FEATURE-BASED PROCESS PLANNING AND AUTOMATIC NUMERICAL CONTROL PART PROGRAMMING

by

Chih-Cheng Ho

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

The University of Utah

December 1997

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Chih-Cheng Ho

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Richard F. Riesenfeld

Elaine Cohen

Samuel H. Drake

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Chih-Cheng Ho in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Richard F. Riesenfeld
Chair, Supervisory Committee

Approved for the Major Department

Bob Kessler
Chair/Dean

Approved for the Graduate Council

Ann W. Hart
Dean of The Graduate School

ABSTRACT

A well-integrated design and manufacturing system should not only provide a rich set of features making it possible to design complex parts with greater flexibility, but also be able to automatically generate numerical control part programs to produce the designed parts. The goal of this research into the integration of Computer-Aided Design and Manufacturing is to support the design, modification, analysis, optimization, and manufacture of mechanical parts with complex shapes automatically and efficiently within a unified framework.

A feature-based approach is used to combine *parametric design* paradigms and *computer-aided process planning* techniques, which also includes freeform curves and surfaces as higher-order features. The approach is to design a part using predefined, but flexible, hybrid design/manufacturing features, and then use the design procedure as the basis of process planning for the production, mainly machining, of the part. Therefore, a part design not only specifies the geometry and functionality of a product, but also provides a plan to automatically generate the NC part programs to machine the specified features, and hence the whole part.

A prototype system for two-and-one-half-dimensional and three-dimensional features has been designed and implemented to demonstrate key components of the research, which includes feature decomposition, tool selection, feed/speed computation, operation reordering, and code generation. Design issues and algorithms for geometric and manufacturing analysis are presented.

To my wife Amy

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank all the persons who made the completion of this work possible.

I thank my committee, Rich Riesenfeld, Elaine Cohen, and Samuel Drake, for their advice, support, encouragement, and patience. Rich and Elaine provided a unique multidisciplinary research environment and generous funding for my research. Elaine patiently helped in discussing the details of my research, from theory to proof and from algorithm to implementation. She also contributed enormously in reading and polishing this thesis. Sam has been a valuable resource to my research, and is always available when I needed his help. He provided many suggestions that improved my algorithms. He also helped in testing many of my algorithms and provided some of the examples used in this thesis.

I thank people in the Alpha_1 research group, staff and students, for their friendship and effort in maintaining a pleasant research environment. I have learned a lot from working with them. Special thanks go to Beth Cobb and Russ Fish for their help and advice in numerous problems and questions.

Last but not least, my deepest gratitude goes to my family, especially my wife Amy, for their willingness to endure the endless expectation and their unwillingness to let me quit. Without their sacrifice and support, it would not have been possible for me to complete my thesis.

This work was supported in part by DARPA (N00014-92-J-4113) and the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

Design and *manufacturing* are two major components in the engineering aspect of production [41]. A product design that cannot be realized through manufacturing processes is not a good design. On the other hand, manufacturing processes cannot be effective without a thoughtful design and plan.

Traditionally, design and manufacturing are treated as two separate stages and usually handled by two distinct groups of people. The design group often does not anticipate the manufacturing implications of its decisions. After the detail design is completed, it is passed onto the manufacturing group as annotated engineering drawings. The manufacturing group then has to determine a way to make the parts based on its interpretation of the design group's drawings, which may not be the same as the designer's intent. It usually takes a few passes back and forth between the two groups in order to reach a satisfactory part. Each pass may also mean a few scrapped parts and some retooling. Moreover, the design could be modified frequently, sometimes even before an acceptable part is made.

A production cycle like this is a slow and costly process. It is clear that to reduce the time and cost, it is important to achieve a good integration of design and manufacturing which provides a common language for both sides in which they can interact with each other and also anticipate design changes and reflects them to the manufacturing processes. The goal of this research into the integration of computer-aided design and manufacturing is to support the design, modification, analysis, optimization, and manufacture of parts with complex shapes automatically and efficiently within a unified framework.

1.1 Integration of Design/Manufacturing

The idea of using computers and electronic technologies in design and manufacturing [151] can be traced back to the late '50s and early '60s, when Numerical Control (NC) machines and computer graphics were first introduced [26, 50]. Early applications were mainly in computer-assisted NC part programming and computer-aided drafting.

With improved computer power and better understanding of its usages for design and manufacturing, Computer-Aided Design and Computer-Aided Manufacturing (CAD/CAM) systems [57] were used to allow interactive design of geometric models and direct or indirect use of the model data in manufacturing processes. Using a CAD system, a part design can be created, modified, analyzed, or optimized interactively and graphically on a computer. This increases the productivity of the designer, improves the quality of design, and creates a database for manufacturing.

To produce such a part, computers can be used in direct monitoring and control of the manufacturing processes or "*off-line*" to provide plans, schedules, instructions, and information by which the production resources can be managed more effectively [23, 55, 114]. The advantages of CAM include efficient management, control, and operation of manufacturing facilities, improved product quality, accuracy, and flexibility, reduced manufacturing lead time and cost, etc. Related applications in CAM, besides NC machining and programming, include inspection, planning, material handling, robotics, assembly, etc. [151].

To further increase the degree of automation in production processes, Computer-Integrated Manufacturing (CIM) was introduced [56, 122]. CIM includes not only the engineering functions from CAD/CAM, but also the business aspect of a product, such as marketing, accounting, management, etc. The CIM concept is that all of the operations related to the production could be incorporated in an integrated computer system to assist, augment, and automate the processes. The same information could be shared throughout the entire life cycle of a product.

CIM addresses the needs for communication, data exchange, and management between design and manufacturing, as well as other aspects of a product. It integrates design and manufacturing by shortening the transitions between them. Existing CAD/CAM systems are made to work together through data format conversion. Internally, each application works independently and uses its own data representation. A typical model of CIM implementation is a star-shaped structure with a network database as the central node (see Figure 1.1).

One problem of this implementation is that each of the existing subsystems usually uses a different data format or needs some information that is not available from other subsystems. Data format conversion between each subsystem and the central database is often required. Standard data formats, such as Initial Graphics Exchange Specification (IGES) [131], are used to store models in the database. Applications need to import/export data from/to IGES files to/from their internal representations. Sometimes, manual augmentation of the product design is applied to obtain extra information required for a particular application.

For example, the CAD design may contain only points, lines and arcs in the model data while the CAM system needs to know the axis/length/width/depth of a slot which are not available from the part design explicitly. Such information is reconstructed from the CAD drawings/data by the manufacturing department. The reconstruction procedures are usually not automated and may create different interpretations due to incomplete or inconsistent information. This heterogeneous

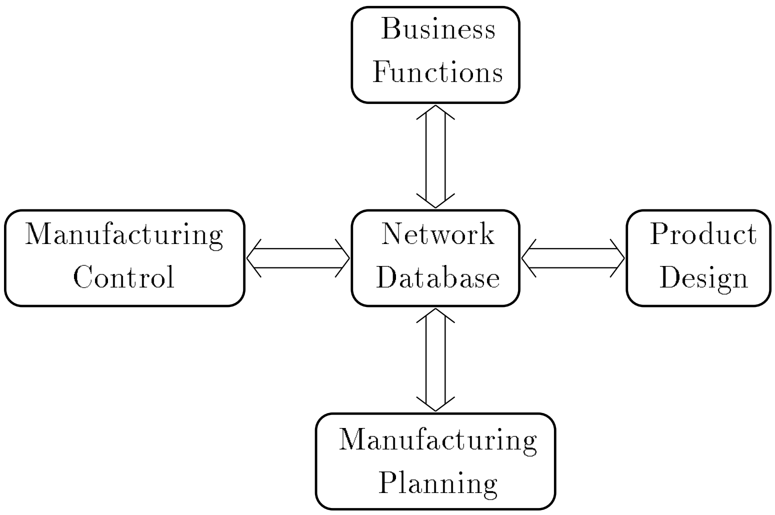


Figure 1.1. A typical model of CIM implementation

approach can easily integrate various existing components. It does, however, build barriers against a full implementation of the CIM concept.

Newer approaches to the integration of design and manufacturing processes use techniques of *design-by-manufacturing* and *feature-based process planning* [27, 20]. They provide an integrated environment and framework for *concurrent* product design and manufacturing process planning. The term "*concurrent*" means to consider both design and manufacturing issues at the same time, instead of as two sequential stages. A part design is composed of a sequence of machining form features, such as holes, slots, counter-bores, etc. A designer could design a part by issuing "virtual" machining commands to subtract features from a given stock. Interactive graphical feedback is provided to display the intermediate result after each command. The same design procedure is also used as a process plan to machine the part. A process planner is then used to improve the machining efficiency by decomposing features into simpler machining operations and reordering them to reduce tool changes [27, 139].

The design-by-manufacturing approach addresses the problems of manufacturability and inconsistency of designs often seen in the CAD/CAM approach [20]. However, due to the small number of supported features, such as holes and simple pockets, which have direct mapping to the manufacturing operations, these systems can handle only a restricted class of parts. Therefore, the design-by-manufacturing approach limits the freedom and flexibility of the designer and the design process. Another problem is the need to make detailed low-level manufacturing decisions early in the process in order to design by manufacturing operations.

1.2 A Feature-Based Approach

A well integrated design and manufacturing system should not only provide a rich set of features so a designer can design complex parts with greater flexibility, but also be able to generate NC part programs to produce the designed parts automatically. Most existing CAD/CAM systems support only a limited set of design and manufacturing features using mostly two- and two-and-one-half-dimensional geometry, such as holes, slots, pockets, etc. Also, the process planning and manufacturing support is incomplete and requires extensive manual intervention and specification.

In this research, we generalize the feature-based approach to the integration of design and manufacturing to include sculptured surfaces as higher-order features, which combines both *parametric design* paradigms and *computer-aided process planning* techniques. To encourage the idea of "design *for* manufacturing," which is not equivalent to "design *by* manufacturing," the restrictions and limitations mentioned earlier must be removed. The compromise is to elevate the ability and flexibility of "what can be manufactured" to an equivalent level of "what will be designed," without lowering the latter too much. We extend the manufacturing features and operations to cover "*freeform features*" as well. Detailed manufacturing decisions are deferred until necessary and are made automatically as much as possible. Along with process simulation, these automatic decisions can give the designer a form of feedback for alternatives in the part design which may improve the efficiency of the manufacturing processes. With a focus on machined parts, this means being able to generate NC part programs directly and automatically from each of the design/manufacturing features. Machining parameters, such as tool selection, feed and speed computation, etc., are determined automatically.

This paradigm provides an integrated environment and framework for concurrent product design, planning, manufacturing, and inspection. A part can be designed with, or decomposed into a hierarchy of known entities, called *parametric features* (see Figure 1.2). Each feature is specified using a set of geometric and numeric parameters. A parametric feature type represents a class of objects that has similar geometric functionalities and can be machined using similar operations. A feature can be decomposed into subfeatures, or included in another higher-level composite feature. Such feature-based hierarchical decomposition is then used as a basis of process planning for producing the part.

These hybrid design/manufacturing features not only define the geometrical specification of the design, but also provide a set of expert agents to automati-

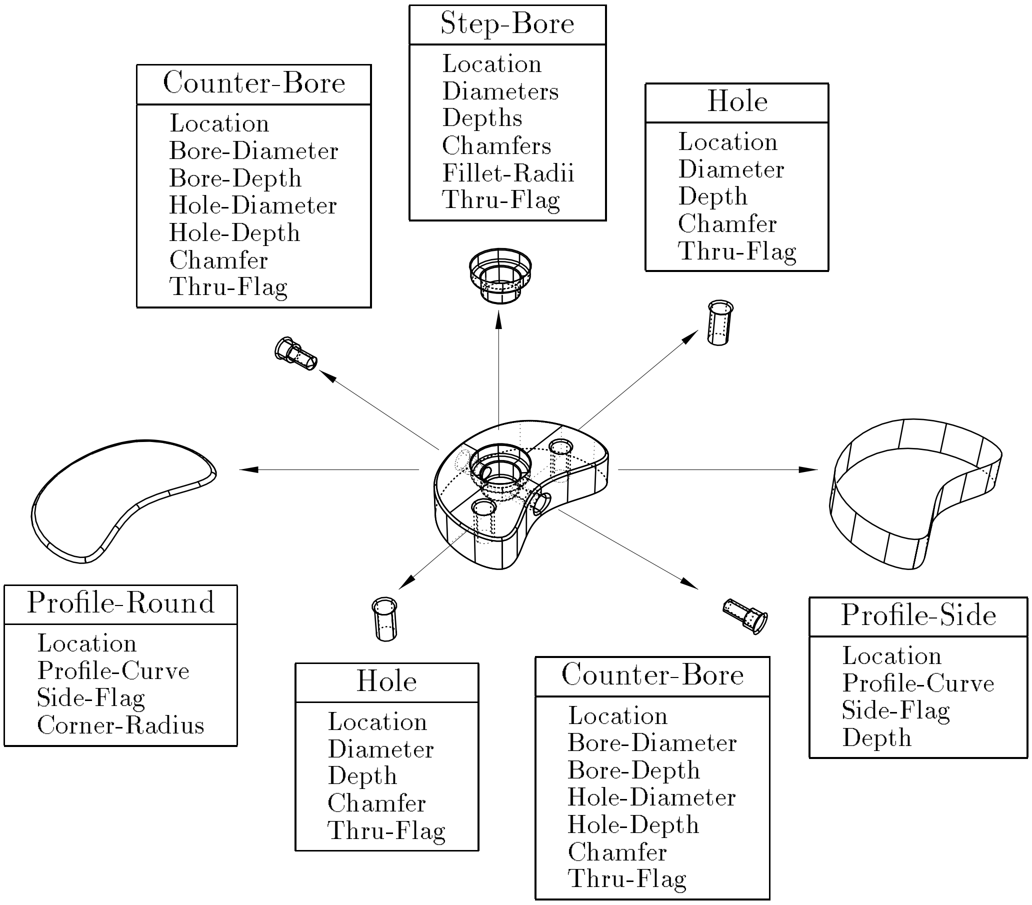


Figure 1.2. A part designed with parametric features

cally determine the machining parameters and generate NC part programs for the designed part. Manufacturing information, such as tolerance, fixture and machine setup, could also be incorporated into the design hierarchy. For example, in [141], an automated part inspection system was presented to inspect parts designed and manufactured using the same feature-based part specification. By including generic freeform curves and surfaces as "parameters," higher-order parametric features can be used to design and manufacture parts with complex shapes, such as a volume bounded by freeform surfaces.

1.3 Process Plan and Part Program

The main focus of this research is to automatically generate the required instructions, i.e., part programs, for the machining of mechanical parts designed using the above feature-based approach on NC milling machines. A part program is defined to consist of the tool path, i.e., locations of the tool movement, and also the machining procedures and decisions of what tools to use and how fast they move [25]. These decisions are usually referred to as (part of) the process plan of a part [23, 144].

For example, to machine the part shown in Figure 1.2, the following issues must be addressed:

- What material to use,
- What manufacturing processes to use,
- What operations to use for each process,
- What the order of operations sequence should be,
- What the setup and tooling requirements for each operation should be,
- What the operating conditions are, and finally
- What the detailed operation instructions are.

Within the proposed feature-based design/manufacturing framework, some of the questions are considered or planned as part of the design. For example, the issues regarding fixturing and feature interaction ordering of setups are assumed to be resolved elsewhere and are available in a priori hierarchical structure.

One advantage of having a feature-based hierarchical part description is the ability to use the same design across multiple applications in design, analysis, planning and manufacturing. The selection of features and their ordering in the hierarchical structure can be used as an initial sequence of operations. This initial plan can then be refined and optimized through feature decomposition and operations reordering. In generating the part program, decisions of what tools to use and how fast they should move are as important as deciding the path that the tools should follow. In fact, these decisions influence how the tool path should be generated. Moreover, performance of the part program and quality of the machined part also depend on the machining parameters chosen for each operation, especially when machining hard materials.

The ultimate goal of automatic process planning and NC part programming is to automatically generate sequences of machining operations and commands, including fixtures, that produce the designed part *economically*, *efficiently*, *accurately*, and *safely*. Using a feature-based approach, the size of this problem can be reduced by decomposing the part design into features, then handling the features one at a time. We can define the problem as:

Given: A feature-based part design.

Determine:

1. The *decomposition* of features for machining this part.
2. The sequence of operations to make each feature.
3. The tooling requirements to machine each feature.
4. The detailed NC commands for each feature.

Previous research on this problem handles mainly parts designed using simple machining features, such as holes, slots, counter-sinks and counter-bores [27, 20, 128, 139]. For more complex features, such as pockets with arbitrary profiles or freeform surfaces, direct symbolic decomposition into machining operations cannot be done. Complex geometric analysis and reasoning must be employed on each feature to determine a proper decomposition. To derive the NC part programs requires more sophisticated algorithms.

By generalizing the definition of a "feature" to include freeform shapes, a part design can be thought of as a single feature that is decomposed into simpler subfeatures. Further, if the definition of feature is extended to include low-level NC operations, the process of NC part programming and tool path generation can also be viewed as a different form of subfeature decomposition. Design and manufacturing could be integrated using such a unified feature-based approach, and it could also provide a uniform definition for feature based on a recursive subfeature decomposition.

Given: A "feature" (the part design).

Determine:

1. Can this "feature" be machined?

2. If it can be, what is a good NC part program and setup to machine it?

3. If it cannot be, what is a good alternative (decomposition)?

The result of this recursive subfeature decomposition would be a list of NC part programs for a machinable part design, or the reasons why a part cannot be made.

1.4 An Overview

In this thesis, we use a feature-based approach to process planning and NC machining of two-and-one-half-dimensional ($2\,^1\!/_2$ D) and three-dimensional (3D) features defined by freeform curves and surfaces. In Chapter 2, we describe the framework of our feature-based process planning and NC machining system. Key components of our system include feature decomposition, tool selection and feed/speed computation, operations reordering, and NC code generation. We describe process planning strategies for $2\,^1\!/_2$ D and 3D milling features.

In Chapter 3, we discuss some requirements of NC part program and automatic NC part programming. We present computational algorithms to compute tool paths of features defined by freeform curves and surfaces. A fast offset tool path clipping algorithm based on binary-weighted curve skeleton is developed to remove unproductive tool cuts for semi-open regions defined by closed or open profiles. By removing unproductive tool path, we improve the efficiency of the automatically generated part program of an "*open pocket.*" The open pocket feature is also used as the foundation for rough cutting 3D features. Also, we present improvements to existing 3D surface contouring tool path generation algorithms.

In Chapter 4, we investigate new technologies to improve the machining efficiency for features with small corners and details. We develop a tool-driven decomposition strategy to machine a milling feature that uses multiple tools of different diameters. The fast tool path clipping algorithm described in Chapter 3 is utilized to produce efficient tool paths for using multiple tools. We present methods and heuristics to find the number of tools and the combination of their diameters such that the machining time of a given feature is minimized. The improved tool paths can reduce the total machining time by more than an order of magnitude, depending on the feature geometry.

We introduce an analysis tool for efficiency called "cut index" of tools. Tool cut index is a dimensionless form which has strong correlation to the machining time of a tool and can be computed efficiently without generating any tool paths. Definitions, properties, and algorithms for cut index analysis are introduced. Further, we present optimization algorithms for the combination of tools which has the minimum total cut index. We also extend cut index analysis to the whole part as well as 3D features. By minimizing the global total cut index, we find a set of tools that can cut the entire part efficiently.

In Chapter 5, we define constant distance offset of curves and surfaces, which is the basis of generating gouge-free tool paths. We present algorithms to compute constant distance offsets of a surface by first offsetting the surface in its normal directions, then trimming away any self-intersections in the offset surface. An adaptive minimum-cover refinement strategy is created to reduce the number of new knots or new control points to improve the accuracy of the approximation. Offset surface methods for tangent discontinuous edges and corners are computed to fill the gaps between the offset surfaces. We also present a new modeling technique based on stereographic projection to model arbitrary spherical patches bounded by arcs of great circles. Examples of using these offset techniques in solid modeling are presented.

In Chapter 6, we present definitions and properties of curve and surface self-intersections. We derive and mathematically prove necessary conditions for self-intersecting curves and surfaces. Using these conditions, we modify existing surface/surface intersection algorithms to find the self-intersection curves efficiently.

Algorithms to classify self-intersection curves are presented that trim away surface self-intersections. By trimming away self-intersecting regions of an offset surface, the machinable regions of a 3D feature of a given tool are computed. Surface area of these regions is used by tool cut index analysis in the tool-driven decompositions of 3D features presented in Chapter 4.

Finally, in Chapter 7, conclusions and contributions of this thesis and directions for future work are discussed.

CHAPTER 2

FEATURE-BASED PROCESS PLANNING

As stated earlier, to have an effective manufacturing process, a designer should consider more than just the functionality and geometry of the part. The goals of concurrent engineering require tight interaction between the design, planning, and manufacturing processes in order to evaluate the manufacturability of a part design. Geometric entities, such as points, lines, arcs, etc., are not enough to describe the part design in such an environment. Rule-based process planning systems usually use a symbolic description of a machined part as a collection of machining features [99, 139]. A unified representation that can facilitate solutions across different applications in design and manufacturing requires a representation that encapsulates the geometric properties for design and analysis, and also exposes the important constraints for planning and manufacturing. This eliminates the need for "lossy" conversion between systems and processes, which is usually not an automatic chore.

This chapter provides the background and motivations of why we use a feature-based approach to process planning and machining. We introduce advantages of using features directly in part design and present different methods to automatically obtain a feature-based part design. Then, we present key components of a feature-based framework for process planning and NC machining. We use rapid prototyping solid freeform fabrication (SFF) processes as counter-examples to the machining processes. In SFF, answers to the questions and requirements of process planning discussed in previous chapter are simple and straightforward. For machining processes, we use a divide-and-conquer approach, which decomposes a feature-based part design into a list of simpler machining operations. We present strategies for the decomposition of $2\,^1\!/_2\,$D and 3D milling features. Examples of feature decomposition are presented. Details of the computation and algorithms for the planning and machining of milling features are discussed in later chapters.

2.1 Feature-Based Design

One advantage of having a feature-based description of part design is the ability to modify and reuse existing designs more easily. Modification of such a design can be done by adding, deleting or modifying features. This type of modification is more tractable than directly changing the underlying geometric description and representation [24]. Changes are either confined to being within the modified features, such as changing the corner radii of a square pocket, or be between related features and features that intersect with the modified ones, such as modifying the distance between two holes or changing the depth of a pocket which has a hole feature at its bottom. Reuse of designs could be achieved by cut-and-paste on a per-feature basis, or by a group of features, to a new design, or even to a different place in the same design. A result of easy design modification and reuse would be the ability to create variants of similar designs rapidly, by changing one or two parameters. This could also provide a database for future design and manufacturing, as well as catalog parts, such as bolts and bearings, where the designer would simply choose existing part components by the specified standard parameters [5, 110].

From software engineering point of view, using a feature-based approach is like the *object-oriented programming* techniques in software design, which provides a uniform interface and good extensibility to the system. Adding new features or combining existing ones for different design and manufacturing purposes could be integrated into the object-oriented interface without changing other parts of the system. It is easier for a designer or an engineer to use a system if the underlying representations and algorithms are encapsulated into commonly used terminology through a feature-based interface. A user would not have to change existing designs or the way one uses the system when modifications are made to the system, while the efficiency or robustness of the results might be significantly improved.

Although specific design procedures are not part of the theme in this thesis, it is important to understand how one could get feature-based designs for planning and manufacturing. The ability to automatically obtain feature-based descriptions from part designs could be a key component to the success of integrating design and manufacturing systems. Two predominant methods to automatically obtain feature-based descriptions from a CAD design are:

1. Automatic feature recognition and extraction.

2. Design by features.

Many solid modeling systems describe parts by their geometric shapes. With automatic feature recognition and extraction, manufacturing features are determined from the part geometry automatically. For example, intrinsic geometric features, such as corners, edges and faces, can be detected from the solid model and grouped based on their spatial relationship and connectivity. Then, within each group, relationships, such as parallel edges and faces, concave and convex edges and their adjacent faces, loop of edges, etc., are analyzed and some sort of adjacency graph is built [28, 53, 71]. By applying pattern-matching and ruled-based recognition techniques, such as *subgraph isomorphism* [11], machining features such as holes, slots or pockets can be extracted. To handle features that intersect with each other, matched features are removed from the graph immediately and the remaining graph is reanalyzed.

Another feature recognition strategy is *volumetric decomposition* [38, 49, 74, 106, 150] of the boundary representation model. A part model is decomposed into a Constructive Solid Geometry (CSG) tree of convex components with boolean set-difference operations. This decomposition is called the *alternating sum of volumes* decomposition because the sign (positive v.s. negative volume contribution) of components changes between each consecutive set-difference operation. Such decomposition can then be converted into subtractive or additive form feature entities using various combination methods on adjacent components of opposite or same volume contributions. Finally the combined entities are classified into different form features and a form feature decomposition of the original part design is obtained.

Although it may be the only way to automatically obtain feature-based descriptions for existing designs created by geometry-based CAD systems, feature recognition and extraction is a very hard problem, especially for complex real world part designs. The assumption that features can be determined by geometry alone is not always true. Different manufacturing processes or features may result in very similar geometric shapes. Nongeometric attributes, such as surface finish and feature specific machining parameters, are hard to specify and retrieve from a geometric model. Intermittent features, such as holes and tabs for fixturing, which are transient and exist only during machining processes will not be retrieved at all.

"*Design by features*" [36, 77, 127] has been defined to mean to design parts directly using features and to generate the geometric model from the features. Once the design is done, then, there is no need to recognize or extract features from the geometry. For planning and manufacturing purposes, predefined features that can be translated into manufacturing operations are used to subtract volume from a raw stock [35]. However, because only a small set of features have direct mappings to manufacturing operations, there is not enough flexibility for the designer in many existing systems that uses this approach. Another problem with these systems is that they require the user to fill in manufacturing details while designing, instead of concentrating on the functionality and geometry of the designed part. This makes them hard to use in the early, more preliminary, stages of design. However, depending on how much detailed information can be filled in automatically, the need for a designer to think about manufacturing issues while designing is not a totally bad thing. The problem of multiple interpretation of features for a given design still exists [72].

Despite of these problems, features are more powerful design tools than low-level geometric primitives. Feature-based CAD systems are more intuitive to learn and use, and usually take less time to create a design. Various advantages of using features in design and manufacturing have been explained earlier. Our approach is to provide more manufacturing capabilities to a richer set of design features and to build intelligent translations between design and manufacturing features. Therefore, the integrated system could provide a set of hybrid design/manufacturing features that are both flexible for design and powerful for manufacturing. The hope is that the designer has to give only one reasonable feature-based interpretation to start with, which will then be transformed and improved by the process planner to an equivalent but more efficient list of features.

The feature-based framework could be extensible for a particular design and manufacturing job if necessary, such as part assembly and inspection [40, 141]. Although we prefer the feature-based design approach, and used features to prepare most of our examples, the algorithms and analysis techniques presented in this thesis for process planning and NC machining can be applied to any part design with a hierarchical feature-based representation, whether it is extracted from the part geometry, augmented from an existing design, or constructed using a hybrid of different methods.

2.2 Process Planning

Before a part design can be realized through manufacturing processes, one must answer at least two questions: "Can it be made?" and "How should it be made?" The processes of determining the sequence of operations for manufacturing a part from its design is called "*process planning*" [23, 41, 144]. Process planning is one of the first activities which bridge design and manufacture. There are two types of computer-aided process planning systems: variant process planning and generative process planning [23, 144]. In variant process planning, existing parts and designs are categorized into groups of families. Using group technology [54], parts are coded and classified by their similarity. Process plans and pregenerated part programs are stored in a database, which can be retrieved in the production stage.

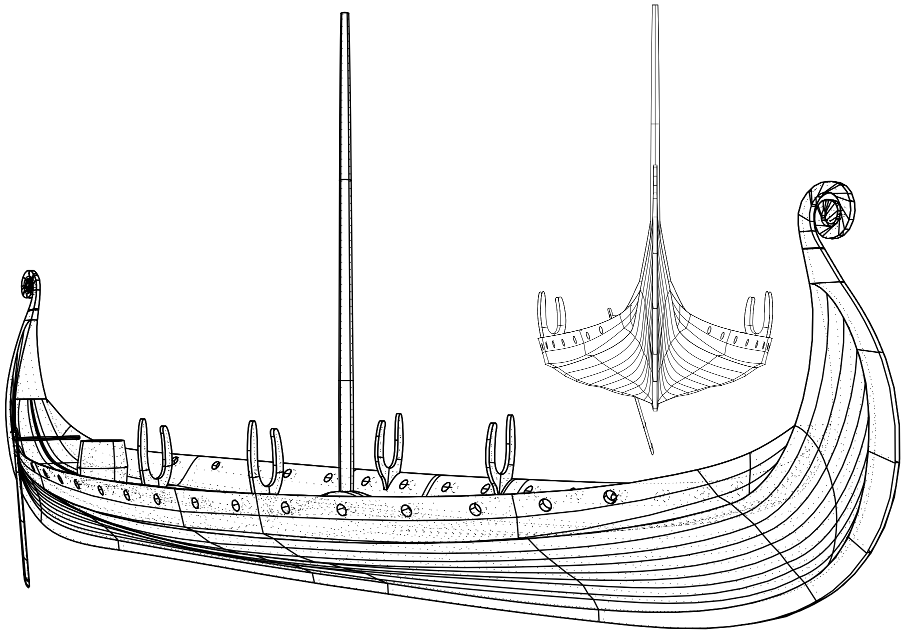
In generative process planning, process information of a part is used to construct a process plan which optimizes the manufacturing processes and generates the required part programs to machine the part. In this research, we use a feature-based approach in which the process information as well as the part design itself are described in a hierarchical structure of features. Before discussing our feature-based process planning approach to the machining processes, we present examples of new rapid prototyping processes that have significantly different requirements for planning and manufacturing.

2.2.1 Solid Freeform Fabrication

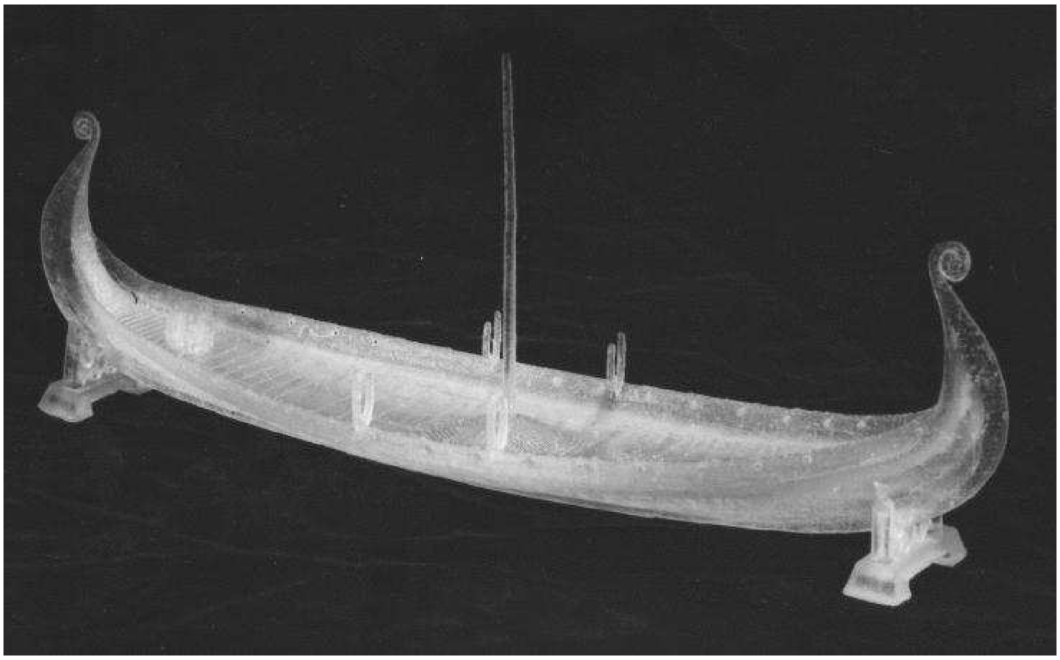
Some of the planning decisions that need to be made in order to produce a physical part from a given design are listed in Section 1.3. Answers to these questions can be very simple if certain rapid prototyping fabrication processes are used. Several new rapid prototyping systems utilize the layered technologies and can produce solid parts directly from CAD 3D solid models [8, 149]. These processes are usually called *Solid Freeform Fabrication* (SFF). A CAD solid model, represented as a collection of 3D triangular facets, is first sliced into horizontal layers, then each layer is formed and glued together, which usually starts from the bottom layer. Since the layered technology constructs parts one layer at a time, it can create parts with undercuts, voids, or even interlinked assembly as long as each layer can be supported and secured in place.

For example, Figures 2.1 and 2.2 show CAD solid models and their stereolithography polymer parts for a Viking ship and a B1-B bomber assembly, respectively. We model both parts using nonuniform rational B-spline (NURBS) surfaces and set boolean operations supported by the Alpha_1 solid modeling system [5] developed at the University of Utah. To manufacture the parts, the trimmed NURBS surfaces which form closed solids are first approximated with triangle facets using adaptive subdivision techniques [33]. Then, we convert the triangulated solids into the STL format [2], an industry-standard for SFF facet data, and use the 3D Systems' slice program to produce the SLI slice data.

The Stereolithography Apparatus (SLA) by 3D Systems [1] drives a laser beam on the top of a vat of photo-sensitive liquid monomer and cures the liquid into solid, i.e., polymerizes the monomer, wherever the laser draws. To form a solid layer, the

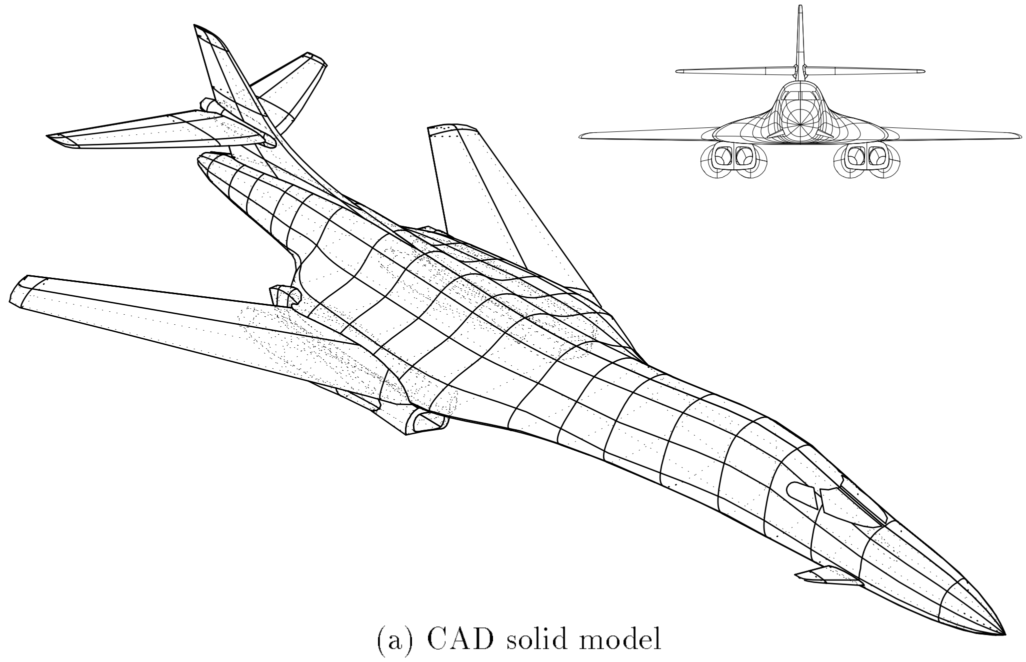


(a) CAD solid model

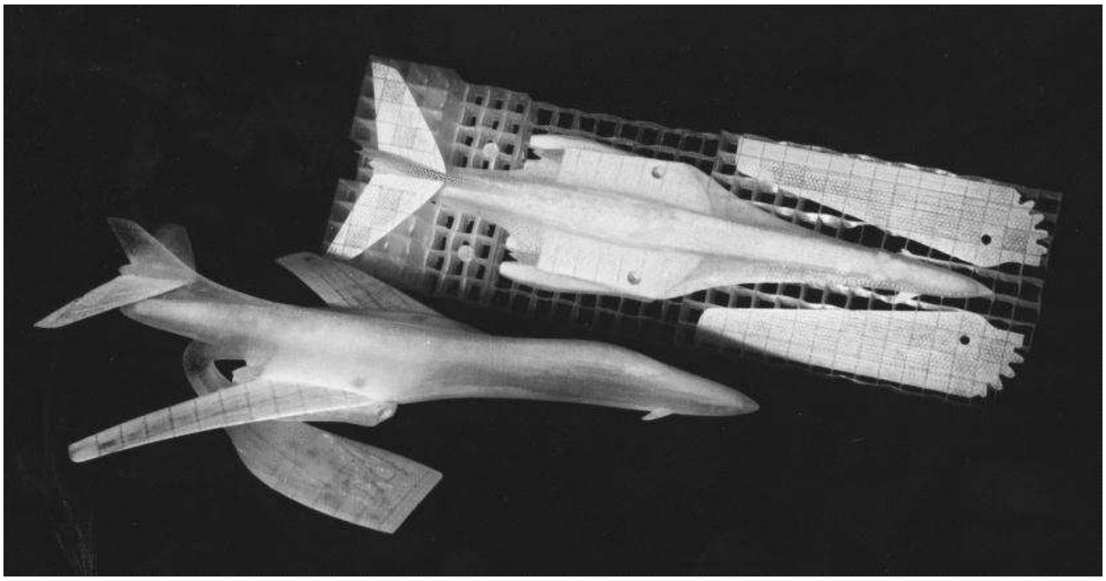


(b) SLA polymer part

Figure 2.1. A Viking ship CAD model and its SLA part



(a) CAD solid model



(b) SLA polymer part

Figure 2.2. A B1-B bomber CAD model and its SLA part

layer contours are hatched with crossing or weaving lines that cover the part area. At the completion of each layer, new material is added on the top for the next layer by lowering the cured layers and letting the liquid monomer flow over the cured material. Since a new layer is built on the top and bonds to the previous layer, a solid part is created in the vat surrounded by uncured liquid after the top-most layer is completed. The Viking ship and the B1-B bomber example parts shown above were built on an SLA-250 machine in the Advanced Manufacturing Laboratory at the University of Utah [4], and each took from 11 to 14 hours to build.

Other SFF systems use different techniques to form the layers and to bond them together. For instance, the Selective Laser Sintering (SLS) machine by DTM Corp. uses a laser to fuse layers of heat-fusible powder, such as wax, thermoplastic, or metal, together, and the Laminate Object Manufacturing (LOM) machine by Helysis uses a laser to cut sheet material, such as paper, into shapes of the part cross-sections, and then laminates them into solid parts. Compared to traditional manufacturing methods, SFF technologies require less lead time in making solid prototypes of complex parts and can automate their processes from the CAD solid models, which is good for visualizing and verifying the CAD designs. However, there are still limitations in current SFF technologies for the production of real functioning mechanical parts. The main limitations of SFF are in the material properties, the processing time, the part accuracy, and the operation costs. Although there have been improvements in many aspects of SFF [9, 67, 98], especially in operation procedures and part accuracy, traditional manufacturing processes such as NC machining are still the mainstream methods for production manufacturing of parts and prototypes.

2.2.2 Feature-Based Process Planning

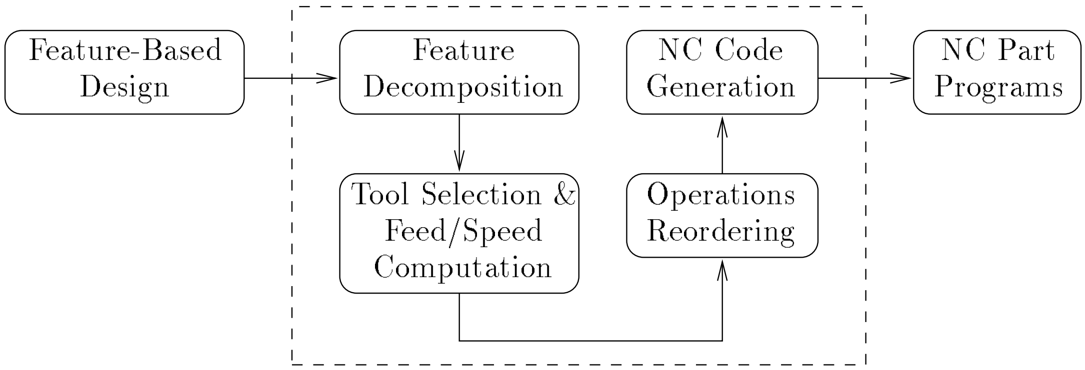
With a feature-based hierarchical description of the part design, many planning decisions could be made based on individual features or groups of features. Using a feature-based approach would allow us to automate or semiautomate the processes from design to manufacturing. By semiautomatic, we mean that once the initial planning/manufacturing decisions have been made, the requirements of adjustment to accommodate design modifications or refinements to improve the effectiveness of manufacturing operations could be achieved systematically with little or no human intervention.

From [27, 139], a straightforward feature-based process planning system for hole features consists of four sequential steps: 1) feature decomposition, 2) tool selection and feed/speed computation, 3) operations reordering, and 4) NC code generation (see Figure 2.3(a)). For parts consists of hole features, such as drilled holes, counter-bores, and counter-sinks, the features are first decomposed into lists of point-to-point operations, such as spot-drill, drill, ream, bore, or chamfer. Then, tools are selected from a tool library and assigned to each of the operations, and appropriate feed and speed are calculated for each tool. In step 3, the low-level operations are reordered to minimize fixture and tool changes, as well as to reduce machining tolerances for features that intersect with each other. The last step is to generate machining commands for each machining operation. It is important that the partial ordering of the operations within each feature are maintained during the reordering stage. For example, the spot-drill operation should come before the drill operation, and the ream operation must come after the drill operation. NC code generation for hole features is straightforward since most CNC controllers support canned cycles, i.e., subroutines for point-to-point operations.

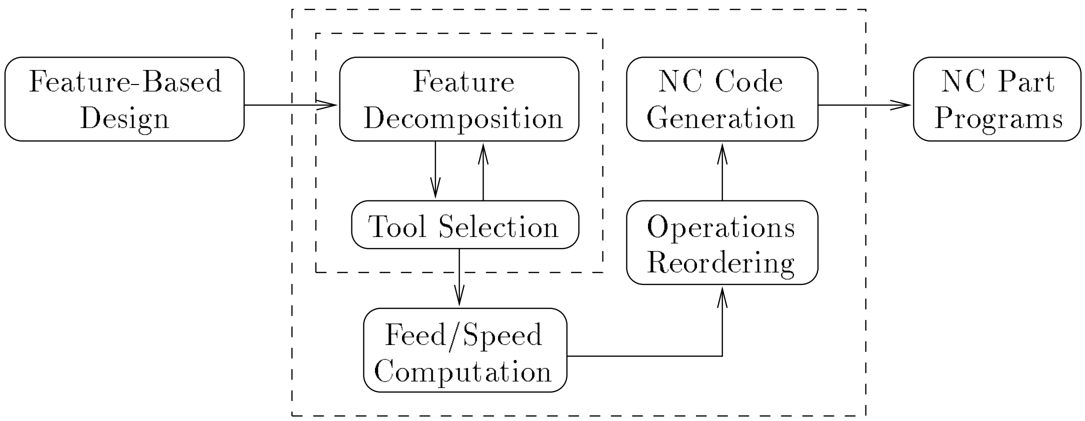
Decomposition of hole features depends mainly on the type and tolerance requirements of the feature [27]. Results of tool selection normally do not affect the decisions made in the decomposition stage. However, for $2\,{}^1\!/_2$ D and 3D milling features, these two steps are more closely related to each other. The sequence of steps outlined above for hole features would not be suitable for milling features. In milling operations, a large tool is stronger and can cut deeper at a single time. Therefore, in decomposing a pocket into layers of 2D roughing operations, the depth of a layer depends on the diameter and length of the roughing tool. That is, it requires fewer roughing passes if a larger tool is selected. The same situation occurs in 3D milling features as well, where the selected tool diameter affects the depth and total number of roughing layers, and also also can change the shape of each layer if the surface has different cross-sections at different Z levels.

In this research, we extend feature-based process planning to encompass milling features by using a tool-driven decomposition scheme (see Figure 2.3(b)). We first perform geometric analysis to compute the tool size requirements and constraints from the milling feature. Then, we decompose the feature using the results of a preliminary tool selection based on available tool sizes in the tool library. Finally, specific tools are selected from a tool library for the decomposed operations.

Besides being able to determine the correct depth and number of roughing layers, the tool-driven decomposition also provides several advantages over the sequential planning used in other systems. The first advantage is the ability to decompose



(a) Sequential steps for hole features



(b) Tool-driven decomposition for milling features

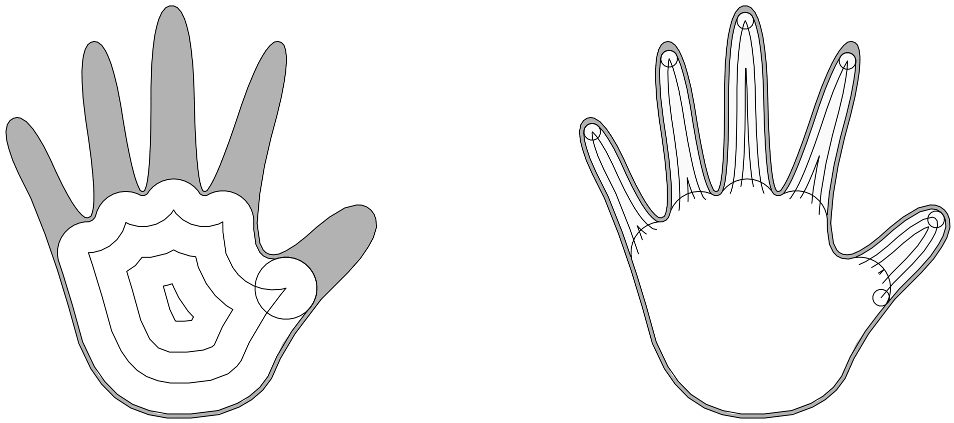
Figure 2.3. Feature-based process planning

a feature into different types of operations depending on the availability of tools in the tool library, which could be configured to reflect the actual configuration of a machine shop. For example, the decomposition of a step-bore feature which is basically a cascade of circular pockets with optional boring operation for tighter diameter tolerance would normally use pocketing operations with end-mill tools and chamfer operations with chamfer tools. However, in the case of small bore diameters, a chamfer tool may not be available, since it has an inverted trapezoidal shape and can fit only if the bore diameter is larger than the inner diameter of the chamfer tool. An alternative for the step-bore feature decomposition would be to use a counter-sink operation and tool, in a way similar to the chamfering of hole features.

This type of tool-driven decomposition to automatically use different types of tools and operations could benefit hole features as well. For instance, a hole with an odd diameter might be decomposed into point-to-point operations that later could not find any suitable tool. Using a sequential approach to process planning, the user is required to change the hole feature into a milling feature, such as a through pocket or step-bore, that uses an end-mill tool. Tool-driven decomposition could automatically perform the conversion or add an additional milling operation. This makes editing a design much easier, and also can generate a more efficient tool path by using more efficient operations whenever the tools are available.

Another advantage of tool-driven decomposition is the possibility of using multiple tools to optimize milling operations. To generate a gouge-free tool path that cuts a feature completely, the selected tool must be small enough to access the smallest corners or the narrowest passageway and details of the feature. However, a smaller tool would require more passes to machine a given feature, and therefore, is less efficient to use. For features that have big areas that can be machined more efficiently with a big tool, but have small corners and details that require a small tool to cut completely, we have derived algorithms to decompose the feature into regions based on the accessibility of different tool diameters.

For example, Figure 2.4 shows the decomposition of a hand-shaped pocket into



(a) First roughing with a big tool (b) Second roughing with a small tool

Figure 2.4. Optimization of milling operations using multiple tools

two roughing operations. On the left, a big tool is first used to machine away the palm area it can fit. The shaded area represents the regions uncut by the big tool. On the right, a small tool is then used to cut the remaining fingers area. In both operations, a small region around the pocket boundary is left uncut. This is the allowance for the finish operation to produce a better surface finish and a more accurate part in case of tool deflection from the roughing passes. Note that the selection of tool diameters shown in Figure 2.4 is not necessarily the best combination of tools that can machine the given pocket in the shortest total machining time.

The machining time of a milling feature, excluding the stock/fixture setup time and other operation overhead, has contributions mainly from two factors, the tool overhead time and the actual milling time. Figure 2.5 shows the relationship between the machining time and the number of tools used to machine a milling feature. The machining time is measured as the minimum among different combinations of tool diameters of the same number of tools. The tool overhead time includes tool setup time, tool change time, and plunge/retract time due to the use of multiple tools. Obviously, there will be a higher overhead if more tools are used.

The milling time depends on the total distance of tool movements and the feed

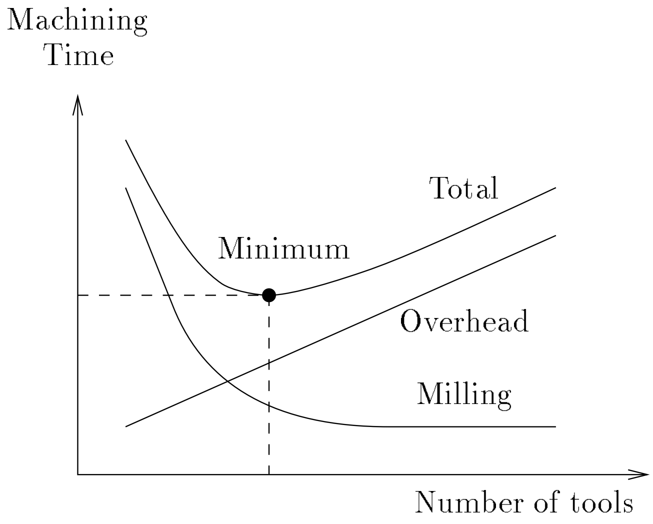


Figure 2.5. Machining time of multiple tools milling operations

rate of each tool motion. Since a larger tool can usually cut faster and requires fewer passes than a smaller tool, using a larger number tools of the "right" diameters could cut the same feature in equal or less time than using one small tool. The sum of both components, i.e., the total machining time, has a minimum value which corresponds to the best number of tools and the combination of tool diameters for the given feature. Details of the geometric analysis and tool path generation algorithms for multiple tools decomposition are presented in Chapter 4.

The approach adopts different methods to find a combination of tool diameters to machine a milling feature in a short time. First, we generate NC tool paths and minimize the simulation time of tool paths of various tool combinations. Then, we define a *tool cut index* to estimate the contribution of different tool diameters. The tool cut index is computed from the geometric shapes of milling features, so therefore, does not require generating numerous tool paths, which could be very time consuming. Unfortunately, the tool overhead time depends on both the features and also the actual configuration and operation of the individual machine shop, a complex issue to quantify. A simplified linear per-tool overhead is adopted to roughly approximate this component.

2.3 Feature Decomposition Strategies

In this section, we present strategies to automatically decompose $2\,^1\!/_2\,$D and 3D milling features into simpler machining operations. Unlike hole features, milling features defined by freeform curves and surfaces require more extensive geometric analysis and reasoning for process planning and part programming. Using the tool-driven decomposition method presented in Section 2.2.2, a feature can be decomposed into multiple stages of roughing and finishing operations. Each operation could use a different tool for better machining efficiency. We introduce the "open pocket" operation to generalize the classification of 2D regions. Techniques for generating the machining commands for each operation are discussed in Chapter 3.

2.3.1 Features

Machining features can be categorized into four groups [5]:

- Hole features: These are the features that can be machined by point-to-point operations, which include plain hole, tapped-hole, counter-sink, counter-bore, counter-drill, etc. Definitions and process planning strategies for hole features can be found in [27, 39].

- Simple milling features: This category includes milling features that can be described in standard forms, such as slot, rectangular-pocket, face, straight-step, etc. Decomposition and tool path generation strategies for these features can be found in many existing feature-based CAD/CAM systems are relatively simple and straightforward [36, 56].

- Complex milling features: These are the general $2\,^1\!/_2\,$D and 3D milling features defined by freeform curves and surfaces, which include profile-pocket, profile-boss, profile-side, profile-groove, profile-chamfer, profile-round, and 3D freeform surfaces. Process planning and part programming strategies for features in this category are the main focus of this research.

- Compound features: Features in this category consist of multiple features oriented in a special relationship, which include rectangular-pattern, circular-pattern, point-list-pattern, and step-bore. A compound feature is first decomposed into a list of subfeatures. Then, each subfeature is decomposed further into machining operations and is treated in the same way as an individual feature.

In addition to milling features, turning features for lathe operations are common in parts with an axial symmetry, such as surface-of-revolution, circular-groove, thread, shaft, etc. A part can be designed using a mixture of different types of features. The feature-based process plan consists of multiple setups which can be manufactured on different machines, such as machining centers, turning centers, or electro discharge machines (EDM). A simple example part with different features is shown in Figure 1.2. More complicated examples are presented in Section 2.4.

2.3.2 Automatic Feature Decomposition

Feature decomposition is the most crucial component in our feature-based approach to process planning and machining. Simple features, such as holes and other point-to-point operations, have well established decomposition strategies to machining canned cycles [27, 99, 139, 144]. Traditionally, complex features such as arbitrary $2\,{}^1\!/\!_2$ D regions and 3D sculptured volumes are treated as special geometric entities to generate machining tool paths in CAD/CAM applications [30]. To expand the vocabulary of feature-based process planning to include freeform curves and surfaces, complex milling features should also be decomposed into simpler machining operations. Conceptually, these operations could be treated by the process planner in the same way as the point-to-point operations.

In general, a milling feature is first decomposed into a roughing and a finishing operations. Then, each operation could be decomposed into subregions using different tool diameters to improve machining efficiency. Finally, each subregion could be decomposed into layers of different Z depths to reduce tool load and to increase the accuracy. We extend the classification of 2D regions from the conventional definition of a pocket [61] to include *open pocket*, *pocket with holes*, and *open profiles* (see Figure 2.6). A pocket, possibly with islands, is usually defined

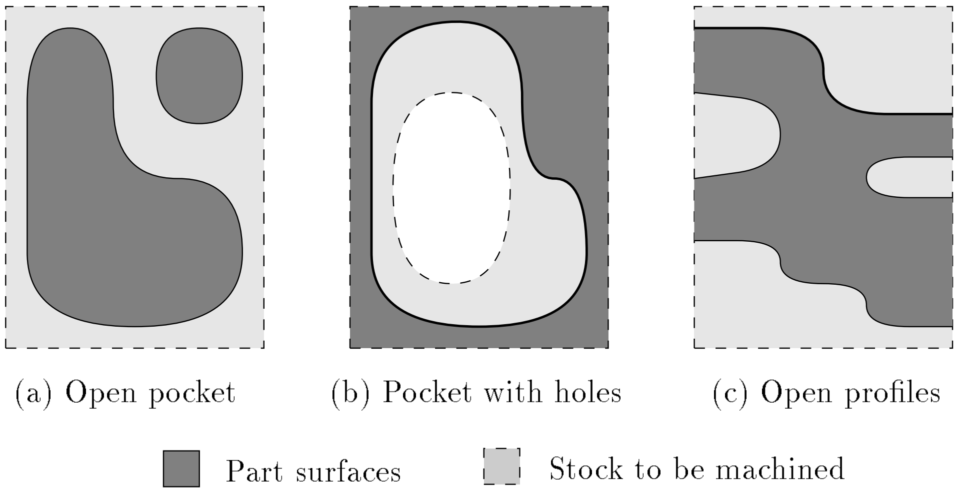


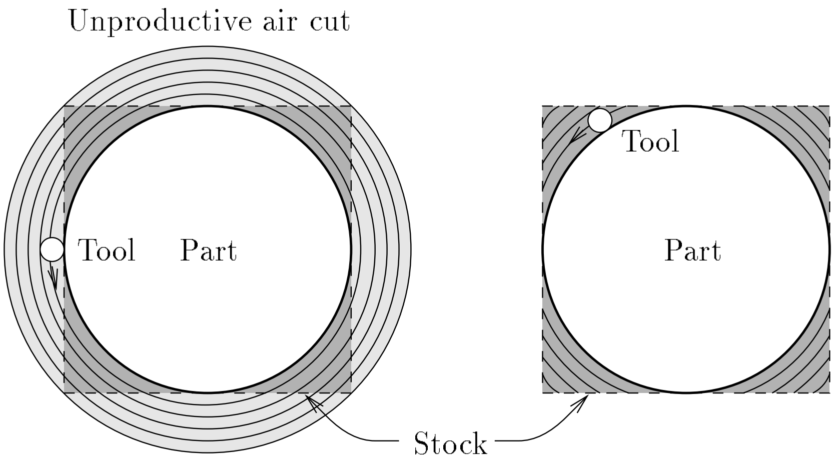
Figure 2.6. Classification of 2D regions

as the inside regions bounded by simply connected contours. The complement of a pocket (for simplicity sake, assuming it has no island) is the outside semi-infinite region. An open pocket can be defined as a finite subset of this open region constrained by the workpiece boundaries. The stock boundary could also occur inside a pocket when there are holes on the stock. We would call this type of pocket a pocket with holes.

With the help of the stock boundary, a user could specify the part boundary by open curves. In this case, part of the region boundaries are part boundary and part of them are stock boundary. This type of region classification with open curves is useful in machining because the stock boundary would not need to be machined. Therefore, the tool path could be more efficient, and it is easier for the user to specify such a feature with different combinations of part profiles and stock profiles. Moreover, the decomposition of features with 3D surfaces could be automated more easily if the surface cross-sections are not required to be closed curves. We decompose 3D features into operations of 2D regions for the roughing stage. Open profiles would make such a task much easier. One precaution should be taken when one defines regions with open curves. Multiple open curves could produce ambiguous classifications of 2D regions that cannot be classified as either interior or exterior. Fortunately, in machining applications of real-world parts, the open regions defined by open part profiles and stock profiles are usually cross-sections of a valid solid part, and therefore, the ambiguous situation should not happen.

Open pocket, pocket with holes, and open profiles, or simply *open pockets*, are more general features than the traditional pocket and profile. Open pockets could be used as a fundamental machining operations for the decomposition and machining of $2\,^1\!/_2$ D features, as well as the roughing of 3D features. In Chapter 3, we introduce a unified algorithm based on the notion of *open skeleton* of curves to generate efficient machining tool paths for open pockets. We present an efficient algorithm to clip the offset tool paths so that the unproductive tool cuts are removed.

Figure 2.7 shows a simple example of cutting a circular boss with successive offsets of the boss boundary. The boss is defined by a unit circle in the middle of a unit square stock. On the left, the effective machining area by successive offsets is $0.5\pi - 0.25\pi = 0.7854$, the region between the outer-most offset and the boss boundary. However, on the right, the actual area needs to be machined is $1.0 - 0.25\pi = 0.2146$ between the square stock and the circular boss. More than half of the machining time could be saved by not driving the tool cutting air.



(a) Effective machining area
$= 0.5\pi - 0.25\pi = 0.7854$

(b) Actual machined area
$= 1.0 - 0.25\pi = 0.2146$

Figure 2.7. Unproductive tool path from successive offsets

As for $2\,{}^{1}\!/_{2}$ D features, we decompose features of 3D freeform surfaces into roughing and finishing operations. For the roughing operation, we extract constant-Z cross-sections of the 3D surfaces to form a cascade of 2D open pockets which could be machined using flat-end milling tools for more efficient roughing. An optional second roughing stage using ball-end tools could remove the wedding-cake-like stair steps resulting from the Z-contour roughing steps. This would produce a more uniform cutting depth for the finish stage, which could result in better surface finish and more accurate parts. The finish stage could also be decomposed into regions that use multiple tools of different diameters to improve the machining efficiency. For surface contouring using ball-end tools, a bigger tool can cut a surface to within a prescribed scallop height in fewer passes, and therefore, is more efficient. On the other hand, a smaller tool can access small details without gouging and produces a more accurate part. Relationships between the tool diameter, the stepover size, and the scallop height are discussed in Chapter 3.

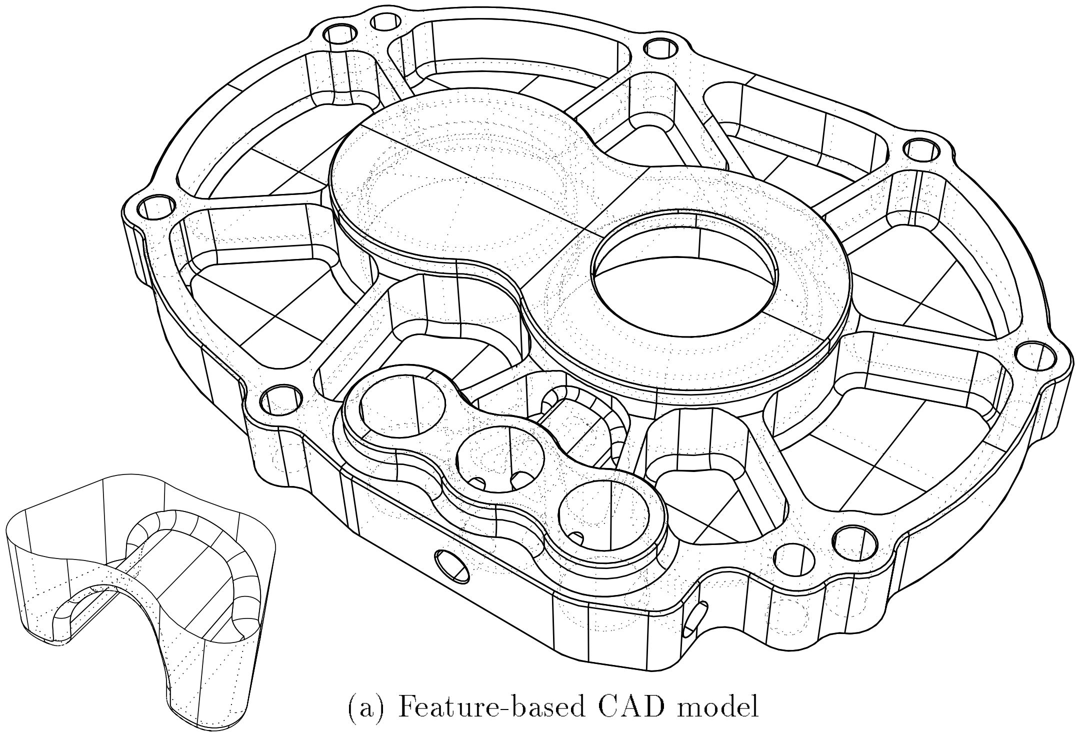
To finish a surface using multiple tools, the ball-end tools would be applied in a decreasing diameter order. Therefore, a bigger tool could machine regions that are machinable by the tool more efficiently. A smaller tool could machine the remaining regions that the big tool cannot cut. Details of the tool-driven surface decomposition and multiple tool selection are discussed in Chapter 4. We compute the tool locations from the constant distance offsets of the part surfaces. Definitions and properties of the constant distance offsets are presented in Chapter 5. Algorithms to compute the constant distance offsets by first offsetting the surfaces then trimming away any self-intersections are presented in Chapters 5 and 6, respectively.

The contouring tool path could be either the intersections or projections from the guide surfaces or curves to the offset surfaces, or along isoparametric curves on the offset surfaces. In Chapter 3, we present an approximation method to extract evenly spaced isolines from a NURBS surface, which is an improvement over a previous adaptive isolines method [42]. Although the offset computation and self-intersection trimming algorithms, as well as the multiple tools analysis for decomposition and tool selection presented in this thesis could be applied to multiaxis machining, the automatic NC part programs generation assumes the surface are accessible using fixed-axis machining.

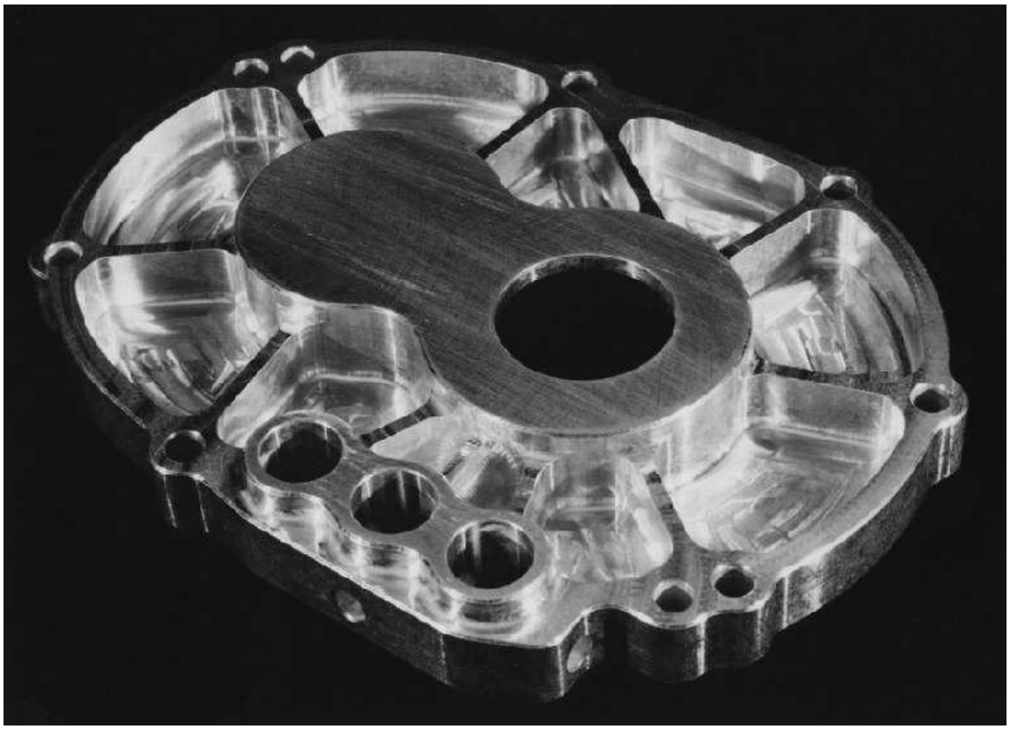
2.4 Examples

Figure 2.8 shows a feature-based CAD model and the machined parts for a transmission case endplate. The endplate consists of many features described in Section 2.3.1, including hole features, $2\,{}^1\!/_2$ D features with arbitrary profiles, and a 3D feature with freeform surfaces. A closeup view of the 3D pocketing feature is shown on the lower left corner of Figure 2.8(a). It has three surfaces. The side and bottom surfaces are constructed using the extrusion operator. Then, the constant radius fillet is constructed by intersecting offsets of the side and bottom surfaces. The intersection curves of the offset surfaces are the centers of the fillet cross-sections defined by a rolling ball between the surfaces [48, 76]. By inverse offsetting of the trimmed offset surfaces, edges of the 3D pocket are rounded at a constant radius. Details of the fillet operation are discussed in Section 5.7. Figure 2.8(b) shows the machined metal part of the endplate model, which is performed on a 3-axis machining center with four stages of fixture setups.

Figure 2.9 shows an axial flow compressor disk. The compressor disk model shown in Figure 2.9(a) consists of hole features, turning features, i.e., surfaces of revolution, and 3D milling features for the blades. Note that the milling features for the blades are defined by the volume between adjacent blades that are to be subtracted from the disk stock. A closeup view of the 3D milling feature is shown on the lower left corner of Figure 2.9(a). This feature is repeated 16 times around the center axis to form the 16 blades. To machine the compressor disk part shown in Figure 2.9(b), two setups on a turning center with live tooling capability are first used. Then, the blades are machined on a 4-axis machining center. The fourth axis of the machining center is used primarily as an index table to orient the workpiece such that each 3D milling feature is accessible using fixed-axis machining. Note that the compressor disk model has sharp edges between the blades and the base surface. Using a ball-end milling tool, the machined part blends the sharp edges

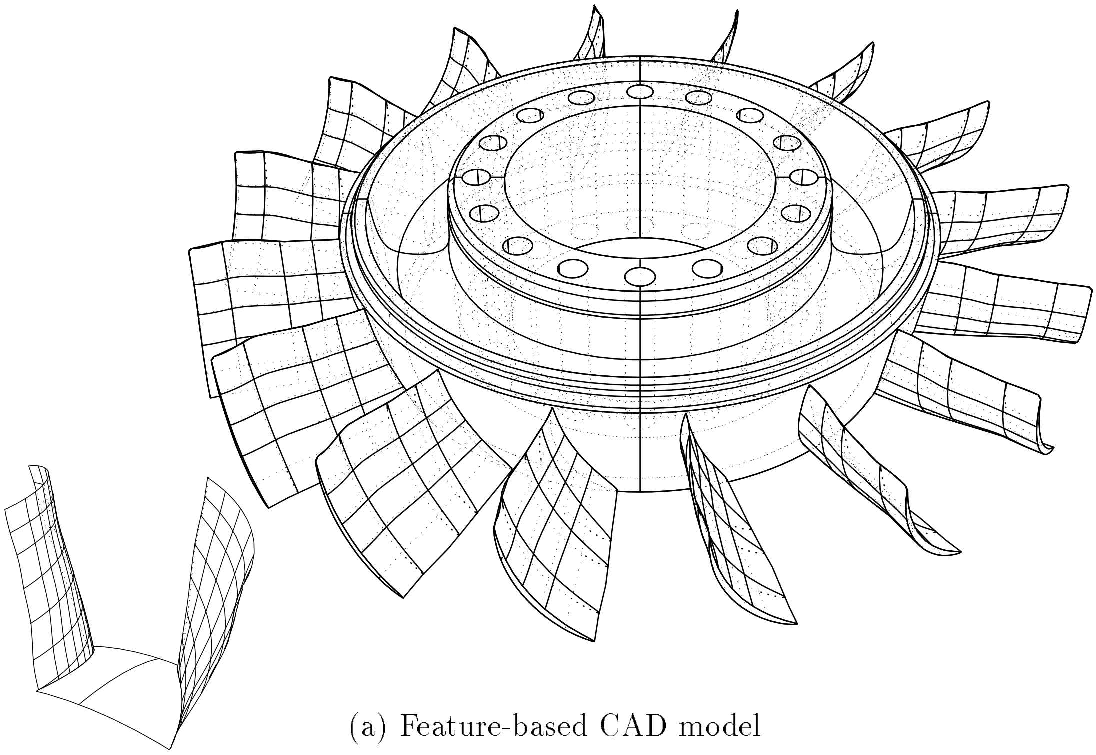


(a) Feature-based CAD model

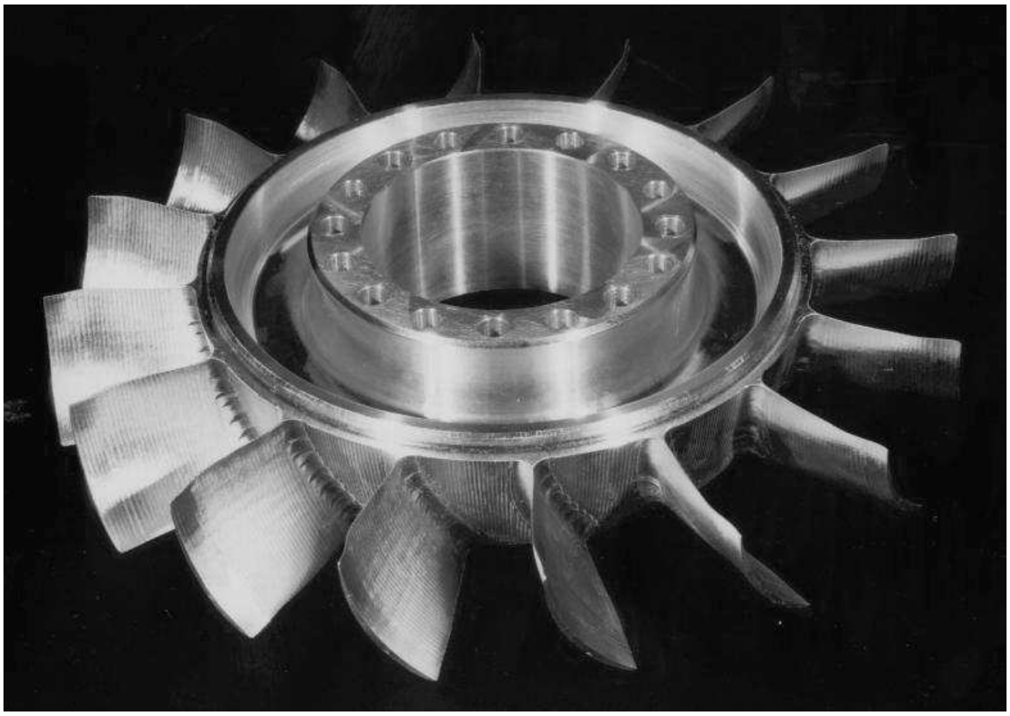


(b) Machined metal part

Figure 2.8. A transmission case endplate CAD model and its machined part



(a) Feature-based CAD model

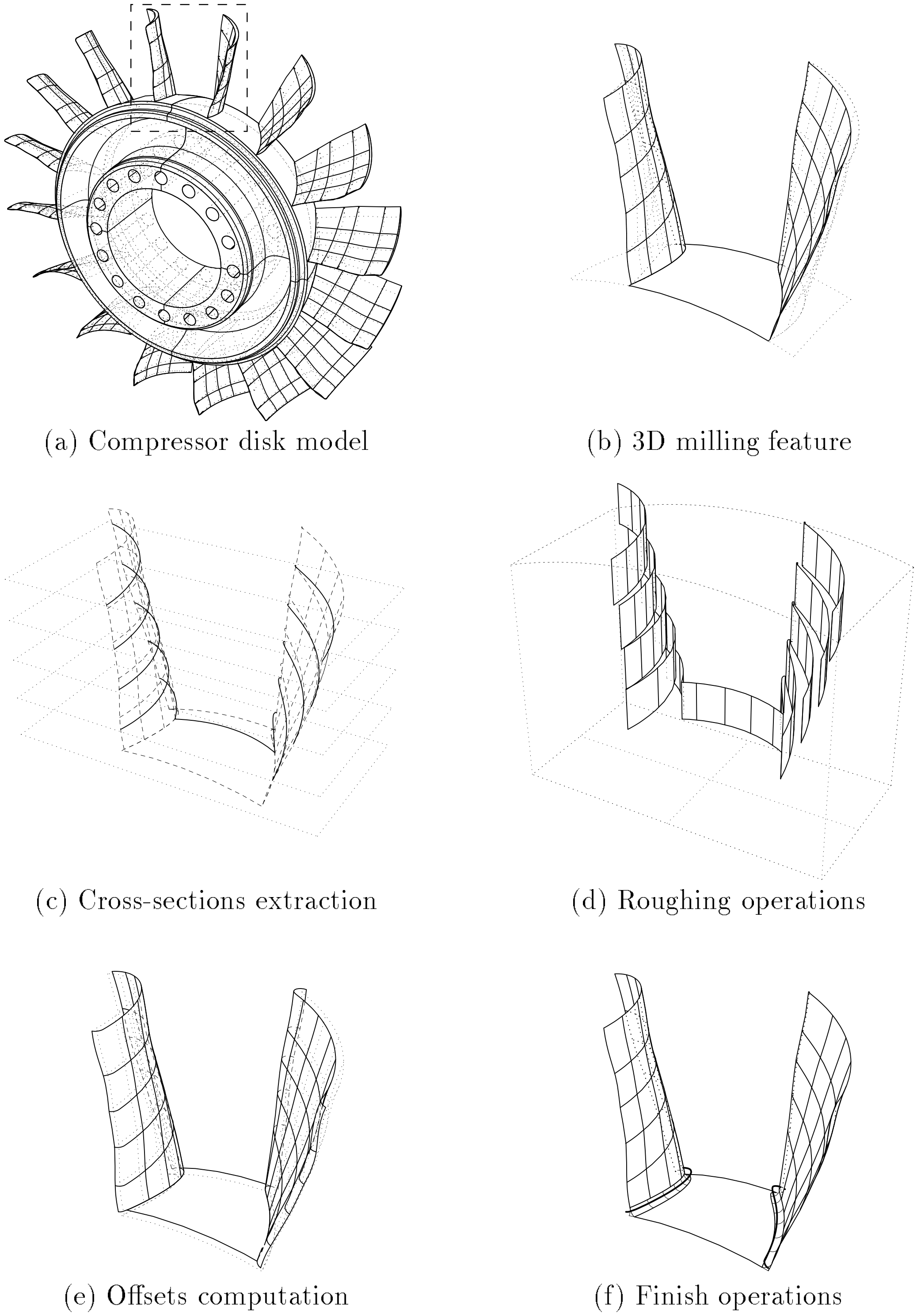


(b) Machined metal part

Figure 2.9. A compressor disk CAD model and its machined part

by constant radius fillets which have the same radius as the ball-end tool.

Figure 2.10 shows examples of 3D feature decomposition for the compressor disk part. Figures (a) and (b) show the machining orientation of the compressor disk and the subtractive 3D features. Figure (c) shows the constant-Z cross-sections extraction for roughing operations. Five layers of cross-section curves are computed by intersecting the 3D feature with parallel planes separated at a distance equal to one tool diameter of the roughing tool. Figure (d) shows open pockets for the roughing operations. Without the support of open regions in $2\,{}^{1}\!/_{2}$ D machining, the automatic decomposition of 3D features of open surfaces would not be possible. Figure (e) shows the constant distance offsets of the part surfaces, which have self-intersections trimmed away for gouge checking and prevention, including the open edges of the part surfaces. Algorithms to compute the offset surfaces and self-intersections are presented in Chapters 5 and 6, respectively. Finally, Figure (f) shows the finish operations of the surface regions that are machinable by a ball-end tool. Two feature curves are extracted from the offset surfaces self-intersection to machine the fillet surfaces between the blades and the base surface. Tool path computation for the roughing and finishing operations are presented in next Chapter. Note that it is possible to use multiple ball-end tools for more uniform and more efficient finish cuts. Details of using multiple tools of different diameters on the same feature are discussed in Chapter 4.



(a) Compressor disk model

(b) 3D milling feature

(c) Cross-sections extraction

(d) Roughing operations

(e) Offsets computation

(f) Finish operations

Figure 2.10. Examples of 3D feature decomposition

CHAPTER 3

AUTOMATIC NC PART PROGRAMMING

In the previous chapter, we described the framework of our feature-based approach to process planning and NC machining. To generate NC part programs for machining a given part, we first decompose each feature of the part design into simpler subfeatures and operations which can then be handled as individual machining entities. We use an "*open pocket*," a generalization of $2\,{}^1\!/_2$ D milling operations, for the roughing and finishing of $2\,{}^1\!/_2$ D milling features. The $2\,{}^1\!/_2$ D milling operation is also used in the roughing of 3D features by extracting constant-Z contours from the 3D surfaces.

This chapter focuses on the machining side of our feature-based approach, i.e., how to automatically generate NC part programs. We start with an introduction to the components and requirements of NC part programs and part programming. A brief overview of NC part programming techniques and history is presented. Then, we review different pocketing algorithms to generate tool paths for regions bounded by closed 2D profiles. In this research, we extend the conventional offset tool path pocketing algorithms to handle open pockets, i.e., semiopen regions defined by open/closed 2D part/stock profiles.

Geometric analysis tools and algorithms are presented to generate and optimize tool paths for open pockets. These include better tool selection based on the pocket geometry and tool path clipping for open pockets. We present a fast offset tool path clipping algorithm to remove the unproductive tool path in the outside regions of a stock. In next chapter, the fast tool path clipping algorithm is extended to generate efficient tool paths for machining a feature using multiple tools of different diameters. Finally, computational issues for generating tool paths of 3D features defined by freeform surfaces are discussed. Details of the algorithms and analysis tools for 3D features decomposition and tool path computation are presented in Chapters 5 and 6.

3.1 NC Machine and Part Program

Numerical control (NC) means to control the operations of machine tools by a series of coded instructions called the *part program*, which consists mainly of numbers and letters [41, 56]. The sequence of instructions is preplanned by the part programmer and the performed operations are predictable. Unlike manual machines, NC machines can execute the same program repeatedly to produce multiple copies of the same part, each within the machining tolerances. Unlike specialized machines, NC machines can produce different parts by running different programs. The flexibility and accuracy of NC machining make it suitable for low to medium volume with high variation and complexity.

There are two types of computer-controlled NC systems [26]. *Direct numerical control* (DNC) was introduced in the mid to late sixties, in which a central mainframe is used to direct the operations of a number of separate NC machines. Machining instructions are sent from the central computer to each individual machine's requests through communication links. DNC was used when computers were big and expensive. One major disadvantage of DNC is that when the central computer is down, none of the NC machines can operate.

Computer numerical control (CNC) was commercially offered during the early seventies, where a dedicated mini/micro computer is used to perform some or all of the basic NC control functions [114]. Part programs are stored in the computer's main memory. Program instructions are processed by the control computer into actions of the machine tool. For example, canned cycles and circular interpolations are carried out by the computer's software/hardware instead of explicit low-level machining sequences in the part program.

With the increasing power and decreasing cost of personal computers, newer CNC machines use a separate personal computer as the front-end for user control interface and off-line storage and operations. A user can perform tool path simulation on the program that is currently running or a program under editing while a different program is running. Another add-on component of the new CNC machines is the capability of shop-floor programming. An operator can either create new part programs using high-level commands while the machine is running another part, or modify an existing program for any necessary adjustment on the programming parameters to produce a desirable part.

The machining process removes material from the workpiece by driving a rotational cutter on the part surface. The cutter has sharp teeth which cut the raw stock into thin chips while the tool rotates. Rate of the tool angular rotation is called the spindle speed, and rate of the tool movement is called the feed rate. Path of the tool motion, i.e., the trajectory of the tool center point, is called the tool path. It is important that the tool path is carefully designed to cover the whole area that needs to be machined, but does not gouge into the part surface. For NC machining, a part program consists of instructions, also known as G-codes, which specify [25]:

- Which tool to use (tool selection),
- How fast the tool turns (spindle speed),
- How fast the tool moves (feed rate),
- How the tool moves (machining mode), and
- Where the tool moves to (coordinates).

In the following, we examine each of the components for the sake of NC part programming.

For milling operations using end milling cutters, the basic programming considerations in tool selection, besides the tool material and shape, are the tool diameter and tool length. For efficiency and accuracy reasons, it is better to select a tool with a larger diameter and a shorter length, so long as it is not too big or short to reach the small or deep corners. A larger tool is stronger and can cut more efficiently, provided the machine has enough horsepower to handle its tool load. A shorter tool causes less deflection and vibration and can produce more accurate parts.

Tool size and length are usually constrained by the geometry and shape of the machined part. Note that there is only a finite number of standard tool diameters, each in a discrete increment. A tool with a common diameter is usually cheaper and preferable than a tool with an odd diameter. For resharpened tools which may have different actual diameters than the original specifications, we can either use the measured actual diameters for tool selection and tool path generation, or use their original diameters and the tool compensation capability of NC machines if the differences are small.

3.1.1 Feed and Speed Computation

With an appropriate tool selected, the next step in a milling operation is to set a suitable feed and speed to drive the tool. The spindle speed is usually set in revolutions-per-minute (RPM) for NC milling machines, which can be calculated from the tool diameter and the recommended surface cutting speed. The surface cutting speed is determined by the tool geometry and the material properties of the tool and the workpiece [87]. Using an improper spindle speed may result in excessive tool wear or poor surface finish.

Feed rate is the rate of tool traversal relative to the workpiece (most NC machines move tool spindle and workpiece table in different axes to achieve the programmed relative motion). In NC part programming, the feed rate is usually specified in feed-per-minute (FPM). However, the recommended feed rate which depends on the material properties and cutting conditions is usually expressed in feed-per-tooth (FPT) [87]. By multiplying the per-tooth feed rate with the number of teeth of the tool and the spindle speed RPM, one can convert the feed rate from FPT to FPM. It is important to adjust the feed rate FPM proportionally if the programmed spindle speed of a part program is to be modified, either from program editing or operator overriding. Most NC machines also support a different method of specifying the feed rate as feed-per-revolution (FPR). Using FPR, the feed rate and spindle speed of a part program can be adjusted independently of each other.

Note that the cutting conditions, e.g., width and depth of cut, are usually not the same throughout the entire milling operation. Sometimes they change rapidly even within a single tool movement. It is difficult to determine a single feed value that is optimal for the whole milling operation. Usually, the feed rate is set to accommodate the worst cutting conditions, i.e., a slower rate for the wider and deeper cuts. To adjust the feed rate automatically in real-time, *adaptive control* NC machines were introduced during the sixties [21].

Using adaptive control, one or more process variables, e.g., cutting force, torque, spindle horsepower, etc., are measured and fed back to the machine controller to regulate the machining feed and/or speed to compensate for any undesirable changes in the cutting conditions [56]. Despite its advantages, adaptive control machining has not been widely used because of its high cost and the low percentage of in-process machining time compare to the overall production processes. With increasing demand for *unmanned machining*, such as in Flexible Manufacturing System (FMS) and Flexible Manufacturing Cells (FMC) [60, 115], low in-process human intervention makes adaptive control systems interesting and important.

A software approach is used to adaptively adjust the feed rate to achieve a near constant material removal rate, thus yields nearly uniform cutting forces and machining loads [145, 146]. This not only reduces the overall machining time, but also prevents high-load peaks, which could cause tool deflection and vibration, or even tool breakage. Material remove rate can be computed using off-line NC machining simulation [135, 143, 146]. Experiments have shown 30–50% reduction in total machining time using adaptive feed rate adjustment [146].

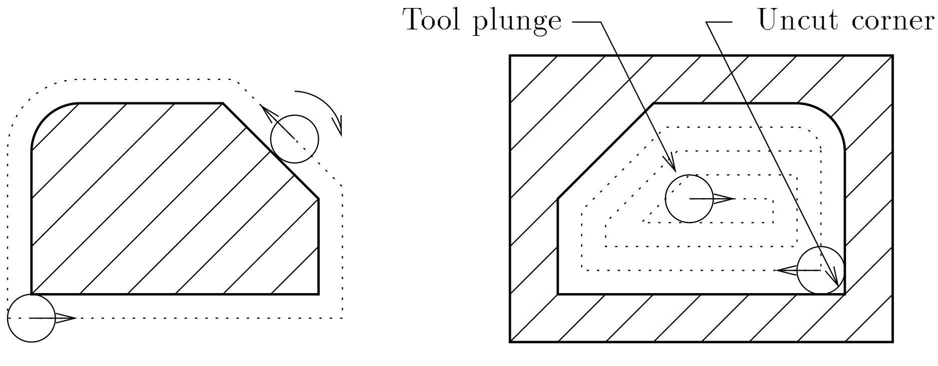
3.1.2 Tool Path Design

From a programmer's point of view, an NC machine moves its tool by specifying coordinates of the destination points and the machining modes, e.g., rapid traversal and linear or circular interpolations, under which the tool moves to the new locations. Tool location is usually specified by the center point of the tool tip.

Tool path design depends mainly on the geometry of the part to be machined and the size and shape of the selected tool. Basic requirements for tool path generation include:

- The union of all tool movements, i.e., its *envelope* [147], must cover the entire area of the material to be removed.
- There is no *gouging*, i.e., unwanted cut, or *collision* between the tool and the part surface or any part of the machining setup, e.g., fixtures.
- The machined part surface must be within the required *tolerances* specified in the part design.

One fundamental task in tool path design is to compute the tool center offset from the part surface. Figure 3.1 shows two examples of tool path design for $2\,{}^1\!/_2$ D milling operations. On the left is a profile operation by driving the tool along a single tool radius offset from the part boundary. On the right is a pocketing operation where material inside the pocket profile is to be removed. The strategy used in this example uses consecutive offsets from the boundary with a constant stepover size [61, 107]. The tool path starts from the center, where the tool needs to plunge or ramp down in Z direction, then spirals along the offsets one at a time towards the outer boundary. Note that the sharp corners of the pocket cannot be machined completely without gouging. The machined part has rounded corners



(a) Profiling (b) Pocketing

Figure 3.1. Examples of profiling and pocketing tool path

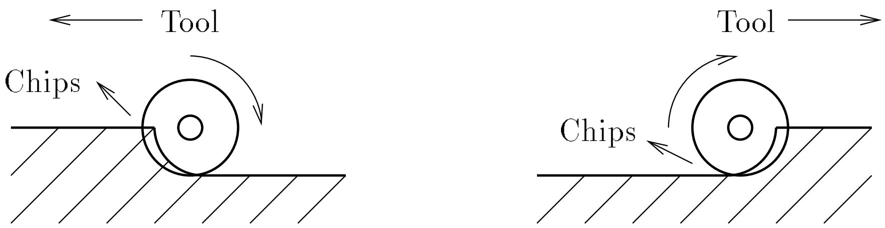
with radius equal to the tool radius.

One interesting observation on the tool paths shown in Figure 3.1 is that while the tool moves in different orientations, counter-clockwise for the profile and clockwise for the pocket, the workpiece is always cut from the left side of the tool, viewed from the direction of tool motion (except for the first cut of the pocket, which is slot cutting). For a tool that cuts in clockwise spindle rotation, the workpiece is fed against the tool when cutting from the left side. This type of milling is called *conventional milling*, as demonstrated in Figure 3.2(a).

Figure 3.2(b) shows another milling type, called *climb milling*, where the workpiece is fed in the same direction as the tool rotation at the contact point. Climb milling can give better surface finish and is normally used for CNC machining. However, conventional milling is preferred in situations when the tool or the fixture is not stiff enough [41]. In automatic tool path generation, it is important that a user can specify the type of cutting method for each operation easily.

For milling 3D surfaces, ball-end milling tools are usually used for their capability of handling both convex and concave surfaces at the same time. Figure 3.3 illustrates the basic computation of tool paths for 3D surfaces. The tool center points are offsets of the tool/surface contact points in the surface normal directions by the tool radius. Coordinates of the tool tip points can then be obtained by translating the center points in the tool axis direction.

The uncut region between successive tool cuts is usually called *scallop*. Assuming the part surface between two tool cuts is planar (see Figure 3.4), the distance between adjacent tool cuts d can be computed from the tool radius R and the



(a) Conventional milling (b) Climb milling

Figure 3.2. Conventional milling and climb milling

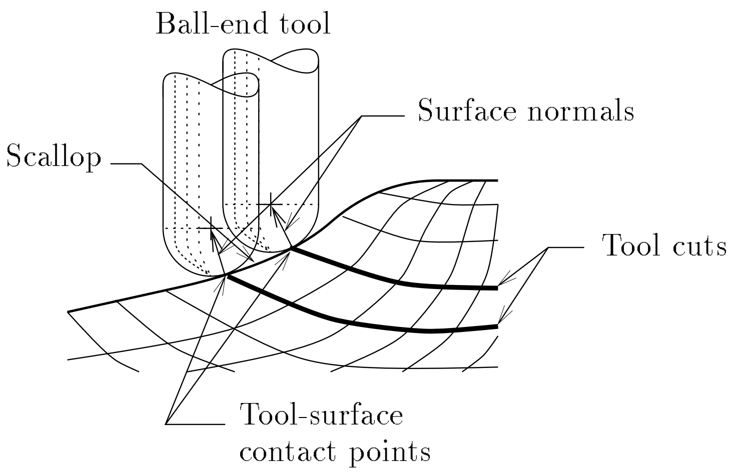


Figure 3.3. Ball-end milling tool offsets with surface normals

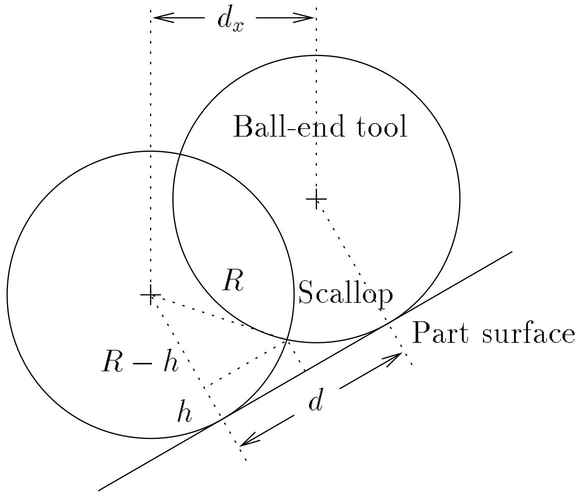


Figure 3.4. Stepover size estimation from scallop height

desired scallop height h by:

$$d = 2\sqrt{R^2 - (R-h)^2}. \tag{3.1}$$

Note that the horizontal stepover size d_x is usually not the same as the surface stepover size d, depending on the slope and curvature of the part surface. It is important to adjust the stepover size or the distance between adjacent tool cuts such that the maximum scallop height is within a prescribed tolerance. A smaller stepover size gives a smaller scallop height. However, it also requires more passes to cover the same surface and, therefore, takes longer to machine.

Using a larger tool diameter can also produce smaller scallops with the same stepover size. However, like in the pocketing operation, the tool radius must not exceed the minimum radius of curvature of the part surface. Otherwise, high curvature concave regions of the part surface cannot be machined cleanly without gouging. In other words, these regions must be avoided to prevent gouging when generating tool path with a large tool. This also applies to the narrow clearance regions.

3.2 NC Part Programming

There are four different methods for NC part programming [56]:

- Manual Part Programming.
- Computer-Assisted Part Programming.
- CAD/CAM Systems.
- Computer-Automated Part Programming.

In manual part programming, all the computations of the part programs are performed by the part programmer manually. A part programmer needs to calculate the appropriate machining feeds/speeds as well as the tool offsets for tool path design using algebra, geometry and particularly trigonometry. A part programmer must also be familiar with the operations of the NC machine and machining practices in order to list every operation the NC machine is to perform. Although this is not too complicated with *point-to-point* machining, such as drilled-holes, counter-sinks, or counter-bores, since most machines support canned cycles, it can be time consuming and difficult with contour milling such as profiling or pocketing operations. It would certainly be impossible to manually program for surface milling of parts with complex geometry. Compared to computer programming languages, manual part programming is similar to programming in assembly languages (G-codes) with limited system library subroutines (canned cycles).

Computer-assisted part programming is more like programming in a high-level computer language. Part programs are written using part programming languages, where software engineering techniques such as abstraction can be applied. The most commonly and widely used part programming language is the Automatically Programmed Tools (APT) system [79]. In APT, the programmer first describes the part geometry using symbolic commands (keywords) and numbers. Geometric entities such as points, lines and circles are constructed and assigned to variables which can be referenced while computing the tool path. Machining parameters, such as tool selection, feed rate and spindle speeds, must be determined by the programmer as in manual programming, although the commands to specify them in APT are machine independent.

In APT, motion of the tool is usually described explicitly with references to geometric constraining surfaces, called check surfaces, or coordinate specification. Decisions must be made with respect to machining sequences, tool location reference points, part clamping procedures, and so on. The general procedure to direct the tool motion consists of three components: the *part surface*, the *drive surface*, and the *check surface*. The drive surface is the surface that guides the side of the cutting tool. The part surface is the surface on which the bottom of the tool rides. The check surface is the surface that stops the movement of the tool in its current direction. For 2D machining, the check surface and drive surface can simply be lines or circles in the XY plane since the part surface will be on a plane parallel to the XY plane. A user can program the tool path using commands to move the tool along the drive surface to/on/past the check surface (see Figure 3.5). For 3D contour milling, numerical methods are used to check the intersection of the tool against the part/check surfaces and to adjust tool locations to prevent gouging.

Using a CAD/CAM system, a portion of the procedure usually done by the part programmer is instead done by the computer. Geometric information can be retrieved from the CAD model for tool path generation. This eliminates the time-consuming and error-prone geometry definition step in computer-assisted programming. Using interactive computer graphics, tool path can be generated sim-

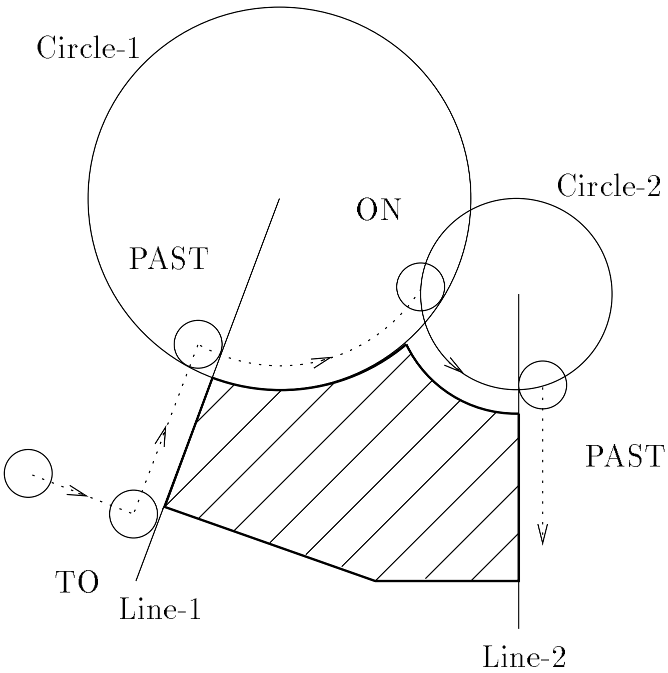


Figure 3.5. Tool path design using APT

ilar to the computer-assisted part programming, but with quick graphical input and immediate graphical feedback. The graphics system also provides a visual validation of the part program. More advanced CAD/CAM systems also support automatic software routines that can handle some common machining cycles such as point-to-point operations, 2D profiles and pockets, as well as some 3D surface contouring [30, 58, 61, 66, 68, 69, 70, 80, 107, 134]. Tool paths for machining these compound geometric entities can be generated with the user supplied machining parameters.

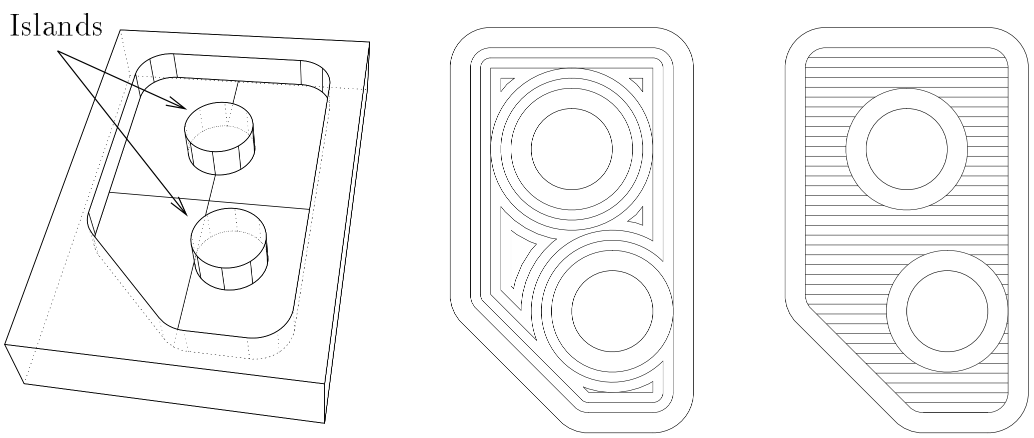
Several aspects of the part programming procedure have been automated on certain classes of part geometry in the CAD/CAM approach to NC part programming. Combined with computer-automated process planning, an automatic NC part programming system could generate the part program completely without relying on the user for detailed machining parameters, such as tool selection and feeds/speeds calculation. Nevertheless, such system should also allow the user to override the machining parameters and make intelligent adjustment to the generated part program.

Automatic NC part programming lifts the burden of many detailed machining decision-making processes from the designers and engineers, such as tool selection, tool path generation, and feed/speed computations, etc. Thus its existence would allow the design/manufacturing engineer to concentrate on the higher-level design and production problems which are more difficult to automate. Presently, this can only be done with well-defined, relatively simple geometries, such as drilled-holes and various point-to-point operations [27].

3.3 Overview of Pocketing Algorithms

In this section, we review some commonly used algorithms for pocketing tool path generation. Pocketing is a milling operation that removes regions bounded by one or more planar profiles [57]. A $2\,{}^1\!/_2$ D pocket has a flat bottom and can be machined using a flat-end milling tool at a fixed Z-depth, or multiple Z-steps. Figure 3.6(a) shows a pocket consisting of one outer boundary and two island profiles, i.e., negative regions. Two predominant pocketing tool path strategies have been used for pocket machining in numerous literature [30, 59, 61, 107]. Figures 3.6(b) and (c) show the two types of tool path generation strategies, contour parallel and direction parallel, respectively [61].

In contour parallel pocketing, also known as spiral cutting, the cutter moves



(a) Pocket with islands (b) Spiral tool path (c) Zig-zag tool path

Figure 3.6. Contour parallel and direction parallel pocketing tool path

along a path of equal distance to the boundary profiles, starting from the innermost ones towards the extreme boundaries. These "parallel" contours are "offsets" of the boundary profiles, each at a constant distance. Regardless of the shape of the boundaries, it can be easily proven that a pocket can be cut cleanly by consecutive offsets with a spacing no greater than the tool radius, provided that the tool is small enough to access the smallest corner and the narrowest passageway of the pocket. An in-depth discussion of offset curves and surfaces is presented in Chapter 5.

In direction parallel pocketing, also known as zig-zag cutting, the tool moves in straight lines parallel to each other, usually aligned with one of the machining axes. The maximum step-over spacing between passes is less than or equal to the tool diameter to ensure a clean cut of the pocket. However, in practice, a smaller step-over size, such as half of the diameter or even less, is usually used to reduce the tool load. A final cut along an offset of the boundary is required to clear the uncut scallops on the boundary [61].

In spiral cutting, the tool always stays on the same side of the cutting surface, left or right, relative to the direction of tool movement, which is a major advantage of the spiral tool path over the zig-zag tool path. With a constant spindle rotation direction, the tool either cuts along the spindle direction (conventional milling) or against the spindle direction (climb milling), as shown in Figure 3.2. However, in zig-zag cutting, the tool moves back and forth in alternate directions between passes. Depending on the specific application and material properties of the stock, this may not be acceptable.

To achieve a uniform cutting direction in direction parallel pocketing, the tool has to retract at the end of each pass and plunge down to the beginning of the next pass. Contrary to a drilling tool, a center-cutting end-mill does not plunge well and a non-center-cutting end-mill simply cannot plunge at all. For hard materials such as steel, holes may be predrilled at plunge locations to solve this problem. However, it is impractical to predrill or plunge before every tool movement. Another method is to move the tool at a ramping angle, e.g., 15 degrees down, instead of plunging straight down. To have an effective ramping, the horizontal travel diameter should be as big as the tool diameter. One drawback of ramping is the additional machining time and tool movement. It is better to reduce the number of retract/plunge/ramping and maintain a continuous cutting at the same Z-level as much as possible.

From computational geometry, an offset of a profile may require trimming of self-intersections, which can be done in $O(n \log n)$ time for a profile of n segments [59]. For m offsets, it takes $O(mn \log n)$ if the same algorithm is applied m times. The zig-zag tool path can also be computed in $O(n \log n)$ time using a scanline sweep algorithm [61]. For m scanlines, the total time complexity of a zig-zag tool path is $O(m+n \log n)$ using a sweep algorithm, or $O(mn)$ if a straightforward line clipping algorithm is repeated m times. Interestingly, when n is big but m is small such that $m < \log n$, the latter becomes more efficient, i.e., when the number of scanlines is less than the log of the number of segments.

3.4 Skeleton and Offset Tool Path

A special technique for computing offsets of profiles consisting of arcs and lines uses the Voronoi diagram [107, 113], also known as skeleton or medial axis of the profiles, which can be computed in $O(n \log n)$ time [30, 61, 81, 152]. Then an offset profile can be computed using the profile skeleton in $O(n)$ time for a boundary of n segments, in this case, points, lines, and circular arcs. Therefore, to compute m offset contours requires only $O(mn + n \log n)$ total computation time, which is an improvement over the above offset/self-intersection trimming method, especially when m becomes large.

In the following, a short introduction of the Voronoi diagram and skeleton is first presented. Then, a merge algorithm to compute the skeleton of profiles consist of connected arcs and lines segments are presented [81, 61]. Note that the computational complexity analysis is mainly based on the average-case complexity which gives better performance measurement for real world applications than the worst-case complexity.

3.4.1 Voronoi Diagram and Skeleton

A Voronoi diagram of n sites in a plane is a partition of the 2D space into the regions that are closest to a particular site. The structure of Voronoi diagram provides a quick answer to all nearest neighbors problems. For example, Figure 3.7 shows a Voronoi diagram for 50 points. The polygon surrounding each point is called the Voronoi polygon of the corresponding point. Voronoi polygons form the boundaries of Voronoi regions. Any point inside a Voronoi region is closer to the corresponding site of that region than to any other site. An edge of a Voronoi polygon is called a Voronoi edge. Every Voronoi edge is a bisector between two sites, called generators of the bisector. Any point on a Voronoi edge is equidistant from its two generator sites. The bisector $\mathcal{B}$ between sites p_1 and p_2 can be defined as:

$$\mathcal{B}(p_1, p_2) = \{x \mid \|x - p_1\| = \|x - p_2\|\}. \tag{3.2}$$

Mathematical definitions and properties of Voronoi diagram can be found in literature [61, 78, 82, 113].

A divide-and-conquer algorithm to compute the Voronoi diagram of points in a plane first splits the points into two linearly separable subsets, i.e., points that can be separated by a line [113]. Then, Voronoi diagrams of each subset are computed. Finally, the two subdiagrams are merged together. Since merge of two separable

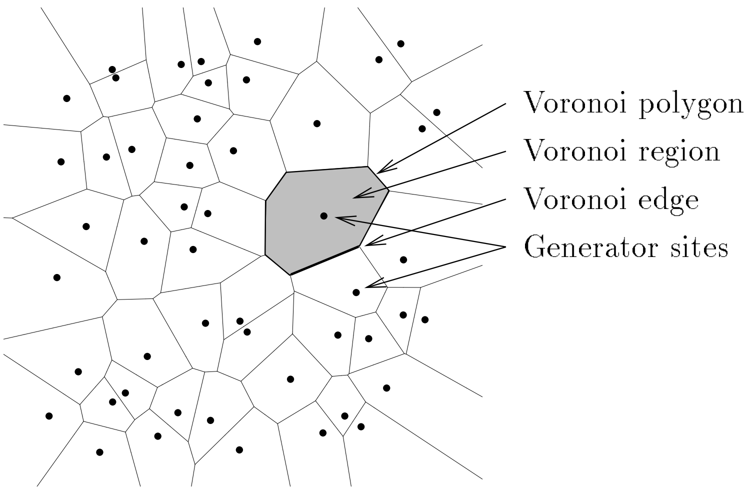
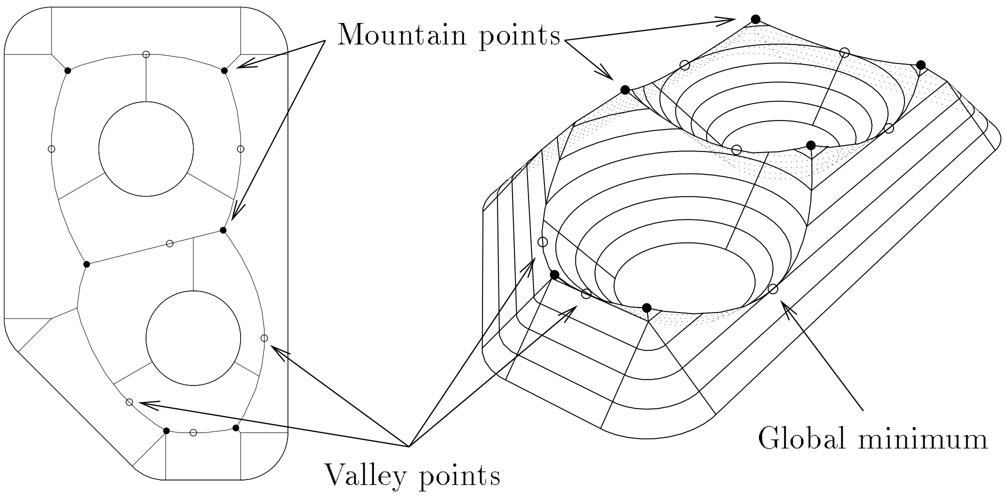


Figure 3.7. Voronoi diagram of 2D points

subdiagrams can be done in $O(n)$ time with an $O(n)$ chain, the whole algorithm is $O(n \log n)$. A different algorithm to compute the Voronoi diagram of 2D points finds the intersection curves between cones of the same angle, perpendicular to the plane, and centered at each point [51]. Any point on the intersection curve between two cones is equidistant to the cone axes, as well as the tip point sites. Therefore, the projection of the intersection curves onto the plane yields the Voronoi diagram. Using a sweep algorithm, the intersections can be found in $O(n \log n)$ time.

The Voronoi diagram of piecewise continuous planar profiles of lines and arcs in a half space, i.e., inside the bounded region, is usually called the skeleton of the profiles. Lee [81] extended the above merge algorithm to compute skeletons of connected piecewise linear profiles, which was extended later to include circular arcs as well [61]. Figure 3.8(a) shows the skeleton of the pocket from Figure 3.6. Details of the skeleton computation merge algorithm will be discussed in next section.

In Figure 3.8, critical points, i.e., local maxima and minima of the bisector functions are shown as filled dots and circular markers, respectively. These points are called the mountain points (maxima) and valley points (minima) of bisectors. There are six mountain points and seven valley points in the skeleton of this pocket. Figure 3.8(b) shows a 3D mapping of the skeleton and offsets such that $z = t$. From



(a) Skeleton (b) Mountain and valley points

Figure 3.8. Skeleton and critical points

the figure, it is clear why the critical points are named. Later, we'll show that these mountain and valley points of skeleton are useful in NC tool path generation and process planning.

3.4.2 Merge Skeleton Algorithm

Recall that the divide-and-conquer merge algorithm first splits the data sites into two subsets, then recursively computes the subdiagram of each subset, and finally merges the two subdiagrams together. For a simply connected profile, i.e., not self-intersecting, the two split subprofiles are separable by an $O(n)$ chain of bisectors. An outline of the merge skeleton algorithm is listed below [81].

```
ALGORITHM MergeSkel( SubSkel1, SubSkel2 )
BEGIN
    Let Edge1 and Edge2 be the starting edges of
            SubSkel1 and SubSkel2.
    REPEAT
        Compute the merge Bisector of Edge1 and Edge2.
        Find first intersection points Pt1 and Pt2 between the
                Bisector and the Voronoi polygons of Edge1 and
                Edge2.  Let VorEdge1 and VorEdge2 be the Voronoi
                edges where the intersections occur.
        Add Bisector to the two Voronoi polygons and trim away
                Voronoi edges that are at the other side of the
                intersection points.
        IF only Pt1 exists or Pt1 is hit first THEN
            Set Edge1 to the other generator of VorEdge1.
        ELSE IF only Pt2 exists or Pt2 is hit first THEN
            Set Edge2 to the other generator of VorEdge2.
        ELSE IF Pt1 and Pt2 exist and are the same THEN
            Set Edge1 and Edge2 to the other generators of
                    VorEdge1 and VorEdge2.
    UNTIL Pt1 and Pt2 do not exist or Edge1 and Edge2 are the
            last edges of SubSkel1 and SubSkel2
END
```

Figure 3.9 illustrates key steps of the merge skeleton algorithm. First, points are inserted at every concave corner, e.g., site 4 in (a). Then, the profile is split into two subprofiles, 1–4 and 5–7, and their skeletons are computed recursively. The two top-level subskeletons are shown in (b). Finally, the two subskeletons are merged together. Starting from the splitting junction point, the bisector between

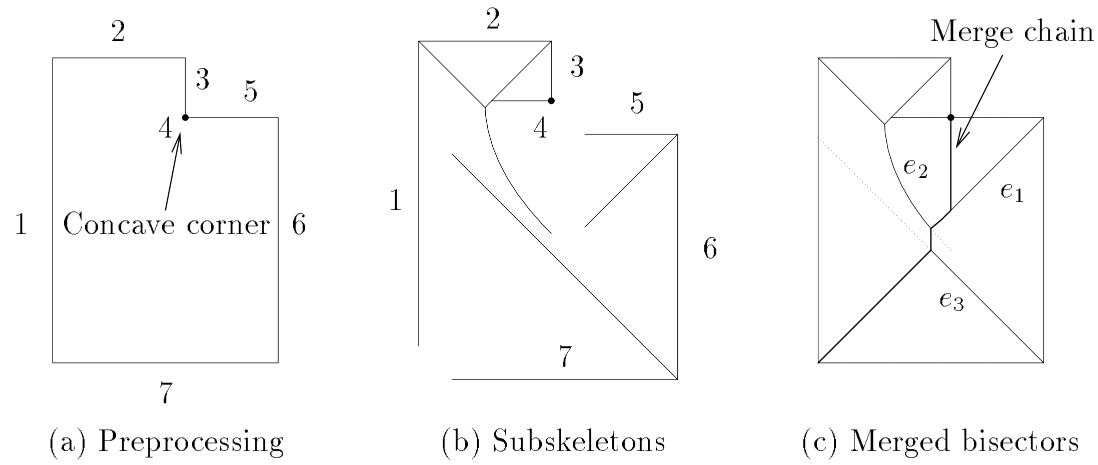


Figure 3.9. A divide-and-conquer merge skeleton algorithm

edges of the two subprofiles are computed. Intersections of the new merge bisector and the Voronoi polygons of its two generators are computed. Following the merge bisector direction, the first intersection point is used to find the next closest edge, and the merge process continues. Results of the merged skeleton is shown in (c). The merge chain intersects and trims Voronoi edges e_1, e_2, and e_3.

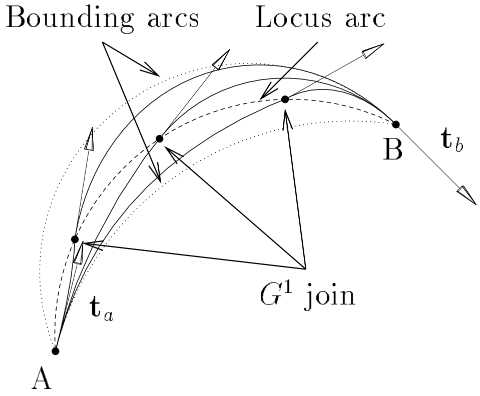
3.5 Biarc Curves

Most NC controllers supports only linear and circular interpolations. To machine an arbitrary profile, sampled points are usually used to approximate the curve as linear segments [30]. The resulting NC file size can be huge if a tight tolerance is required. This presents problems in storing part programs with the limited available memory on NC machines, and in loading the NC codes to the controller which creates a bottleneck to achieve the requested feed rate. Also, the piecewise linear approximation creates tangent discontinuities that may not be acceptable in some part design. A method to reduce the code size and the loading overhead is to approximate the linear tool movements with circular arcs, then utilize the circular interpolation capability of the NC machine [112]. However, it is used only for rough cutting and the resulting arcs are not necessarily tangent continuous.

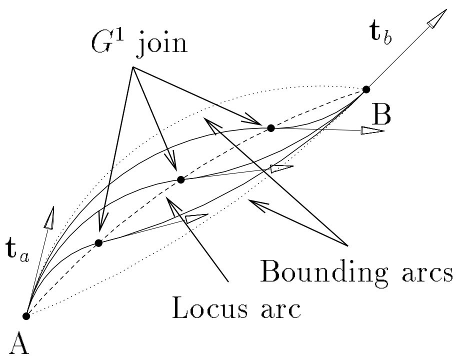
A numerical curve fitting based on chains of circular arc and straight line segments was designed for shipbuilding industry in 1970 [19]. Two tangent continuous circular arcs are used between each pair of data points along nonstraight portions of the curve. Results of this type of curve fitting are called biarc curves, or arc splines. Biarc curves were intended as an alternative to cubic spline interpolation for easy offsets, intersections, and length measurement computations for NC programming in ship building.

A biarc is a composite curve constructed from two circular arcs so that there is tangent continuity at the join. Figure 3.10 shows examples of biarc interpolation of Hermite data, i.e., end points and end tangents. A general formulation of biarcs includes C-shaped biarcs and S-shaped biarcs. A biarc could degenerate into a single arc when angles between the chord AB and the end tangents, $\mathbf{t}_a$ and $\mathbf{t}_b$, are the same. It could even degenerate into a single line segment if $\mathbf{t}_a$ and $\mathbf{t}_b$ are aligned with chord AB. Some important properties of biarcs are described below.

From the figure, S-shaped biarcs have inflection points at the join, while C-shaped biarcs do not have inflection points unless the junction point falls outside of the end tangent lines. With a given set of Hermite data, there is one degree of freedom in determining the junction point of the biarc. An important property of biarcs is that the locus of the junction point is a circular arc passing through the two end points [100, 129]. This property holds for both planar and space biarcs. Obviously, the end tangents of the locus arc are not the same as the end tangents of the curve, except when the two centers of the biarc are actually the same point.



(a) C-shaped biarcs



(b) S-shaped biarcs

Figure 3.10. Interpolate end points and tangents with biarcs

In which case, the biarc degenerates into a single arc. The locus arcs are shown as dashed lines in Figure 3.10.

Another degeneracy of biarcs occurs when one of the arcs has zero radius, i.e., the junction point is at one of the end points. Two circular arcs can be created under this degenerate condition, with each of them matching only one end tangent. These two arcs form a region that bounds all biarcs matching the same Hermite data [93]. Bounding arcs are shown as dotted lines in Figure 3.10.

Several special types of plane and space biarcs have been used [100]. An equal-chord-length biarc is a biarc which has the same chord length between the two arcs. For plane biarcs, an equal-chord-length biarc also gives the minimum curvature ($^1/_r$) change between the two arcs. Another interesting type of biarc is the equal-winding-angle biarc, in which, the two arcs have the same winding angles. For space biarcs, an equal-winding-angle biarc also gives the minimum twist angle at the biarc junction point. Detailed discussion and proof of the above properties can be found in reference [100].

Depending on the input data, there are several different biarc curve fitting approaches. The first one is biarc curve interpolation of sparse data points, such as points of an airfoil [19, 95, 102, 101, 123]. In this approach, tangent directions at the data points are first approximated. Then, tangent continuous biarcs are used to interpolate the data. Optimization techniques can be applied to adjust the tangent directions and/or the biarc junction point locations, such that the total curvature change or strain energy is minimized. Another use of biarc curves is to approximate a list of dense data points [91, 92, 119, 133]. The objective is to find the minimum number of smooth piecewise circular arcs that satisfies a prescribed approximation tolerance.

A method to approximate quadratic NURBS curves uses the bounding arcs [93]. For quadratic NURBS curves, the bounding arcs of biarcs matching the end points and tangents also bounds the curve. Therefore, an upper bound of the biarc approximation error is to compute the maximum distance between the two bounding arcs. Using this method, a quadratic spline curve is first subdivided until the error of the bounding arcs of each subcurve is within a prescribed tolerance. Thus, any biarc approximation of a subcurve is also within the tolerance. A disadvantage of this method is the extra subdivisions performed because of the coarse error bound method. Also, it is only for quadratic NURBS curves, i.e., conic sections.

3.6 Our Approach

In this section, we first discuss the requirements of automatic NC part programming systems, particularly for a feature-based part programming system. Then, we present our approach to $2\,{}^1\!/_2$ D offset tool path generation and introduce a biarc approximation method for NURBS curves.

3.6.1 Feature-Based NC Part Programming

To automate the design/manufacturing integration, this research uses a feature-based approach for process planning and part programming. Under this approach, an automatic part programming system should at least do the following automatically:

- The system should provide a rich set of general machining operations to cover complex features and their subfeature decomposition. These core operations should be flexible enough to support freeform surfaces as well as well-defined form features as special cases. The number of such operations need not be too many. However, multiple tool path generation algorithms could be used for the same operation under different context of subfeature decomposition.

- Tool selection should be done automatically for each operation, including freeform features. The default strategy would be to select a tool, or tools, that can machine the target feature completely and efficiently. When a preferred tool is specified by the user, a gouge-free tool path should be produced to machine as much area as possible.

- Feed and speed calculations should be done automatically. The system should provide intelligent default values for other machining parameters, such as the horizontal step size and cutting depth. However, it should allow user override to be specified in the process plan. An optional adaptive feed rate adjustment could be used as a back-end process to improve the machining efficiency.

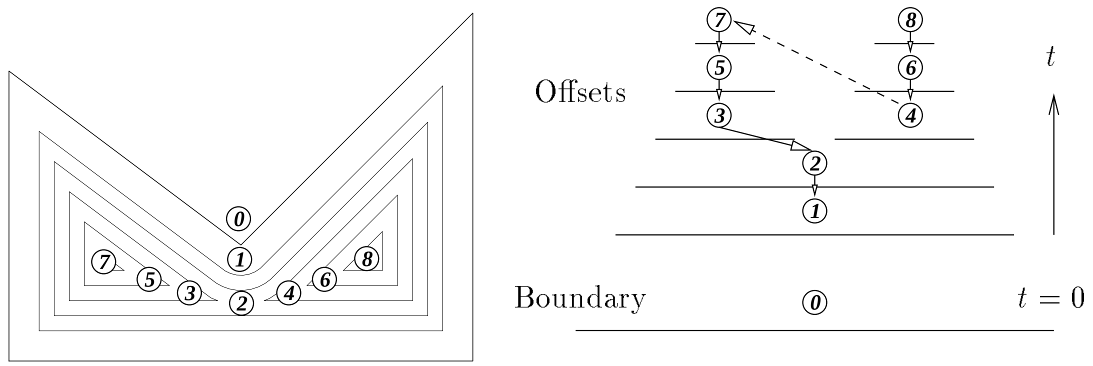
- The system should provide some sort of NC simulation and validation facilities to ensure the correctness and safety of the generated part programs. Although the goal of automatic part programming is to generate correct part program, these facilities could also be used as development tools as well as sources of feedback for a user to search better alternatives in design.
- The tool path generation algorithms should cover both roughing and finishing stages. Although roughing does not show up in the final part geometry, it may occupy more than 50% of the total machining time.

3.6.2 Tool Path Generation

With the profile skeleton computed, offset profiles can be computed by evaluating the bisector functions and connecting the evaluated points to form offset lines or arcs. Offsets of a line segment are parallel lines, and offsets of an arc segment are concentric arcs. Since the profile skeleton provides continuous distance information within the boundaries, no trimming of self-intersection is necessary. To simplify the computation, we subdivide any quadratic bisector with an internal critical point at the critical point. Therefore, each Voronoi edge, except one with constant clearance, has at most one point for a given offset distance. And, each offset segment should start and end at different Voronoi edges.

To generate the tool path, we use a method based on the premise that one wants to cut an offset at a greater offset distance before any other offsets with overlapped boundary segments and smaller offset distance. If the boundary profile and offsets are unfolded into stacks of horizontal lines (see Figure 3.11), with the offset distance aligned with the $+y$ direction, a simple rule for sorting the offset segments is to pick a segment only when there is nothing else on top of it.

An inverted tree structure can be used to represent the partial ordered list for a pocket with a simply connected boundary profile. The root is the boundary profile



(a) Boundary and offsets (b) Offsets sorting

Figure 3.11. Offsets sorting for tool path generation

at the bottom, so the inverse of a depth-first traversal from the root produces an efficient topological sort of the offset segments. For pockets with islands, in general, the partial ordered list is not a tree but an acyclic digraph [3]. The depth-first traversal strategy must be modified so that an offset node is visited after all its predecessors are visited. Figure 3.12 shows the resulting tool path for the example pocket shown in Figure 3.6. The dashed lines are rapid traversal of the tool, and the dotted lines are the original pocket.

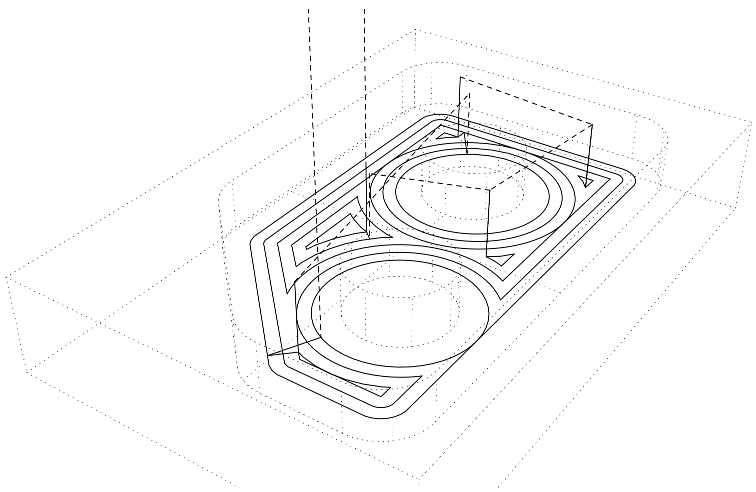
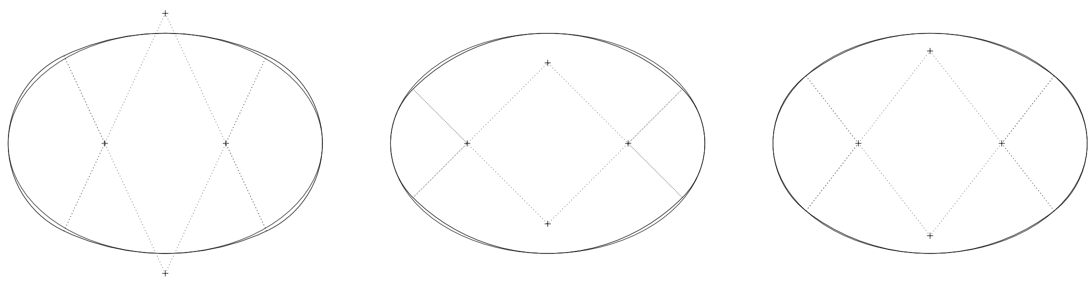


Figure 3.12. Tool path generation of a pocket with islands

3.6.3 Biarc Approximation of NURBS Curves

This research introduces a general divide-and-conquer approach to the biarc approximation for 3D NURBS curves of arbitrary degree. The approximation error is computed to ensure a tighter global tolerance bound. We symbolically compute the distance squared between the curve and the arc center point using a NURBS representation, then numerically bound the distance to the arc center, i.e., d_{min} and d_{max}. Since the distance error $\mathcal{E}$ from any point to an arc is the difference between the distance to the arc center and the arc radius, i.e., $\mathcal{E} = |d - r|$, the maximum error between any arc and a subregion of the curve is the greater of $|d_{max} - r|$ and $|d_{min} - r|$.

Figure 3.13 shows approximation of an ellipse (quadratic NURBS curve) using several different types of biarcs. Approximations using equal-chord-length biarcs and equal-winding-angle biarc, described above, are shown in (a) and (b). From the figure, both equal-chord-length and equal-winding-angle biarcs have arcs on one side of the ellipse, outside and inside respectively. To reduce the approximation error, it is better to have the arcs in both sides of the curve [91]. In (c), we select the closest point on the locus arc to the original curve, i.e., the intersection point between them, to join the biarc. Therefore, end points of the circular arcs are all on the original curve. In all three cases, the number of arcs is set to four and the maximum distance error $\mathcal{E}_{max}$ is calculated. The closest-mid-point biarc has arcs



(a) Equal-chord-length, $\mathcal{E}_{max} = 0.02529$ (b) Equal-winding-angle, $\mathcal{E}_{max} = 0.01845$ (c) Closest-mid-point, $\mathcal{E}_{max} = 0.00738$

Figure 3.13. Biarc approximation of an ellipse ($N = 4$)

in both sides of the ellipse and achieves the smallest error among the three with the same number of arcs.

Interestingly, approximating an ellipse by four tangent continuous circular arcs is a well-known problem in traditional engineering drawing [52]. Figure 3.14 shows two different methods. Both methods can construct the four arcs using traditional drawing tools such as triangles, T-squares, and compasses. In the first method, shown on the left, we first join A and D with a straight line and lay off DF equal to $AO - DO$. This can be done graphically by drawing two circles centered at O and D, respectively (see Figure 3.14(a)). Then, we bisect AF by a perpendicular line which intersects AB and DE at G and H, respectively. G and H are two of the center points of the four arcs. AG and DH are the radii of the two tangent continuous arcs. In the second method, shown on the right, we find the center points by the following equations:

$$OH = AB - DE \text{ and } OG = \frac{3}{4}OH. \tag{3.3}$$

This method should be used only when the minor axis DE is at least two thirds of the major axis AB. Both methods give tighter approximation than the equal-chord-length and equal-winding-angle biarcs methods for this test ellipse. However, the closest-mid-point biarc method still gives a slightly better result than the engineering drawing methods.

Figure 3.15 shows results of the adaptive subdivision biarc approximation of an

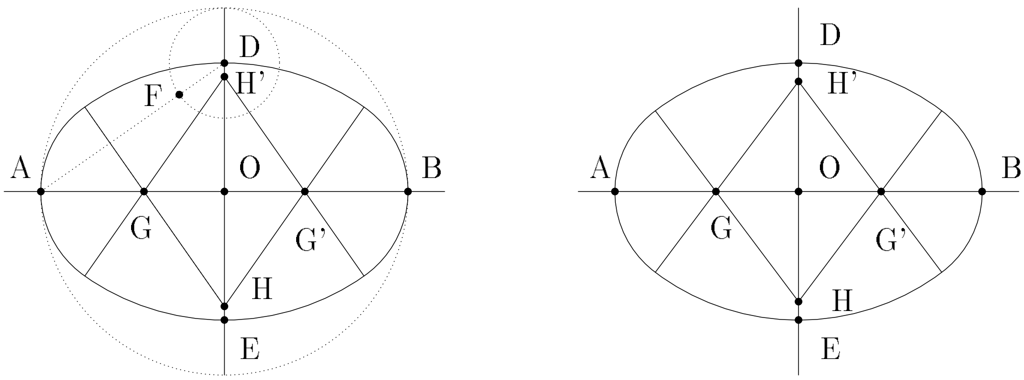
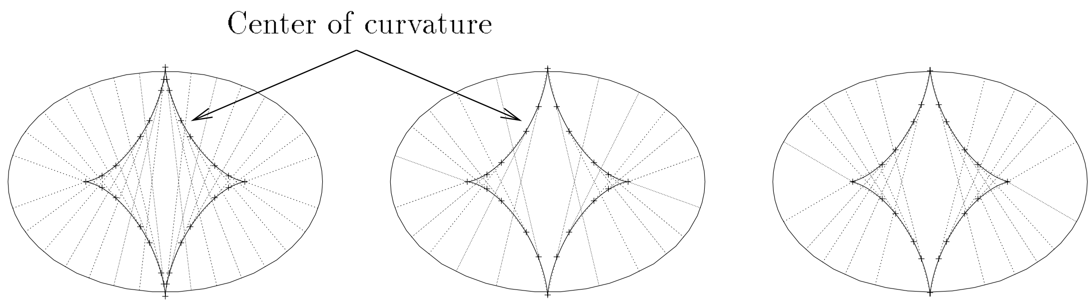


Figure 3.14. Engineering drawing methods to draw an ellipse with four arcs



(a) Equal-chord-length $\mathcal{E}_{max} = 0.0004$ $N = 28$

(b) Equal-winding-angle $\mathcal{E}_{max} = 0.0008$ $N = 20$

(c) Closest-mid-point $\mathcal{E}_{max} = 0.0006$ $N = 20$

Figure 3.15. Adaptive biarc approximation of an ellipse ($\epsilon = 0.001$)

ellipse. Again, the closest-mid-point biarc approximation gives slightly smaller error with the same number of arcs. Table 3.1 lists comparison of the three methods at different tolerances. It shows all three methods converges at about the same rate, despite small differences in numbers of arcs and maximum errors.

One interesting observation is that centers of the arcs follow closely to the centers of curvature of the curve. The center of curvature is the center of the osculating circle [94]:

$$\mathcal{K}(t) = \mathbf{C}(t) + \frac{1}{\kappa(t)}\mathbf{N}(t), \tag{3.4}$$

where $1/\kappa(t)$ is the radius of the osculating circle, called the radius of curvature. For a NURBS curve, the center of curvature can be computed symbolically using a NURBS representation:

Table 3.1. Comparison of biarc approximations at different tolerances

Tolerance	Equal-chord-length		Equal-winding-angle		Closest-mid-point	
0.1	4	0.02529	4	0.01843	4	0.00738
0.01	12	0.00156	12	0.00216	4	0.00738
0.001	28	0.00037	20	0.00079	20	0.00059
0.0001	52	5.215e-5	52	4.691e-5	36	9.972e-5
0.00001	100	9.993e-6	100	7.722e-6	100	9.089e-6
0.000001	228	9.495e-7	220	9.629e-7	244	9.569e-7

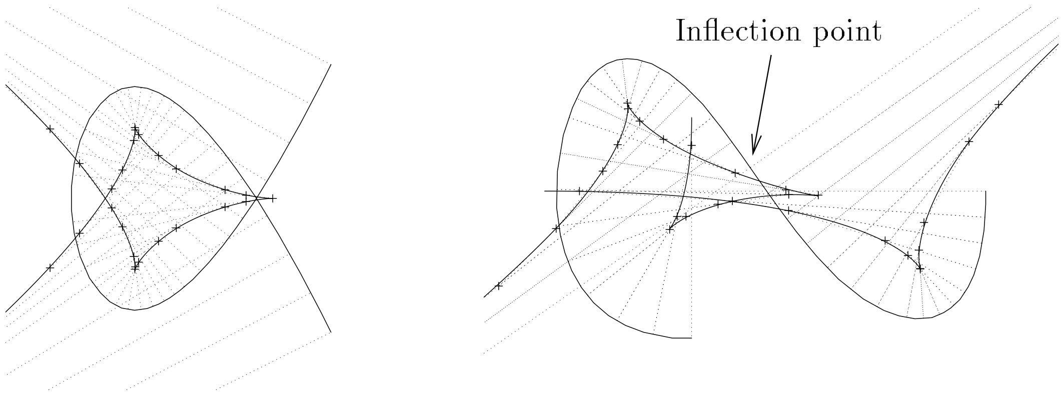
$$
\begin{aligned}
\frac{\mathbf{N}}{\kappa} &= \frac{\kappa\,\mathbf{N}}{\kappa^2} \\
&= \frac{\kappa\,(\mathbf{B}\times\mathbf{T})}{\kappa^2} \\
&= \frac{(\kappa\,\mathbf{B})\times\mathbf{T}}{\kappa^2} \\
&= \left(\frac{\mathbf{C}'\times\mathbf{C}''}{\|\mathbf{C}'\|^3}\times\frac{\mathbf{C}'}{\|\mathbf{C}'\|}\right)\Big/\left(\frac{\|\mathbf{C}'\times\mathbf{C}''\|}{\|\mathbf{C}'\|^3}\right)^2 \\
&= \frac{\|\mathbf{C}'\|^2\,(\mathbf{C}'\times\mathbf{C}''\times\mathbf{C}')}{\|\mathbf{C}'\times\mathbf{C}''\|^2} \\
&= \frac{(\mathbf{C}'\cdot\mathbf{C}')\,(\mathbf{C}'\times\mathbf{C}''\times\mathbf{C}')}{(\mathbf{C}'\times\mathbf{C}'')\cdot(\mathbf{C}'\times\mathbf{C}'')} \qquad (3.5)
\end{aligned}
$$

At inflection points and points on a straight line, where $\|\mathbf{C}''\| = 0$, the center of curvature is not defined.

Figure 3.16 shows two examples of biarc approximation of cubic NURBS curves. Note that, in (b), the curve has an inflection point. The center of curvature is discontinuous at inflection points. It flips sides around the inflection point. The adaptive biarc approximations behaved well under this condition, without the need to split the curve at inflection points.

3.7 Open Pocket and Tool Path Clipping

Very often one needs to machine the outside profile of a mechanical part from the raw stock, or regions (partially) specified by open curves. Conventional pocketing



(a) $N = 39$, $\mathcal{E}_{max} = 0.0005$ (b) $N = 34$, $\mathcal{E}_{max} = 0.0007$

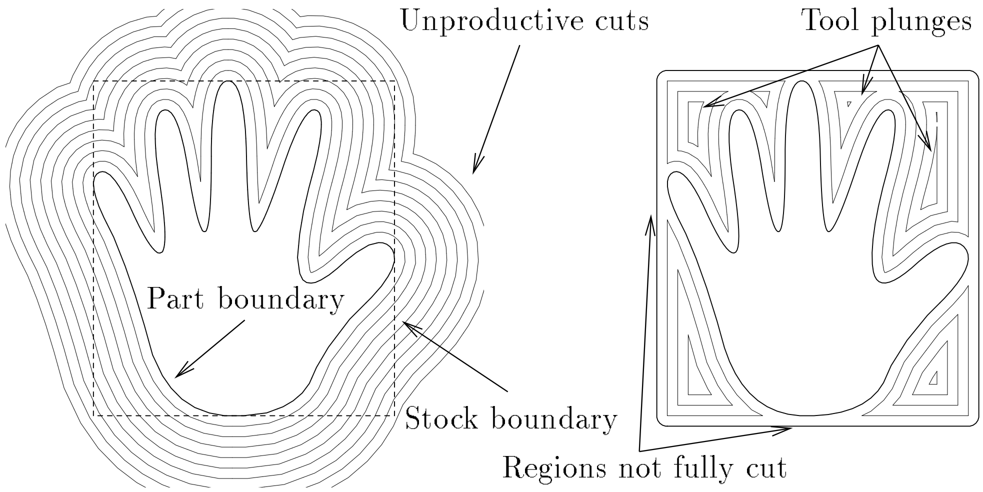
Figure 3.16. Adaptive biarc approximation of cubic NURBS curves ($\epsilon = 0.001$)

algorithms handle only the interior regions of closed profiles. In this section, we extend the skeleton computation algorithms to generate offset tool paths for open regions, which we call "*open pockets.*"

3.7.1 Open Pocket

"Open pocket" is an abstract machining operation to describe the type of milling operations that are common in $2\,{}^1\!/_2$ D features such as profile, pocket, and boss. An open pocket can be defined by a list of curves, each of which can be closed or open, and its total depth. Some of the curves represent boundaries, outer or inner, of the part surface, whereas others represent regions, positive or negative, of the raw stock to be machined. It is assumed that it is safe for the tool to move in regions outside of the stock profile without collision.

The objective of open pocket machining is to make sure that the tool path covers the whole area to be machined, does not gouge into any specified part boundaries, and does not waste time in unproductive cuts outside of the stock boundaries. For example, Figure 3.17 shows a hand-shaped outer profile with a square stock boundary. As shown in (a), the straightforward successive offsets tool path includes many unproductive air cuts. To avoid this problem, traditional pocketing algorithms treat the outer profile as an island of a pocket large enough



(a) Offsets of outer profile (b) Offsets of a pocket with island

Figure 3.17. Machining of outer profile

to enclose the stock. Figure 3.17(b) shows an offset tool path for such a pocket. Note that it is important to make sure that the enclosing profile is at least one tool diameter larger than the part profile. Otherwise, regions of the profile may not get cut cleanly, as indicated in the figure.

An examination of the tool path reveals that instead of machining from the outside of the stock, the tool plunges several times in the middle of the stock. Also, when the tool cuts near the outer boundary of the stock, thin walls of uncut metal are created, which may cause a chattering problem, or break away and damage tool. A better way is to follow the same offset tool path as shown in Figure 3.17(a), but without the portions that are farther than half a tool diameter from the stock outline. Figure 3.18 shows a clipped tool path for the same outer profile and stock boundary. The clipped tool path does not have the above problems.

If a direct clipping method is used, tool path clipping can be computed in $O(lmn)$ time, where l is the number of segments in the stock boundary, m is the number of offset profiles, and n is the number of segments in the part profile. This is in addition to the computation of the skeleton and offsets, $O(mn + n \log n)$, as discussed in Section 3.4. An improvement of the tool path clipping uses a sweep algorithm to find all the intersection points, which requires sorting of the clipping and offsets profiles for $O(l \log l + mn \log n)$.

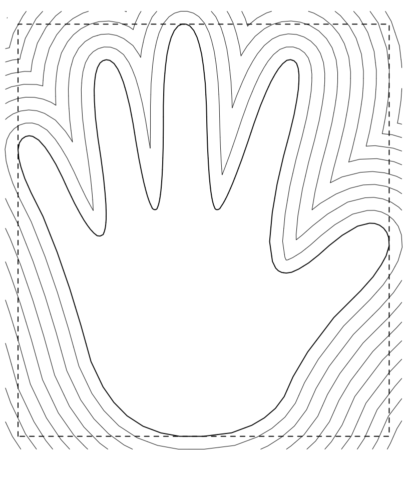


Figure 3.18. Tool path clipping for outer profile

3.7.2 A Fast Offset Clipping Method

In this section, we introduce a fast offset tool path clipping method. This method uses the geometric information in the curve skeleton to reduce the size of the inputs to the clipping algorithm. It identifies the offsets that will be clipped away and the nonexisting intersection points for totally contained offset segments, and does not perform the associated computation. If regions of the Voronoi edges outside the clipping boundary do not generate any offset points, totally clipped offsets segments are not generated. The skeleton is trimmed so it does not create unwanted offset points, so totally outside Voronoi edges are removed, and so partially outside ones are trimmed.

Figure 3.19 shows the clipped skeleton from which the clipped tool path shown in Figure 3.18 was generated. Since the number of Voronoi edges is $O(n)$, skeleton clipping can be computed in $O(l \log l + n \log n)$ time by using a sweep algorithm, where l is the number of segments of the piecewise linear clipping profile. Note that the number of offset profiles, m, is not a factor in the computational complexity of skeleton clipping. To compute m clipped offsets from a clipped skeleton, which has $O(l+n)$ Voronoi edges, needs $O(m(l+n))$ time complexity.

A yet more efficient algorithm for skeleton clipping treats the clipped skeleton as a binary-weighted skeleton. A weighted Voronoi diagram is a Voronoi diagram, in

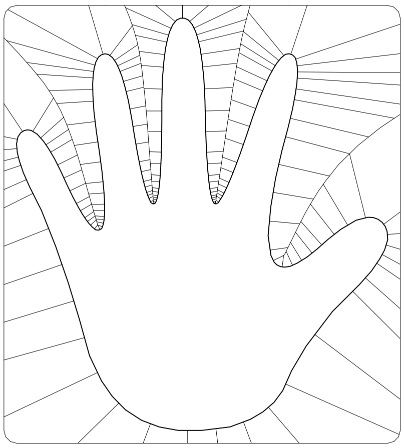


Figure 3.19. Clipped skeleton for an outer profile

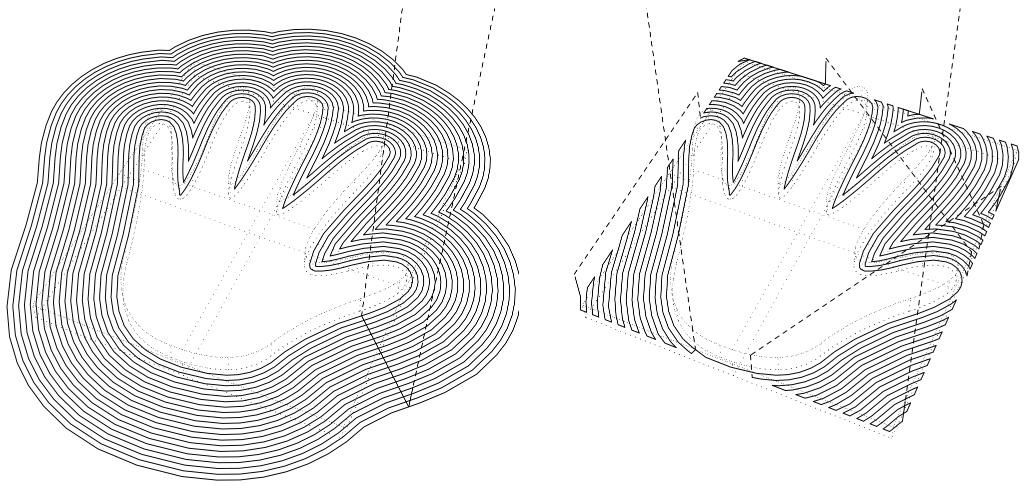
which, the distance to each site is weighted by a weighted factor of that site [78, 116]. A weighted bisector $\mathcal{B}_w$ of sites p_1 and p_2 with weight factors w_1 and w_2 can be defined as:

$$\mathcal{B}_w(p_1, p_2) = \{x| \, \|x - p_1\|/w_1 = \|x - p_2\|/w_2\}. \tag{3.6}$$

A binary-weighted Voronoi diagram is a weighted Voronoi diagram whose weight factors are either zero or one.

Bisectors between sites of unit weight are the same as those of a regular Voronoi diagram. Bisectors between sites of different weight factors, i.e., one and zero, are simply the zero-weighted generators. By assigning a weight factor of zero to the clipping profiles, and one to the pocket profiles, the merge skeleton algorithm can be modified to construct the binary-weighted skeleton that is the same as the clipped skeleton. The merge can be done in $O(n + l)$ time, on the average. Starting from an intersection point, the merge chain is the same as the clipping profile.

Figure 3.20 shows the tool path and simulated machining time for the outer profile using successive offsets, with and without stock boundary tool path clipping. In both cases, all plunges are off the stock material. Table 3.2 lists the machining parameters used in both cases.



(a) Offset tool path (25.9 min.) (b) Clipped tool path (11.4 min.)

Figure 3.20. Tool path simulation of open pocket machining

Table 3.2. Machining parameters for an outer profile

Tool diameter	$^1/_{16}$ inch
Stepover size	$^1/_{32}$ inch
Cut depth	$^1/_8$ inch
Spindle speed	5200 rpm
Feed rate	6.93 ipm

Machining time calculation of the tool path simulator is tuned for the Monarch VMC45 vertical machining center in the Advanced Manufacturing Laboratory [4]. The clipped tool path computation is 11.4 minutes, a 55% reduction in machining time from 25.9 minutes for the unclipped. The parameters of the tool path simulation include the average tool change time, the rapid traversal rate, and the acceleration/deceleration for each axis, etc.

3.8 3D Surface Contouring

In this section, we discuss basic computation issues for generating tool paths for 3D surface contouring using ball-end tools, the finish or the final roughing operation from our feature decomposition. As discussed in Section 3.1.2, center of the ball-end moves on a surface that is an offset constant distance from the part surface. A common approach used in many CAM systems to generate tool paths for sculptured surfaces uses a point-based tool offset sampling technique [58, 66, 68]. In this method, the tool is guided along a 2D pattern defined by the drive surfaces, such as parallel lines or isoparametric curves, or the offset contours of 2D profiles similar to those in a pocketing operation [66, 134]. At each point on the guide curves, offset surface points are sampled by projecting a ray through the curve points in the tool axis direction. To produce a gouge-free tool path, the first tool-surface contact point along the projection ray is computed to set the Z coordinates of the cutter locations.

Various methods have been used to improve the efficiency of the tool/surface interference tests. One method uses an adaptive subdivision-based approximation method to find the first contact points of the tool with the part surface [58]. In another method, a simulation-based approach is used [68]. The part surface is first rendered with sampled points set. The tool/surface intersection is computed using the sampled surface points. A surface normal vector is associated with each point and used to improve the tool position using tangent planes. In a third method, the sampled surface points are converted into a polyhedron with triangular facets [66]. Tool paths are then generated from the polyhedron.

Advantages of these offset points sampling methods include global gouge detection, support for multiple surfaces, choices of tool path patterns. However, there are drawbacks in this type of tool path generation methods too. The computation can be slow if a tight tolerance is required, in that case more offset points need to be computed. Another problem is controlling the scallop height around regions of nearly vertical surfaces. It is difficult to adjust the horizontal stepover size so a uniform scallop height can be maintained, especially when there is significant surface slope variation along a guide curve.

A different approach to surface tool path generation method computes offset surfaces first [42, 80]. The offset operator can be defined by moving each point on the surface in its normal direction for a constant distance, i.e.,

$$\mathcal{O}(\mathbf{S}(u,v), d) = \mathbf{S}(u,v) + d\,\mathbf{n}(u,v), \tag{3.7}$$

where $\mathbf{n}$ is the unit surface normal vector and d is the offset distance. The offset of a NURBS surface usually is not a NURBS surface. Definitions, properties, and algorithms to approximate offset surfaces are presented in Chapter 5.

In one method, intersections between the offset surface and the drive surface is computed. The offset surface is approximated as a set of polygons [80]. To prevent gouging, self-intersections of the piecewise linear intersection curve are removed, and gaps of the intersection curve around tangent discontinuous ridges are filled with linearly approximated circular arcs. In another method, the ball-end tool is moved along isolines on the offset surface. Symbolic computation is used to ensure a global bound on the surface offset errors [42]. An adaptive isoline method is used, which measures the distances between corresponding parametric points of two isolines, i.e.,

$$\|\mathbf{S}(u_1, v) - \mathbf{S}(u_2, v)\| \text{ or } \|\mathbf{S}(u, v_1) - \mathbf{S}(u, v_2)\|, \tag{3.8}$$

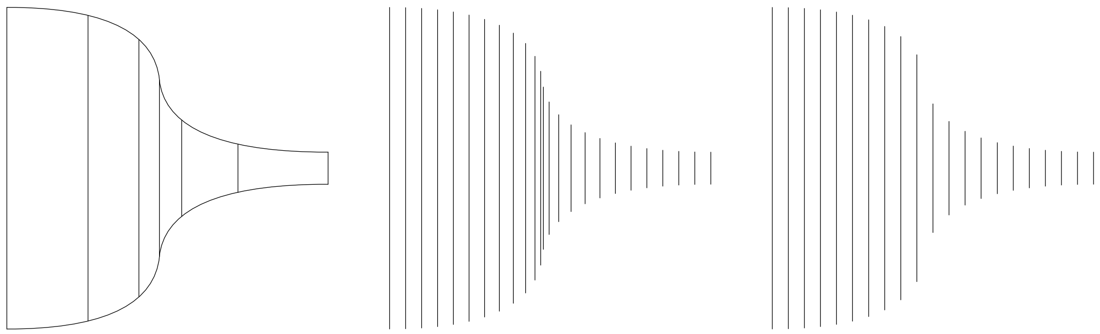
and inserts an isoline in between if the maximum distance is greater than a specified value. The adaptive isoline method can also remove parts of the isolines that are too close together by a threshold of the tool path spacing.

In this research, we improve the adaptive isoline method in several ways. First, the surface offset method is improved which converges faster than that in reference [42]. An adaptive surface refinement method based on minimum knot insertion is used to improve the offset errors until a prescribed tolerance is met. Details of our offset algorithms and quantitative comparison between different surface offset methods are presented in Chapter 5.

Second, self-intersections of the offset surfaces are trimmed away to prevent gouging. Using global properties of the surface, we derive a property shared by self-intersecting surfaces. This property is used to modify existing surface/surface intersection methods to find the self-intersection curves of surfaces. Mathematical definitions and proofs of the self-intersection conditions are presented in Chapter 6.

Third, the distances between isolines are measured to extract more evenly-spaced isolines. The adaptive isoline method measures distance between the corresponding parametric points of isolines. This is not the same as the distance between tool cuts since the corresponding points are usually not the closest points to each other on the isolines. We have created an approximation method which gives more evenly-spaced isolines even for surfaces whose isolines are not perpendicular to each other in both parametric directions.

Figure 3.21 shows a ruled-surface between two planar NURBS curves and its extracted isolines. Using the adaptive isoline method as shown in Figure 3.21(b), more isolines are added in the middle region of the surface. Despite the mathematical guarantee of a global bound on the maximum distance between isolines, the unevenly-spaced nature of the adaptive isoline method is a drawback for surfaces whose isolines are not perpendicular in both parametric directions.



(a) A ruled-surface (b) Adaptive isolines (c) Evenly-spaced isolines

Figure 3.21. Adaptive isolines and evenly-spaced isolines of a ruled surface

To extract evenly-spaced isolines as shown in Figure 3.21(c), one needs to measure the maximum spacing between isolines. Unfortunately, such computation is very complicated for freeform surfaces and requires costly numerical computation. We use an approximation method that first refines the surface and computes the distance between the segments of the refined control mesh as if a tool is to be swept along one of the parametric directions. We establish an approximation of the parametric/Euclidean distances mapping between mesh segments. An evenly-spaced sampling on the Euclidean distances of the mapping function gives us the parametric values at which we extract the isolines (see Figure 3.22).

We compute the distance between two mesh segments by first observing it is constant if the segments are parallel. Otherwise, we find the intersection point between the two lines, or average of the two closest points if the lines are skewed.

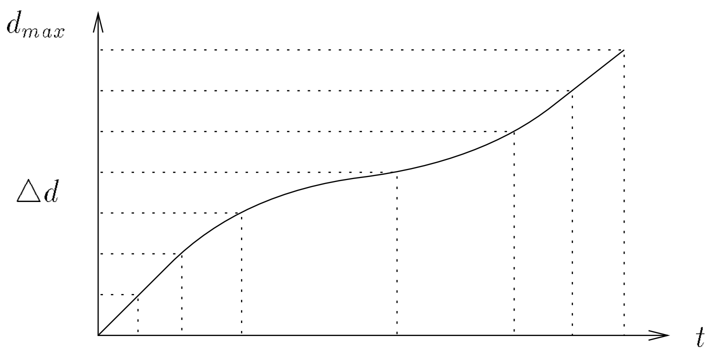


Figure 3.22. Evenly-spaced resampling of the distance mapping function

Using this point as the center point, we find the smallest sphere that bounds the two line segments, and the distance between the intersection points of the sphere and the two line equations (see Figure 3.23).

Figure 3.24 shows the rough and finish tool paths for the compressor disk example shown in Figure 2.10. To compute the skeletons and the offsets of the open profiles, point sites are added at the open end points, in a way similar to the preprocessing of concave corners. In the finish tool path, the intersection curves of the offset surfaces are included to machine the fillet surfaces at the bottom of the blades.

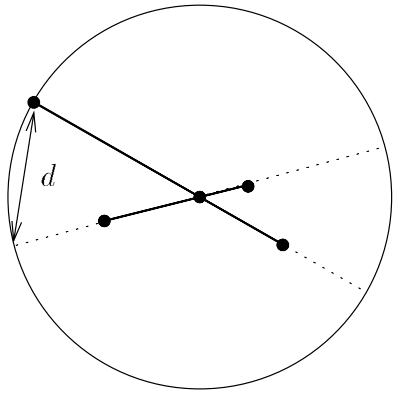


Figure 3.23. Distance computation between isoline segments

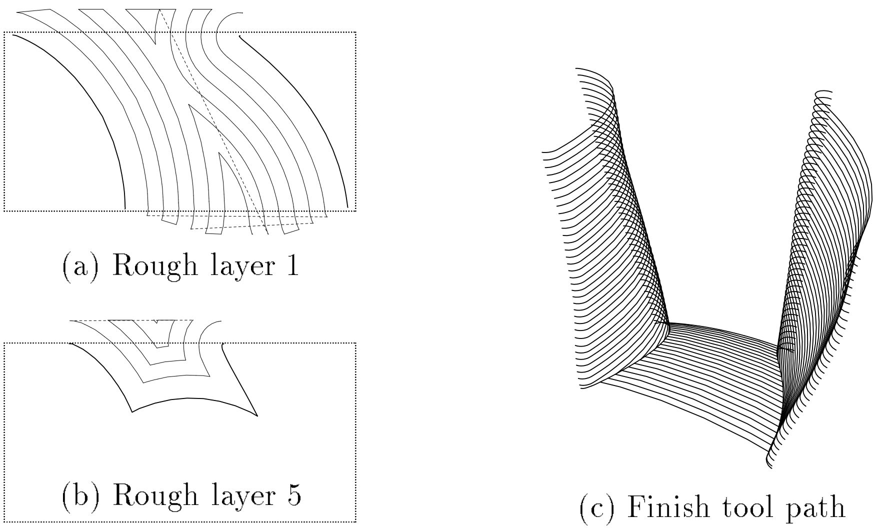


Figure 3.24. Roughing and finish tool paths for a compressor disk

CHAPTER 4

TOOL-DRIVEN DECOMPOSITION

A crucial component of the feature-based approach to process planning and NC part programming described in Chapter 2 is to decompose features in a feature-based design into sequences of machining operations. A machining operation, such as drilling, tapping, reaming, pocketing, or contouring, is a machining entity that can be handled by a single tool and a fixed strategy for generating the tool path.

For example, a hole feature can be decomposed into spot drilling, drilling, chamfering, and reaming operations (see Figure 4.1). Depending on the feature parameters, e.g., hole diameter and depth, and the type of stock material, a tool is selected from the tool crib and driven at an appropriate feed rate and spindle speed to accomplish the designated operation [27]. Most NC machines support canned

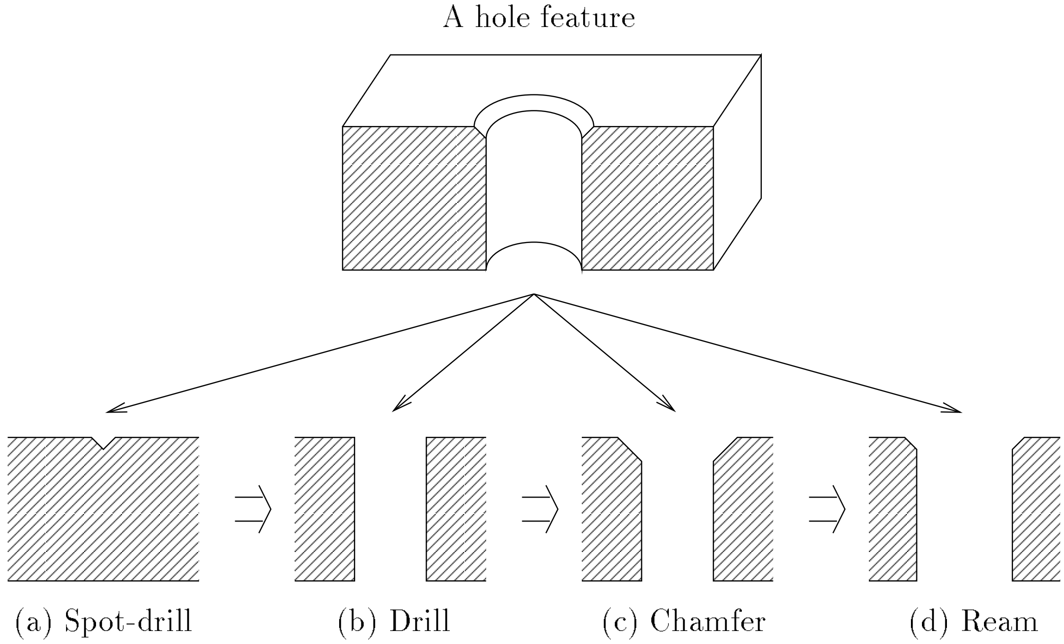


Figure 4.1. Decomposition of a hole feature

cycles, i.e., fixed subroutines of linear and rapid moves, to simplify programming of point-to-point operations. For milling operations such as pocketing and contouring, however, tool path generation strategies are more complicated, usually include the computation of offset curves or surfaces.

4.1 Multiple Tools Decomposition

In Figure 4.1, a sequence of four operations is used to machine a feature as simple as a hole, which basically has a cylindrical shape and is probably modeled with a cylinder primitive in many CAD systems. Each operation requires a different type of tool and uses a different feed rate and spindle speed. A "tool-driven feature decomposition" decomposes a feature according to the functionality of the available tools such that, each operation can be performed in an efficient way. Of course, the use of multiple operations and tools imposes the extra overhead of tool setup and tool change. This overhead can be reduced when machining a pattern of the same feature, or multiple copies of the same part, by reordering the decomposed operations and reusing the same tool whenever possible.

With so many different types of specialized tools for point-to-point operations, it is a common practice to utilize multiple tools in machining hole features. In fact, hole features were designed to accommodate many commonly used combinations of point-to-point operations in mechanical part machining [39]. Examples of automatic decomposition for hole features can be found in reference [27]. The decomposition is straightforward, so are the tool selection and tool path generation.

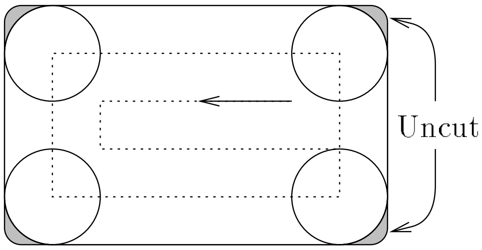
However, tool-driven decomposition for $2\,{}^1\!/_2\,$D or 3D milling features is more complicated than for hole features. Traditionally, the only type of automatic decomposition for milling features has been to have a roughing operation followed by a finish operation. The main purpose of having the second finish operation is to achieve a desirable surface finish and accuracy, by using a slower feed rate, a faster spindle speed, a smaller depth of cut, and a sharper tool. Sometimes, the same milling tool is used in both operations to reduce manual tool setup overhead, especially when machining prototypes using soft materials such as wax or aluminum.

One problem of using a single tool or multiple tools of the same diameter to machine the whole milling feature is that the tool diameter has to be small enough to cut the smallest details and corners of a feature. However, a smaller tool requires a smaller depth of cut. Moreover, since the tool is not as strong, a smaller tool also needs to be driven at a slower per-tooth feed rate to prevent excessive tool deflection or tool breakage. A simple rule of thumb in tool selection for milling features is to use the largest tool that can fit the smallest corner of a feature.

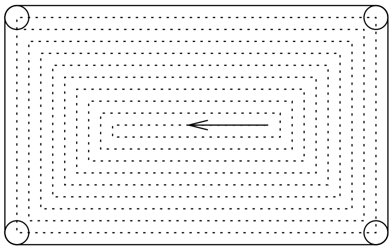
For example, Figure 4.2 shows a rectangular pocket with small corner radii. On the left, the big tool only needs few passes but leaves the corners uncut. On the right, the small tool cuts the whole pocket cleanly but requires many passes. A more efficient way of machining a feature like this is to decompose it into two, or more, operations. The first operation uses a big tool to rapidly machine away the majority of the area. The second operation uses a small tool for the remaining uncut corner regions. This type of tool-driven feature decomposition can also be applied to 3D features with sculptured surfaces. To machine a surface using a ball-end milling tool, radius of the ball-end tool must be less than or equal to the minimum radius of curvature. Otherwise, the part surface is either gouged by the tool or not cut cleanly at regions of high curvature.

In this chapter, as well as the following chapters, computational algorithms and approaches to support practical use of the tool-driven decomposition for milling features are discussed. These include, but are not restricted to:

- Identifying regions that can be machined by a larger tool and generating a



(a) Big tool cannot cut small corners.



(b) Small tool needs many passes.

Figure 4.2. Machining a pocket with small corner radii

gouge-free tool path for it, i.e., cut only the regions a tool can cut.

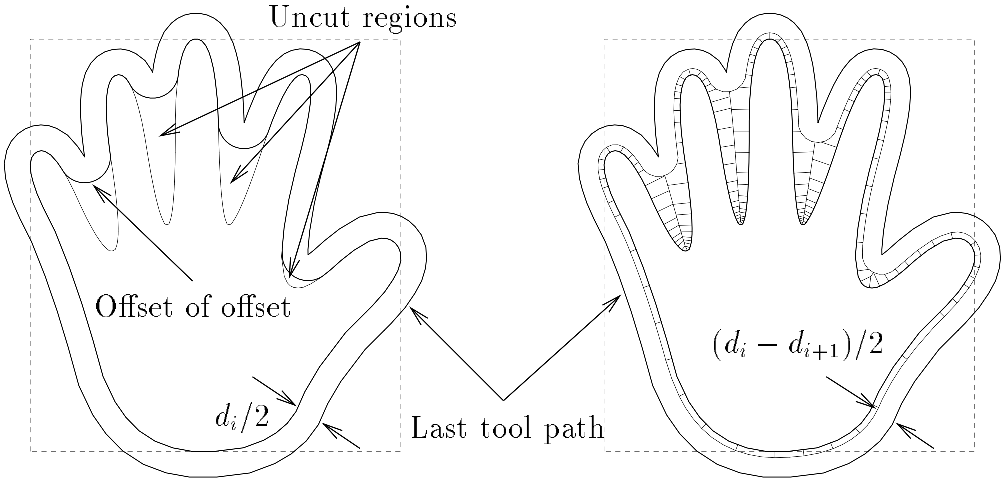
- Identifying the remaining uncut regions and generating an efficient tool path using a smaller tool. i.e., cut only the regions need to be cut.
- Determining an efficient combination of different diameter tools for any given feature, i.e., deciding how many tools to use and what they should be.

This chapter first presents techniques and algorithms for using multiple tools on $2\,{}^1\!/_2$ D features defined by planar curves. Generalizations to features defined by 3D surfaces are then presented.

4.2 Multiple Tools Path Generation

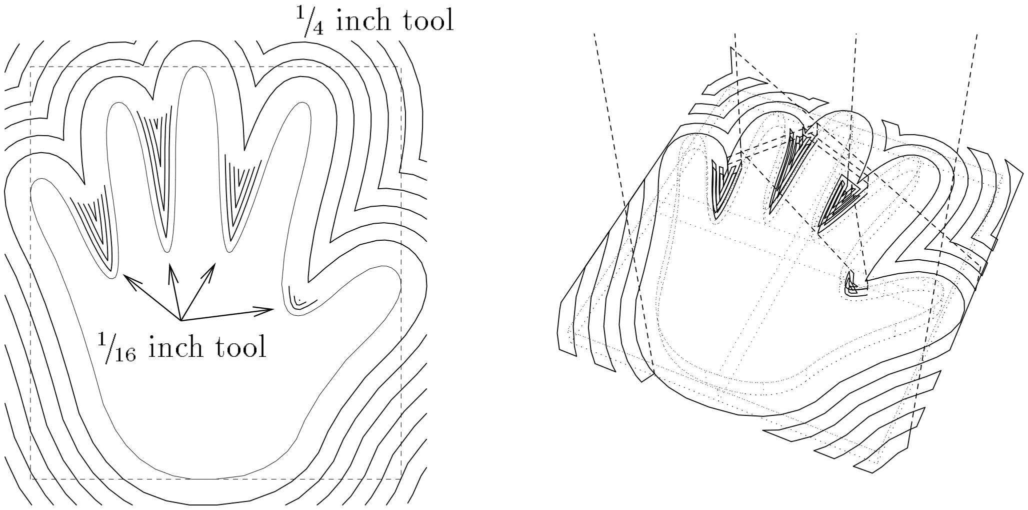
In this section, we discuss tool path generation using multiple tools of different diameters to greatly reduce the total machining time. To machine a milling feature using multiple tools, the tools are applied in order of decreasing diameter. Let d_i and d_{i+1} be the respective diameters of tool i and tool $i+1$ from two consecutive operations of the same feature, where $d_i > d_{i+1}$. To identify the uncut regions of tool i, the offset of the last tool path at $-d_i/2$ is computed (see Figure 4.3(a)). This is the cut boundary of the i-th operation. The area between the cut boundary and the feature boundary is the uncut region that tool $i+1$ must cut. The tool path for tool $i+1$ is clipped against the cut boundary (see Figure 4.3(b)). Figure 4.4 shows the clipped offsets and tool path for using multiple tools, ${}^1\!/_4$ inch and ${}^1\!/_{16}$ inch, on the hand-shaped profile.

Multiple tool decomposition can greatly reduce the total machining time, especially when multiple Z-steps are required for a small tool. Table 4.1 lists different machining parameters for tools of different diameters. The hand-shaped profile has a ${}^1\!/_8$ inch total depth. It will take two Z-steps if a ${}^1\!/_{16}$ inch tool is used. Table 4.2 lists the per-tool machining time (minutes) for process plans with different tool diameter combinations. Plan 1 uses only one tool, but the rest use two tools. Plan 4 gives the lowest total machining time, 5.87 minutes. That is a 500% improvement in total machining time over the 31.6 minutes for plan 1.



(a) Uncut regions (b) Clipped skeleton

Figure 4.3. Uncut regions and clipped skeleton for multiple tools machining



(a) Clipped offsets (b) Clipped tool path

Figure 4.4. Clipped offsets and tool path for multiple tools machining

Table 4.1. Machining parameters of different tool diameters

Tool diameter (inch)	$^1/_{16}$	$^1/_8$	$^3/_{16}$	$^1/_4$	$^3/_8$	$^1/_2$
Stepover size (inch)	$^1/_{48}$	$^1/_{24}$	$^1/_{16}$	$^1/_{12}$	$^1/_8$	$^1/_6$
Cut depth (inch)	$^1/_{16}$	$^1/_8$	$^1/_8$	$^1/_8$	$^1/_8$	$^1/_8$
Spindle speed (rpm)	5200	5200	5200	5200	4074	3055
Feed rate (ipm)	6.93	8.50	10.40	13.97	16.30	18.33
Number of Z-steps	2	1	1	1	1	1

Table 4.2. Machining time (minutes) of different tool combinations

Diameter	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5	Plan 6
$^1/_{16}$	31.60	0.87	1.89	2.96	4.93	5.98
$^1/_8$	—	7.62	—	—	—	—
$^3/_{16}$	—	—	4.93	—	—	—
$^1/_4$	—	—	—	2.91	—	—
$^3/_8$	—	—	—	—	2.06	—
$^1/_2$	—	—	—	—	—	1.36
Total	31.60	8.49	6.82	5.87	6.98	7.34

In the above example, combinations of two tools are used to cut the same part. For features of complex shapes, more tools could be used to reduce the machining time. To compensate for the tool setup overhead, a per-tool setup cost is added. We can find a decomposition to reduce the machining time for a given feature by tool path simulation of different numbers and diameters of tools. The smallest tool is determined by the minimum clearance of the skeleton. Any tool larger than that is a possible candidate for the tool-driven decomposition. Tools that are too big to cut anything can be excluded immediately.

For n available tool diameters and m maximum number of selected tools, the total number of tool combinations is $\sum_{i=1}^{m} C\binom{n}{i}$, where $1 \leq m \leq n$. Suppose there are fifty different diameters in all end-mill tools. Among those, 15 tools are commonly available in a machine shop. For the given sized pocket, half of them are suitable for the job. And, assume the maximum number of tools for a feature is four. Excluding the final tool, we have $n = 7$ and $m = 3$. Then, the total number of tool combinations is

$$C\binom{7}{1} + C\binom{7}{2} + C\binom{7}{3} = 7 + 21 + 35 = 63.$$

That is a lot of tool paths to generate if the "brute-force" exhaustive search approach is used. Unless the number of tools, both n and m, is very small, the search space is probably too big for practical use. In the following, we present heuristics and geometric analysis that help narrow the search space.

4.3 Multiple Tool Selection

One approach for selecting multiple tools is to keep the ratio of previous tool size to next tool size constant, for example $1/2$. That is, the next tool is twice the size of the previous tool. This is terminated when either the tool diameter is too big to have a significant contribution, or the maximum number of tools is reached. Such an algorithm is common and frequently used for search algorithms, compression schemes, and multiresolution representations.

Figure 4.5 shows a pocket with complex island profiles which resemble a Chinese character, and the clipped offsets for tool path generation using multiple tools. The simulated tool path is shown in Figure 4.6, and the per-tool machining time is listed in Table 4.3. Machining parameters for each tool are the same as the ones listed in Table 4.1. Using the doubling approach, four tools are used to machine a complex pocket. More than nineteen times speedup, from 235.62 to 12.39 minutes, is achieved by this multiple tool decomposition.

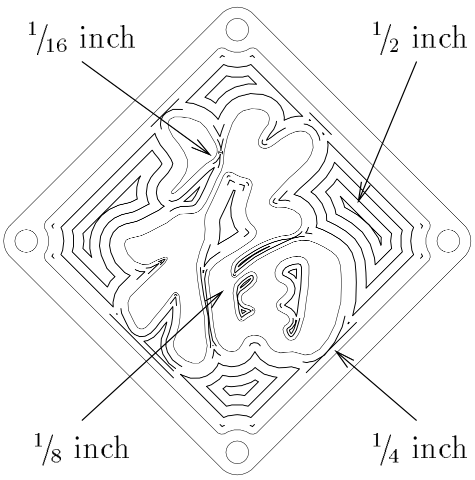


Figure 4.5. Clipped offsets of a complex pocket

Table 4.3. Machining time (minutes) of a complex pocket

Multiple tools	Tool 1	Tool 2	Tool 3	Tool 4	Total	Single tool
Diameter (inch)	$1/2$	$1/4$	$1/8$	$1/16$		$1/16$
Time (minutes)	3.74	3.68	3.65	1.32	12.39	235.62

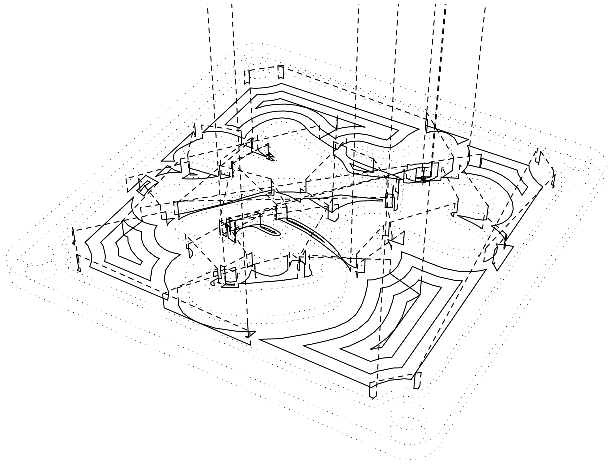


Figure 4.6. Tool path simulation of a complex pocket using multiple tools

The selected diameters are as important. Sometimes, if the tools are not chosen wisely, using more tools may actually increase the total machining time. For example, Table 4.4 lists the machining time of different tool combinations for the pocket shown in Figure 4.7. Plan 5 uses three tools but needs more time than plan 2, which only uses two tools.

An adaptive strategy that we have developed starts from a reasonable combination, then adjusts tool diameters locally. In previous section, we presented algorithms for clipping profiles for multiple tools path generation that depend only on the diameters of the current and the previous tools. Therefore, changing one tool diameter affects only the machining time of two operations, itself and the one follows it.

Table 4.4. Machining time (minutes) of different tools

Diameter	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5
$^1/_8$	41.35	1.47	1.47	1.47	10.78
$^1/_4$	—	5.70	2.55	3.02	—
$^3/_8$	—	—	—	—	1.28
$^1/_2$	—	—	1.58	—	—
$^3/_4$	—	—	—	0.63	0.63
Total	41.35	7.17	5.60	5.12	12.69

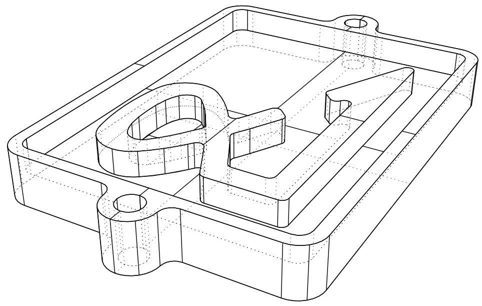


Figure 4.7. A 2 ½ D part

From examples given in this section, we found that when the per-tool contribution is evenly distributed, the total machining time is usually among the lowest. For example, in Table 4.2, the best tried combination, plan 4, spends about the same amount of time on both tools. Whereas in Table 4.4, plan 2 and plan 5 use too much of the ¼ inch and the ⅛ inch tools, respectively. Thus a second approach for multiple tool selection is to use tools equally as much as possible.

The constant ratio heuristic can be used to find an initial tool selection. Then, starting from the smaller tool, locally adjust the diameters to reduce the total machining time by increasing/decreasing the diameters of over/under utilized tools. Since the time spent on a tool is proportional, although not linearly, to the diameter of the previous tool, the direction of searching can be determined by whether the tool is over-used or under-used. For each tool, we find a diameter that gives a balanced usage from among its neighbors. This search can be done linearly until such a tool is found, or by using a binary search on the monotonic time/diameter function.

4.4 Tool Cut Index

In this section, we present a new technique to obtain a better multiple tool selection. Instead of minimizing the machining time or the simulation time, which requires the tool path to be generated and is dependent on a particular machine type, we use an objective function that can be computed from the geometry of a feature and the tool diameters.

We define the "cut index" of a tool and analyze it to provide an efficient estimation of the contribution from each tool. By minimizing the total cut index, the selected tool diameters and the total number of tools can be optimized based on the geometry and shape of the feature, without generating tool paths. Other factors are assumed to be constant among all tools and ignored in the following derivation and equations.

Definition 4.1 *The* cut index *of a tool for a pocket is the ratio between the area of the region cut by a particular tool and the (cross-sectional) area of the tool itself. Let A be the area of the region cut by a tool with diameter d. Ignoring the ${}^4\!/_\pi$ constant, the cut index CI is defined as:*

$$CI = A/d^2. \tag{4.1}$$

Imagine the milling machine as a stamping machine, although mechanically the milling process is in no way related to the stamping process. Then, intuitively, the cut index represents the number of punches required to cover an area using the given tool. The smaller the tool diameter is, the longer it takes to cover a given area. For a given shape of pockets, the length of tool path is roughly proportional to the area A of the cut region divided by the step over size, which is usually a ratio of the tool diameter d for a particular machining job. Therefore, we have $T \propto A/(d\,f)$, where f is the feed-per-minute feed rate.

The net effect of different tool diameters over the feed rate is very complex. A complete analysis of the problem would also include the rigidity of the machine tool and the fixture setup, as well as the thermodynamics of the cooling systems [146]. In general, for same depth of cut, a bigger tool can be driven at a faster feed rate than a smaller tool. We use $f \propto d^\alpha$ as a simplified mathematical model for the relationship between the feed rate and the tool diameter, where $\alpha \geq 1$. If $\alpha = 1$, we have $T \propto A/(d\,f) \propto A/d^2 = CI$. Therefore, the cut index of a tool can be used as a relative measure of contribution the tool has on a pocketing operation.

To compute the cut indices CI_i for a sequence of pocketing operations with decreasing tool diameters d_i, the cut area A_i for each tool must be computed first.

Unfortunately, the cut area A_i for tool i depends not only on its diameter d_i but also on the diameter d_{i-1} of the tool before it. Instead of computing the cut index directly, which requires the sequence of tool diameters being determined first, we compute the cumulative histogram of the cut area. The cumulative cut area $\widehat{A}_i$ for tool i is the area cut by all tools through d_i, inclusive, i.e., $\widehat{A}_i = \sum_{j=1}^{i} A_j$. Since tools are applied in a decreasing diameter order, $\widehat{A}_i$ is the same as the machinable area if a single tool of diameter d_i were used. Figure 4.8 shows a cumulative histogram of the cut area as a function of the tool diameter.

The domain of $\widehat{A}$ is restricted to the interval $[d_{min}, d_{max}]$, where

1. d_{max} is the diameter of the largest tool that can fit in the pocket. Any tool diameter larger than d_{max} will not cut anything.

2. d_{min} is the diameter of the largest tool that can cut the pocket cleanly. Any tool diameter smaller than d_{min} can be used to cut the whole pocket. There should be only one tool selected that is equal to or less than d_{min}.

3. As the tool diameter decreases, the cumulative cut area increases until it reaches the total area of the pocket A_{total}.

4. The cut area of tool i can be computed from the cumulative histogram for any combination of diameters d_{i-1} and d_i, i.e., $A_i = \widehat{A}_i - \widehat{A}_{i-1}$.

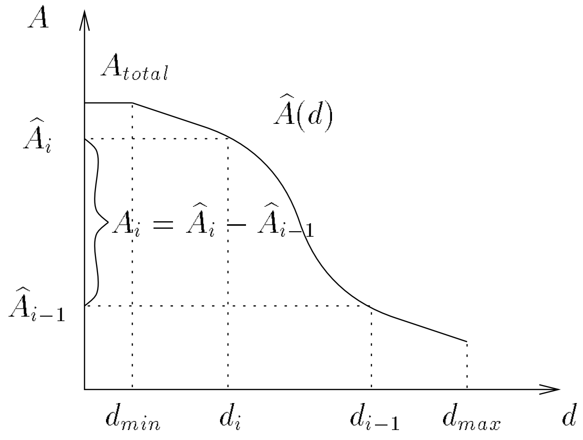
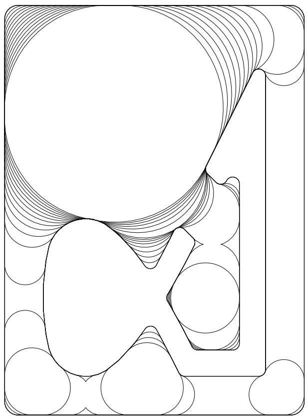


Figure 4.8. Cumulative histogram of the cut area

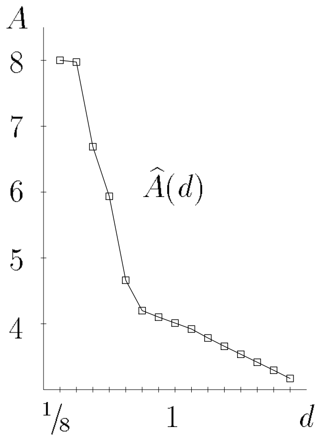
4.5 Cumulative Cut Area

The cumulative cut area histogram can be computed in many ways. Computing the area enclosed by the offset curves plus half the tool diameter is one such way. Figure 4.9(a) shows the cut boundaries of different tools. The cumulative cut area histogram is shown in (b). In this example, $d_{min} = {}^1\!/_8$ and $d_{max} = 1\,{}^7\!/_8$, with increments of ${}^1\!/_8$ inch.

Table 4.5 lists the cut indices of tool combinations from Table 4.4. The cut index results are comparable to the simulation time results. The small differences between the two could be from the extra time spent to start/end each tool path away from previous cuts. To accurately account for this extra cut area is not easy, but an error like this is small compared to the other assumptions/approximations



(a) Cut profiles



(b) Cumulative histogram

Figure 4.9. Cut profiles and cumulative histogram

Table 4.5. Cut indices of different tool combinations

Diameter	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5
${}^1\!/_8$	512	2	2	2	83
${}^1\!/_4$	—	128	33	60	—
${}^3\!/_8$	—	—	—	—	18
${}^1\!/_2$	—	—	24	—	—
${}^3\!/_4$	—	—	—	7	7
Total	512	130	59	69	108

already made in the cut index formulation.

The computation of each offset area requires at least $O(n \log n)$, where n is the number of boundary/offset segments. For m tools, there are m areas, so, the total computational complexity is $O(mn \log n)$, which could be as expensive as computing the clipped skeletons/offsets for m tools. An approximate method to compute the cumulative cut area is presented below. It uses integral properties of plane offset curves to approximate the area between the offset tool path and the cut profile. Then, the cumulative cut area $\widehat{A}$ is the sum of the area enclosed by the offset tool path A_p and area of the offset region A_o, i.e., $\widehat{A} = A_p + A_o$

Lemma 4.1 *Let* $\mathbf{C}$ *be a regular plane curve with total length* S*. The total length* S_o *of a nondegenerate offset* $\mathbf{C}_o$ *at signed distance* t *from* $\mathbf{C}$ *is given by:*

$$S_o = |S + t\,\Delta\theta|, \tag{4.2}$$

where $\Delta\theta$ *is the total rotation of the normal* $\mathbf{N}$ *to* $\mathbf{C}$*, measured in the right-handed sense defined by* $\mathbf{z}$*. The area* A_o *between* $\mathbf{C}$ *and* $\mathbf{C}_o$ *is given by:*

$$A_o = \frac{1}{2}(S + S_o)|t|. \tag{4.3}$$

Detail discussion and proof of Lemma 4.1 can be found in reference [47]. For cut profile $\mathbf{C}_o$ and offset tool path $\mathbf{C}$, both t and S_o are nonnegative. Combining Equations 4.2 and 4.3 together, we get:

$$\begin{aligned} A_o &= \frac{1}{2}(S + S_o)t \\ &= \frac{1}{2}(S + S + t\,\Delta\theta)t \\ &= S\,t + \frac{1}{2}\Delta\theta\,t^2. \end{aligned} \tag{4.4}$$

Table 4.6 lists the exact cumulative cut area $\widehat{A}$ and the area approximated by $A_p + A_o$ for the example shown in Figure 4.9 to three decimal places. Note that the numbers are different only at two diameters, $^3/_8$ inch and $^1/_2$ inch, when the cut profile contains valley points and is self-intersecting (see Figure 4.9(a)). In this situation, the approximation method counted the overlapped area twice.

Table 4.6. Cumulative cut area approximation for different tool diameters

$d = 2t$	$^1/_8$	$^1/_4$	$^3/_8$	$^1/_2$	$^5/_8$	$^3/_4$	$^7/_8$	1
$\widehat{A}$	8.001	7.975	6.689	5.936	4.662	4.200	4.102	4.012
$A_p + A_o$	8.001	7.975	6.703	5.939	4.662	4.200	4.102	4.012

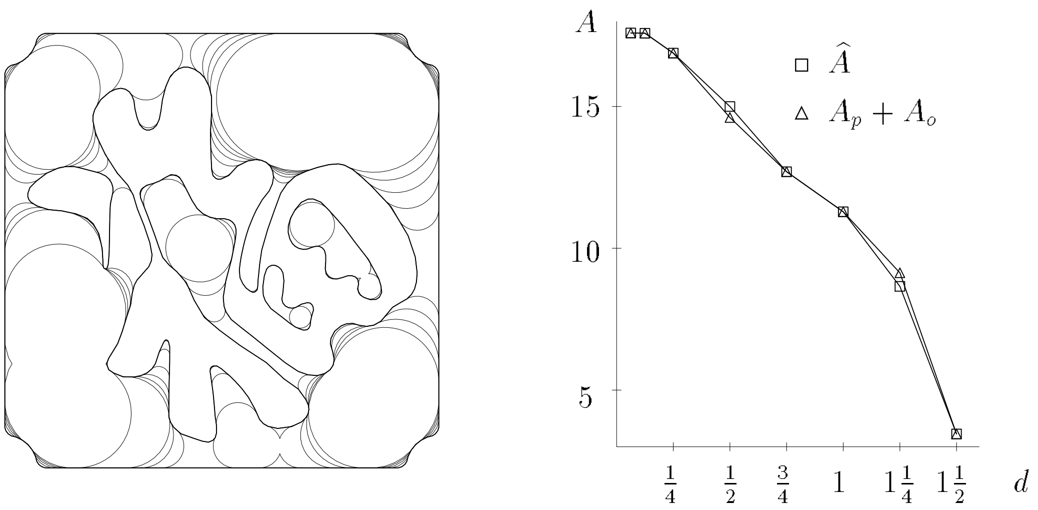
Figure 4.10 shows the cut profiles and the cumulative cut area histogram for the complex pocket from Figure 4.5. The $^1/_{16}$ inch tool diameter is the d_{min} for this pocket. Again, the approximation method gives very close results to the exact cut area.

4.6 Automatic Multiple Tool Selection

Instead of using the tool path simulation time, the tool cut index is used to measure the contribution of each tool. Two variations of cut index functions are defined below.

Definition 4.2 *The* cumulative cut index *function* $\widehat{CI}(d)$ *of a feature is the ratio between the cumulative cut area* $\widehat{A}(d)$ *and the cross-sectional area of the tool. Ignoring the* $^4/_\pi$ *constant, the cumulative cut index is defined as:*

$$\widehat{CI}(d) = \widehat{A}(d)/d^2, \tag{4.5}$$



(a) Cut profiles (b) Cumulative cut area histogram

Figure 4.10. Cut profiles and cumulative cut area histogram of a pocket

where d is the tool diameter.

Corollary 4.2 *The cumulative cut index function* $\widehat{CI}(d)$ *is monotonic decreasing with increasing* d*.*

This is obvious since the cumulative cut area $\widehat{A}$ is a monotonic nonincreasing function with increasing d.

Corollary 4.3 *Let* d_i*,* $i = 1, \ldots, n$*, be the sequence of tools used in a pocketing operation, where* $d_i < d_{i-1}$*. Then, for the first tool* d_1*, the cumulative cut index and the cut index are the same, i.e.,*

$$\widehat{CI}(d_1) = CI(d_1).$$

Proof: Since there is no cut profile tool path clipping for the first tool, the cut area A is the same as the cumulative cut area $\widehat{A}$. Therefore,

$$\widehat{CI}(d_1) = \widehat{A}_1/d_1^2 = A_1/d_1^2 = CI(d_1).$$

■

Definition 4.3 *The* incremental cut index *function* $CI_i(d)$ *of a tool with diameter* d_i *is the cut index obtained by applying the tool* d_i *immediately after a tool with diameter* d*. The incremental cut index can be computed from the cumulative cut area histogram* $\widehat{A}(d)$ *by:*

$$CI_i(d) = \begin{cases} (\widehat{A}(d_i) - \widehat{A}(d))/d_i^2 & \text{if } d > d_i \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

Corollary 4.4 *The incremental cut index function* $CI_i(d)$ *is monotonic increasing with increasing* d*, and, for a fixed* d*,* $CI_i(d)$ *is monotonic decreasing with increasing* d_i*, except when* $CI_i(d) = 0$*.*

Proof: The first part is true since the cumulative cut area $\widehat{A}(d)$ is a monotonic nonincreasing function with increasing d. From Equation 4.6, The incremental cut index $CI_i(d)$ is monotonic decreasing with increasing d, when $d > d_i \geq d_{min}$. Otherwise, $CI_i(d) = 0$.

Let d_1 and d_2 be two tool diameters with incremental cut index functions $CI_1(d)$ and $CI_2(d)$, respectively. Assuming $d > d_1 > d_2$, we have:

$$\begin{aligned} CI_1(d) - CI_2(d) &= (\widehat{A}(d_1) - \widehat{A}(d))/d_1^2 - (\widehat{A}(d_2) - \widehat{A}(d))/d_2^2 \\ &< (\widehat{A}(d_1) - \widehat{A}(d))/d_1^2 - (\widehat{A}(d_2) - \widehat{A}(d))/d_1^2 \\ &= (\widehat{A}_1 - \widehat{A}_2)/d_1^2 \\ &\leq 0 \end{aligned}$$

Therefore, $CI_i(d)$ is monotonic decreasing with increasing d_i. ■

Lemma 4.5 *For a tool with diameter d_i, the cumulative cut index $\widehat{CI}(d_i)$ is an upper bound of the incremental cut index function $CI_i(d)$, i.e., $\forall d$, $CI_i(d) \leq \widehat{CI}(d_i)$.*

Proof: This is true since the cumulative cut area is nonnegative, i.e., $\widehat{A}(d) \geq 0$. When $d > d_i$, from Equations 4.5 and 4.6, we have

$$\begin{aligned} \widehat{CI}(d_i) - CI_i(d) &= \widehat{A}(d_i)/d_i^2 - (\widehat{A}(d_i) - \widehat{A}(d))/d_i^2 \\ &= \widehat{A}(d_i)/d_i^2 - \widehat{A}(d_i)/d_i^2 + \widehat{A}(d)/d_i^2 \\ &= \widehat{A}(d)/d_i^2 \\ &\geq 0. \end{aligned}$$

Otherwise, $CI_i(d) = 0 \leq \widehat{CI}(d_i)$. ■

To find the combination of tool diameters that has the lowest total cut index, the cumulative and incremental cut index functions are evaluated on all available tool diameters. The result is a triangle matrix of cut indices, which has all zero on the upper-right side. Figures 4.11(a) and (b) shows the cumulative and incremental cut index functions, respectively, for the pocket shown in Figure 4.9. Note that the cumulative cut index function in (a) has a lower bound at $\pi/4$ for all nonzero values. This is because any tool that can fit inside a pocket, or closed inner profiles, should have at least one tool cross-section cumulative cut area.

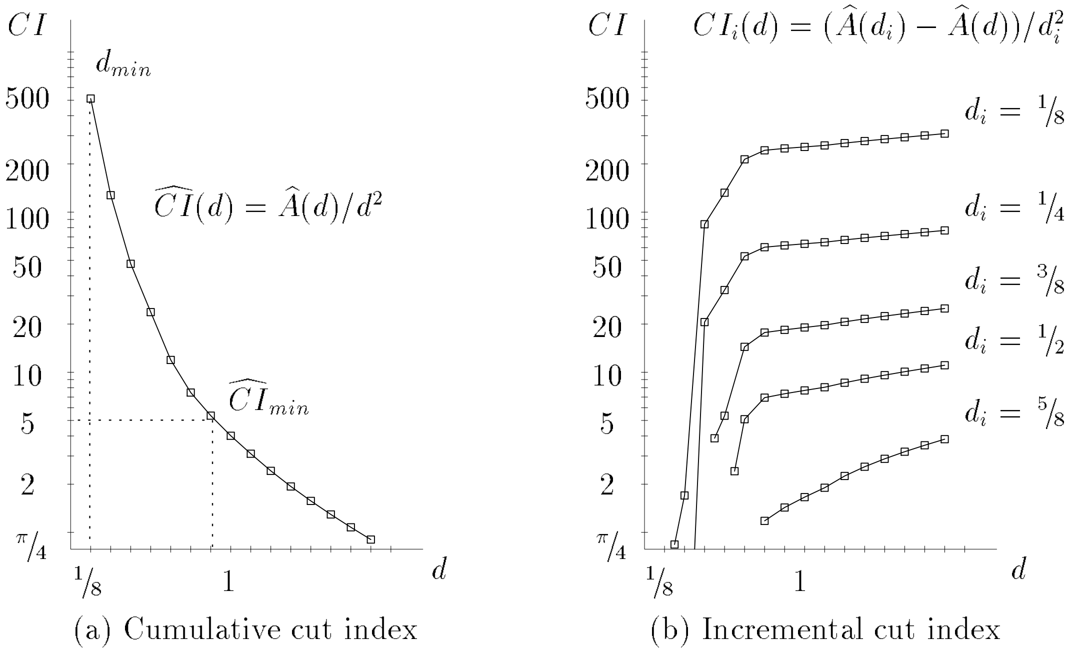


Figure 4.11. Cumulative and incremental cut index functions

The range of workable tool diameters for a feature is bounded by d_{min}, the minimum clearance of the skeleton, and $\widehat{CI}_{min}$, the minimum cut index a tool must contribute (see Figure 4.11(a)). The upper bound of tool diameter is determined by $\widehat{CI}_{min}$ and the cumulative cut index function. For inside pockets, $\widehat{CI}_{min}$ must be at least $\pi/4$, the global lower bound of $\widehat{CI}(d)$. Table 4.7 lists the evaluated cut index matrix of the example shown in Figure 4.9, for tools from $1/8$ inch up to $7/8$ inch diameters with an increment of $1/16$ inch. The diagonal is the cumulative cut indices $\widehat{CI}(d)$, and the lower-left half is the incremental cut indices $CI_i(d)$.

Definition 4.4 *Let $CM(i,j)$, where $i,j = 1,\ldots,n$, be the cut index matrix of a pocket with n different tool diameters, d_i, sorted in an increasing tool diameter order, i.e., $d_i < d_{i+1}$. The cut index $CM(i,j)$ of the i-th column and j-th row element can be computed by:*

$$CM(i,j) = \begin{cases} \widehat{CI}(d_i) & \textit{if } i = j \\ CI_i(d_j) & \textit{if } i < j \\ 0 & \textit{otherwise.} \end{cases} \tag{4.7}$$

Table 4.7. Cut index matrix for different tool diameters

	d_i												
d_j	$^1/_8$	$^3/_{16}$	$^1/_4$	$^5/_{16}$	$^3/_8$	$^7/_{16}$	$^1/_2$	$^9/_{16}$	$^5/_8$	$^{11}/_{16}$	$^3/_4$	$^{13}/_{16}$	$^7/_8$
$^1/_8$	512												
$^3/_{16}$	0.8	227											
$^1/_4$	1.7	0.4	128										
$^5/_{16}$	3.2	1.0	0.4	81									
$^3/_8$	83	37	20	13	48								
$^7/_{16}$	116	51	29	18	3.9	32							
$^1/_2$	132	58	33	21	5.4	1.1	24						
$^9/_{16}$	171	76	42	27	10	4.2	2.4	17					
$^5/_8$	214	95	53	34	15	7.7	5.1	2.1	12				
$^{11}/_{16}$	239	106	59	38	17	10	6.7	3.4	1.0	9.0			
$^3/_4$	243	108	60	38	18	10	6.9	3.6	1.2	0.1	7.5		
$^{13}/_{16}$	247	109	61	39	18	11	7.2	3.7	1.3	0.2	0.1	6.3	
$^7/_8$	250	111	62	39	18	11	7.3	3.9	1	0.3	0.2	0.1	5.4

Then, machining the pocket using a sequence of m tools, among the n tool diameters, with indices $t_i \in [1, n]$, where $i = 1, \ldots, m$, will have a total cut index

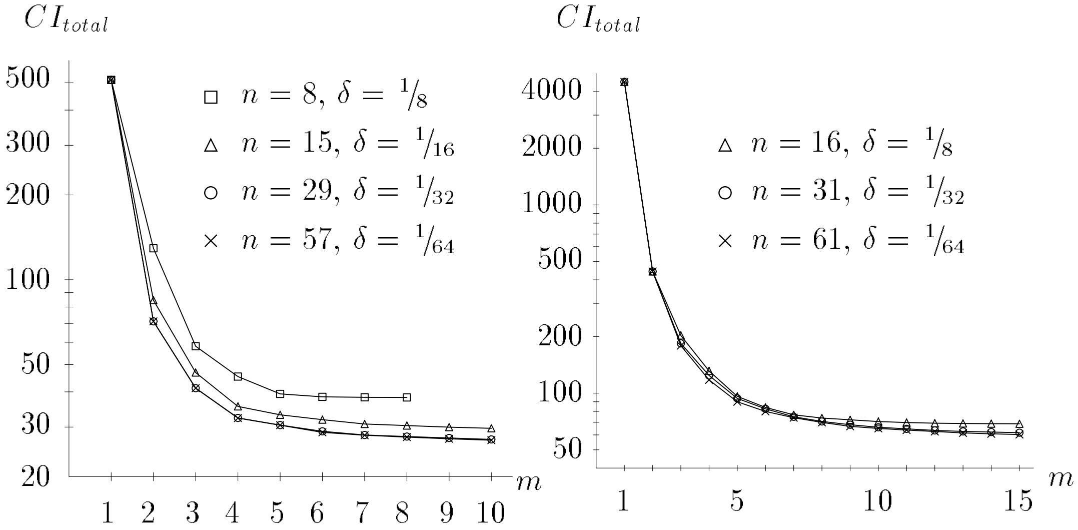
$$CI_{total} = CM(t_1, t_1) + \sum_{i=2}^{m} CM(t_i, t_{i-1}), \tag{4.8}$$

assuming t_i is sorted in an decreasing order, i.e., $t_i < t_{i-1}$.

Definition 4.5 *For a pocket with cut index matrix CM of n different tool diameters, the multiple tool selection problem is to minimize CI_{total} by selecting m tools with indices $\{t_i\}$ from n tool diameters, where $i = 1, \ldots, m$. The function $\mathcal{M}(i, t_i)$ to find the minimum subtotal of cut index for i tools with tool t_i as the last tool can be defined as:*

$$\mathcal{M}(i, t_i) = \begin{cases} \min_{j=t_i+1}^{n}(CM(t_i, j) + \mathcal{M}(i-1, j)) & \textit{if } i > 1 \\ CM(t_1, t_1) & \textit{otherwise.} \end{cases} \tag{4.9}$$

Note that the smallest tool with diameter d_{min} is required to cut the pocket cleanly. Since the tools are sorted in decreasing diameter order, the last tool index $t_m = 1$. The minimum total cut index can be computed by evaluating $\mathcal{M}(m, 1)$. Using dynamic programming techniques [3], we have developed an efficient algorithm to evaluate the recursive function. Figures 4.12(a) and (b) show results of



(a) Pocket from Figure 4.7 (b) Pocket from Figure 4.6

Figure 4.12. Minimum total cut index for different number of tools

the minimum cut index for pockets from Figures 4.7 and 4.6, respectively. Tool diameters up to 1 inch are considered, with different increment and total number n. In all cases, the minimum total cut index decreases with increasing number of tools m, and converges to a lower bound when $m = n$.

Table 4.8 lists the minimum total cut index and its corresponding tool diameters at various m for the cut index matrix shown in Table 4.7. The numbers follow the tool diameters are the corresponding cut matrix elements for each sequence of tools. Table 4.9 lists the tool combinations of minimum total cut index for the complex pocket shown in Figure 4.6. Using the cut index analysis, even two tools can give better machining time than the four tools listed in Table 4.3 with the simple constant ratio heuristic.

Table 4.8. Minimum total cut index and the corresponding tools

m	CI_{total}	Tool 1		Tool 2		Tool 3		Tool 4		Tool 5	
1	512	1/8	512								
2	84.6	5/16	81.4	1/8	3.2						
3	46.8	9/16	16.9	5/16	26.8	1/8	3.2				
4	36.5	7/8	5.4	1/2	7.3	5/16	20.6	1/8	3.2		
5	34.1	7/8	5.4	1/2	7.3	3/8	5.4	5/16	12.9	1/8	3.2

Table 4.9. Multiple tool selection for a complex pocket ($n = 61$)

Machining time (minutes) and CI								
d	$m = 1$		$m = 2$		$m = 3$		$m = 4$	
$^1/_{16}$	235.62	4504	8.79	274	1.96	6.5	1.96	6.5
$^9/_{64}$	—	—	—	—	5.52	83	5.52	68
$^5/_{16}$	—	—	2.96	169	—	—	—	—
$^{25}/_{64}$	—	—	—	—	—	—	2.90	31
$^{27}/_{64}$	—	—	—	—	2.80	90	—	—
$^{31}/_{32}$	—	—	—	—	—	—	1.09	12
Total	235.62	4504	11.75	443	10.28	179	11.47	118

4.7 Global Cut Index Analysis

Very often, a part contains many features. For example, Figure 4.13 shows a link part for an educational robot arm assembly. It has two setups, for machining from both sides. Each of the setups consists of many $2\,^1\!/_2$ D milling features, such as profiles, pockets, and step bores. Note that the total depth of cut within features is different in the different features.

Using multiple tool selection techniques presented in previous sections, a set of tools can be selected for each individual feature to minimize the per-feature machining time and cost. However, the overall machining time and number of tools may not be minimal. It is necessary also to consolidate different tool diameters between different features to select a common set of tools for the whole part if a global minimum is to be achieved.

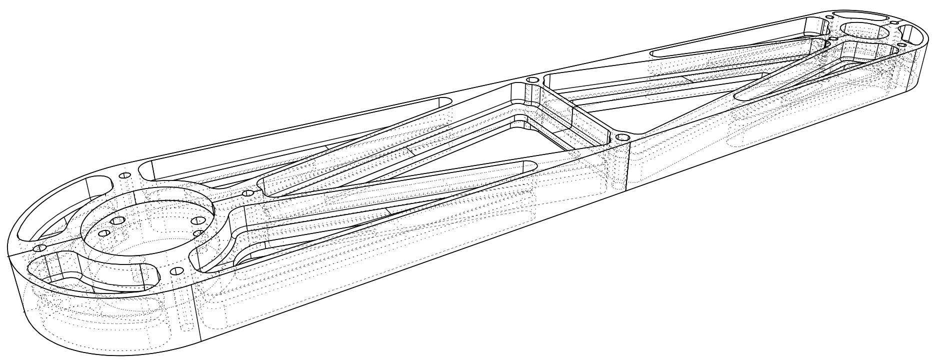


Figure 4.13. A robot arm link part with many $2\,^1\!/_2$ D features

A global tool-driven decomposition and tool path generation strategy would be to use a common set of tools and apply them in a decreasing diameter order on every feature. If any feature is too small to fit or to produce a significant tool path, i.e., $\widehat{CI}(d_i) < \widehat{CI}_{min}$, that tool would not cut the feature. It would be left uncut for the next tool. Any feature that is so big that is has been cut already cleanly by a larger tool is also skipped. To select the best tool combination for the whole part, we introduce a global cut index analysis.

Definition 4.6 *The* global cumulative cut area $\widehat{\mathcal{A}}(d)$ *and* cut index $\widehat{\mathcal{CI}}(d)$ *functions of a part consisting of many features are the cumulative cut area and cut index of all features, respectively. Then,*

$$\widehat{\mathcal{CI}}(d) = \widehat{\mathcal{A}}(d)/d^2. \tag{4.10}$$

Under our feature-based process planning framework, describe in Chapter 2, the process plan and its features are organized in a hierarchical structure. So the cumulative cut area of a feature is independent of that of other features, and is additive between features of the same depth. For features of different cut depth, a weighted sum by the relative cut depth of each feature is formed to compute the global cumulative cut area and cut index of a part.

Corollary 4.6 *Let w_j be the relative cut depth of feature j. The global cumulative cut area $\widehat{\mathcal{A}}(d)$ and cut index $\widehat{\mathcal{CI}}(d)$ functions of a part containing multiple features is defined as*

$$\widehat{\mathcal{A}}(d) = \sum_j w_j \widehat{A}_j(d) \tag{4.11}$$

$$\widehat{\mathcal{CI}}(d) = \sum_j w_j \widehat{CI}_j(d), \tag{4.12}$$

where $\widehat{A}_j(d)$ and $\widehat{CI}_j(d)$ are the individual cumulative cut area and cut index functions of feature j, respectively.

Proof: From Equations 4.11 and 4.5,

$$\widehat{\mathcal{CI}}(d) = \widehat{\mathcal{A}}(d)/d^2$$

$$
\begin{aligned}
&= (\sum_j w_j \widehat{A}_j(d))/d^2 \\
&= \sum_j w_j (\widehat{A}_j(d)/d^2) \\
&= \sum_j w_j \widehat{CI}_j(d).
\end{aligned}
$$

■

To perform the global cut index analysis, the global incremental cut index functions $\mathcal{CI}_i(d)$ for tool d_i is needed. It can be defined as:

Definition 4.7 *The* global incremental cut index *function $\mathcal{CI}_i(d)$ of a tool with diameter d_i is the global cut index by applying that tool immediately after a tool with diameter d on the whole part, i.e.,*

$$
\mathcal{CI}_i(d) = \begin{cases} (\widehat{\mathcal{A}}(d_i) - \widehat{\mathcal{A}}(d))/d_i^2 & \textit{if } d > d_i \\ 0 & \textit{otherwise.} \end{cases} \tag{4.13}
$$

Similarly, the global incremental cut index functions $\mathcal{CI}_i(d)$ can be defined as a weighted sum of the incremental cut index functions $CI_{ij}(d)$ of each feature.

Corollary 4.7 *Let $CI_{ij}(d)$ be the incremental cut index function of tool d_i for feature j. Then, the global incremental cut index function $\mathcal{CI}_i(d)$ is defined as*

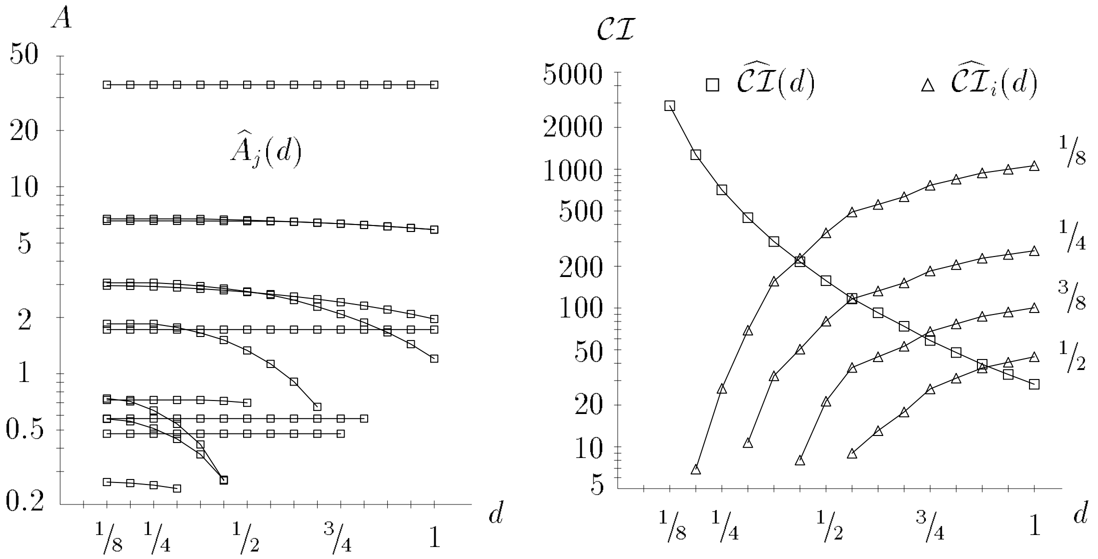
$$
\mathcal{CI}_i(d) = \sum_j w_j CI_{ij}(d). \tag{4.14}
$$

Proof: From Equations 4.13, 4.11, and 4.6,

$$
\begin{aligned}
\mathcal{CI}_i(d) &= (\widehat{\mathcal{A}}(d_i) - \widehat{\mathcal{A}}(d))/d_i^2 \\
&= (\sum_j w_j \widehat{A}_j(d_i) - \sum_j w_j \widehat{A}_j(d))/d_i^2 \\
&= \sum_j w_j (\widehat{A}_j(d_i) - \widehat{A}_j(d))/d_i^2 \\
&= \sum_j w_j CI_{ij}(d).
\end{aligned}
$$

■

Figure 4.14 shows the global cut index analysis on the link part shown in Figure 4.13. Table 4.10 lists the minimum global cut index for different total number of tools and its corresponding machining time.



(a) Cumulative cut area (b) Global cut indices

Figure 4.14. Cumulative cut area and global cut indices

Table 4.10. Multiple tool selection using global cut index analysis

Machining time (minutes) and $\mathcal{CI}$								
d	$m=1$		$m=2$		$m=3$		$m=4$	
1/8	334.24	2867	32.8	228	9.27	69	3.46	26
1/4	—	—	—	—	—	—	8.65	51
5/16	—	—	—	—	9.86	90	—	—
7/16	—	—	16.08	215	—	—	4.80	63
11/16	—	—	—	—	7.97	74	—	—
15/16	—	—	—	—	—	—	5.98	33
Total	334.24	2867	48.88	443	27.10	233	21.54	173

4.8 Extensions to 3D Features

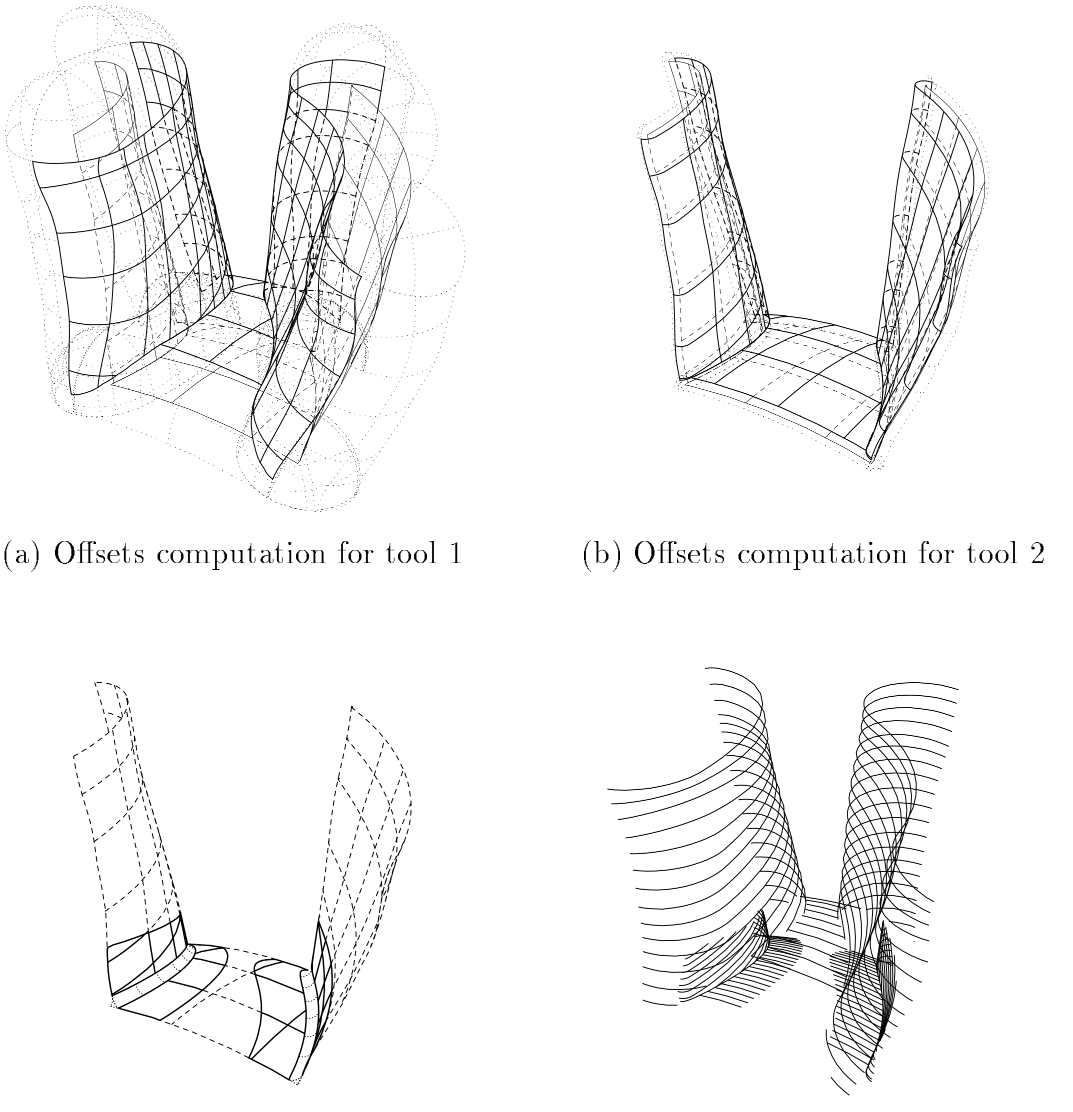
The tool-driven decomposition techniques can be extended to features defined with 3D freeform surfaces. For surface contouring operations using ball-end tools, constant distance offset surfaces can be computed to efficiently obtain the tool center locations. Assuming the offset surfaces are self-intersection free, i.e., either there is no self-intersection or the self-intersections have been trimmed away, the following algorithm presents a tool-driven decomposition for 3D features.

```
ALGORITHM SurfDecomp( Surfaces, Tools )
BEGIN
    Sort Tools in a decreasing tool diameter order.
    FOR each tool in the sorted Tools DO
    BEGIN
        Compute offsets of Surfaces at a distance
                equal to the tool radius.
        Compute the cumulative cut area of the tool.
        Compute the cumulative and incremental cut
                indices of the tool.
    END
    Perform cut index analysis and select a subset of tools
        that gives the minimum total cut index.
    Identify the uncut regions of each operation.
    Return the trimmed subregions and the selected tools.
END
```

The offset surfaces are computed using symbolic NURBS composition. To identify the regions uncut by tool d_{i-1} and to generate a gouge-free tool path for tool d_i, trimming loops of the two consecutive trimmed offset surfaces are superimposed on each other. A boolean operation can be applied to find the remaining cut regions for tool d_i.

Figure 4.15 shows the tool-driven decomposition for the finish cuts of the 3D feature from the compress disk example presented in Chapter 2. Two tools, $^3/_4$ and $^1/_8$ inch diameters, are selected for the finish operations. Figures (a) and (b) show the computation of the constant distance offsets for the tools. The dotted lines represent the offset surfaces, which include offsets from the open edges and corners of the feature surfaces. Then, intersections and self-intersections among the offset surfaces are computed and trimmed away.

Regions of the decomposed operations are shown in Figure 4.15(c). Regions shown in dashed lines are the first finish operation using a $^3/_4$ inch tool. Regions shown in thick solid lines are the second finish operation for the remaining uncut area using a $^1/_8$ tool. The fillet surfaces are shown in dotted lines, which are offsets of the C^1 discontinuous intersection curves of the trimmed offset surfaces shown in Figures 4.15(b). Tool center locations for the offset isoline tool paths are shown in Figure 4.15(d).



(a) Offsets computation for tool 1 (b) Offsets computation for tool 2

(c) Tool-driven decomposition (d) Offset isoline tool path generation

Figure 4.15. Tool-driven decomposition for 3D features

CHAPTER 5

OFFSETS

The computation of curve and surface offsets is an essential task in many engineering applications, such as NC tool path generation, robot path planning, mold design and solid rocket fuel analysis [86, 107, 138]. The offset operation is also important in geometric modeling. It is used directly or indirectly in other geometric operations, such as fillet blending, surface shelling and sweeping [18, 31, 76, 109, 120].

In this chapter, we first define the constant distance offset operation. Then, we review and compare different offset techniques, and present a new hybrid method to compute the offsets of NURBS curves and surfaces. We introduce algorithms and techniques to improve the offset approximation accuracy and efficiency, to handle offsets of degenerate surfaces, and to offset C^1 discontinuous edges and corners. Finally, examples of using these curve and surface offset techniques in solid modeling are presented.

5.1 Constant Distance Offsets

In previous chapters, offset curves and surfaces are used in NC tool path generation for features having freeform curves and surfaces. Figure 5.1 illustrates the basic use of offsets in tool path generation. To machine the part surface, shown as the solid curve, the center of a ball-end tool moves along a curve offset from the part by the tool radius, shown as the dashed curve. The edge of the tool forms an envelope that is the inverse offset of the tool path. This gives us the part surface. In essence, tool path generation consists of finding the cutter center locations that are at a constant distance, usually equal to the tool radius, to the part surface.

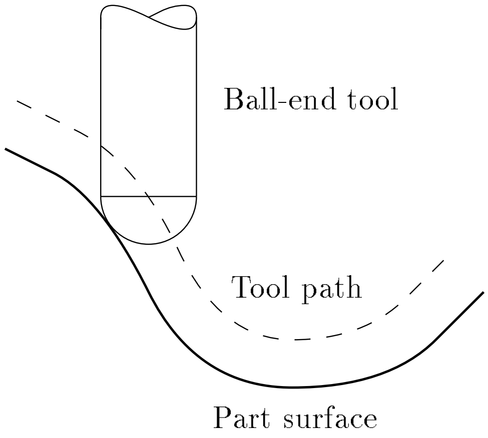


Figure 5.1. Offset of part surface and tool path

Definition 5.1 *Let* $\mathbf{S}$ *be a continuous surface. A* constant distance offset *of* $\mathbf{S}$ *at distance* d *is a surface of points whose distance to* $\mathbf{S}$ *equals* d*,*

$$\mathbf{S}_d = \{\mathbf{p} | D(\mathbf{p}, \mathbf{S}) = d\}, \tag{5.1}$$

where $D(\mathbf{p}, \mathbf{S})$ *is the shortest distance from* $\mathbf{p}$ *to* $\mathbf{S}$*.*

For an orientable surface, there are two constant distance offset surfaces at d, one on each side (see Figure 5.2). To distinguish between them, by a convention, the offset surface on the opposite side of the surface normal is denoted by a negative offset distance. Similarly, the constant distance offset operation can also be applied to curves. Note that the constant distance offset operation is a many-to-many

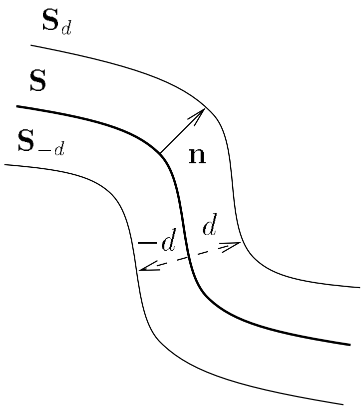


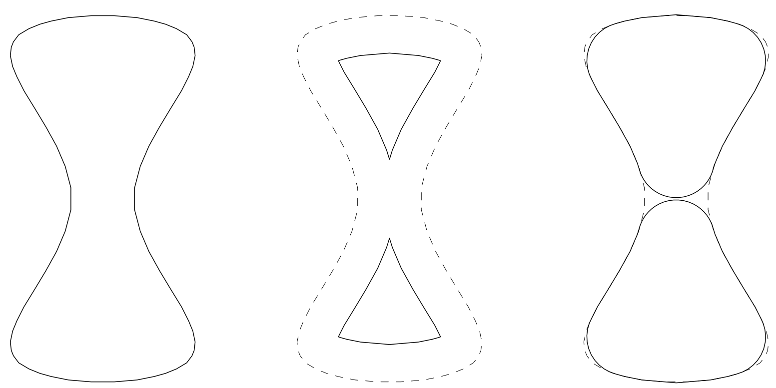
Figure 5.2. Constant distance offset surfaces

mapping and is not invertible. That is, offset in the negative direction of an offset curve or surface does not necessarily result in the original shape. For example, in Figure 5.3, the offset curve and its offset have two profiles, unlike the original curve which has only one. An interesting observation is that an additional offset operation on the offset of an offset curve will yield the same offset curve,

$$\mathcal{O}(\mathcal{O}(\mathbf{P}, -d), d) = \mathbf{P}, \text{ if } \mathbf{P} = \mathcal{O}(\mathbf{C}, d). \tag{5.2}$$

With the above definition, how can one compute the constant distance offset of curves or surfaces? In reference [120], a spatial subdivision method was proposed to identify rough machining regions. The suggested method was to use a spatial indexing scheme, such as an octree, and successively subdivide any cell that might contain the offset curve (see Figure 5.4). However, there are at least two problems with this method. First, obtaining a reasonable accuracy and smoothness requires many subdivisions that could be costly in both space and time. Second, an efficient offset curve containment test is hard to implement. A "brute force" implementation requires the computation of the closest distance from the center of a cell to the curve, which is not trivial for freeform curves.

In Section 3.4, constant distance offsets of curves from 2D line segments and circular arcs are computed with the help of the curve skeleton. The skeleton structure provides an efficient representation to describe the closest distance from



(a) Original curve (b) Offset curve (c) Offset of the offset

Figure 5.3. Constant distance offset operation is not invertible

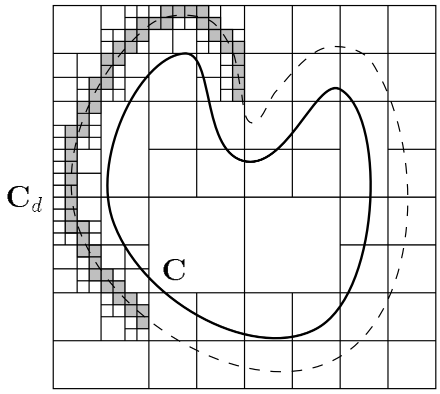


Figure 5.4. Constant distance offset by spatial subdivision

any point to the boundary curve. However, the computation of 3D surface skeleton is not easy and is usually approximated by piecewise linear facets [140]. Another way to compute the constant distance offset surface is to apply the mathematical offset operation, which can be computed using the differential geometry of the surface.

The offset map of a regular surface is an operation that associates every point on the surface with a point in the normal direction (see Figure 5.5). If the offset distance is the same across the surface, we can define the offset surface as

$$\widehat{\mathbf{S}}_d = \mathbf{S} + d\,\mathbf{n}, \tag{5.3}$$

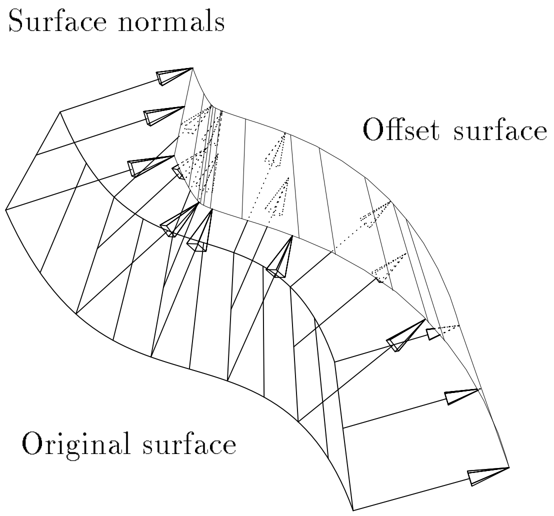


Figure 5.5. Offset surface in the normal direction

where $\mathbf{n}$ is the surface unit normal vector and is globally defined on a regular surface [94]. For a regular surface defined in a close domain, the surface has a closest point to any point on its constant distance offset, and the direction between them is perpendicular to the original surface [136]. In other words, the constant distance offset of a regular surface is a subset of its offset surface at the same distance, i.e., $\mathbf{S}_d \subseteq \widehat{\mathbf{S}}_d$. To compute the constant distance offset surface, we first compute the mathematical offset surface, then trim away any self-intersection from the offset surface.

5.2 Offset Techniques

For certain special forms, such as line, circle, plane and sphere, the offset can be exactly represented with the same representation. However, for freeform NURBS curves and surfaces, the exact offset is usually not a NURBS and must be approximated. Below are some of the requirements for NURBS offset computation:

1. It should be easy to understand and be extensible to general 3D surfaces.
2. It should create an exact offset when possible.
3. It should include global tolerance control as a parameter.
4. It should maintain smoothness.
5. It should handle tangent discontinuity.
6. It should handle degeneracy in the parametrization.

5.2.1 Review of Offset Techniques

Numerous techniques found in the literature for computing offsets of NURBS curves and surfaces produce an approximation to the exact offsets [31, 34, 42, 63, 64, 75, 108, 138]. Instead of giving an overview of what each of them did, which is available from the literature [45, 109], a comparison of the key differences between these techniques is summarized here.

- *Direct offset v.s. sampled offset.* Most NURBS offset techniques can be categorized into two groups. The direct offset techniques offset the NURBS control points directly. Cobb [31] approximated the offsets of NURBS curves and surfaces by moving each control point in the normal direction at the associated node by the offset distance. Coquillart [34] modified Cobb's method to produce exact offset curves for circular arcs by using the curvature and normal at the control node of each control point, i.e., the closest point on the curve. She also used the "quasinormal" of the curve to handle the inflection points for 3D offset curves. However, unlike Cobb's method, Coquillart's method applies only to curves. Patrikalakis and Prakash [104] approximated the offsets of NURBS surfaces by offsetting the control points in the "pseudo-normal" directions. The pseudo-normal of a control point is defined as the average of the plane normals of the (eight) triangles formed by the control point and its neighbors. Tiller and Hanson [138] approximated the offsets of planar curves by offsetting segments of the control polygon.

  The sampled offset techniques approximate the offset using exact offsets of the sampled curve or surface points. Tiller and Hanson [138] also tried different fitting methods to offset planar curves, which include least squares fitting to point data, interpolation to point data, and interpolation to point and first derivative data. Pham [108] used uniform B-splines with double control points at the ends to interpolate the offset of 2D curves evaluated at knot points. Klass [75] approximated the offsets of planar cubic spline curves by interpolating the points, tangent directions, and curvatures of the exact offset at the two end points. Farouki [46] took a similar approach to approximating the offsets of bicubic surfaces by interpolating the points, the tangent vectors, and the twist vectors of the exact offset at the four corner points. Klass's and Farouki's methods apply to only cubic and bicubic curves and surfaces. Hoschek and Wissel [63, 64] used least square approximation method with G^k-continuous cubic Bézier splines of degree $2k-1$, where k is from 1 to 4. Then, they used parameter transformations and nonlinear optimization techniques to reduce the offset errors. The least square methods are difficult to extend to general 3D offset surfaces. Lee et al. [83] first approximated the normal vectors of a 2D curve by reparametrization and subdivision of the unit circle as piecewise quadratic Bézier curve segments. Then, the offset curves are computed using symbolic composition which results in higher-order NURBS offset curves. Their method is for 2D offset curves only.

- *Exact offset when possible.* The motivating premise is that the approximation scheme should produce the exact offset wherever it is possible. Circular arcs and linear segments have exact NURBS offsets and are used frequently in engineering design. Coquillart's control points offset method was originally designed to reproduce offset circles as quadratic and cubic NURBS. The planar control polygon offset method by Tiller and Hanson generates exact offsets for lines and circles as well. The other approximation methods do not reproduce exact offsets of circles [45].

- *Approximation improvement and tolerance control.* To improve the approximation accuracy, all the techniques used either subdivision to divide the curve into two halves [75, 63, 138] or add more knots to refine the original curve [31, 34, 42, 104, 108]. Hoschek and Wissel [64] used parametrization improvement to reduce the error as well as a split and merge method for Bézier curves to produce an optimal approximation. Most offset methods rely on the user to refine or subdivide the original curve or surface. Tiller and Hanson [138] automatically subdivided the curve by measuring the offset deviation at the middle point of each span. Patrikalakis and Prakash [104] measured offset errors by examining the length of the vector computed by intersecting a line in the direction of the surface normal at a given point with the approximate offset surface. The approximation is examined at knot points and at uniform samples within each parametric region. The surface is refined adaptively until all samples are within the tolerance. Elber [42] used spline symbolic and numerical computation to bound the approximation error globally. A new knot is added in the middle of the region where the error exceeds the tolerance. He also introduced a control points perturbation method [42] to move control points, originally offset using Cobb's method, to improve the approximation accuracy.

- *Smoothness maintenance.* The B-spline refinement approximation improvement methods maintain G^{k-1} continuity to a G^k continuous curve or surface. The subdivision methods can introduce tangent discontinuity at the subdivision boundary. The 2D polygon offset method by Tiller and Hanson maintains G^1 continuity for offsets of smooth curves. Our 3D control polygon and control mesh offset method also maintains G^1 continuity for offsetting smooth curves and surfaces, including the end-to-end tangent continuity for closed curves and surfaces. Cobb's and Coquillart's direct control point offset methods can introduce tangent discontinuity if the original curve or surface has $order - 1$ multiplicity in its knot vectors. Using Hoschek's G^k continuous Bézier interpolation method, it is hard to maintain smoothness for surfaces.

- *Tangent discontinuity and normal vector degeneracy.* Most offset methods assume the input curve or surface is smooth and nondegenerate. Tiller and Hanson [138] filled the gaps between offsets of tangent discontinuous 2D curves by either extending both offsets linearly in their tangent directions or by adding circular arcs tangent to the extension lines. We fill the gaps between offsets of tangent discontinuous 3D surfaces by tangent continuous constant radius fillet surfaces and spherical patches, which are offsets of the C^1 discontinuous edges and corners.

5.2.2 A Hybrid Approach

In this section, we present a hybrid approach to approximating 3D offset curves and surfaces to within a prescribed tolerance. This offset algorithm has many of the features found across the reviewed offset techniques. Based on the requirements described in the previous section, we modify and extend the 2D curve offset techniques to offsetting 3D NURBS surfaces.

```
ALGORITHM SurfOffset( Surface, Distance, Tolerance )
BEGIN
    Subdivide Surface at every C1 discontinuity.
    FOR each subdivided C1_Surface DO
    BEGIN
        Offset C1_Surface at Distance.
        WHILE OffsetError( C1_Surface, C1_Offset ) > Tolerance DO
        BEGIN
            Refine C1_Surface by adding new knots at
                    regions where error > Tolerance.
            Offset the refined C1_Surface by the same method.
        END
    END
    Identify gaps between the C1_Offset surfaces and construct
            constant radius blend surfaces to fill the gaps.
    Identify gaps between the constant radius blend surfaces
            and construct spherical patches to fill the gaps.
END
```

Evaluation of the offset function requires a unique normal at each point. To offset curves and surfaces with C^1 discontinuity, the curve or surface needs to be subdivided at every C^1 discontinuity. Then, circular arcs or fillets are added to fill gaps between the C^1 offsets (see Figure 5.6).

Two different 3D curve and surface offset methods have been developed in this research. The general NURBS offset interpolation method approximates the offset curve and surface by interpolating exact offsets of the node points. The 3D control

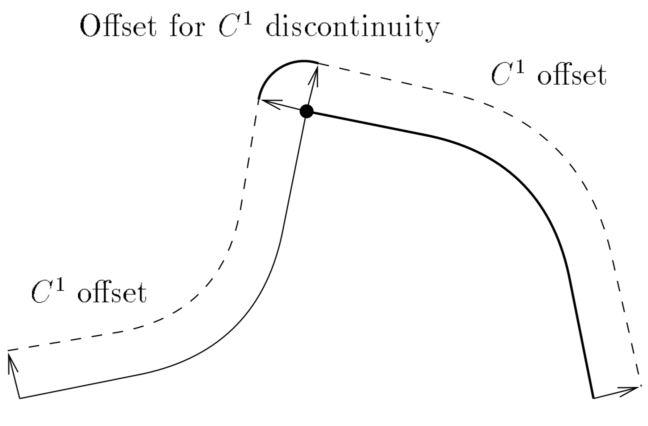


Figure 5.6. Offsets with C^1 discontinuity

polygon and control mesh offset method approximates the offset curve and surface by offsetting the control polygon and control mesh directly. Figure 5.7 shows the basic operations of the two offset methods.

We adapt Elber's method to bound the offset error globally by symbolic computation of the distance squared between the corresponding points on the original surface and the offset approximation [42]. We improve his adaptive refinement method to select a better location and direction for the refinement based on the magnitude of the error function. Note that the error function is defined as the maximum deviation from the offset distance, i.e.,

$$\mathcal{E}_d \geq |d - \|\tilde{\mathbf{S}}_d(u,v) - \mathbf{S}(u,v)\||, \forall u, v. \tag{5.4}$$

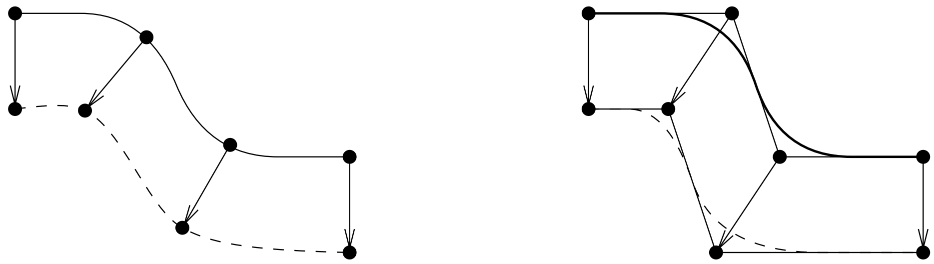
It does not include the deviation of the offset direction from the normal direction. However, error to the true offset cannot be computed directly using a NURBS representation, which is defined as:

$$\delta_d(u,v) = \|\tilde{\mathbf{S}}_d(u,v) - \widehat{\mathbf{S}}_d(u,v)\|. \tag{5.5}$$

5.3 Offset Surface Approximation

From Equation 5.3, the surface offset operation requires the evaluation of surface unit normal vectors. From differential geometry [94], the surface normal can be computed from the cross product of the surface partial derivatives,

$$\mathbf{n}(u,v) = \frac{\mathbf{S}_u(u,v) \times \mathbf{S}_v(u,v)}{\|\mathbf{S}_u(u,v) \times \mathbf{S}_v(u,v)\|}. \tag{5.6}$$



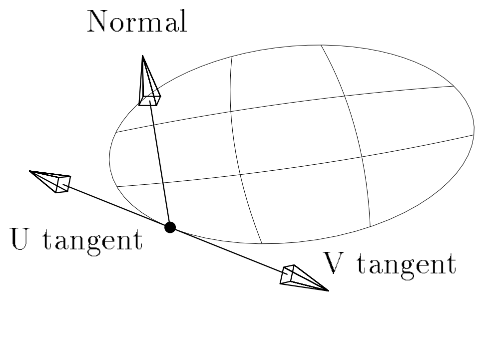
(a) Offset node points (b) Offset control polygon

Figure 5.7. Interpolation and direct NURBS offset methods

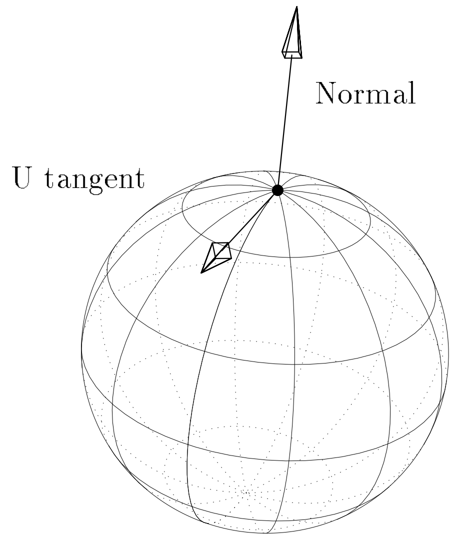
However, the cross product has zero magnitude when either one of the partials has zero magnitude or when the two partials align in the same or opposite directions (see Figure 5.8). Many surface offset methods found in the literature simply ignore this problem or assume the surface is regularly parametrized [31, 42, 46, 109].

Clearly, from the geometry, i.e., in an arc length parametrization, the tangent planes exist in both cases. For the disk example, the tangent plane is the plane that contains the planar surface. For the sphere example, the tangent plane at any point $\mathbf{p}$ on the sphere is the plane containing $\mathbf{p}$ and perpendicular to the direction from the center to $\mathbf{p}$. In fact, a zero magnitude cross product of partial derivatives occurs very often in many widely used surfaces, such as Coons patches with G^1 continuous boundaries, ruled-surfaces between two end-connected curves, surfaces of revolution where end points of the cross-section curve lie on the axis line, and general sweep/variable radii fillet surfaces where the cross-section vanishes (at the end points).

A formula using one-sided limits for computing unit normal vectors along a degenerate edge is given by Berger and Kallay [17]. It applies l'Hôpital's rule [136] when a degenerate edge is encountered. The direction of degeneracy is first iden-



Tangent vectors in opposite directions



Degenerate V tangent vector

Figure 5.8. Degenerate cross product of surface partial derivatives

tified, then higher order partial derivatives in the other direction are taken until a nonzero value is found. In this research, we develop a geometric method that can be applied to both degenerate and regular points of NURBS surfaces. Instead of computing the cross-product of the partial derivatives, the surface tangent plane is computed geometrically from the surface control mesh.

5.3.1 Offset Interpolation and Normal Evaluation

Unlike Pham's 2D curve offset method for uniform splines, refinement of a NURBS curve or surface does not change its geometry. Our general NURBS offset interpolation method first computes the exact offsets of the node points, then interpolates them using the original knot vectors and weight function of rational spline. A robust surface normal evaluation algorithm is the first step towards a robust surface offset algorithm. Taking the cross product of partial derivatives does not always produce a valid unit normal vector. To evaluate the unit normal vector at $\mathbf{S}(u_0, v_0)$ of a C^1 continuous surface, we first refine the surface at (u_0, v_0) to $order - 1$ multiplicity in both directions, then compute the surface tangent plane from the refined control mesh.

From the formulation of knot insertion and point evaluation of B-splines [33, 37], a NURBS surface with $order - 1$ multiplicity in both parametric directions satisfies the following conditions (see Figure 5.9 for notation):

$$
\begin{aligned}
\mathbf{p}_{i,j} &= \mathbf{S}(u_0, v_0) \\
\mathbf{u}_1 &= \mathbf{p}_{i,j} - \mathbf{p}_{i-1,j} = \frac{\|\mathbf{p}_{i,j} - \mathbf{p}_{i-1,j}\|}{\|\mathbf{S}_u^-(u_0, v_0)\|}\, \mathbf{S}_u^-(u_0, v_0) \\
\mathbf{u}_2 &= \mathbf{p}_{i+1,j} - \mathbf{p}_{i,j} = \frac{\|\mathbf{p}_{i+1,j} - \mathbf{p}_{i,j}\|}{\|\mathbf{S}_u^+(u_0, v_0)\|}\, \mathbf{S}_u^+(u_0, v_0) \\
\mathbf{v}_1 &= \mathbf{p}_{i,j} - \mathbf{p}_{i,j-1} = \frac{\|\mathbf{p}_{i,j} - \mathbf{p}_{i,j-1}\|}{\|\mathbf{S}_v^-(u_0, v_0)\|}\, \mathbf{S}_v^-(u_0, v_0) \\
\mathbf{v}_2 &= \mathbf{p}_{i,j+1} - \mathbf{p}_{i,j} = \frac{\|\mathbf{p}_{i,j+1} - \mathbf{p}_{i,j}\|}{\|\mathbf{S}_u^+(u_0, v_0)\|}\, \mathbf{S}_v^+(u_0, v_0),
\end{aligned}
$$

where the $\mathbf{S}_u^\pm$ and $\mathbf{S}_v^\pm$ are the one-sided partial derivatives of the surface. In other words, the refined control mesh contains the evaluated surface point and the directions of the (one-sided) surface partial derivatives.

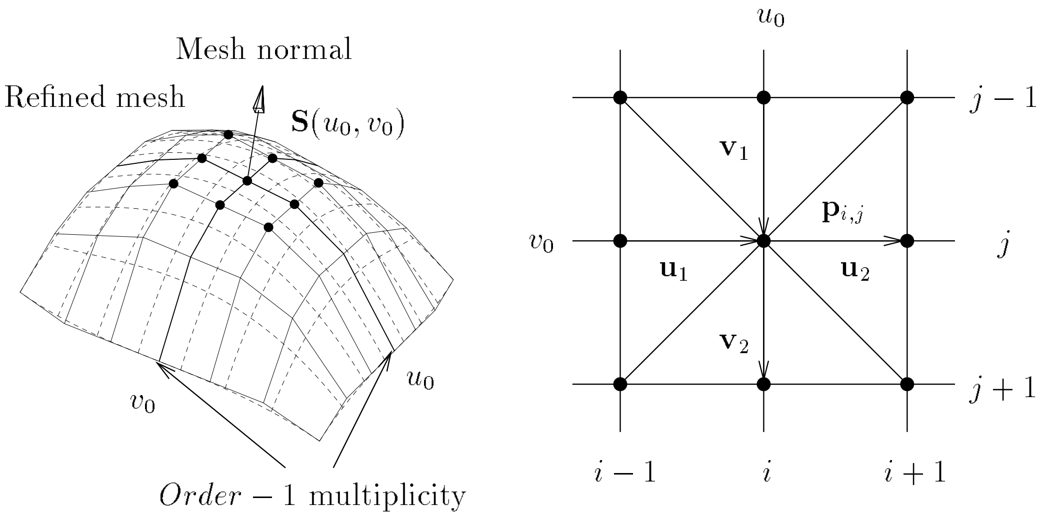


Figure 5.9. Surface normal from refined control mesh

At a regular point, the refined point $\mathbf{p}_{i,j}$ and its 4-neighbor points, $\mathbf{p}_{i\pm1,j}$ and $\mathbf{p}_{i,j\pm1}$, are coplanar. These points define a unique tangent plane and unit normal vector at $\mathbf{S}(u_0, v_0)$. If the cross-product of the partial derivatives is degenerate but the surface is tangent plane continuous at $\mathbf{p}_{i,j}$, i.e., the 4-neighbor points are coincident or collinear, the 8-neighbor mesh points $\mathbf{p}_{i\pm1,j\pm1}$, in addition to the 4-neighbor points, are used. A maximum of eight triangles are formed around $\mathbf{p}_{i,j}$. The surface normal vector is computed as the average of the plane normals of the nondegenerate triangles among them.

In case of ill-formed control meshes, ones that have multiple rows or columns of control points which coincide with each other, neighbor point propagation can be used to skip coincident control points. Figure 5.10 shows offset surfaces of two degenerately parametrized surfaces, a circular disk and a sphere, using the node point offset interpolation method and the refined mesh normal evaluation algorithm. Node points on the original surfaces are shown as circular dots along with their surface normal vectors. Exact offsets are found in both examples.

5.3.2 Control Mesh Offset

This research extends the 2D control polygon offset method by Tiller and Hanson [138] to 3D NURBS curves and surfaces. We compute the control points of

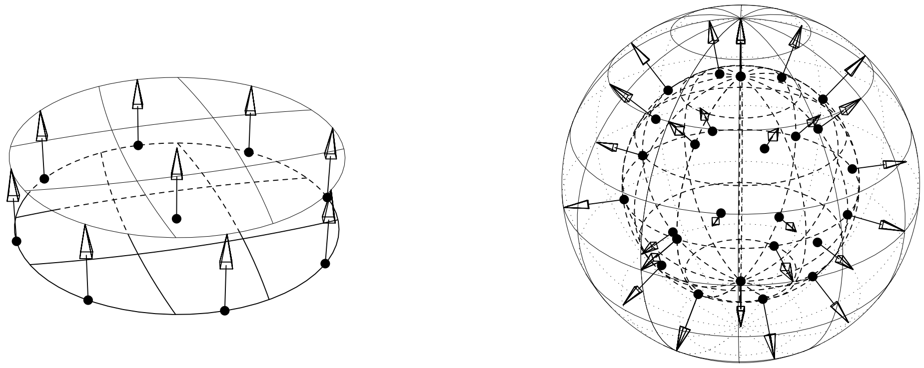


Figure 5.10. Offset interpolation of degenerate surfaces

the offset curve approximation by intersecting three planes at each point. The first plane contains the two polygon legs. Each of the last two planes is perpendicular to the first plane and parallel to its respective polygon leg at a distance equal to the offset value (see Figure 5.11). We adapt Coquillart's quasinormal to handle special cases such as degenerate and collinear control segments, and to adjust normals at inflection points of the control polygon.

In a similar approach, we offset planes of the control mesh triangles from the 4-neighbor points, and the diagonal 8-neighbor points if the 4-neighbor points are collinear. The mesh triangles could have up to four different normal directions.

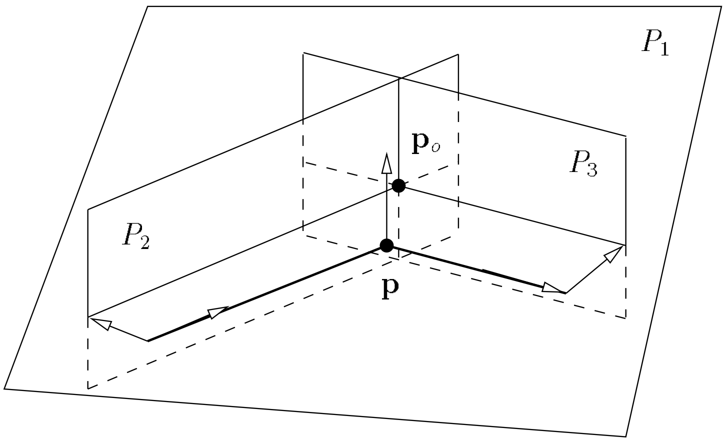


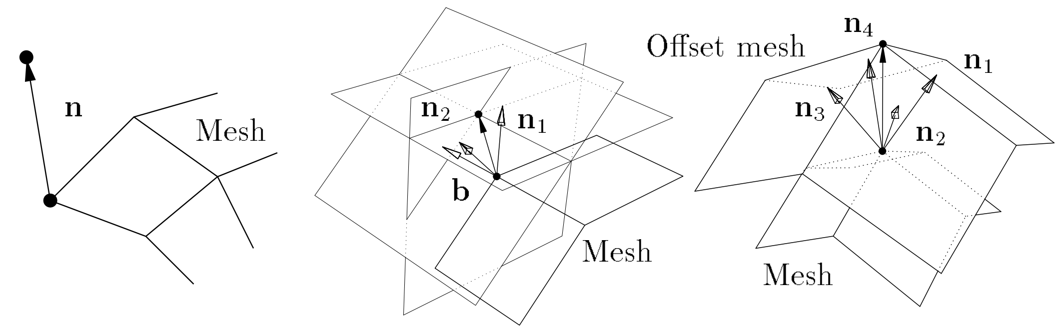
Figure 5.11. Control polygon offset for 3D curves

Depending on the number of independent mesh normal vectors, we compute the offset control points as following:

- One independent mesh normal vector: The control point of the offset surface is the offset of the control point of the original surface in the mesh normal direction (see Figure 5.12(a)). Examples of this case include mesh corners and flat regions.

- Two independent mesh normal vectors: This case is very similar to the 3D control polygon offset case shown in Figure 5.11 where the control point for the offset is the intersection of three planes. In Figure 5.12(b), each of the first two planes is parallel to its respective mesh triangle at a distance equal to the offset value. The third plane contains the control point and perpendicular to the first two planes. This case happens often on boundaries of the control mesh.

- Three or four independent mesh normal vectors: This is the most common case for interior mesh points. If only three independent mesh normals exist, the offset control point is the intersection of the three offset normal planes. When all four mesh normal vectors are independent, $C\binom{4}{3} = 4$ combinations of normal planes are selected to intersect with each other. The average of the four intersection points is used as the offset control point (see Figure 5.12(c)).



(a) One mesh normal (b) Two mesh normals (c) Four mesh normals

Figure 5.12. Control mesh offset method

For ill-formed control meshes, a valid mesh normal may not exist if only immediate neighbor points are used. Again, neighbor point propagation can be used to skip coincident control points. The mesh offset method also handles degenerate surfaces such as the disk and sphere examples correctly and produces exact offset surfaces for them (see Figure 5.13).

5.4 Adaptive Offset Refinement

To improve the offset accuracy, we adaptively refine the curve and surface until the maximum offset error is within the tolerance. In curve offset refinement, the number of added knots at each interval is proportional to the magnitude of the distance error and is distributed based on the error function. The purpose is to speed-up the convergence while reducing the total number of inserted points. In surface offset refinement, we add new knots at every parametric regions that have errors exceeding the prescribed tolerance. Besides the locations of the new knots, an additional degree of freedom is the parametric direction of the knot insertions.

A frequently used straightforward method is always to insert new knots in both directions, or to let one direction alternate between iterations. Figure 5.14(a) shows a matrix of parametric regions and locations where excessive errors are found, marked by X's. Note that the cells indicate the number of intervals of the knot

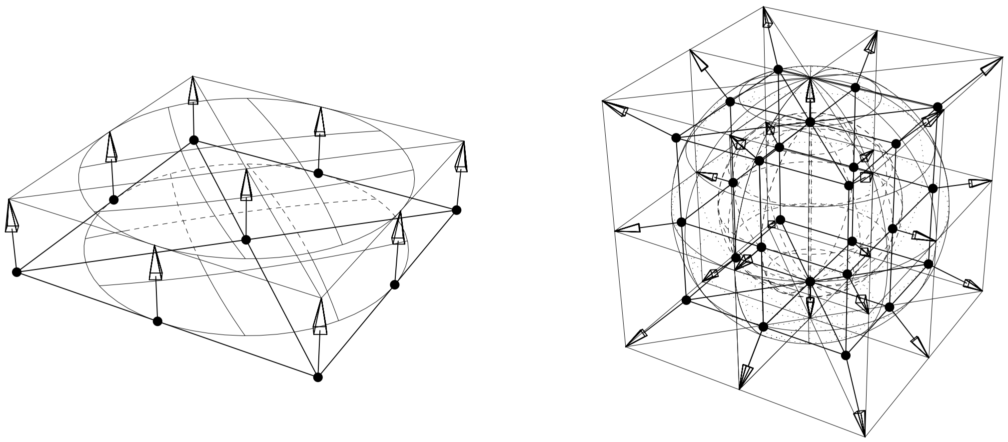


Figure 5.13. Control mesh offset of degenerate surfaces

vectors, and do not reflect the actual spacing or multiplicity of the knots. A cross line, horizontal or vertical, is drawn to indicate a new knot inserted in that direction. Adding knots in both directions at every marked cell almost doubles the number of knots in both parametric directions, and quadruples the number of control points in this example. We introduce a more efficient method, in which, a minimal number of new knots, or new control points, are inserted to cover every marked cell at each iteration. For example, Figure 5.14(b) shows only three new knots are required to cover all marked cells.

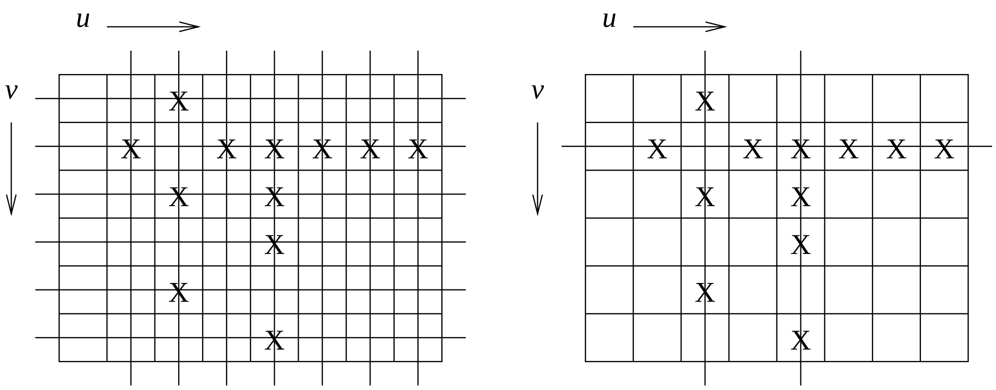
Definition 5.2 *An* integer linear problem *minimizes* $\sum_{j=1}^{n} c_j x_j$ *subject to the constraints:*

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i,\ i = 1, \ldots, m$$

where x_j *are nonnegative integers.*

The *minimum cover* problem is a class of integer linear problem where a_{ij} are nonnegative integers, $b_i = 1$, and $c_j \geq 0$. Such problems arise in many applications, such as crew scheduling, switching circuit design, and recycle systems simulation [111, 117, 118].

A marked cell in the knot matrix can be refined in row, column, or both directions. To minimize the overall number of new points, and to maximize the number of inserted points at each iteration, so the number of iterations is minimal,



(a) Add knots in both directions (b) Minimum cover knot insertion

Figure 5.14. Offset surface refinement directions

we have a heuristic created that inserts the minimum number of knots to cover every marked cells at each iteration. A new knot is added at the location where the offset error is a maximum within the marked parametric region. The minimum refinement problem can be transformed into a minimum cover problem in the following steps:

1. Let boolean variables x_j represent the rows and columns where refinements may be required, i.e., which contain at least one marked cell. These are the rows and columns that would have been refined if the trivial refinement method were used. They are numbered from 1 to n, $n = 13$ as shown in Figure 5.15(a). Then, variable $x_j = 1$ if the j-th interval of row/column is to be refined, and $x_j = 0$ otherwise.

2. Define the constraints as the cells that must be covered by at least one refinement. If the marked cells are numbered from 1 to m, $m = 12$ in this example, an $m \times n$ boolean matrix is constructed as the coefficients of the constraints (see Figure 5.15(b)). The matrix element $a_{ij} = 1$ if the i-th marked cell is from the j-th row/column of the knot matrix. Otherwise $a_{ij} = 0$. Note that each row of the constraint matrix has exactly two nonzero elements, one from

		7	8	9	10	11	12	13
1			*1*					
2		*2*		*3*	*4*	*5*	*6*	*7*
3			*8*		*9*			
4					*10*			
5			*11*					
6					*12*			

(a) Knot matrix numbering scheme

	rows						columns						
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1							1					
2		1					1						
3		1							1				
4		1								1			
5		1									1		
6		1										1	
7		1											1
8			1					1					
9			1							1			
10				1						1			
11					1			1					
12						1				1			
c_j	9	9	9	9	9	9	7	7	7	7	7	7	7

(b) Matrix a_{ij} and coefficient c_j

Figure 5.15. Transform knot matrix to a minimum cover problem

each parametric direction.

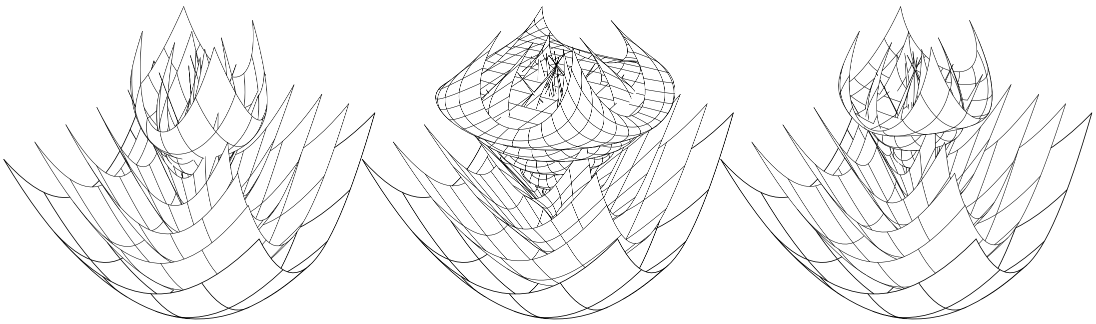
3. The coefficients c_j are the refinement weight factors. When all c_j are equal, the minimum number of new knots is computed. To minimize the number of control points, c_j can be set to the row/column size of the knot matrix, i.e., the number of added control points by inserting a new knot in the parametric directions (see Figure 5.15(b)).

4. A weighting factor of zero is assigned to force a refinement at the j-th row or column, i.e., $c_j = 0$ and $x_j = 1$. This heuristic is used when the whole row or column are marked, analogous to the way more cross-sections are added in sweeping operations by refining the axis curve [18]. The magnitude of the distance error function can also be used to assign the weight factors. The weight factors are inversely proportional to the offset error to favor more knots at places of higher error.

In general, a linear problem can have more than one solution. Global minima of covering problems can be determined by techniques such as linear programming [130, 142]. However, for problems of large m and n (≥ 100), such approaches can be impractical. Reduction methods that give local minimum for unweighted and weighted covering problems are given by Roth [117] and by Rho and Lapidus [111], respectively. We have modified them to find the locations and directions of refinements. Even though minimum refinement at each iteration does not guarantee an overall minimum of new knots or control points, the overall reduction in adaptive refinement compared to the full refinement method can be enormous.

5.5 Comparison of Surface Offset Results

Figure 5.16 shows the three different surface offset techniques on a bicubic NURBS surface, with distances from 0.25 to 2. The first one (from left) uses the control points offset method given by Cobb [31]. The second and the third examples use the new node points offset interpolation and control mesh offset methods, respectively. Note that the offset surfaces start self-intersecting half way,



(a) Offset control points (b) Offset node points (c) Offset control mesh

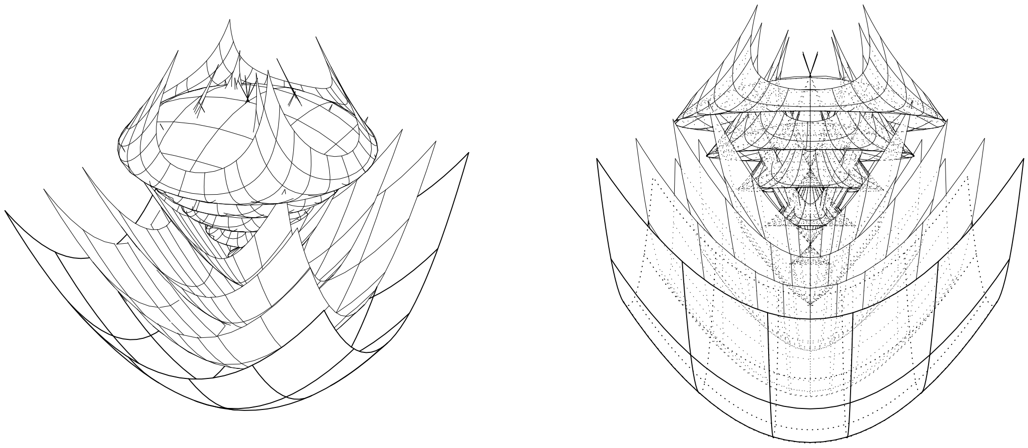
Figure 5.16. Offset surfaces of a bicubic B-spline surface

around $d = 1$, and eventually flip direction at the end. Table 5.1 lists the maximum distance error of the offset surfaces, which is proportional to the offset distance. The interpolation method performs best for this example.

Figure 5.17 shows offset surfaces using the adaptive minimum cover refinement method until the maximum error is within the prescribed tolerance, $\epsilon = 0.001$. Results in isometric and front views from only the interpolation method are shown. The other two methods also converge but require more refinements for the same tolerance. Table 5.2 lists the number of inserted points at different offset distances. Table 5.3 lists the number of inserted points for offset distance $d = 0.5$ at various offset tolerances, from 0.1 to 0.000001. A comparison between the minimum cover

Table 5.1. Offset errors at different offset distances

Offset Distance	Maximum distance error		
	Points offset	Interpolation	Mesh offset
0.25	0.06689	0.02612	0.03844
0.50	0.13244	0.05212	0.07638
0.75	0.19859	0.07817	0.11437
1.00	0.26456	0.10423	0.15246
1.25	0.33071	0.13027	0.19058
1.50	0.39674	0.15632	0.22868
1.75	0.46289	0.18238	0.26675
2.00	0.52899	0.20843	0.30485



(a) Isometric view (b) Front view

Figure 5.17. Adaptive refinement offset surfaces

Table 5.2. Number of inserted points at different offset distances

Offset Distance	Inserted points ($\epsilon = 0.001$)		
	Points offset	Interpolation	Mesh offset
0.25	825	105	273
0.50	1505	105	660
0.75	2189	128	713
1.00	2585	128	1007
1.25	3696	153	1208
1.50	4472	180	1140
1.75	5384	180	1544
2.00	6464	153	1962

Table 5.3. Number of inserted points at different tolerances

Offset Tolerance	Inserted points ($d = 0.5$)		
	Points offset	Interpolation	Mesh offset
0.1	20	9	9
0.01	209	33	33
0.001	1505	105	660
0.0001	15545	240	5684
0.00001	137609	1140	34394
0.000001	N/A	1584	N/A

refinement and the full refinement methods is given in Table 5.4 using the interpolation method. The offset interpolation method with adaptive minimum cover refinement shows orders of magnitude improvement over the other methods for the examples shown.

A method to compute successive offsets of a surface from a list of offset distances is also presented which splits the computation of the surface offset operation, as defined in Equation 5.3, into two stages. In the first stage, the unit normal vector field $\mathbf{n}(u,v)$ of the surface is approximated. This stage is performed only once for each surface and contains most of the computationally intensive iterations.

With the surface unit normal vector field computed, the rest of the computation involves scaling and addition of $\tilde{\mathbf{n}}(u,v)$ and $\mathbf{S}(u,v)$, which can be done symbolically in real time. To compute the offset surface at a different offset distance, only the second state of the offset operation is recomputed. Since we use the same order and weight function as the original surface in the adaptive refinement interpolation, the resulting offset surfaces have the same order as the original surface, for both polynomial and rational surfaces. Using this method, offset surfaces are constructed to within the prescribed tolerance, i.e.,

$$\begin{aligned}
\mathcal{E}_d &= |d - \|\tilde{\mathbf{S}}_d - \mathbf{S}\|| \\
&= |d - \|\mathbf{S} + d\tilde{\mathbf{n}} - \mathbf{S}\|| \\
&= |d\,(1 - \|\tilde{\mathbf{n}}\|)| \\
&\leq |d|\,\epsilon_n \\
&= |d/d_{max}|\,\epsilon_o \\
&\leq \epsilon_o.
\end{aligned}$$

Table 5.4. Number of inserted points using different refinement methods

Adaptive offset interpolation ($d = 0.5$)						
Offset tolerance	0.1	0.01	0.001	0.0001	0.00001	0.000001
Minimum cover	9	33	105	240	1140	1584
Full refinement	65	105	468	2100	5025	28545

Figure 5.18 shows the unit normal vector field of the bicubic NURBS surface used in the offset surface examples. The surface unit normal vector field is also called the Gauss mapping or the spherical image of surfaces [94]. Properties of Gauss mappings can be found in reference [12]. For surfaces with parabolic points, that is, where the Gaussian curvature is zero, the Gauss mappings can be very complex. Figure 5.19 shows such a bicubic NURBS surface and its Gauss mapping. Offset surfaces of this surface at distances 0.25, 0.5, and 0.75 are shown in Figure 5.20.

The two-stages surface offset method can also be used for geometric modeling

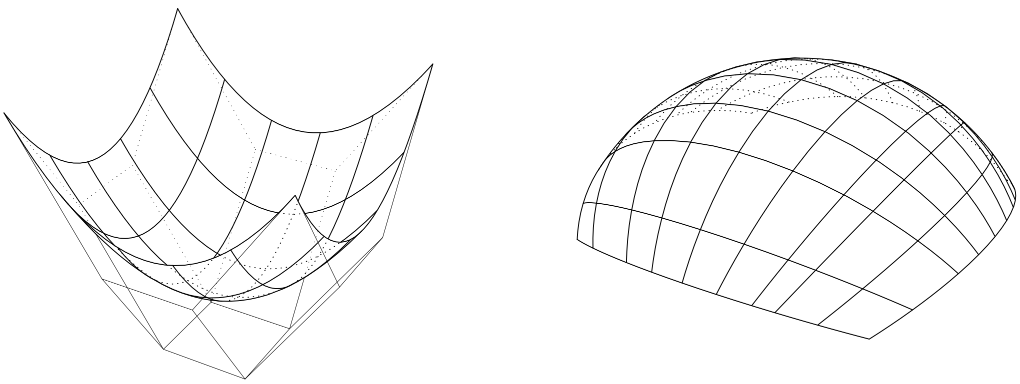


Figure 5.18. Spherical image of a bicubic NURBS surface

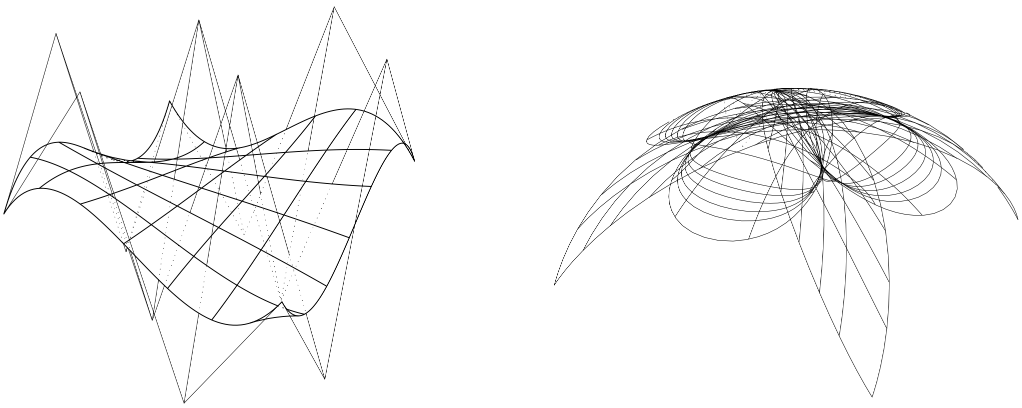


Figure 5.19. Spherical image of a bicubic surface with parabolic points

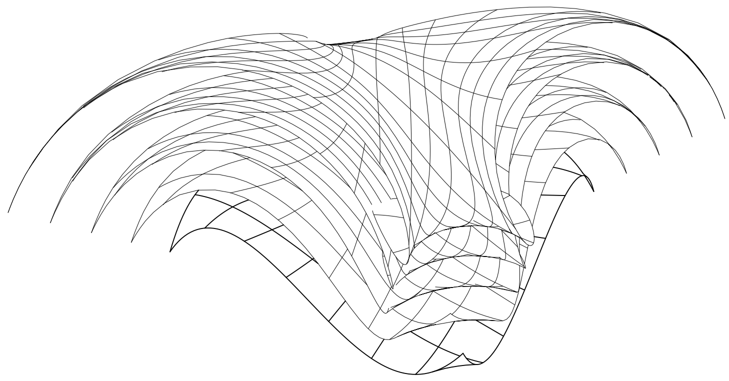


Figure 5.20. Offset surfaces examples

in an interactive design system. A graphical user interface to construct offset surfaces interactively lets the user pick the surface graphically, then control the offset distances by dragging a pointing device or using a scroll bar. The Gauss mapping is computed once after the user picked the surface. At each change of the offset distance, only the second stage is performed and the offset surface can be displayed in real time.

5.6 Offset Surfaces with C^1 Discontinuity

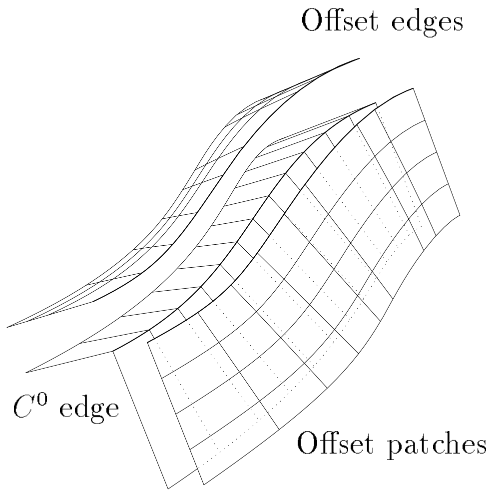
The algorithms that have been presented for offset approximation assume C^1 continuous curves and surfaces. In this section, we present procedures to construct offsets from C^0 continuous, but C^1 discontinuous surfaces. The approach used is to 1) subdivide curves or surfaces along C^1 discontinuities, 2) compute the offset surface of each C^1 subsurface, 3) construct offsets of sharp edges and corners to fill the gaps between offset patches. The third step includes constant radius fillet surfaces and spherical patches, called *spherical polygons*. A technique to model spherical polygons as biquadratic trimmed NURBS surfaces has been developed which handles offsets of corners from arbitrary number of edges. Finally, we extend the same techniques to compute offsets of solid models consisting of multiple trimmed NURBS surfaces. Examples of modeling operations using offset surfaces are presented.

5.6.1 Offsets of Edges and Corners

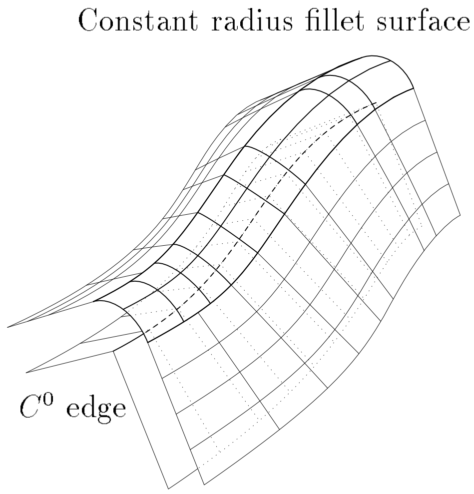
Figure 5.21(a) shows a surface with a C^1 discontinuity. The surface is first subdivided along its C^0 edge, then offsets of the subsurfaces are computed to within a tolerance. Note that two boundary edges of the offset surfaces are both offsets of the same C^0 edge, in different directions. To smoothly fill the gaps between the two offset edges, a constant radius fillet surface is constructed (see Figure 5.21(b)).

Circular arcs are first constructed using center points on the C^0 edge of the original surface and end points on the two offset edges. The two offset edges, called rail curves, are equidistant to the C^0 edge, the center curve, within the offset tolerance. Then, a fillet surface is constructed to interpolate these arcs and the rail curves.

The next step is to fill the gaps, if any, between offset fillet surfaces. Figure 5.22 shows the steps to offset a surface with six C^0 edges that meet at a sharp corner. After offsetting subdivided surfaces and edges, gaps are found between the six offset fillets (see Figure 5.22(b)). The top edges of the six offset fillets are offsets from the sharp corner point shared by the six C^0 edges. They are arcs of great circles on a sphere centered at the corner point. These arcs form a spherical polygon that is an

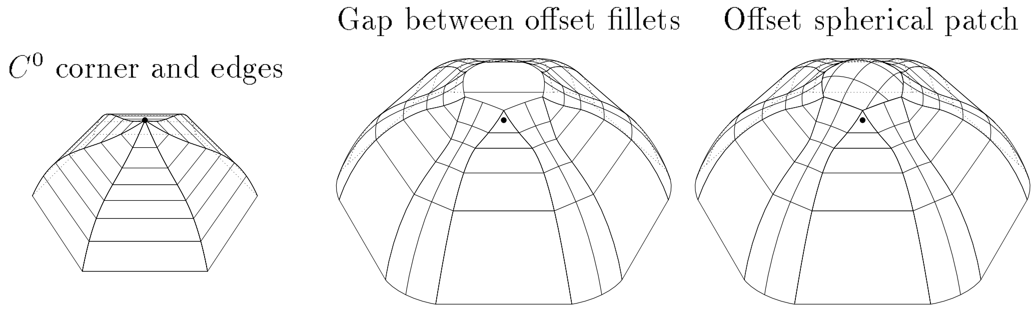


(a) Offsets of C^1 subpatches



(b) Offset of C^0 edge

Figure 5.21. Offset surfaces with C^0 edges



(a) C^0 surface (b) C^0 edge offsets (c) C^0 corner offset

Figure 5.22. Offset surfaces with C^0 corners

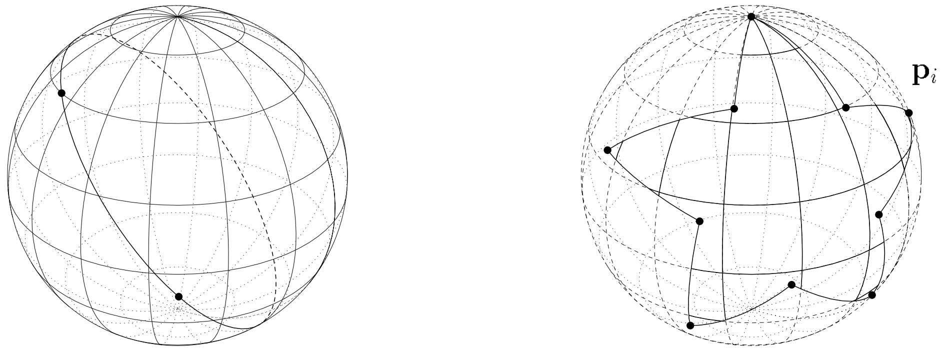
offset of the sharp corner point. Figure 5.22(c) shows the offset spherical polygon represented as a trimmed biquadratic NURBS surface with quadratic trimming curves.

5.6.2 Spherical Polygons

Definition 5.3 *A* great circle *between two points on a sphere is the largest circle on the sphere through the two points. A* spherical polygon *from vertices* $\{\mathbf{p}_i\}$ *on a sphere is the region bounded by the shortest arcs of great circles between vertices (see Figure 5.23).*

It is easy to see that radius of a great circle is the same as radius of the sphere, and center of a great circle is at center of the sphere.

Spherical triangles of equal angles can be modeled easily as regions from sur-



(a) Great circle (b) Spherical polygon

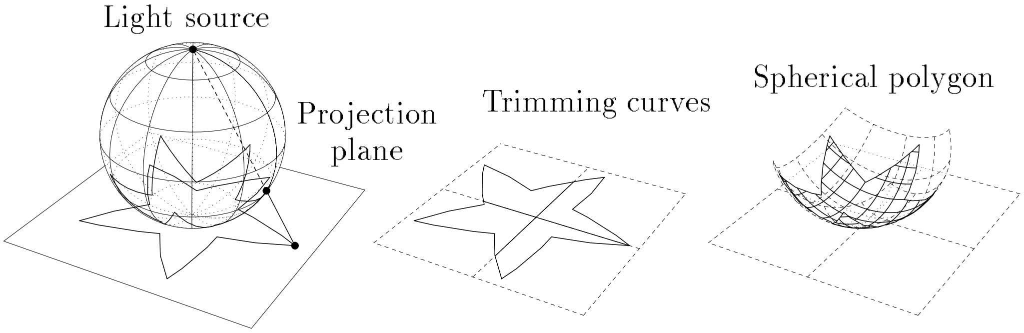
Figure 5.23. Great circle and spherical polygon

face of revolution. However, this cannot be done for arbitrary spherical patches. J. Cobb [32] used a technique based on stereographic projection and symbolic computation to model spherical square patches as biquartic, order 5, rational Bézier patches. His method has three steps: 1) project spherical boundaries to the projection plane, 2) construct a biquadratic planar surface patch bounded by the projected boundaries, 3) inverse map planar surface back to the sphere using symbolic computation.

A similar approach is used here to model arbitrary spherical polygons as biquadratic trimmed NURBS surfaces (see Figure 5.24). The first step, as shown in (a), is the same as in Cobb's method. The stereographic projection is a conformal projection, one that preserves angles. The location of the "light source" is the antipode of the center of the projection; that is, it is the point exactly opposite the point of tangency [22]. Consider the projection of a unit sphere centered at the origin by placing the projection plane ($z = -1$) tangent to the south pole and the light source at the north pole. Any point $\mathbf{p}$ on the sphere, except the north pole, has a unique projection point $\widehat{\mathbf{p}}$ on the $z = -1$ plane:

$$\begin{aligned} \widehat{p}_x &= 2p_x/(1-p_z) \\ \widehat{p}_y &= 2p_y/(1-p_z) \\ \widehat{p}_z &= -1. \end{aligned} \tag{5.7}$$

A unique aspect of the stereographic projection is that, regardless of its orien-



(a) Stereographic projection (b) Trimmed surface (c) Inverse mapping

Figure 5.24. Modeling of spherical polygons

tation, each circle on the sphere appears as a circle on the projected map [132]. Projections from boundaries of any spherical polygon are also circular arcs. They can be constructed by projecting three points, start, end, and mid-point of the spherical boundaries. It is important to make sure that the projection light source is not within the spherical polygon. The centroid of polygon is used to align the projection plane.

In step 2, instead of constructing surface patches from the projected boundaries, which is not always possible without tessellation, a bilinear trimmed surface is used to cover the projected area (see Figure 5.24(b)). Finally, inverse mappings of stereographic projection are taken on the surface data while keeping the same parametric trimming curves. The inverse mapping from $\widehat{\mathbf{p}}$ onto a point $\mathbf{p}'$ on the sphere,

$$\begin{aligned} p'_x &= 4\widehat{p}_x \\ p'_y &= 4\widehat{p}_y \\ p'_z &= \widehat{p}_x^2 + \widehat{p}_y^2 - 4 \\ p'_w &= \widehat{p}_x^2 + \widehat{p}_y^2 + 4. \end{aligned} \tag{5.8}$$

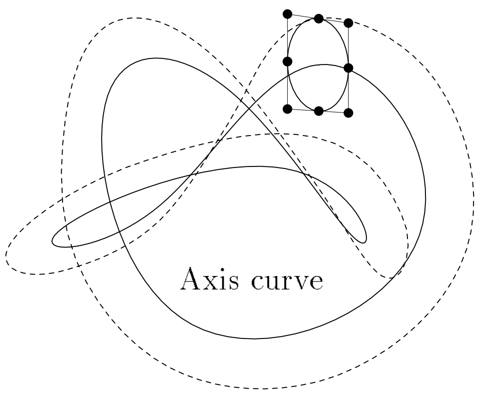
Substituting Equations 5.7 into Equations 5.8, one can easily verify that

$$p_i = p'_i/p'_w \text{ and } \sum p_i^2 = 1, \;\; i = x, y, z.$$

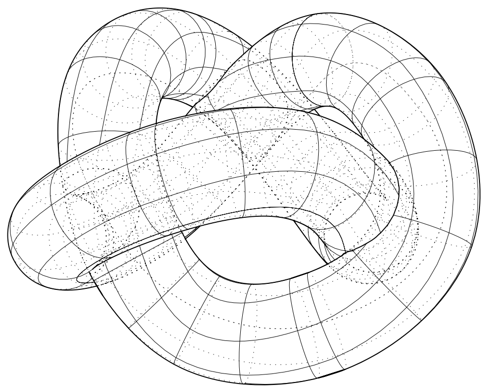
Computation of the final step involves spline function multiplication, which raises surface order from k to $2k - 1$. Therefore, the bilinear planar trimmed surface becomes a biquadratic rational spherical surfaces with the same quadratic trimming curves (see Figure 5.24(c)).

5.7 Offsets and Solid Modeling

There are many applications of offset curves and surfaces in solid modeling [18]. For instance, Figure 5.25 shows an example of using the 3D curve offset technique to perform a sweeping operation. The axis curve and a circular cross-section curve is shown on the left. The offset of the axis curve at the radius of the cross-section



(a) Offset of the axis curve



(b) Sweep surface

Figure 5.25. Sweeping operation using adaptive polygon offset method

is first computed using the 3D adaptive polygon offset method, shown as the dashed curve. In this example, the radius equals 0.2, and the tolerance is 0.01. This adaptively refined offset curve is then used to determine the locations and orientations to place the control points of the sweeping circular cross-sections. The resulting surface is shown on the right. Because the polygon offset method preserves tangent continuities, even at the end-to-end junction of a closed curve, the two ends of the sweep surface meet and line-up with each other.

Offsets of solid models with multiple trimmed surfaces can be computed in a similar manner. Trimming curves are split along subdivision boundaries and assigned to subdivided patches, later transferred to their offsets. The offset of a trimmed surface is also a trimmed surface with the same parametric trimming curves. Edge/corner offset techniques are also applied to inter-surface discontinuities, including trimmed surface boundaries. Figure 5.26 shows a curved box from six surfaces and its offset — a curved round box.

Surface offset techniques provide a basis for many geometric operations in solid modeling. Rounding and filleting are operations used to smooth sharp edges and corners. A constant radius rounding operation can be achieved by first shrinking the trimmed model, i.e., offsetting with a negative distance. Then excessive regions should be trimmed away. Finally, a positive offset of the negative offset model

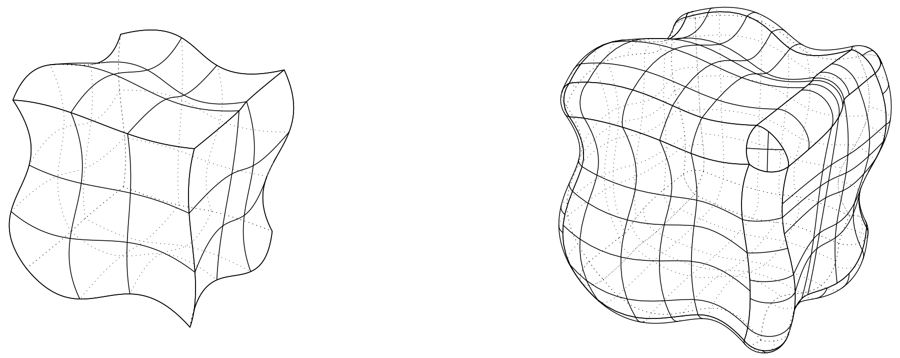


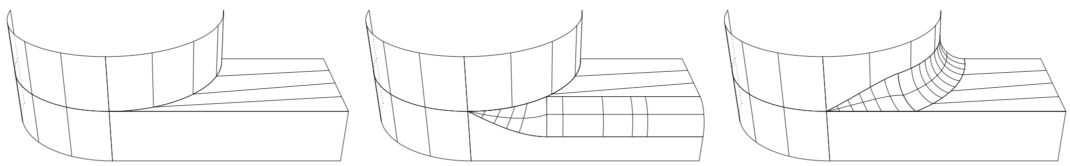
Figure 5.26. Offset of a curved box from six surfaces

should be created. Reverse the order, constant radius fillets are constructed to remove concave edges and corners.

Figure 5.27(a) shows a joint model commonly found in mechanical part designs. It has two sharp edges, one concave and one convex, which are usually not desirable because of poor mechanical properties. Figures 5.27(b) and (c) show results from applying rounding and filleting operations on the joint model respectively. Both edges are replaced by constant radius fillet surfaces. Note that the degenerate end of fillet surfaces are from arcs of zero degrees, not zero radius.

A very popular operation in mechanical part design is the "round-all" operation, usually denoted by a typ. (typical) radius in engineering drawings. Every sharp edge/corner is replaced by an arc of same radius. Figure 5.28(a) shows a joint model with filleting and rounding applied to both edges. Figure 5.28(b) shows a telephone handle model with all edges rounded to the same radius.

Shelling is an operation to convert surfaces into solids of constant thickness. It



(a) Joint model (b) Rounding (c) Filleting

Figure 5.27. Rounding and filleting operations

can be done by offsetting surfaces at half the thickness in both sides. Fillets can also be constructed for open boundaries and corners. Figure 5.28(c) shows a rounded shell constructed from a single surface of the curved box example shown before.

We conclude this chapter by formal definitions of the rounding, filleting, round-all, and shelling operations as compositions of the constant distance offset operations:

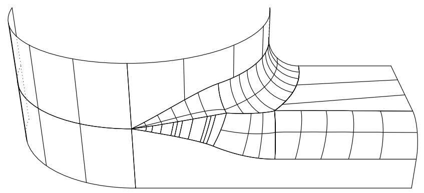
$$\textit{Rounding} \ : \ \mathcal{O}^*(\mathbf{S}) = \mathcal{O}(\mathcal{O}(\mathbf{S}, -d), d) \tag{5.9}$$

$$\textit{Filleting} \ : \ \mathcal{O}^+(\mathbf{S}) = \mathcal{O}(\mathcal{O}(\mathbf{S}, d), -d) \tag{5.10}$$

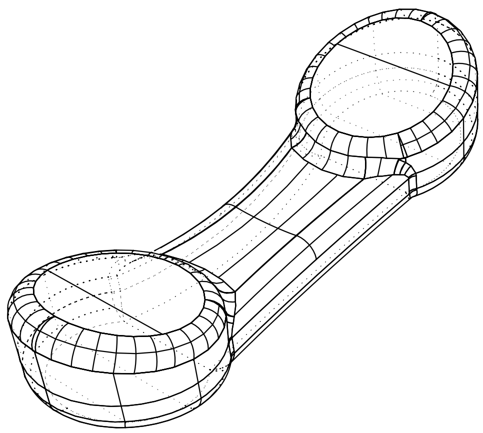
$$\textit{Round-all} \ : \ \mathcal{O}^*(\mathcal{O}^+(\mathbf{S})) \text{ and } \mathcal{O}^+(\mathcal{O}^*(\mathbf{S})) \tag{5.11}$$

$$\textit{Shelling} \ : \ \mathcal{O}(\mathbf{S}, d/2) \cup \mathcal{O}(\mathbf{S}, -d/2). \tag{5.12}$$

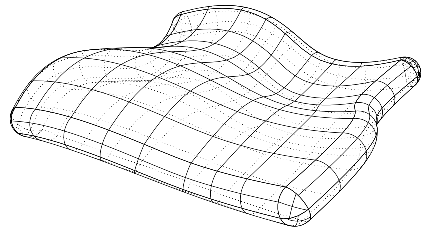
Note that, since the offset operator is not invertible, the rounding and filleting operations do not commute. The two round-all operations may result in different models. A required operation to use offset surfaces in solid modeling is being able to trim away self-intersections. Constant distance offsets are (mathematical) offsets with self-intersections trimmed away.



(a) Rounded joint model



(b) Rounded phone handle model



(c) Open edge offsets

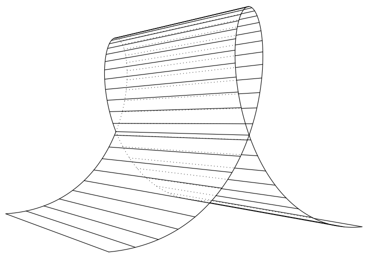
Figure 5.28. Round-all and shelling operations

CHAPTER 6

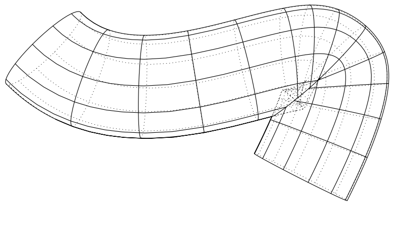
SELF-INTERSECTION

One fundamental problem of using boundary representations in solid modeling is that if the boundary surface self-intersects, it may not represent a solid volume unambiguously [62]. Surface self-intersection can occur in many B-rep geometric modeling operations, such as bending, sweeping, interpolation, and offsetting [18, 31]. Figure 6.1 shows some examples of self-intersecting surfaces. For solid modeling, surface self-intersections must be trimmed away to obtain a valid solid model.

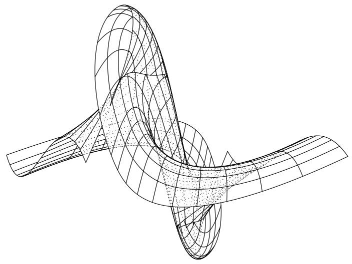
Self-intersection of offset curves and surfaces may occur when the offset distance is greater than the minimum radius of curvature [47]. An offset has cusps at points where the radius of curvature is equal to the offset distance. Local properties such



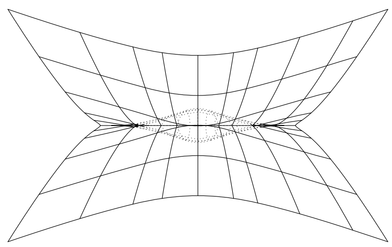
(a) Extrude



(b) Sweep



(c) Interpolate



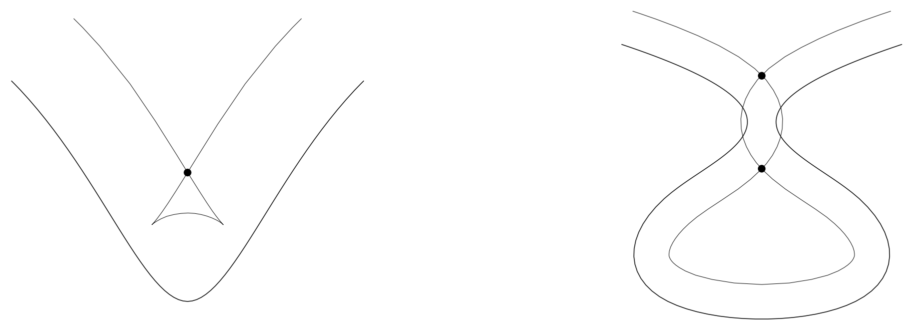
(d) Offset

Figure 6.1. Examples of self-intersecting surfaces

as curvature, normal, and tangent vectors have been used in identifying cusps and self-intersections of offset curves [42, 47].

However, high curvature is not a necessary condition for self-intersection to occur in offsets. Figure 6.2 shows two different types of self-intersection of offset curves. The local self-intersection shown in Figure 6.2(a) indeed is caused by high curvature, relative to the offset distance. However, the global self-intersection shown on the right happens between two convex regions. Curvature analysis alone would not predict self-intersection for this case. Moreover, self-intersection may occur in any curve, not just offset curves. This indicates that local properties alone are not enough to characterize the occurrence of self-intersection, even in offsets. To reliably and efficiently detect and identify self-intersection, one must consider global properties of curves and surfaces as well.

To better understand the characteristics of surface self-intersections, this chapter starts with an overview of the intersection problem of NURBS curves and surfaces, i.e., to detect and locate the intersection points between two curves or surfaces. Then, problems that may arise if an intersection algorithm is to be applied on the same curve or surface are discussed. A necessary condition of self-intersection is presented. It is used to prune away curves or surfaces that are not self-intersecting. By just focusing on areas where self-intersection may occur, it greatly reduces the amount of numerical computation. Algorithms to find self-intersections and to trim the surfaces to create valid boundary surfaces are presented. Finally, results and



(a) Local self-intersection (b) Global self-intersections

Figure 6.2. Self-intersections of offset curves

examples of self-intersections for solid modeling and machining analysis are given.

6.1 Review of Intersection Techniques

The main purpose of this section is to give an overview of what an intersection problem is and to understand the basic approaches to intersection detection and finding. Intersections between curves are first discussed, followed by surfaces.

6.1.1 Curve Intersection

Definition 6.1 *Let* $\mathbf{C}_1(t)$ *and* $\mathbf{C}_2(t)$ *be two continuous curves.* $\mathbf{C}_1$ *and* $\mathbf{C}_2$ intersect *if and only if:*

$$\exists t_1, t_2,\ \mathbf{C}_1(t_1) = \mathbf{C}_2(t_2) \tag{6.1}$$

The points where two intersecting curves meet are called the intersection points *of* $\mathbf{C}_1$ *and* $\mathbf{C}_2$ *(see Figure 6.3).*

The intersection problem of curves is to find all pairs of t_1 and t_2 that correspond to the intersection points of two curves. The intersection of two continuous curves is either 1) empty, 2) a list of points, 3) a list of curves, i.e., overlapped regions, or 4) combination of 2 and 3. In the following, the two curves are assumed to be in "general" position and have intersection type 1 or 2.

Since an analytic solution to self-intersections of freeform NURBS curves does not always exist, approximations of the true intersection points are found instead. A tolerance ϵ is usually used to satisfy the intersection condition. Intersections between two curves can be found by first approximating both curves by line seg-

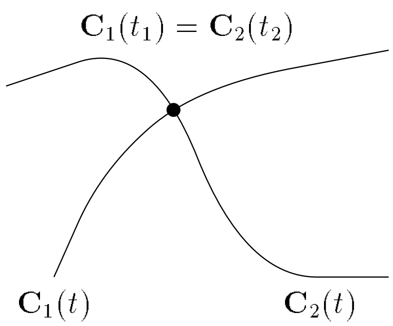


Figure 6.3. Intersection point of curves

ments. Then, intersection points of segments from each curve can be computed in $O(n \log n)$ time using a sweep algorithm [113].

To localize the approximation and intersection, a divide-and-conquer algorithm can be used [125],

```
ALGORITHM CrvInter( Crv1, Crv2, Tolerance )
BEGIN
    IF Crv1 and Crv2 may intersect THEN
    BEGIN
        Subdivide Crv1 and Crv2 if not straight within Tolerance.
        If either curve is subdivided, return the
                intersections from the recursive calls of
                CrvInter on the subdivided subcurves.
        Approximate Crv1 and Crv2 by line segments
        Return the intersection point of the line segments.
    END
END
```

Intersection of bounding boxes or convex hull of curves is often used to test whether two curves may intersect or not. For faster convergence, bounding arcs and arcs intersection can be used, which gives cubic convergence [126].

6.1.2 Surface Intersection

The surface intersection problem can be categorized into four groups:

1. Intersection between two surfaces.
2. Intersection between two offset surfaces.
3. Self-intersection of an offset surface.
4. Self-intersection of a surface.

The intersection between two surfaces can be defined as:

Definition 6.2 *Let* $\mathbf{S}_1(u, v)$ *and* $\mathbf{S}_2(u, v)$ *be two continuous surfaces.* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ intersect *if and only if:*

$$\exists (u_1, v_1), (u_2, v_2),\ \mathbf{S}_1(u_1, v_1) = \mathbf{S}_2(u_2, v_2) \tag{6.2}$$

The points where two intersecting surfaces meet are called the intersection points *of* $\mathbf{S}_1$ *and* $\mathbf{S}_2$*.*

The intersection of two continuous surfaces is either empty, a list of points, a list of curves, a list of surfaces, i.e., overlapped regions, or a combination of them [13]. Most surface/surface intersection algorithms assume the two surfaces are in "general" position and focus on the problem of finding the (possible) intersection curves between the two surfaces [10, 13, 14, 65, 96, 97, 105, 137].

Definition 6.3 *An* intersection curve *of two intersecting surfaces is the surface curve of intersection points, i.e.:*

$$\forall t,\ \mathbf{C}(t) = \mathbf{S}_1(u_1(t), v_1(t)) = \mathbf{S}_2(u_2(t), v_2(t)) \tag{6.3}$$

A complete representation of an intersection curve consists of three curves, a Euclidean curve and two parametric curves. The intersection problem of surfaces is to find all intersection curves between two surfaces within a prescribed tolerance.

There are two predominant methods to find the intersection curves between two NURBS surfaces. The first one is a subdivision-based divide-and-conquer method [65, 105, 137]. The two surfaces are adaptively subdivided into flat polygons, usually triangles. Then, intersections between the triangles are computed and linked together to form the intersection curves. An optional numerical refinement can be applied to improve the accuracy of the intersection points [89]. The subdivision method can also be parallelized due to its nonsequential nature [103].

Another technique is to numerically trace the intersection curves [10, 13]. Rough subdivision is first used to find some initial intersection points [97]. These points are then numerically improved and used as starting points to trace out the intersection curves [96]. The step size used at each point is estimated based on the surface curvature. To ensure that all branches of intersection curves are traced, analysis of the surface geometry is performed to ensure that no loop of an intersection curve is embedded inside the original subdivision patches [124].

A hybrid method [14] applies the adaptive subdivision technique used in the divide-and-conquer method as a front end to find the initial intersection points. Then an adaptive numerical improvement scheme is used as a marching method to refine the intersection curves. There are also algebraic methods for special case surfaces, or approximations of the original surfaces using nonlinear surface patches [7, 85, 121].

Intersection between two offset surfaces is a special application of the surface/surface intersection techniques on offsets of two surfaces [15, 48, 76, 148]. It is mainly used in the construction of constant radius fillet surfaces defined by rolling balls between two surfaces. Centers of the rolling balls are at a constant distance to the two surfaces and can be computed as the intersection curves between the offsets of the two surfaces. Evaluation of the surface offset functions is incorporated into the surface intersection condition of Definition 6.2. Therefore, the intersections are computed directly from the original surfaces without computing their offsets first, which could introduce additional offset approximation error. The start point is found by either a subdivision method or searching from the intersection of the two original surfaces. Then, the surface intersection marching method is modified to trace the fillet center curve.

Finding self-intersections of offset surfaces is a hard but important task in the use of offset surfaces [109]. There have been two approaches to finding the self-intersections of offset surfaces. The first method constructs approximation of the offset surface first, then finds the self-intersections of the offset surface [16, 42]. The offset surface is subdivided and approximated with triangles, and intersections between the triangles are computed. However, due to the lack of any self-intersection detection and pruning method, localization of the subdivision found in the divide-and-conquer surface intersection techniques is not applicable, and therefore, the subdivision and triangles intersection are performed everywhere on the surface. Numerical methods are applied on the self-intersection points to improve the accuracy and to refine the self-intersection curves. However, existing numerical surface intersection methods assume nondegenerate surfaces whose normal is defined everywhere, and can fail on the degenerate offset surfaces which occur when the offset distance is greater than the radius of curvature.

The second method to find the self-intersections of offset surfaces applies the surface/surface offset intersection techniques on the same surface [6, 15, 88]. Self-intersection points of the offset are computed by minimizing an objective function which equals the Euclidean distance between offset points divided by their parametric distance. Then, if any self-intersection point is found, self-intersection curves are calculated by solving simultaneous homogeneous differential equations using numerical integration, in which the points found in the first step are used as the starting points.

To ensure that all branches of the self-intersection curves are found, differential geometry and global distance functions are used to subdivide the surface at all the extrema of principal curvatures and at all collinear normal points whose distance is smaller than twice the offset distance [88]. These are the possibilities for having closed loops of self-intersection curves due to local differential geometry properties and global distance function properties. The global self-intersection is computed by solving simultaneous equations of four variables. Searching for the collinear normal points could be slow, and there is no guarantee of finding all solutions. A major problem of applying the surface/surface intersection techniques directly on the same surface is that the self-intersection equations contain infinite number of trivial roots which correspond to the same point on the surface. Detection of these trivial self-intersections requires many subdivisions, and it is difficult to distinguish between trivial roots and parametrically nearby self-intersection points.

Construction of robust algorithms for computing self-intersections of general parametric surfaces is a very difficult problem [16]. So far, techniques to find self-intersections of surfaces subdivide the whole surface into polygons. This can be slow and inaccurate. Applying the surface/surface intersection techniques directly on the same surface faces the same difficulties of trivial roots pruning as those found in the offset surface self-intersection techniques. In the following sections, we address these problems and introduce necessary conditions for occurrence of self-intersections in curves and surfaces. Using these conditions, we present a hybrid approach to self-intersection finding and trimming.

6.2 Curve Self-Intersection

A direct extension from Definition 6.1 to curve self-intersection is to replace the two intersecting curves with the same curve. However, an immediate observation on such a definition is that every point on the curve is a self-intersection point to itself. The definitions given below resolve this problem.

It is the author's belief that self-intersection should be a (global) intrinsic property of curves and surfaces, independent of their parametrization. In the following, we start by giving a definition for self-intersecting arc length parametrized curve. Then, we extend the definition to any parametric curve. To simplify the discussion, all curves are assumed to be continuous.

Definition 6.4 *Let* $\mathbf{C}(s)$ *be an arc length parametrized curve, i.e., a unit-speed curve. Then,* $\mathbf{C}(s)$ *is* self-intersecting *if and only if*

$$\exists s_1 \neq s_2,\ \mathbf{C}(s_1) = \mathbf{C}(s_2) \tag{6.4}$$

For a parametric curve $\mathbf{C}(t)$, we want to ensure that any self-intersecting curve is still self-intersecting after a reparametrization by its arc length. The importance of this condition is frequently overlooked.

Definition 6.5 *Curve* $\mathbf{C}(t)$ *is* degenerate *between* $[t_1, t_2]$, $t_1 \neq t_2$, *if*

$$\forall t \in [t_1, t_2],\ \mathbf{C}(t) = \mathbf{C}(t_1) = \mathbf{C}(t_2) \tag{6.5}$$

Definition 6.6 *Curve* $\mathbf{C}(t)$ *is* self-intersecting *if and only if*

$$\exists t_1 \neq t_2,\ \mathbf{C}(t_1) = \mathbf{C}(t_2)\ \textit{and}\ \exists t \in [t_1, t_2],\ \mathbf{C}(t) \neq \mathbf{C}(t_1) \tag{6.6}$$

Points $\mathbf{C}(t_1)$ *and* $\mathbf{C}(t_2)$ *are called the* self-intersection *points of* $\mathbf{C}(t)$ *(see Figure 6.4).*

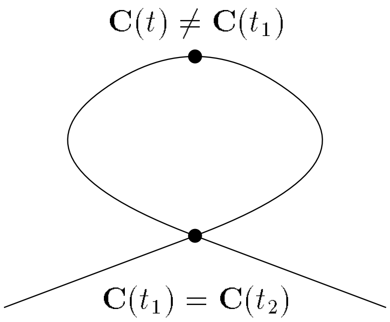


Figure 6.4. A self-intersecting curve

Corollary 6.1 *A self-intersecting curve must form a nondegenerate closed loop, simple or complex, between its self-intersection point pair.*

This is obvious from Definitions 6.5 and 6.6.

Algorithm `CrvInter` can be modified to find self-intersection points of curves. Assume the curve does not overlap with itself, i.e., there are finite number of self-intersection points.

```
ALGORITHM CrvSelfInter( Crv, Tolerance )
BEGIN
    IF Crv is not straight within Tolerance and
            may self-intersect THEN
    BEGIN
        Subdivide Crv into SubCrv1 and SubCrv2.
        Return CrvInter( SubCrv1, SubCrv2, Tolerance ) and
               CrvSelfInter( SubCrv1, Tolerance ) and
               CrvSelfInter( SubCrv2, Tolerance ).
    END
END
```

Two missing components need to be added to make this algorithm practical. First, a necessary condition for curve self-intersection should be used to prune away curves that cannot intersect with themselves. It should be simple enough to test whether a curve may self-intersect in a fashion similar to the bounding box test used in curve intersection. The second missing component is in the computation of intersections between two neighboring subcurves. Since the two subcurves connect to each other, their bounding boxes will intersect. This situation remains in one of the recursive calls in the curve intersection algorithm, subdivision after subdivision.

A test between neighbors to see if there might be any intersection at other than the junction point should be performed in addition to the bounding box test. Without these two additions, the divide-and-conquer adaptive subdivision algorithm will subdivide the curve everywhere until it is approximated by a polyline. Although the algorithm may still work if junction points between neighbors are ignored, it is probably less efficient than a simple and straightforward sweep algorithm.

6.2.1 A Necessary Condition

A necessary condition for curve self-intersection is developed in the section utilizing global theory of curves from differential geometry [94].

Definition 6.7 *Let* $\mathbf{S}^2$ *be the set of unit vectors, i.e., a unit sphere. The* open hemisphere *with pole* $\mathbf{a}$ *is* $\{\mathbf{x} \in \mathbf{S}^2 | \, \mathbf{a} \cdot \mathbf{x} > 0\}$. *The* closed hemisphere *with pole* $\mathbf{a}$ *is* $\{\mathbf{x} \in \mathbf{S}^2 | \, \mathbf{a} \cdot \mathbf{x} \geq 0\}$.

Definition 6.8 *Let* $\mathbf{C}$ *be a unit-speed curve. Its unit tangent vector* $\mathbf{T} = \mathbf{C}'$ *defines a curve on* $\mathbf{S}^2$, *i.e., the unit sphere about the origin. This curve is called the* tangent spherical image *of* $\mathbf{C}$.

Lemma 6.2 *If* $\mathbf{C}$ *is a regular closed curve, then its tangent spherical image does not lie in any open hemisphere.*

Proof: This can be proved by contradiction. Suppose that the tangent spherical image of a closed curve $\mathbf{C}$ lies in the open hemisphere with pole $\mathbf{a}$, so that $\mathbf{a} \cdot \mathbf{T} > 0$. Then

$$0 < \int_0^L \mathbf{a} \cdot \mathbf{T} \, ds = \int_0^L \mathbf{a} \cdot \mathbf{C}' \, ds = \int_0^L (\mathbf{a} \cdot \mathbf{C})' \, ds = (\mathbf{a} \cdot \mathbf{C}) \Big|_0^L = 0$$

(since $\mathbf{C}$ is closed), which is impossible. ■

Corollary 6.3 *If* $\mathbf{C}$ *is a piecewise smooth closed curve and its spherical image is defined so that the disconnected segments of its* C^1 *spherical images are joined by great circle segments of length* $\Delta\theta_i$, *the jump angles at the junction points, then this extended spherical image does not lie in any open hemisphere.*

Proof of this corollary is essentially the same as the proof for Lemma 6.2.

Definition 6.9 *A* direction cone $\langle \mathbf{a}, \theta \rangle = \{\mathbf{x} \in \mathbf{S}^2 | \mathbf{x} \cdot \mathbf{a} \geq \cos \frac{\theta}{2}\}$ *is a 3D geometry defined by its axis unit vector* $\mathbf{a}$ *and spanning angle* θ *(see Figure 6.5(a)).*

It is easy to see that the direction cone $\langle \mathbf{a}, \theta \rangle$ defines a region bounded by a circle in $\mathbf{S}^2$, i.e., a unit sphere. The circle has radius equal to $\sin \frac{\theta}{2}$ and center point aligned with the axis vector $\mathbf{a}$, or in the opposite direction of $\mathbf{a}$ if $\theta > \pi$. When $\theta = \pi$, the center point is at the center of the unit sphere. In this case, the direction cone defines a closed hemisphere with pole at its axis vector $\mathbf{a}$.

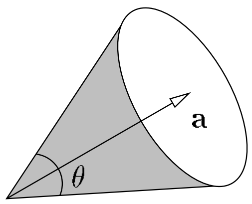
Definition 6.10 *A* tangent bounding cone *of a curve* $\mathbf{C}$ *is a direction cone* $\langle \mathbf{a}, \theta \rangle$ *that contains all unit tangent vectors* $\mathbf{T}$ *of* $\mathbf{C}$ *(see Figure 6.5(b)), i.e.,:*

$$\forall t,\ \mathbf{T}(t) \cdot \mathbf{a} \geq \cos \frac{\theta}{2} \tag{6.7}$$

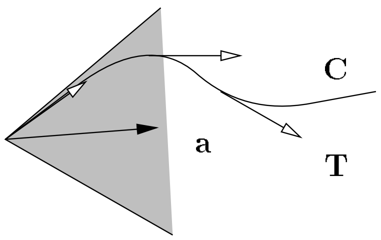
The smallest tangent bounding cone *is the bounding cone with the smallest spanning angle.*

Lemma 6.4 *The smallest tangent bounding cone of a closed curve spans at least* π*.*

Proof: Suppose there exists a tangent bounding cone of the closed curve $\mathbf{C}$ that spans less than π with axis $\mathbf{a}$. Then an open hemisphere with pole at $\mathbf{a}$ would contain the tangent spherical image of $\mathbf{C}$, which contradicts with Lemma 6.2. ∎



(a) Direction cone



(b) Tangent bounding cone

Figure 6.5. Direction cone and tangent bounding cone

Proposition 6.5 *The smallest tangent bounding cone of a self-intersecting curve spans at least* π.

Proof: From Corollary 6.1 and Lemma 6.4, a self-intersecting curve forms a closed subcurve between the two self-intersection points, which has bounding cones with spanning angles at least π. Therefore, the smallest tangent bounding cone of the whole curve spans at least π. ■

Proposition 6.5 provides the key to a test for a self-intersecting curve, that can be answered by "no" or "maybe." For a NURBS curve, tangent bounding cone can be approximated by bounding the direction vectors of its control polygon. A better approximation could be obtained by computing bounds of its first derivative, which also has a NURBS representation.

```
ALGORITHM CanSelfInter( Crv )
BEGIN
    Compute a tangent bounding cone of Crv.
    If angle of the bounding cone < 180 degrees, return FALSE.
    Otherwise, return TRUE.
END
```

To test whether two neighbors can intersect at places other than the junction point, we apply algorithm CanSelfInter on the concatenation of the two curves. A more efficient method to approximate the total bounding cone of neighbors is to take the union of the individual bounding cones.

6.3 Surface Self-Intersection

Similar to curves, we define the self-intersection points of a parametric surface to exclude degenerate self-intersections due to parametrization.

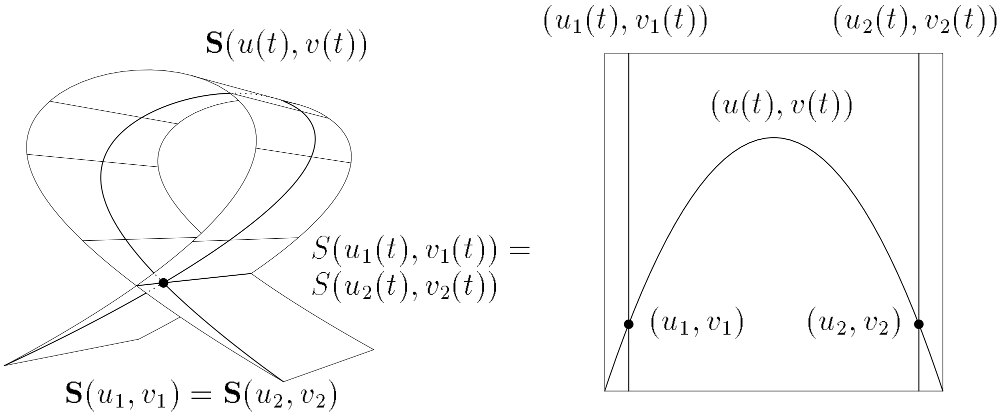
Definition 6.11 *Let* $\mathbf{C}(t)$ *be a continuous curve on surface* $\mathbf{S}(u,v)$ *through points* $\mathbf{S}(u_1,v_1)$ *and* $\mathbf{S}(u_2,v_2)$, *i.e.,*

$$\forall t,\ \mathbf{C}(t) = \mathbf{S}(u(t),v(t)) \text{ and } (u(t_i),v(t_i)) = (u_i,v_i),\ i = 1,2 \tag{6.8}$$

Surface $\mathbf{S}(u,v)$ *is* self-intersecting *if and only if every surface curve* $\mathbf{C}(t)$ *through* $\mathbf{S}(u_1,v_1)$ *and* $\mathbf{S}(u_2,v_2)$ *is self-intersecting at the two surface points. Points* $\mathbf{S}(u_1,v_1)$

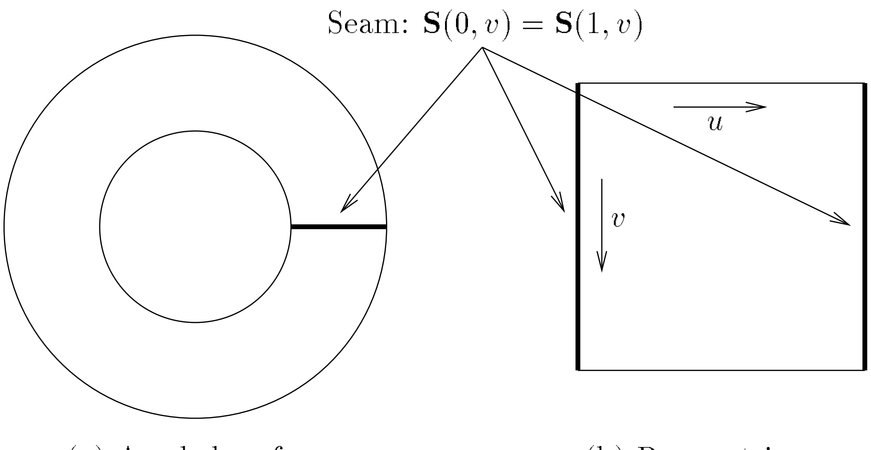
and $\mathbf{S}(u_2, v_2)$ *are called the* self-intersection points *of* $\mathbf{S}(u, v)$ *(see Figure 6.6).*

An immediate surface analogy to Proposition 6.5 would be to assert that the minimum bounding cone of surface normal vectors must span at least π. For examples shown in Figure 6.1, at least, this condition holds. However, it is insufficient, in general, since planar surfaces could also self-intersect. An easy example is a planar ruled-surface between two concentric circles, which intersects itself along the seam but has normals pointing in one direction (see Figure 6.7). In the following, a necessary condition for self-intersecting planar surfaces is first discussed. Then, any



(a) Euclidean space (b) Parametric space

Figure 6.6. A self-intersecting surface



(a) A ruled-surface (b) Parametric space

Figure 6.7. An example of planar self-intersecting surface

planar bijective projection of a 3D self-intersecting surface should also self-intersect and satisfy the condition.

6.3.1 A Necessary Condition

Definition 6.12 *A* minimal self-intersecting surface $\mathbf{S}$ *is a self-intersecting surface such that every nontrivial subdivision of* $\mathbf{S}$ *results in subsurfaces that are not self-intersecting. A nontrivial subdivision means that the subdivision does not occur along a boundary in the subdivision direction.*

Lemma 6.6 *Self-intersection of a minimal self-intersecting surface occurs only at the boundaries.*

Proof: Suppose one of the self-intersection points of a minimal self-intersecting surface lies in the interior. Then a nontrivial subdivision exists such that the two intersection points remain in the same side of the subdivision line, for example, through the interior self-intersection point. Therefore, at least one of the subdivided surfaces is still self-intersecting. This contradicts the definition. ■

Lemma 6.7 *The intersection points of a minimal self-intersecting surface are located at either the two diagonal corners of the surface or at the end points of an isoline. In the latter case, the minimal self-intersecting surface is degenerate and its image is a single isoline curve.*

Proof: If the two self-intersection points lie on the same isoline, the surface can always be subdivided by a nontrivial subdivision that leaves the isoline segment intact until the surface degenerates into a single isoline starting and ending at the two points. Otherwise, from lemma 6.6, self-intersection only occurs on boundaries for a minimal self-intersecting surface. The self-intersection points must be at one of the two diagonal corners. ■

Definition 6.13 *A* normal bounding cone *of surface* $\mathbf{S}(u, v)$ *is a direction cone that contains all surface normal vectors. A* tangent bounding cone *of* $\mathbf{S}(u, v)$ *in an isoparametric direction is a direction cone that contains tangent vectors of all* *isolines in the isoparametric direction [73]. The smallest normal bounding cone and smallest tangent bounding cones of* $\mathbf{S}(u,v)$ *are the corresponding bounding cones with the smallest spanning angles, respectively. A surface has a smallest normal bounding cone and two smallest tangent bounding cones, one in each parametric direction.*

$$\begin{aligned} \forall(u,v),\ \mathbf{n}(u,v)\cdot\mathbf{a}_n &\geq \cos\frac{\theta_n}{2} \\ \mathbf{T}_u(u,v)\cdot\mathbf{a}_u &\geq \cos\frac{\theta_u}{2} \\ \mathbf{T}_v(u,v)\cdot\mathbf{a}_v &\geq \cos\frac{\theta_v}{2} \end{aligned} \tag{6.9}$$

Corollary 6.8 *For planar surfaces, the spanning angle of the minimum normal bounding cone is either 0 or* π*.*

Proof: Normal vector of a planar surface can either parallel or anti-parallel to the normal direction of the plane it is on. The spanning angle equals π if the normal vectors flip directions. Otherwise, it is 0. ■

Theorem 6.1 (Rotation Index Theorem) *The rotation angle of a continuous simple closed plane curve is* $\pm 2\pi$*.*

A proof of the rotation index theorem can be found in differential geometry [94].

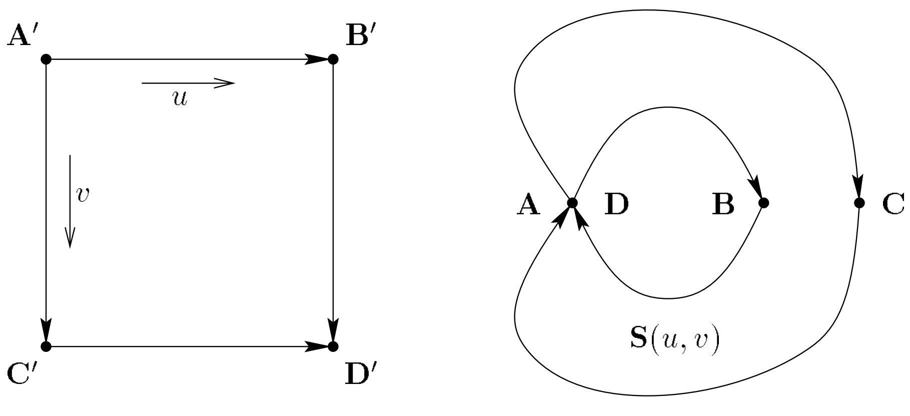
Proposition 6.9 *A self-intersecting planar surface satisfies at least one of the following conditions:*

1. *spanning angle of the smallest normal bounding cone is* π*, or*
2. *one of the two smallest tangent bounding cone spans at least* π*.*

Proof: Since the smallest bounding cones of a surface are always greater than or equal to those of a subsurface, all we need to show is that the conditions hold for any planar minimal self-intersecting surface. From Lemma 6.7, there are only two types of minimal self-intersecting surfaces. In the case of a self-intersecting isoline, condition 2 holds because of Proposition 6.5. That leaves only the diagonal corners case.

Consider a planar minimal self-intersecting surface $\mathbf{S}(u,v)$ with four corner points $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ and the corresponding parametric points $\mathbf{A}'$, $\mathbf{B}'$, $\mathbf{C}'$, and $\mathbf{D}'$ (see Figure 6.8). It is necessary to show that when condition 1 is false, i.e., when surface normals do not flip, one of the smallest isoparametric tangent bounding cones must span at least π, i.e., condition 2 is true.

Assume that $\mathbf{A}$ and $\mathbf{D}$ are the two intersecting corners ($\mathbf{B}$ and $\mathbf{C}$ intersecting is the same as reversing the surface orientation, which does not change spanning angles of the bounding cones). In Figure 6.8(b), the boundary curves $\mathbf{AB}$ and $\mathbf{BD}$ form a simple closed loop from $\mathbf{A}$ through $\mathbf{B}$ to $\mathbf{D}$, and have $\pm 2\pi$ total rotation angle. The boundary curves $\mathbf{AC}$ and $\mathbf{CD}$ also form a closed loop from $\mathbf{A}$ through $\mathbf{C}$ to $\mathbf{D}$, and have $\pm 2\pi$ total rotation angle. Since $\mathbf{S}$ intersects itself only at $\mathbf{A}$ and $\mathbf{D}$, the two loops do not cross each other in any other places. They can either be on the same side of the self-intersection points, i.e., one inside the other like one shown in Figure 6.8, or on the opposite sides of the points as two disjoint regions. Additionally, the two loops can be in the same orientation or in opposite orientation. There are only four possible configurations on a plane geometry, with the other four being mirrors.



(a) Parametric space (b) Euclidean space

Figure 6.8. Planar minimal self-intersecting surface

When condition 1 does not hold, Corollary 6.8 requires that all surface normal vectors point in the same direction. Two of the four possible configurations have surface normals at **A** and **D** in opposite directions and can be ruled out quickly. Among the remaining two cases, the one with two disjoint regions and opposite loop orientations, like the shape of 8, is also impossible. In such a configuration, isolines in same parametric direction will cross each other, which contradicts the assumption that the surface self-intersects only at **A** and **D**.

The only valid configuration has both loops in the same orientation, one inside the other and touching at points **A** and **D** (see Figure 6.8). Since $\mathbf{A} = \mathbf{D}$, the four boundary edges form a double loop **ABACA** whose total rotation angle is $\pm 4\pi$. The double loop can be broken at points **B** and **C** into **BAC** and **CAB**, each representing the two boundary isolines of the same parametric direction. Since the jump angles at **B** and **C** are at most π, sum of the spanning angles of **ABD** and **CAB** is at least $4\pi - 2\pi = 2\pi$. That means one of them has to span at least π, and therefore, the smallest isoparametric tangent bounding cone at the corresponding direction also spans at least π. ■

Proposition 6.9 gives a necessary condition for planar self-intersecting surfaces. To extend this condition to 3D surfaces, we want to project the 3D self-intersecting surface onto a plane. The projected planar surface is also self-intersecting and satisfies the conditions given above.

Definition 6.14 *An* orthographic projection *is a projection that maps points in a direction perpendicular to the projection plane. Let* $\hat{\mathbf{x}}$ *be the projection point of* $\mathbf{x}$*, and* $\mathbf{n}$ *be the unit normal vector of the projection plane. Then,*

$$\hat{\mathbf{x}} = \mathbf{x} - (\mathbf{x} \cdot \mathbf{n})\,\mathbf{n} \tag{6.10}$$

Lemma 6.10 *An orthographic projection of a direction cone* $\langle \mathbf{a}, \theta \rangle$ *has a projected spanning angle less than* π *if and only if*

$$|\mathbf{n} \cdot \mathbf{a}| < \cos \frac{\theta}{2} \tag{6.11}$$

where $\mathbf{n}$ *is the unit normal vector of the projection plane.*

Proof: Let $\hat{\mathbf{a}}$ and $\hat{\mathbf{x}}$ be projections of the axis unit vector $\mathbf{a}$ and any unit vector $\mathbf{x}$, respectively. From definition 6.14,

$$\begin{aligned}\hat{\mathbf{x}} &= \mathbf{x} - (\mathbf{x}\cdot\mathbf{n})\mathbf{n}\\ \hat{\mathbf{a}} &= \mathbf{a} - (\mathbf{a}\cdot\mathbf{n})\mathbf{n}\end{aligned}$$

We want to show that $\hat{\mathbf{x}}\cdot\hat{\mathbf{a}} > 0$ when $\mathbf{x}$ is inside the cone.

$$\begin{aligned}\hat{\mathbf{x}}\cdot\hat{\mathbf{a}} &= (\mathbf{x} - (\mathbf{x}\cdot\mathbf{n})\mathbf{n})\cdot(\mathbf{a} - (\mathbf{a}\cdot\mathbf{n})\mathbf{n})\\ &= \mathbf{x}\cdot\mathbf{a} - (\mathbf{a}\cdot\mathbf{n})(\mathbf{x}\cdot\mathbf{n}) -\\ &\quad (\mathbf{x}\cdot\mathbf{n})(\mathbf{n}\cdot\mathbf{a}) + (\mathbf{x}\cdot\mathbf{n})(\mathbf{a}\cdot\mathbf{n})(\mathbf{n}\cdot\mathbf{n})\end{aligned}$$

Since $\mathbf{n}$ and $\mathbf{x}$ are unit vectors, i.e., $\mathbf{n}\cdot\mathbf{n} = 1$ and $|\mathbf{x}\cdot\mathbf{n}| \leq 1$,

$$\begin{aligned}\hat{\mathbf{x}}\cdot\hat{\mathbf{a}} &= \mathbf{x}\cdot\mathbf{a} - (\mathbf{a}\cdot\mathbf{n})(\mathbf{x}\cdot\mathbf{n})\\ &\geq \mathbf{x}\cdot\mathbf{a} - |\mathbf{a}\cdot\mathbf{n}||\mathbf{x}\cdot\mathbf{n}|\\ &\geq \mathbf{x}\cdot\mathbf{a} - |\mathbf{a}\cdot\mathbf{n}|\end{aligned}$$

From Definition 6.9 and Equation 6.11, $\mathbf{x}\cdot\mathbf{a} \geq \cos\frac{\theta}{2} > |\mathbf{n}\cdot\mathbf{a}|$, therefore

$$\hat{\mathbf{x}}\cdot\hat{\mathbf{a}} > 0$$

∎

Corollary 6.11 *If the spanning angle of a direction cone is at least π, any orthographic projection of it also spans at least π.*

This must be true as $\theta/2$ must be less than $\pi/2$ for Equation 6.11 to be true,

Proposition 6.12 *A self-intersecting nonplanar surface $\mathbf{S}(u,v)$ satisfies at least one of the following conditions (see Equations 6.9 for notation):*

$$\theta_n \geq \pi, \tag{6.12}$$

$$|\mathbf{a}_n\cdot\mathbf{a}_u| \geq \cos\frac{\theta_u}{2}, \text{ or} \tag{6.13}$$

$$|\mathbf{a}_n\cdot\mathbf{a}_v| \geq \cos\frac{\theta_v}{2} \tag{6.14}$$

Proof: If Equation 6.12 is not true, an orthographic projection of $\mathbf{S}$ in direction $\mathbf{a}_n$ is a bijective mapping. The projected surface $\widehat{\mathbf{S}}$ is a self-intersecting planar surface whose normal vectors point in one direction. If Equations 6.13 and 6.14 are also false, from Lemma 6.10, the projected isoparametric tangent bounding cones span less than π. This contradicts Proposition 6.9 and is impossible. ■

The isoparametric tangent bounding cones of surface $\mathbf{S}(u, v)$ can be computed from the partial derivatives $\mathbf{S}_u$ and $\mathbf{S}_v$. The surface normal bounding cone can be approximated using normals of the control mesh. A tighter bound could be obtained by the cross produce of $\mathbf{S}_u$ and $\mathbf{S}_v$, which can be computed symbolically as a NURBS surface. A fast but nonoptimal method to bound any cross product between vectors from the two tangent bounding cones is given in [124]. This creates a conservative normal cone whose axis is mutually perpendicular to the axes of the isoparametric tangent bounding cones. Equation 6.12 dominates the other two conditions with such a loosely bounded normal cone.

6.4 Surface Self-Intersection Curves

From Definition 6.11, the self-intersection of a continuous surface is either empty, a list of points, a list of curves, a list of surfaces, or a combination of them [16]. We modify the divide-and-conquer subdivision method for surface/surface intersections [137] to find the surface self-intersections by incorporating the necessary conditions of Proposition 6.12. To improve the accuracy of the self-intersection curves, we modify the numerical techniques used in the marching method [13]. A new method is presented to handle self-intersections near degenerate regions of a self-intersecting surface. Since the surface/surface intersection algorithms do not support overlapping regions [13, 137], we focus mainly on finding the (possible) self-intersection curves of a continuous surface.

6.4.1 Finding Self-Intersection Curves

Definition 6.15 *A* self-intersection curve $\mathbf{C}(t)$ *of a self-intersecting surface* $\mathbf{S}(u, v)$ *is a curve in the surface consisting of only the self-intersection points (see Figure 6.6), i.e.,*

$$\forall t,\ \mathbf{C}(t) = \mathbf{S}(u_1(t), v_1(t)) = \mathbf{S}(u_2(t), v_2(t)) \tag{6.15}$$

The following divide-and-conquer algorithm computes intersections between two surfaces and can be used for surface self-intersections by passing the same surface twice.

```
ALGORITHM SrfInter( Srf1, Srf2, Tolerance )
BEGIN
    If Srf1 and Srf2 are the same surface and cannot
            self-intersect, return NULL.
    If Srf1 and Srf2 are neighbors and cannot intersect
            elsewhere besides the common boundary, return NULL.
    Otherwise, if the bounding boxes of Srf1 and Srf2
            do not intersect, return NULL.
    Subdivide Srf1 and Srf2 if not flat within Tolerance.
    If either surface is subdivided, return the intersections
            and self-intersections found from the recursive
            calls of SrfInter on the subdivided subsurfaces.
    If Srf1 and Srf2 are the same surface, return NULL.
    Return intersections of Srf1 and Srf2.
END
```

The algorithm uses surface bounding cones and Proposition 6.12 to test whether a surface can self-intersect or not. For neighboring surfaces which share (part of) a common boundary, the combined bounding cones are used to see if there can be intersections at other than the common boundary.

Figure 6.9 shows results of the adaptive subdivision surface self-intersection algorithm applied on examples from Figure 6.1. On the left are the Euclidean space self-intersection curves and subdivision lines. On the right are the corresponding curves in parametric space. The dotted lines indicate subdivision locations and directions. A surface is subdivided whenever an intersection is possible, with itself or with another surface, until the surface is flat or small enough to within a coarse initial tolerance. The two parametric self-intersection curve pair, μ_1 and μ_2, are shown in solid and dashed lines, respectively.

In Figure 6.9(b), the surface has a seam along its top and bottom edges. A topological adjacency has been declared in the surface model. In addition to subdivision lines, a priori adjacency information can be used as part of neighboring surfaces

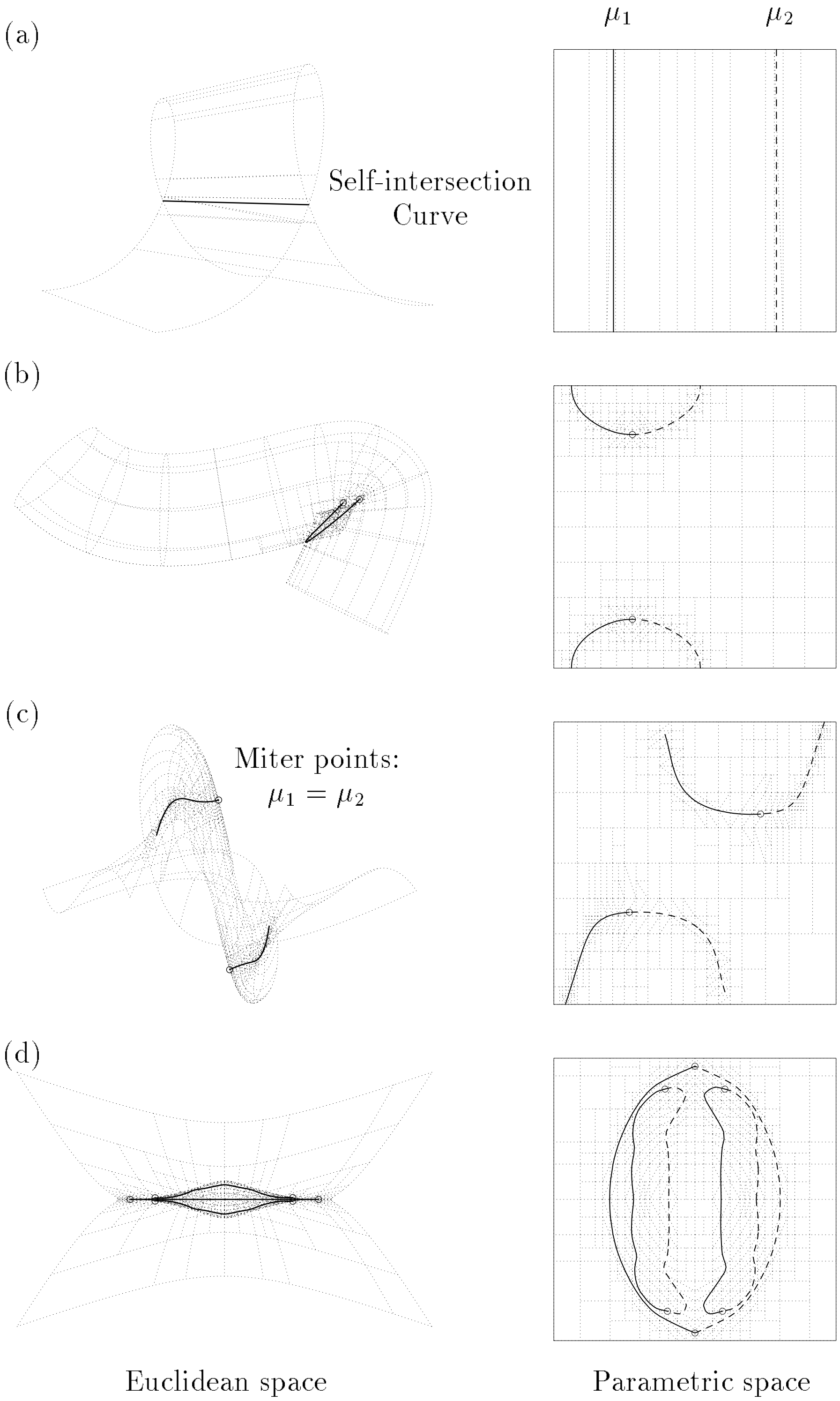


Figure 6.9. Adaptive subdivision for surface self-intersection

detection. Figure 6.9(c) shows a self-intersecting surface, none of whose isolines self-intersect. Thus, we see it would be insufficient to design a self-intersection algorithm based on self-intersecting isolines. Figure 6.9(d) shows three self-intersection curves found in an offset surface. Two of them are from the imperfect cusps of the offset approximation. If the original surface were available, intersections around these regions could be avoided by detecting flips of tangent/normal vectors between the surface and its offset [42].

6.4.2 Self-Intersection Improvement

We use a numerical method based on successive normal projection to tangent planes to improve accuracy of the self-intersection points. Most numerical surface/surface intersection algorithms require the computation of nondegenerate surface normal vectors at intersection points [13, 65]. However, degenerate surface normals frequently occur in self-intersecting surfaces, especially for offset surfaces, so those methods fail. In this research, a new numerical method is developed to handle this kind of special case.

In Figure 6.9, some of the self-intersection curves have circles marked on their end points. From the corresponding parametric space drawings, these are the junction points where parametric self-intersection curve pair join together, i.e., $\mu_1 = \mu_2$.

Definition 6.16 *A* miter point *is an end point of a surface self-intersection curve where $u_1 = u_2$ and $v_1 = v_2$.*

Strictly speaking, miter points are not self-intersection points. A self-intersection curve with miter points is a curve whose domain is defined on an open, or semi-open interval.

Miter points occur frequently in sweep surfaces along an elbow bend or collapsed offset surfaces. However, to pinpoint the exact location of a miter point is not easy. Approaching from one side of a miter point, the surface is not self-intersecting. Approaching from the other side of a miter point, the self-intersection curve vanishes when $\mu_1 \to \mu_2$. Existing numerical methods for surfaces intersection require distinct surface normals at intersection points for intersection computation [97]. If marching is used along a self-intersection curve towards its miter point, any overshoot will converge to a single surface point on the other side of the miter point. There is no easy way to distinguish between them. To closely approximate a miter point, the step size must be very small. This gives slow convergence, at best.

In the following, a necessary condition is established to numerically improve the accuracy of a miter point. The self-intersecting surface is assumed to be C^1 near the miter point.

Corollary 6.13 *Let* $\mathbf{p}$ *be a miter point of a self-intersecting surface* $\mathbf{S}$*. Then every neighborhood of* $\mathbf{p}$ *in* $\mathbf{S}$ *is also a self-intersecting surface.*

Proof: Since a miter point is at the edge of the open interval domain of a self-intersection curve, any neighborhood of $\mathbf{p}$ should also contain part of the self-intersection curve, therefore, is self-intersecting. ∎

This means that unless a subdivision occurs right on the miter point in a direction to separate the two parametric self-intersection curves, i.e., μ_1 and μ_2, a subdivided surface containing a miter point will always pass the self-intersection condition test. To prevent an infinite loop, the subdivision should stop when a surface is too small. A surface smaller than the initial subdivision tolerance is considered to be "flat."

Lemma 6.14 *A self-intersecting surface is not regular at its miter points.*

Proof: Any arbitrarily small neighborhood of a miter point is self-intersecting and satisfies the necessary conditions given in Proposition 6.12. In other words, no unique surface normal can be defined around a miter point. Therefore, the surface is not regular at miter points. ∎

Proposition 6.15 *At a miter point of a* C^1 *surface* $\mathbf{S}(u, v)$*,*

$$\left\| \frac{\partial \mathbf{S}}{\partial u} \times \frac{\partial \mathbf{S}}{\partial v} \right\| = 0 \tag{6.16}$$

Proof: This must be true since the surface is not regular at miter points. For a C^1 surface, the cross product of its partial derivatives equals zero at those points where it is not regular. ■

The condition given above is only a necessary condition for miter points. To verify that a point satisfying the condition is actually a miter point, a self-intersection curve must be traced and end at the miter point. Using this property, potential miter points are located by finding zeros of surface normals, i.e., $\hat{\mathbf{n}}(u,v) = \mathbf{S}_u(u,v) \times \mathbf{S}_v(u,v) = \mathbf{0}$. The iteration starts when a possible miter point is found. Figure 6.10(a) shows subdivided triangles of the surface in (c) using a rather coarse subdivision tolerance, $\epsilon = 0.1$. Six self-intersection points were found in this example. Self-intersection lines and their parameters are shown in thick lines, solid and dashed.

One of the self-intersection points has same parameters and is a candidate for miter point numerical improvement. The miter point improvement algorithm converges relatively fast. Figure 6.10(b) shows the results from numerical improvement of self-intersection points, including the miter point, using a point tolerance $\epsilon = 10^{-5}$. In Figure 6.10(c), the self-intersection curve is adaptively refined until the curve is within a finer tolerance, $\epsilon = 0.001$, to the surface.

6.5 Self-Intersection Classification

Trimmed surfaces can be used to obtain a solid model for self-intersecting surfaces. Boolean set operations [137], union and intersection (see Figure 6.11), are used in surface self-intersection trimming. We use a general approach to trimmed surface construction in this research to trim away surface self-intersections, which also works for trimming surface/surface intersections.

Our trimming procedure starts by intersecting and splitting all intersection curves of a surface and its existing trimming curves, or boundaries if untrimmed, in the parametric space. This can be done in $O(n \log n)$ time using a sweep algorithm [113]. A 2D graph is then constructed to describe the connectivities of intersection and trimming curves on a surface. In the graph, an edge represents

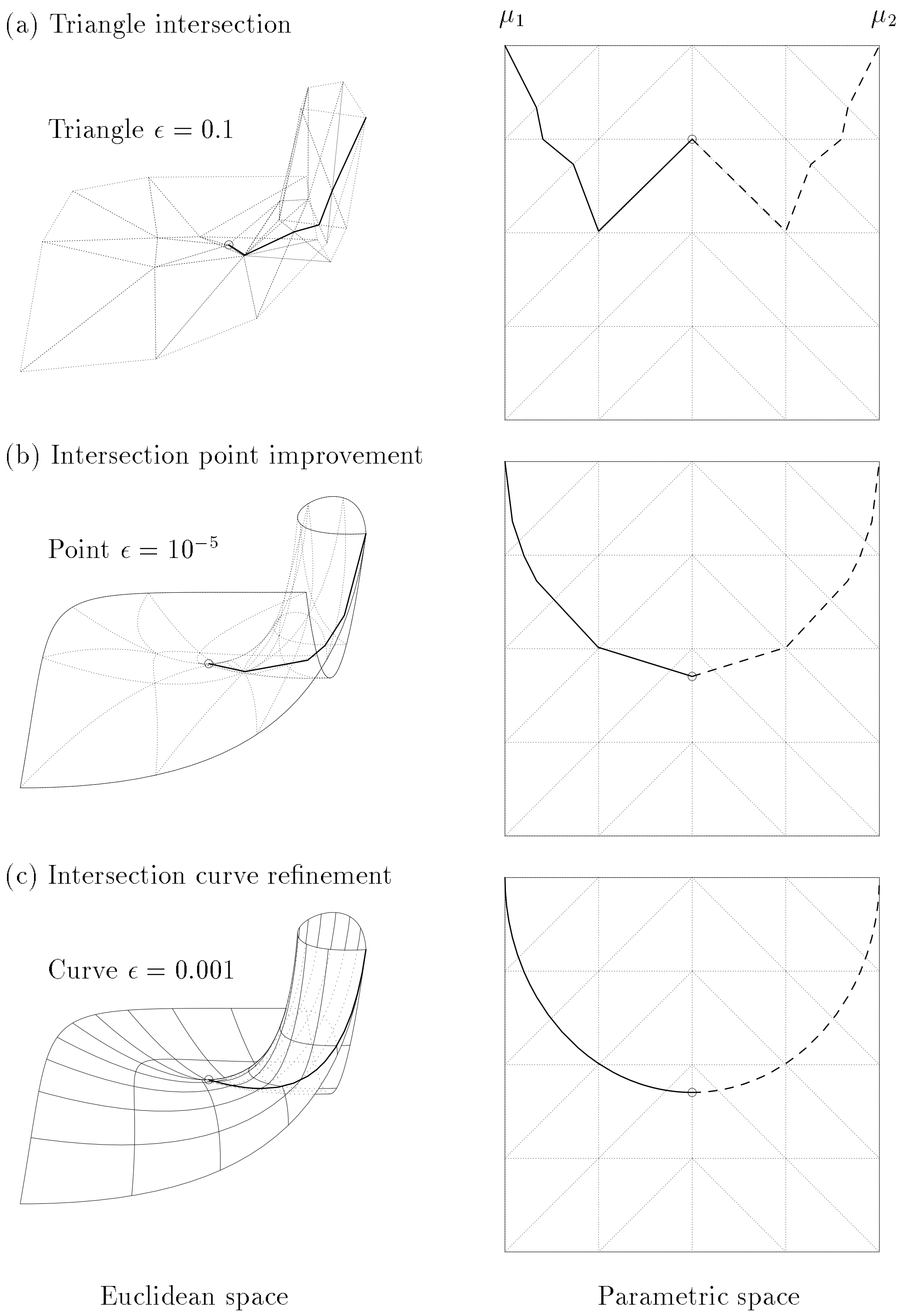
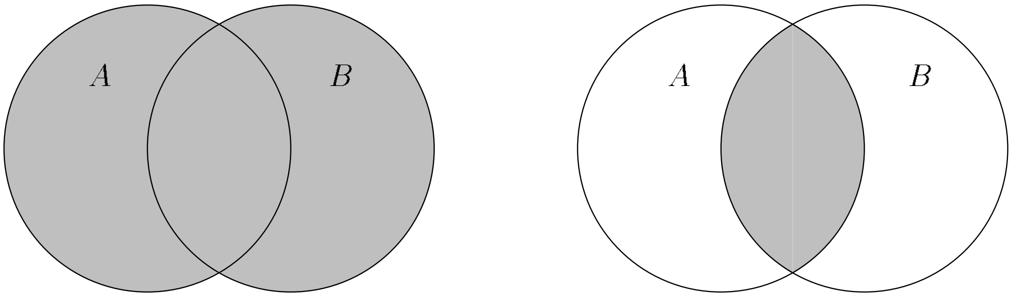


Figure 6.10. Miter point of surface self-intersection



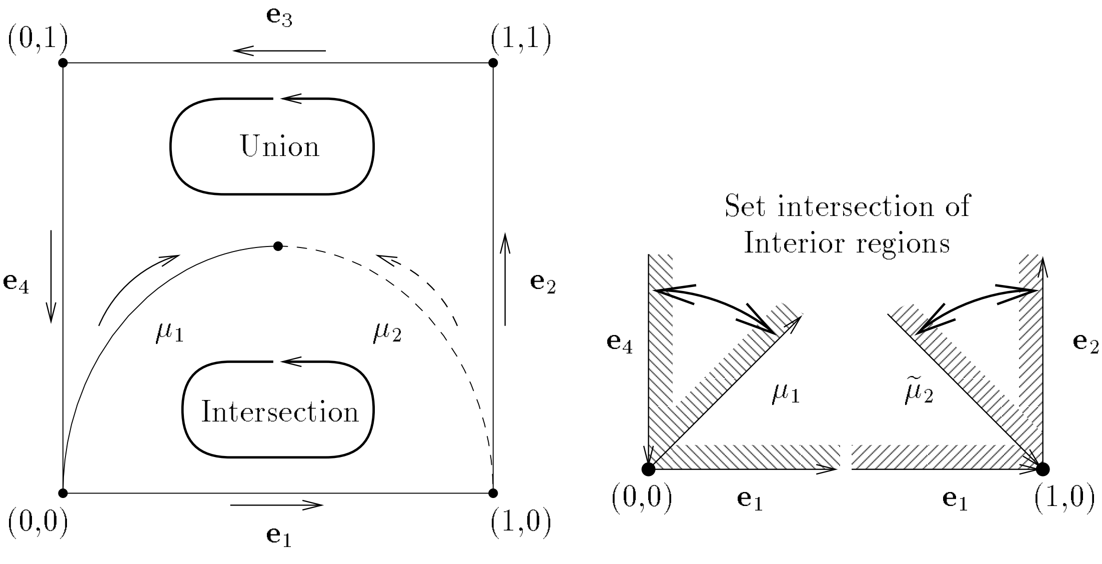
(a) Set union, $A \cup B$
$= \{x | x \in A \vee x \in B\}$

(b) Set intersection, $A \cap B$
$= \{x | x \in A \wedge x \in B\}$

Figure 6.11. Boolean set operations

a simple intersection/trimming curve segment in parametric space, and a node represents a joint of multiple segments. Figure 6.12(a) shows a connectivity graph for the self-intersecting surface shown in Figure 6.10. The graph has five nodes and six edges. Two of the nodes have three edges connecting to them (see Figure 6.12(b)). Edge $\widetilde{\mu}_2$ represents the parametric curve μ_2 (of the self-intersection curve) in reverse orientation.

Figure 6.13 shows the resulting trimmed surfaces for examples from Figure 6.9. Note that two of the end points of the parametric self-intersection curves in Fig-



(a) Parametric trimming loops

(b) Node classification

Figure 6.12. Classification of surface self-intersection curves

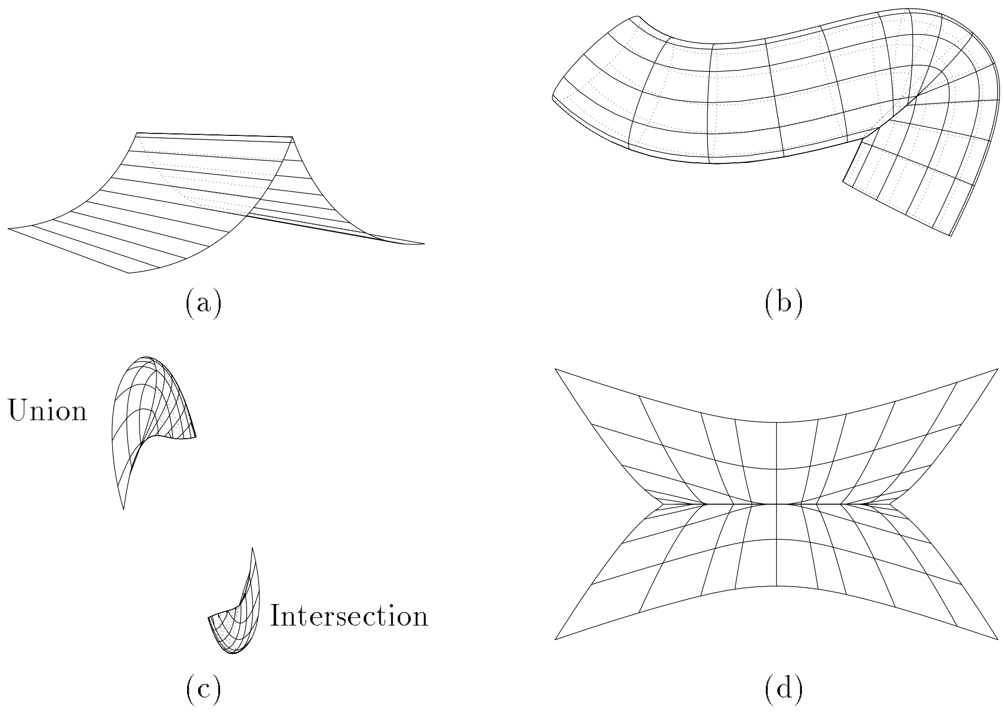


Figure 6.13. Trimmed self-intersecting surfaces

ure 6.9(c) do not touch boundary curves of the surface. In this case, no valid loop can be found from the self-intersection curves. To construct the trimmed surfaces shown in Figure 6.13(c), G^1 extensions of the original surface were made around the four boundaries first. Then, intersections between the surface and the four extension patches were computed, which extend the self-intersection curves to the surface boundary. Results from both set union and set intersection operations are shown as two separate regions.

6.5.1 3D Classification

The trimmed regions obtained from intersection curve classification do not always form a single connected 3D surface as in the above examples. A 3D volume classification, similar to the 2D loop classification, is needed when there is more than one connected surface. Figure 6.14 shows trimming of self-intersecting sweep surfaces of circular cross-sections. The sweep operation is performed using the adaptive polygon offset method presented in Section 5.7 (see Figure 5.25). On the

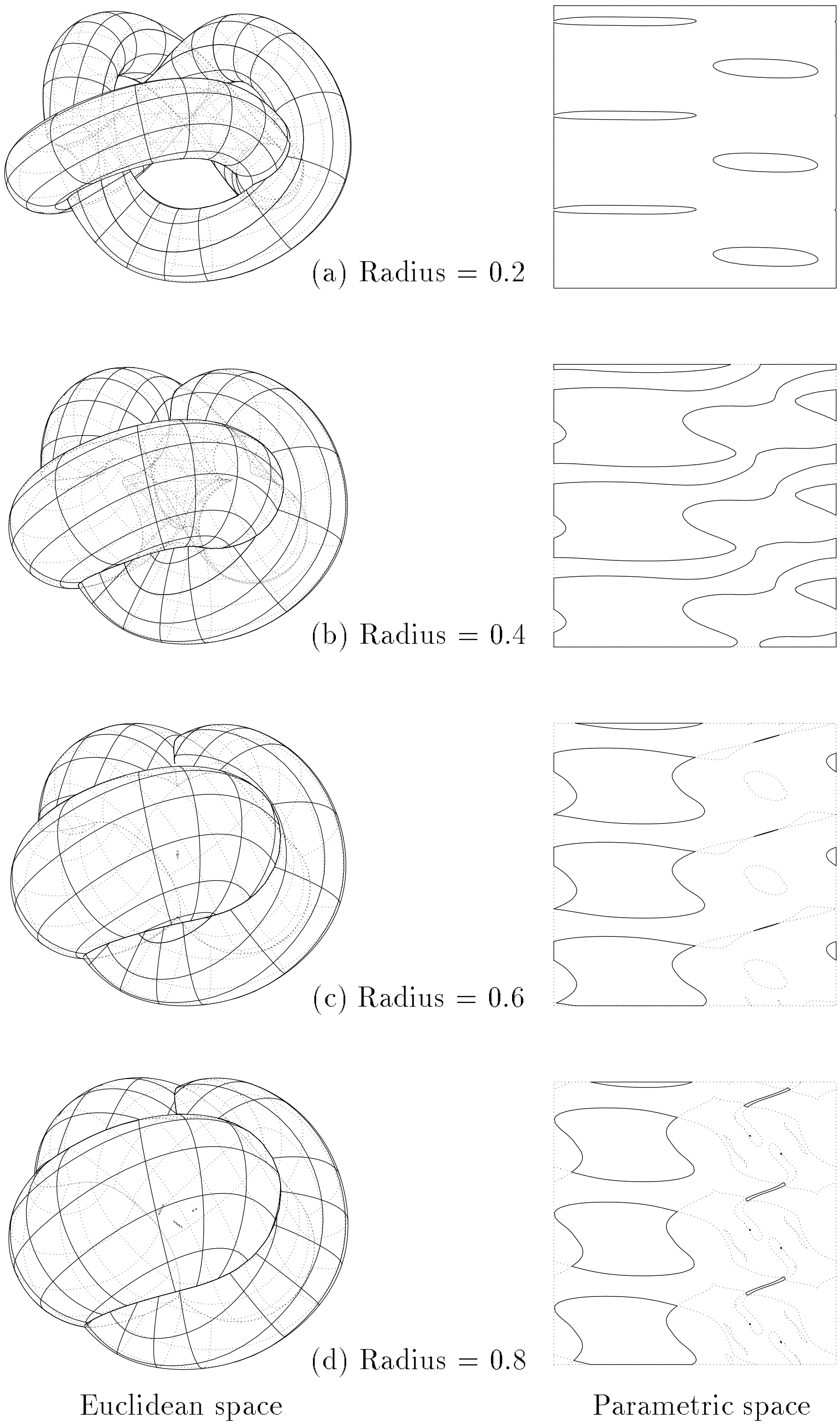


Figure 6.14. Self-intersection trimming of sweep surfaces

left, the four Euclidean space surfaces are scaled differently to fit in the same space. On the right, the dotted lines are the untrimmed parametric self-intersection curves. The classified trimming loops are shown in solid lines.

In Figure 6.14(a), there is only one trimming loop which corresponds to one connected surface to enclose a solid volume. In (b), there are eight trimming loops that also form one connected surface and volume. However, in (c), there are ten trimming loops, three of which form a tiny void, i.e., negative volume, around the center. Therefore, the trimmed surface forms two disconnected surfaces to represent a solid with a void. In (d), there are 13 loops, of which, nine loops form three tiny voids in the middle. Judging from the cross-section radius and axis geometry, unlike (c), these three voids contain points that are closer to the axis curve than the radius of the sweeping circle. If the model should represent a solid that is constant distance to the axis curve, these voids should be removed.

Unfortunately, 2D topology of the surface representation provides no useful information for this kind of 3D global classification. After all, a solid with voids is still valid. In some applications, such as NC machining, voids are not desired. A 3D solid classification can be done by computing the enclosing volume and can be used to remove unwanted regions. For open surfaces that do not enclose a solid volume, connected surfaces with small area can be filtered out instead. Table 6.1 lists surface area and enclosing volume of the connected regions in Figures 6.14(c) and (d). It is clear, only one significant connected region exists in both cases, i.e., the outer boundaries.

6.6 Offsets and Self-Intersection

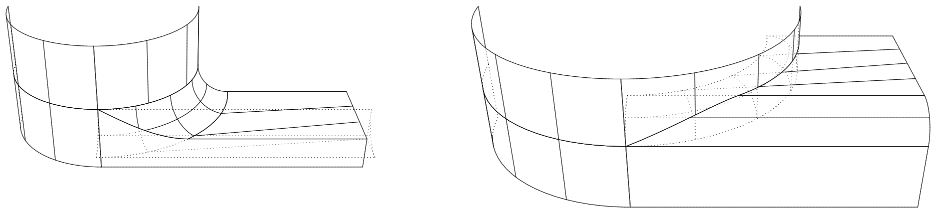
The surface self-intersection finding and trimming algorithms can also be applied to solid models of multiple trimmed surfaces. First, intersections and self-

Table 6.1. Area and volume of connected regions

	Radius = 0.6		Radius = 0.8			
Area	19.5909	0.0030	26.1949	0.0039	0.0038	0.0034
Volume	7.2534	-5.148e-6	11.7658	-6.032e-6	-6.032e-6	-7.543e-6

intersections of surfaces are computed. Then, intersection curve classification algorithms are applied to each surface. Figure 6.15 shows the intermediate offsets and self-intersection trimming for the rounding and filleting operations shown in Figure 5.27. Rounding/filleting operation can also be applied on the same surface. Figure 6.16 shows examples of a "self-filleting" operation. Note that self-intersecting surfaces with miter points are degenerate and may not be offset correctly.

Figure 6.17 shows an example of using offset surface and self-intersection trimming techniques to compute the machinable regions of a surface, which is important in process planning and gouge-free tool path generation for 3D milling features. Figures 6.17(a) and (b) show an open surface and its offset surface. The self-intersection curves of the offset surface indicate that two of the open edges do not self-intersect. To compute the constant distance offset surface, offsets of the



(a) Rounding (b) Filleting

Figure 6.15. Offsets and self-intersections for solid modeling

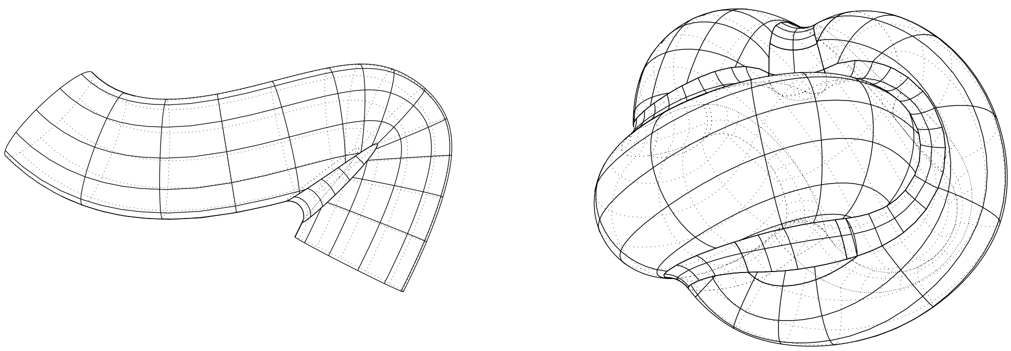
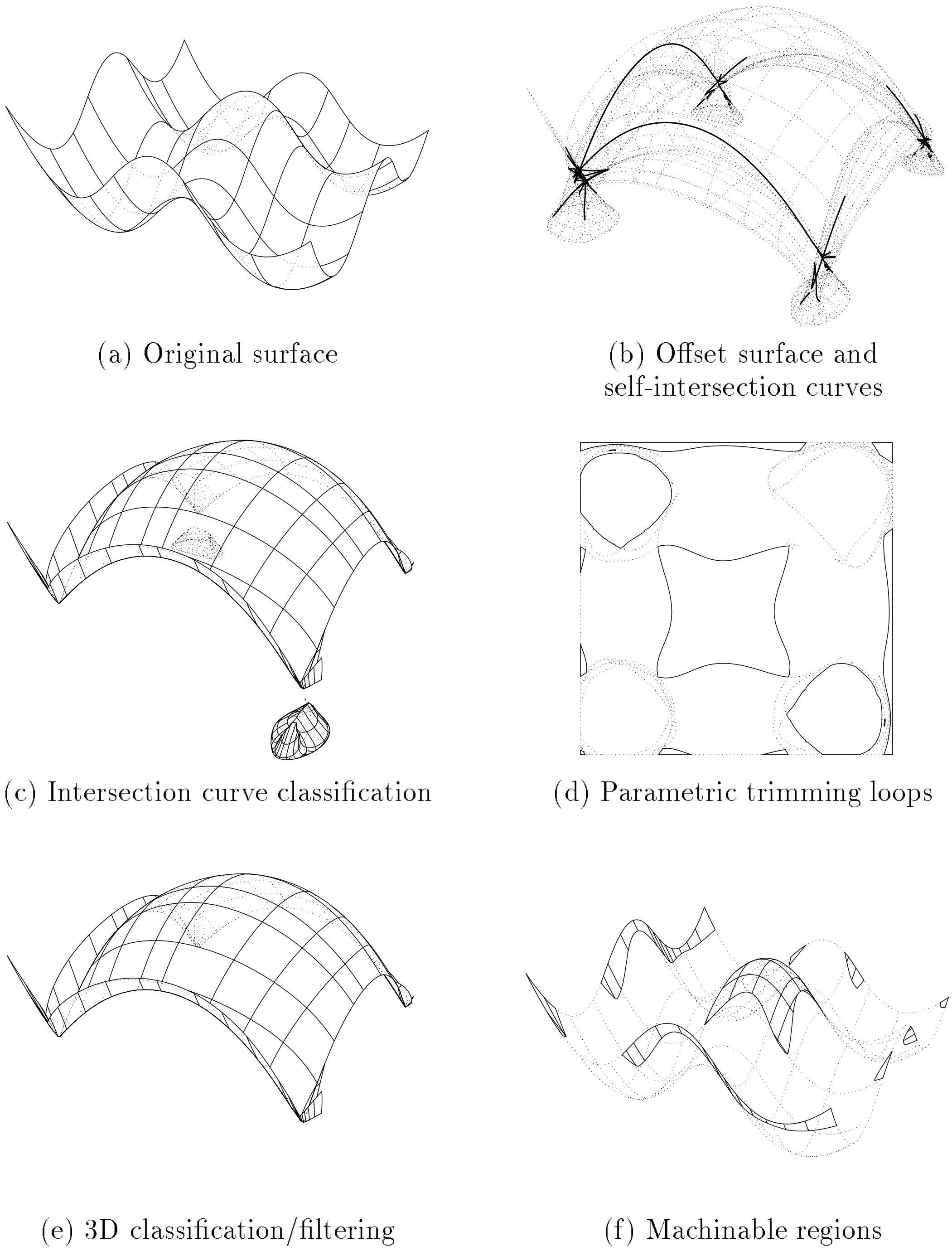


Figure 6.16. Self-filleting operation



(a) Original surface

(b) Offset surface and self-intersection curves

(c) Intersection curve classification

(d) Parametric trimming loops

(e) 3D classification/filtering

(f) Machinable regions

Figure 6.17. Self-intersection trimming of offset surfaces

open boundary are computed to trim against the self-intersecting offset surface. Figure 6.17(c) shows the result of the self-intersection classification. The parametric space self-intersection curves and trimming loops are shown in Figure 6.17(d). The trimmed surface has thirteen trimming loops and five connected regions. Table 6.2 lists the surface area and (open) volume of the connected regions. A 3D classification based on connected surface area filtering is applied to the trimmed surface.

Figure 6.17(e) shows the result, which is a constant distance offset surface of (a). By mapping the trimming loops of the constant distance offset surface back to the original surface, Figure 6.17(e) shows the regions that are machinable by a ball-end tool with radius equal to the offset distance, provided that the regions are accessible by the tool. Area of the trimmed surface is the cumulative cut area $\widehat{A}$ of this tool, as defined in Section 4.4. By repeating the same procedure at different offset distances, optimal tool-driven decomposition and multiple tool selection for the 3D surface feature can be computed (see Chapter 4).

Table 6.2. Area and volume for 3D classification

	Connected regions				
Area	2.2228	0.1175	0.1181	1.763e-6	3.548e-6
Volume	0.5663	-0.0027	-0.0026	1.654e-7	-1.156e-11

CHAPTER 7

CONCLUSIONS

In this thesis, we demonstrate a feature-based design/manufacturing system that can handle parts of complex shapes including freeform surfaces, a requirement of practical use in many applications. Key components of our feature-based approach to process planning and NC machining include feature decomposition, tool selection and feed/speed computation, operations reordering, and NC code generation. Currently, our implementation relies on the user to provide fixture and interfeature information. Therefore, it only works as a semiautomatic process planning system. However, the NC part programming is automatic and can fulfill an immediate need in the integration of concurrent design and manufacturing. It is our hope that with more and more geometric analysis tools and algorithms on interfeature interactions and dependencies, such a system could one day become fully automated.

7.1 Contributions

Several new inventions and discoveries are presented throughout this thesis. The generalization of skeleton computation to handle outside regions and regions defined by open profiles is a direct extension of the merge skeleton algorithm. The fast 2D offset tool path clipping algorithm presented in Section 3.7 not only is used for the open pocket machining but is also used in generating efficient tool paths using multiple tools.

We introduce the tool cut index in Section 4.4 to quantify the effectiveness of milling tools, which can be computed from the feature geometry and tool diameter. Efficient algorithms to compute the cut index and to minimize the total cut index for selecting multiple tools are presented in Chapter 4. With the help of offset surface computation and self-intersection trimming, we can machine 3D features using ball-end tools of different diameters for more efficient roughing and finishing.

We present a hybrid offset algorithm to approximate offsets of NURBS curves and surfaces to within a prescribed tolerance, including offsets of C^1 discontinuities. We transform the offset surface refinement problem into a minimum-cover problem and present an efficient adaptive offset surface refinement algorithm that greatly reduced the inserted new knots or new control points. We introduce a new method to model arbitrary spherical patches bounded by arcs of great circles. Using the inverse stereographic mapping and trimmed surface representation, a method to model such a spherical polygon as a biquadratic NURBS surface with quadratic trimming curves is presented in Section 5.6. We demonstrate many modeling examples that use the constant distance offset operation.

We define self-intersection as a global intrinsic property of curves and surfaces in Chapter 6. Necessary conditions for self-intersecting curves and surfaces are presented and mathematically proved. Using these conditions, we can detect whether a surface or a curve may be self-intersecting from its tangent and normal bounding cones. Previously, surface self-intersection detection and finding were considered to be unsolved problems, the answers to which are crucial in computing the constant distance offsets for a gouge-free tool path of freeform surfaces as well as in many solid modeling operations.

7.2 Future Research

A good algorithm is one that robustly and efficiently handles all conditions and all special cases. In this research, we investigate several problems of geometric analysis and machining planning and provide solutions to some of them. Our approaches to the problem of offsets and self-intersection are influenced mainly by the needs of our application and many other applications to have a constant distance offset of curves and surfaces. Without an efficient method to compute and trim surface self-intersections, the use of offset surfaces is somewhat limited.

Of course, there is always room for improvements, and we only provide limited solutions for a problem that spans many different fields. Below are some improvements and directions for future research:

- We need better algorithms to determine the relationship between features. Currently, we use simple rules to decide the partial ordering of operations. An analysis of the overlapping regions between features would be a start for a more intelligent operations reordering.

- We need to investigate and use 5-axis machining methods for parts that cannot be machined using fixed-axis machining. Although our feature-based system and offset/self-intersection algorithms do not impose any restriction for 5-axis machining, our tool path generation and decomposition strategies for 3D surfaces are mainly for 3-axis machining using ball-end tools. Various 5-axis tool positioning and tool path generation algorithms can be found in the literature [29, 30, 43, 44, 73, 84, 90]. For continuous 5-axis machining, the use of flat-end tools and global gouge-prevention have to be investigated too.

- We could use different forms to define the tool cut index. In this thesis, we use an area-based definition, i.e., the total cut area divided by the cutter cross-section area. One of the reasons to choose this form is because our model of feed computation which is linearly proportional to the tool diameter. A more reasonable feed rate model needs to take the depth of cut into account. We could use a volume-based cut index definition that is the ratio of total cut volume to the effective tool cut volume. Another possibility is to perform a dimensionless analysis, then conduct a series of experiments to determine the coefficients of an empirical equation.

- We could improve the efficiency of self-intersection trimming for offset surfaces. One possibility is to find and classify self-intersection curves adaptively.

REFERENCES

[1] 3D SYSTEMS, INC. *SLA-250 User Reference Manual*, software release 3.60 ed., September 1989.

[2] 3D SYSTEMS, INC. *StereoLithography Interface Specification*. Valencia, California, October 1989.

[3] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *Data Structures and Algorithms*. Computer Science and Information Processing. Addison-Wesley Publishing Company, Reading, Massachusetts, 1983.

[4] ALPHA_1 RESEARCH GROUP. *Alpha_1 Developer's Handbook*. University of Utah, Salt Lake City, Utah, June 1992.

[5] ALPHA_1 RESEARCH GROUP. *Alpha_1 User's Manual*. University of Utah, Salt Lake City, Utah, June 1992.

[6] AOMURA, S., AND UEHARA, T. Self-intersection of an offset surface. *Computer-Aided Design 22*, 7 (September 1990), 417–422.

[7] ARNER, P. R. *Another Look at Surface/Surface Intersection*. PhD thesis, University of Utah, Salt Lake City, Utah, 1987.

[8] ASHLEY, S. Rapid prototyping systems. *Mechanical Engineering* (April 1991), 34–43.

[9] AUSTIN, M. M. Reducing stereolithography processing times by creating hollow models. Master's thesis, Brigham Young University, March 1992.

[10] BAJAJ, C., HOFFMANN, C., LYNCH, R., AND HOPCROFT, J. Tracing surface intersections. *Computer Aided Geometric Design 5* (1988), 285–307.

[11] BALLARD, D. H., AND BROWN, C. M. *Computer Vision*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[12] BANCHOFF, T., GAFFNEY, T., AND MCCRORY, C. *Cusps of Gauss Mappings*. Pitman Books Limited, London, UK, 1982.

[13] BARNHILL, R., FARIN, G., JORDAN, M., AND PIPER, B. Surface/surface intersection. *Computer Aided Geometric Design 4* (1987), 3–16.

[14] BARNHILL, R., AND KERSEY, S. A marching method for parametric surface/surface intersection. *Computer Aided Geometric Design 7* (1990),

257–280.

[15] BARNHILL, R. E., FARIN, G. E., AND CHEN, Q. Constant-radius blending of parametric surfaces. In *Geometric Modelling*, G. Farin, H. Hagen, H. Noltemeier, and W. Knödel, Eds., vol. 8 of *Computing Supplementum*. Springer-Verlag/Wien, New York, New York, 1993, pp. 1–20.

[16] BARNHILL, R. E., FROST, T. M., AND KERSEY, S. N. Self-intersections and offset surfaces. In *Geometry Processing for Design and Manufacturing*, R. E. Barnhill, Ed. Society for Industrial and Applied Mathematics, Philadelphia, 1992, ch. 2, pp. 35–44.

[17] BERGER, D., AND KALLAY, M. Computing the unit normal on a degenerate edge. *Computer Aided Design 24*, 7 (July 1992), 395–396.

[18] BLOOMENTHAL, M., AND RIESENFELD, R. Approximation of sweep surfaces by tensor product NURBS. In *Curves and Surfaces in Computer Vision and Graphics II* (Boston, MA, November 1991), vol. 1610, SPIE, pp. 132–144.

[19] BOLTON, K. Biarc curves. *Computer Aided Design 7*, 2 (April 1975), 89–92.

[20] BROWN, D. R., CUTKOSKY, M. R., AND TENENBAUM, J. M. Next-Cut: A second generation framework for concurrent engineering. In *Computer Aided Cooperative Product Development*, D. Sriram and R. Logcher, Eds. Springer-Verlag, 1991.

[21] BURGAM, P. M. Two tools for the eighties: Machining centers and adaptive control. *Manufacturing Engineering 92*, 5 (May 1984), 56–60.

[22] CAMPBELL, J. *Introductory Cartography*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984, ch. 2.

[23] CHANG, T.-C., AND WYSK, R. A. *An Introduction to Automated Process Planning Systems*. Industrial and Systems Engineering. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[24] CHEN, X., AND HOFFMANN, C. M. On editability of feature-based design. *Computer-Aided Design 27*, 12 (December 1995), 905–914.

[25] CHILDS, J. J. *Numerical Control Part Programming*. Industrial Press Inc., New York, New York, 1973.

[26] CHILDS, J. J. *Principles of Numerical Control*, third ed. Industrial Press Inc., New York, New York, 1982.

[27] CHO, B. S. Rule based process planning system for hole feature machining. Master's thesis, University of Utah, Salt Lake City, Utah, August 1989.

[28] CHOI, B., BARASH, M., AND ANDERSON, D. Automated recognition of machined surfaces from a 3D solid modeler. *Computer Aided Design 16*, 2

(1984).

[29] Choi, B., Park, J., and Jun, C. Cutter-location data optimization in 5-axis surface machining. *Computer Aided Design 25*, 6 (June 1993), 377–386.

[30] Chou, J. J. *NC Milling Machine Toolpath Generation for Regions Bounded by Free Form Curves and Surfaces.* PhD thesis, University of Utah, Salt Lake City, Utah, April 1989.

[31] Cobb, E. S. *Design of Sculptured Surfaces Using the B-spline Representation.* PhD thesis, University of Utah, Salt Lake City, Utah 84112, June 1984.

[32] Cobb, J. E. Tiling the sphere with rational Bezier patches. Tech. Rep. UUCS-88-009, Department of Computer Science, University of Utah, Salt Lake City, Utah 84112, July 1988.

[33] Cohen, E., Lyche, T., and Riesenfeld, R. F. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing 14*, 2 (October 1980), 87–111.

[34] Coquillart, S. Computing offsets of B-spline curves. *Computer Aided Design 19*, 6 (July/August 1987), 305–309.

[35] Cutkosky, M. R., Brown, D. R., and Tenenbaum, J. M. Extending concurrent product and process design. In *ASME WAM Symposium on Concurrent Product and Process Design* (1989).

[36] Cutkosky, M. R., and Tenenbaum, J. M. A methodology and computational framework for concurrent product and process design. *Mechanism and Machine Theory 25*, 3 (June 1990), 365–381.

[37] de Boor, C. *A Practical Guide to Splines.* Applied Mathematical Sciences. Springer-Verlag, New York, New York, 1978.

[38] Dong, J., and Vijayan, S. Manufacturing feature determination and extraction — part I: Optimal volume segmentation. *Computer-Aided Design 29*, 6 (June 1997), 427–440.

[39] Drake, S., and Sela, S. A foundation for features. *Mechanical Engineering 111*, 1 (January 1989), 66–73.

[40] Driskill, E. E. *Towards the Design, Analysis, and Illustration of Assemblies.* PhD thesis, University of Utah, 1997.

[41] El Wakil, S. D. *Processes and Design for Manufacturing.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[42] Elber, G. *Free Form Surface Analysis Using a Hybrid of Symbolic and*

Numeric Computation. PhD thesis, University of Utah, Salt Lake City, UT, December 1992.

[43] ELBER, G. Accessibility in 5-axis milling environment. *Computer-Aided Design 26*, 11 (November 1994), 796–802.

[44] ELBER, G. Freeform surface region optimization for 3-axis and 5-axis milling. *Computer-Aided Design 27*, 6 (June 1995), 465–470.

[45] ELBER, G., LEE, I.-K., AND KIM, M.-S. Comparing offset curve approximation methods. *IEEE Computer Graphics and Applications* (May—June 1997), 62–71.

[46] FAROUKI, R. The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design 3*, 1 (May 1986), 15–43.

[47] FAROUKI, R., AND NEFF, C. Analytic properties of plane offset curves. *Computer Aided Geometric Design 7*, 1–4 (1990), 83–99.

[48] FAROUKI, R. T., AND SVERRISSON, R. Approximation of rolling-ball blends for free-form parametric surfaces. *Computer-Aided Design 28*, 11 (November 1996), 871–878.

[49] FERREIRA, J., AND HINDUJA, S. Convex hull-based feature recognition method for 2.5D components. *Computer-Aided Design 22*, 1 (January/February 1990), 41–49.

[50] FOLEY, J. D., AND VAN DAM, A. *Fundamentals of Interactive Computer Graphics*, second ed. Addison-Wesley Publishing Company, Reading, Massachusetts, 1990.

[51] FORTUNE, S. A sweepline algorithm for Voronoi diagrams. In *Proceedings of Second ACM Symposium on Computational Geometry* (1986), ACM, pp. 313–322.

[52] FRENCH, T. E., AND VIERCK, C. J. *A Manual of Engineering Drawing for Student and Draftsman*, ninth ed. The Engineering Drawing Series. McGraw-Hill Book Company, Inc., New York, New York, 1960.

[53] GADH, R., AND PRINZ, F. Recognition of geometric forms using the differential depth filters. *Computer Aided Design 24*, 11 (November 1922), 583–598.

[54] GALLAGHER, C. C., AND KNIGHT, W. A. *Group Technology Production Methods in Manufacture*. Halsted Press, New York, New York, 1986.

[55] GATELMAND, C. D. A survey of flexible manufacturing systems. *Journal of Manufacturing Systems 1*, 1 (1982), 1–16.

[56] GROOVER, M. P. *Automation, Production Systems, and Computer Inte-*

grated Manufacturing. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[57] Groover, M. P., and Zimmers, Jr., E. W. *CAD/CAM Computer-Aided Design and Manufacturing.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[58] Hansen, A., and Arbab, F. Fixed-axis tool positioning with built-in global interference checking for NC path generation. *IEEE Journal of Robotics and Automation 4*, 6 (December 1988), 610–621.

[59] Hansen, A., and Arbab, F. An algorithm for generating NC tool paths for arbitrarily shaped pockets with islands. *ACM Transactions on Graphics 11*, 2 (April 1992), 152–182.

[60] Hartley, J. *FMS at Work.* IFS (Publications) Ltd., Bedford, UK, and North-Holland, Amsterdam, Netherlands, 1984.

[61] Held, M. *On the Computational Geometry of Pocket Machining*, vol. 500 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, Germany, 1991.

[62] Hoffmann, C. M., and Hopcroft, J. E. Geometric ambiguities in boundary representations. *Computer Aided Design 19*, 3 (April 1987), 141–147.

[63] Hoschek, J. Spline approximation of offset curves. *Computer Aided Geometric Design 20*, 8 (October 1988), 33–40.

[64] Hoschek, J., and Wissel, N. Optimal approximate conversion of spline curves and spline approximation of offset curves. *Computer-Aided Design 20*, 8 (October 1988), 475–483.

[65] Houghton, E. G., Emnett, R. F., Factor, J. D., and Sabharwal, C. L. Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Computer Aided Geometric Design 2* (1985), 173–183.

[66] Hwang, J. Interference-free tool-path generation in the NC machining of parametric compound surfaces. *Computer Aided Design 24*, 12 (December 1992), 667–676.

[67] Jacobs, P. F., and Richter, J. Advances in stereolithography accuracy. Tech. rep., 3D Systems, Inc., Valencia, California, August 1991.

[68] Jerard, R. B., Angleton, J. M., and Drysdale, R. L. Sculptured surface tool path generation with global interference checking. In *Design Productivity Conference* (Honolulu, Hawaii, February 1991).

[69] Jerard, R. B., Angleton, J. M., Drysdale, R. L., and Su, P. The use of surface points sets for generation, simulation, verification, and automatic correction of NC machining programs. In *Proceedings of NSF*

Design and Manufacturing Systems Conference (Tempe, Arizona, January 1990), National Science Foundation, Society of Manufacturing Engineers, pp. 143–148.

[70] Jerard, R. B., Drysdale, R. L., Kauck, K., Schaudt, B., and Magewick, J. Methods for detecting errors in numerically controlled machining of sculptured surfaces. *IEEE Computer Graphics & Applications* (January 1989), 26–39.

[71] Joshi, S., and Chang, T. Graph based heuristics for recognition of machined features from a 3D solid model. *Computer Aided Design 20*, 2 (1988).

[72] Karinthi, R. R., and Nau, D. An algebraic approach to feature interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 4 (April 1992), 469–484.

[73] Kim, D.-S., Papalambros, P. Y., and Woo, T. C. Tangent, normal, and visibility cones on Bézier surfaces. *Computer Aided Geometric Design 12*, 3 (May 1995), 305–320.

[74] Kim, Y. Recognition of form features using convex decomposition. *Computer Aided Design 24*, 9 (September 1992), 461–476.

[75] Klass, R. An offset spline approximation for plane cubic splines. *Computer Aided Design 15*, 3 (1983), 297–299.

[76] Klass, R., and Kuhn, B. Fillet and surface intersections defined by rolling balls. *Computer Aided Geometric Design 9* (1992), 185–193.

[77] Kleeman, M. W. Feature based parametric solids modeling for improving mechanical engineering design. In *Autofact '89* (Detroit, Michigan, October 1989), SME, pp. 21–37.

[78] Klein, R. *Concrete and Abstract Voronoi Diagrams*, vol. 400 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 1989.

[79] Kral, I. H. *Numerical Control Programming in APT*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.

[80] Kuragano, T. FRESDAM system for design of aesthetically pleasing free-form objects and generation of collision-free tool paths. *Computer Aided Design 24*, 11 (November 1992), 573–581.

[81] Lee, D. Medial axis transformation of a planar shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-4*, 4 (July 1982), 363–369.

[82] Lee, D., and Drysdale, R. Generalization of Voronoi diagrams in the plane. *SIAM J. Comput 10*, 1 (February 1981), 73–87.

[83] Lee, I.-K., Kim, M.-S., and Elber, G. Planar curve offset based on circle approximation. *Computer-Aided Design 28*, 8 (August 1996), 617–630.

[84] Lee, Y.-S. Admissible tool orientation control of gouging avoidance for 5-axis complex surface machining. *Computer-Aided Design 29*, 7 (July 1997), 507–521.

[85] Levin, J. Z. Mathematical models for determining the intersections of quadric surfaces. *Computer Graphics and Image Processing 11* (1979), 73–87.

[86] Lperez, M., and Wesley, M. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM 22*, 10 (October 1979), 560–570.

[87] Machinability Data Center. *Machining Data Handbook*, third ed., vol. 1 & 2. Metcut Research Associates Inc., Cincinnati, Ohio, 1980.

[88] Maekawa, T., Cho, W., and Patrikalakis, N. M. Computation of self-intersections of offsets of Bézier surface patches. *Journal of Mechanical Design 119*, 2 (June 1997), 275–283.

[89] Marchant, P. A numerical method for intersection of parametric surfaces. Master's thesis, University of Utah, Salt Lake City, Utah, 1987.

[90] Marciniak, K. Influence of surface shape on admissible tool positions in 5-axis face milling. *Computer-Aided Design 19*, 5 (June 1987), 233–236.

[91] Marciniak, K., and Putz, B. Approximation of spirals by piecewise curves of fewest circular arc segments. *Computer Aided Design 16*, 2 (March 1984), 87–90.

[92] Meek, D., and Walton, D. Approximation of discrete data by G^1 arc splines. *Computer Aided Design 24*, 6 (June 1992), 301–306.

[93] Meek, D., and Walton, D. Approximating quadratic NURBS curves by arc splines. *Computer Aided Design 25*, 6 (June 1993), 371–376.

[94] Millman, R. S., and Parker, G. D. *Elements of Differential Geometry.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[95] Moreton, D., Parkinson, D., and Wu, W. The application of a biarc technique in CNC machining. *Computer-Aided Engineering Journal* (April 1991), 54–60.

[96] Müllenheim, G. Convergence of a surface/surface intersection algorithm. *Computer Aided Geometric Design 7* (1990), 415–423.

[97] Müllenheim, G. On determining start points for a surface/surface intersection algorithm. *Computer Aided Geometric Design 8* (1991), 401–408.

[98] Murphy, E., Ansel, R., and Krajewski, J. Reduced distortion in optical freeform fabrications with UV lasers. *Radiation Curing* (February/May 1989), 3–7.

[99] Nau, D., and Chang, T.-C. Automated process planning using hierarchical abstraction. *Texas Instruments Technical Journal* (1987), 39–46.

[100] Nutbourne, A., and Martin, R. *Differential Geometry Applied to Curve and Surface Design*, vol. 1. Ellis Horwood Limited, West Sussex, England, 1988, ch. 1.

[101] Parkinson, D. Optimized biarc curves with tension. *Computer Aided Geometric Design 9* (1992), 207–218.

[102] Parkinson, D., and Moreton, D. Optimal biarc-curve fitting. *Computer Aided Design 23*, 6 (July 1991), 411–419.

[103] Pates, R. F. A parallel algorithm for surface intersection. Master's thesis, University of Utah, Salt Lake City, Utah, 1988.

[104] Patrikalakis, N. M., and Prakash, P. V. Free-form plate modeling using offset surfaces. In *Computers in offshore and arctic engineering* (1987), J. S. Chung and D. Angelides, Eds., pp. 37–44.

[105] Peng, Q. S. An algorithm for finding the intersection lines between two B-spline surfaces. *Computer Aided Design 16*, 4 (July 1984), 191–196.

[106] Perienté, F., and Kim, Y. S. Incremental and localized update of convex decomposition used for form feature recognition. *Computer-Aided Design 28*, 8 (August 1996), 589–602.

[107] Persson, H. NC machining of arbitrarily shaped pockets. *Computer-Aided Design 10*, 3 (May 1978), 169–174.

[108] Pham, B. Offset approximation of uniform B-splines. *Computer-Aided Design 20*, 8 (October 1988), 471–474.

[109] Pham, B. Offset curves and surfaces: A brief survey. *Computer Aided Design 24*, 4 (April 1992), 223–229.

[110] Phang, P. K. Feature-based design and manufacturing using standard catalog mechanical parts. Master's thesis, University of Utah, Salt Lake City, Utah, 1994.

[111] Pho, T., and Lapidus, L. Topics in computer-aided design: Part I. an optimum tearing algorithm for recycle systems. *AIChE Journal 19*, 6 (November 1973), 1170–1181.

[112] Piegl, L. Curve fitting algorithm for rough cutting. *Computer Aided Design 18* (1986), 79–82.

[113] Preparata, F. P., and Shamos, M. I. *Computational Geometry: An Introduction.* Springer-Verlag, New York, New York, 1985.

[114] Pusztai, J., and Sava, M. *Computer Numerical Control.* Reston Publishing Company, Inc., Reston, Virginia, 1983.

[115] Ranky, P. *The Design and Operation of FMS.* IFS (Publications) Ltd., Bedford, UK, and North-Holland, Amsterdam, Netherlands, 1983.

[116] Rosenberger, H. Order-k Voronoi diagrams of sites with additive weights in the plane. Tech. Rep. UIUCDCS-R-88-1431, Department Of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, May 1988.

[117] Roth, R. Computer solutions to minimum cover problems. *Operations Research 17*, 3 (1969), 455–465.

[118] Rubin, J. A technique for the solution of massive set covering problems with application to airline crew scheduling. Tech. Report 320-3004, IBM, 1971.

[119] Sabin, M. *The Use of Piecewise Forms for the Numerical Representation of Shape.* PhD thesis, Computer and Automation Institute, Budapest, Hungary, 1977.

[120] Saeed, S., de Pennington, A., and Dodsworth, J. Offsetting in geometric modeling. *Computer-Aided Design 20*, 2 (March 1988), 67–74.

[121] Sarraga, R. F. Algebraic methods for intersections of quadric surfaces in GMSOLID. *Computer Vision, Graphics and Image Processing 22* (1983), 222–238.

[122] Scheer, A.-W. *CIM, Computer Integrated Manufacturing, Towards the Factory of the Future*, second, revised and enlarged ed. Springer-Verlag, Berlin, Germany, 1991.

[123] Schönherr, J. Smooth biarc curves. *Computer Aided Design 25*, 6 (June 1993), 365–370.

[124] Sederberg, T. W., and Meyers, R. J. Loop detection in surface patch intersections. *Computer Aided Geometric Design 5* (1988), 161–171.

[125] Sederberg, T. W., and Parry, S. R. Comparison of three curve intersection algorithms. *Computer Aided Design 18*, 1 (January/February 1986), 58–63.

[126] Sederberg, T. W., White, S. C., and Zundel, A. K. Fat arcs: A bounding region with cubic convergence. Tech. Rep. ECGL-88-1, Engineering Computer Graphics Laboratory Brigham Young University, Provo, Utah, June 1988.

[127] Shah, J. J., and Mäntylä, M. *Parametric and Feature-Based CAD/CAM*

: Concepts, Techniques, and Applications. John Wiley & Sons, Inc., New York, New York, 1995.

[128] Shah, J. J., Mäntylä, M., and Nau, D. *Advances in Feature Based Manufacturing.* Elsevier Science Publishers, 1994.

[129] Sharrock, T. J. Biarcs in three dimensions. In *The Mathematics of Surfaces II*, R. R. Martin, Ed. Clarendon Press, Oxford, 1987, pp. 395–411.

[130] Sierksma, G. *Linear and Integer Programming: Theory and Practice.* Dekker, New York, New York, 1996.

[131] Smith, B. M., Rinaudot, G. R., Reed, K. A., and Wright, T. *Initial Graphics Exchange Specification (IGES), Version 4.0.* IGES/PDES Organization, Gaithersburg, MD, June 1988.

[132] Sohon, F. W. *The Stereographic Projection.* Chemical publishing co., inc., Brooklyn, New York, 1941.

[133] Su, B., and Liu, D. *Computational Geometry — Curve and Surface Modeling.* Academic Press, 1989.

[134] Suh, Y. S., and Lee, K. NC milling tool path generation for arbitrary pockets defined by sculptured surfaces. *Computer-Aided Design 22*, 5 (June 1990), 273–284.

[135] Sungurtekin, U. A., and Voelcker, H. B. Graphical simulation & automatic verification of NC machining programs. In *IEEE International Conference on Robotics and Automation* (1986), pp. 156–165.

[136] Thomas, Jr., G. B., and Finney, R. L. *Calculus and Analytic Geometry*, seventh ed. Addison-Wesley Publishing Company, Reading, Massachusetts, 1988.

[137] Thomas, S. W. *Modeling Volumes Bounded by B-spline Surfaces.* PhD thesis, University of Utah, Salt Lake Cite, Utah, June 1984.

[138] Tiller, W., and Hanson, E. G. Offsets of two-dimenstional profiles. *IEEE Computer Graphics and Applications* (September 1984), 36–46.

[139] Tsang, J. PROPEL: An expert system for generating process plans. *SIGMAN Workshop on Manufacturing Planning* (1989).

[140] Turkiyyah, G. M., Storti, D. W., Ganter, M., Chen, H., and Vimawala, M. An accelerated triangulation method for computing the skeletons of free-form solid models. *Computer-Aided Design 29*, 1 (January 1997), 5–19.

[141] Van Thiel, M. T. Feature based automated part inspection. Master's thesis, University of Utah, Salt Lake City, Utah, 1993.

[142] Vanderbei, R. J. *Linear Programming: Foundations and Extensions*. No. 4 in International series in operations research & management. Kluwer Academic Publishers, Boston, Massachusetts, 1996.

[143] Verma, S. Simulation of numerically controlled milling machines. Master's thesis, University of Utah, Salt Lake City, Utah, 1995.

[144] Wang, H.-P., and Li, J.-K. *Computer-Aided Process Planning*, vol. 13 of *Advances in Industrial Engineering*. Elsevier Science Publishers B.V., Amsterdam, Netherlands, 1991.

[145] Wang, W. Integration of solid geometric modeling for computerized process planning. *Computer-Aided/Intelligent Process Planning - PED 19* (November 1985), 177–187.

[146] Wang, W. Solid modeling for optimizing metal removal of three-dimensional NC end milling. *Journal of Manufacturing Systems 7*, 1 (1988), 57–65.

[147] Wang, W., and Wang, K. Geometric modeling for swept volume of moving solids. *IEEE Computer Graphics & Applications* (December 1986), 8–17.

[148] Wang, Y. Intersection of offsets of parametric surfaces. *Computer Aided Geometric Design 13*, 5 (July 1996), 453–465.

[149] Wohlers, T. T. Make fiction fact fast. *Manufacturing Engineering 106*, 3 (March 1991), 44–49.

[150] Woo, T. Feature extraction by volume decomposition. In *Proc. Conf. CAD/CAM in Mechanical Engineering* (Cambridge, MA, March 1982), pp. 39–45.

[151] Woodwark, J. Shape models in computer integrated manufacture — a review. *Computer-Aided Engineering Journal 7*, 6 (June 1988), 103–112.

[152] Yap, C. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Journal of Discrete and Computational Geometry 1*, 4 (1987), 365–393.