REPRESENTATION, EVALUATION AND EDITING OF
FEATURE-BASED AND CONSTRAINT-BASED DESIGN

A Thesis
Submitted to the Faculty
of
Purdue University
by
Xiangping Chen

In Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

May 1995
This thesis is dedicated to my sister, Xiao-Min Chen, who supported my undergraduate study.
ACKNOWLEDGMENTS

I gratefully thank my advisor, Professor Christoph M. Hoffmann, for his guidance, advice, support and encouragement throughout my entire doctoral program at Purdue. I have been deeply influenced by his professional conduct which has been a career model for his students to follow. I thank the other members of my committee, Dr. David C. Anderson, Dr. Jörg Peters and Dr. George Vaněček for their valuable advice.

I thank the others in the Erep project whose work supported this thesis. William J. Bouma coded the Erep graphical user interface. He also shared with me his experience in C++ programming. Ioannis Fudos integrated the constraint solver, whose earliest version was developed in collaboration with J. Cai and R. Paige of New York University. Pamela J. Vermeer shared with me her experiences using the ACIS solid modeler.

The consistent support and encouragement of my family has been very important to my education.

Thanks also go to Jindong Chen, Steve Cutchin, Cassiano Durand, Haitao Jiang, Nianzheng (Douglas) Liu, Xiangning Sean Liu and K. R. Ramanathan for friendship, encouragement and good times. Finally, I thank Dr. William J. Gorman for his careful proofreading of this thesis.

The research of this thesis was supported in part by ONR contract N00014-90-J-1599, by NSF Grant CDA 92-23502, and by NSF Grant ECD 88-03017.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>xi</td>
</tr>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>Thesis Statement</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Thesis Outline</td>
<td>4</td>
</tr>
<tr>
<td>2.</td>
<td>CURRENT CAD PARADIGMS</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Solid Modeling</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Constructive Solid Geometry</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Boundary Representation</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3</td>
<td>Decomposition</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4</td>
<td>Boolean Operations</td>
<td>7</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Boundary Modifications</td>
<td>8</td>
</tr>
<tr>
<td>2.1.6</td>
<td>Deficiencies of Solid Modeling</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Feature-Based Design</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Form Features</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Specification Features</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Constraint-Based Design</td>
<td>15</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Simultaneous Solver</td>
<td>17</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Incremental Solver</td>
<td>17</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Constructive or Mixed Solver</td>
<td>18</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Deficiencies of Constraint-Based Design</td>
<td>18</td>
</tr>
<tr>
<td>3.</td>
<td>A LAYERED DESIGN MODEL</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>A Layered Design Model</td>
<td>22</td>
</tr>
</tbody>
</table>
3.2 Unevaluated and Modeler-Independent Design .................................. 24  
  3.2.1 Generated Features ..................................................... 24  
  3.2.2 Modifying Features ..................................................... 29  
  3.2.3 Datum Features .......................................................... 30  

4. GENERIC NAMING SCHEMA ..................................................... 33  
  4.1 Background ................................................................. 33  
  4.2 Topology-Based Naming ................................................... 36  
      4.2.1 Naming Created Geometric Elements ............................... 37  
      4.2.2 Topological Context Computation ................................ 39  
      4.2.3 Remarks .............................................................. 43  
  4.3 Orientation Information .................................................. 43  
      4.3.1 Local Orientation .................................................... 44  
      4.3.2 Feature Orientation ................................................ 47  
      4.3.3 Combining Local and Feature Orientation ...................... 51  
  4.4 Implementation .......................................................... 53  
      4.4.1 Vertices .............................................................. 54  
      4.4.2 Edges ................................................................. 54  
      4.4.3 Faces ................................................................. 54  

5. DESIGN COMPILATION .......................................................... 56  
  5.1 Matching Generic Names .................................................. 57  
  5.2 Solving Constraints ..................................................... 58  
  5.3 Performing Feature Attachment ........................................ 59  
      5.3.1 Explicitly Bounded Features ...................................... 61  
      5.3.2 Implicitly Bounded Features ..................................... 66  
      5.3.3 Discussion ......................................................... 67  

6. EDITING FEATURE-BASED AND CONSTRAINT-BASED DESIGN ............. 71  
  6.1 Feature Editing .......................................................... 72  
      6.1.1 Editing Operations ................................................ 72  
      6.1.2 Editing Semantics ................................................ 74  
      6.1.3 Editing Examples .................................................. 78  
  6.2 Feature Reevaluation ................................................... 79  
  6.3 Matching a Vertex ...................................................... 81  
  6.4 Matching an Edge ....................................................... 84  
  6.5 Matching a Face ......................................................... 88  
  6.6 Remarks ................................................................. 91
7. PERSPECTIVE .............................................................. 92
  7.1 Specification Features ............................................. 92
  7.2 Instance Features .................................................. 93
  7.3 Assembly ........................................................... 94
  7.4 Future Work ........................................................ 94

BIBLIOGRAPHY .......................................................... 99

APPENDIX  EREP SPECIFICATIONS ..................................... 105
  A.1 Parts and Features ............................................... 105
  A.2 Generated Features .............................................. 105
  A.3 Datum Features .................................................. 107
  A.4 Modifying Features .............................................. 109
  A.5 Naming Schema .................................................. 110
  A.6 Cross Sections ................................................... 111

VITA ................................................................. 115
LIST OF TABLES

Table                                                                 Page

5.1    Compilation steps involved in three classes of features        57
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of feature-based design</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of constraint-based design</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>An example of feature-based and constraint-based design.</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>A design model is layered into an unevaluated, modeler-independent design representation and an evaluated, modeler-dependent design representation.</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>The modeler independence is achieved by a generic naming schema and a design compiler.</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Extrusion of profile C</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>Toroidal revolution of profile C</td>
<td>26</td>
</tr>
<tr>
<td>3.6</td>
<td>Spherical revolution of profile C</td>
<td>27</td>
</tr>
<tr>
<td>3.7</td>
<td>An example of using feature attributes.</td>
<td>29</td>
</tr>
<tr>
<td>3.8</td>
<td>A datum plane defined in Erep.</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Persistent naming error in design variant.</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Generic naming in an extrusion. Left: names assigned by sketcher. Right: names for extrusion.</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Labeling of object with two features. For simplicity, the names omit the feature qualifier.</td>
<td>40</td>
</tr>
<tr>
<td>4.4</td>
<td>Context graph for object of Figure 4.3. Faces indicated by boxes, edges by circles.</td>
<td>40</td>
</tr>
<tr>
<td>4.5</td>
<td>Block with toroidal hole. The edges of the cut cannot be distinguished from the context graph.</td>
<td>44</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.6</td>
<td>Computation of local orientation of an edge.</td>
<td>45</td>
</tr>
<tr>
<td>4.7</td>
<td>Computation of local orientation of a vertex.</td>
<td>46</td>
</tr>
<tr>
<td>4.8</td>
<td>Local orientation is insufficient to identify the four vertices with same</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>direct context.</td>
<td></td>
</tr>
<tr>
<td>4.9</td>
<td>Orientation of torus surface created by revolving a circular profile.</td>
<td>48</td>
</tr>
<tr>
<td>4.10</td>
<td>Curve types on a cylindrical topology.</td>
<td>48</td>
</tr>
<tr>
<td>4.11</td>
<td>Edges $e_1$ and $e_2$ are not distinguished by feature orientation alone.</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>Two phases of feature attachment.</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>Wedge determination for proto feature definition.</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>The two wedges used for revolved features delimited by from and to planes</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>are shown shaded. Revolution seen in a plane perpendicular to axis $A$ and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>from the direction in which $A$ points. $Q$ is the intersection of the from</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and to planes, $p$ a generic point revolved about $A$.</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>From face completely intersects swept volume.</td>
<td>64</td>
</tr>
<tr>
<td>5.5</td>
<td>Definition of a volume in set $S_{in}$. Left: completely intersecting</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>bounding faces. Right: partially intersecting bounding faces.</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>Creating a protrusion via from face to next. The protrusion is from right</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>to left.</td>
<td></td>
</tr>
<tr>
<td>5.7</td>
<td>Alternatives for incompletely intersecting from/to faces: (a) from face $F$</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>incompletely intersects circular proto feature; (b) $F$ is extended as</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mathematical surface, (c) $F$ is extended by tangent directions at the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>boundary, (d) $F$ is extended by a ruling of the boundary perpendicular to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the direction of the sweep.</td>
<td></td>
</tr>
<tr>
<td>5.8</td>
<td>Determining in-between volumes from intervals: (a) use the minimum value</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>of the bounding box to include $V$, (b) use the maximum value of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bounding box to exclude $V$.</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>When the slot (second feature of part $A$) is moved to the right and</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>becomes a step, rounding the edge in part $A'$ should be considered an</td>
<td></td>
</tr>
<tr>
<td></td>
<td>error.</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>When the slot (second feature of part <em>B</em>) is moved to the right and becomes a step, rounding the edge in part <em>B′</em> should be considered an error.</td>
<td>76</td>
</tr>
<tr>
<td>6.3</td>
<td>When the through slot (second feature of part <em>C</em>) is changed to a blind slot, the entire edge of part <em>C′</em> should be rounded. For subsequent editing parts <em>B</em> and <em>C′</em> should be equivalent.</td>
<td>77</td>
</tr>
<tr>
<td>6.4</td>
<td>The slot dimensioning schema is edited. The new schema shown to the right no longer references the protrusion, so the slot position is now independent of the protrusion feature and its variations.</td>
<td>79</td>
</tr>
<tr>
<td>6.5</td>
<td>Editing dimensioning schema and feature shapes. Top: using the dimensioning schema left, editing the protrusion radius will affect the position of the slot feature. Bottom: the cross section of the slot feature has been edited replacing one of the round ends with a square end.</td>
<td>80</td>
</tr>
<tr>
<td>6.6</td>
<td>The vertex <em>V</em> of part <em>D</em> is not found in part <em>D′</em>. As a proto feature vertex, it can be reconstructed from the first feature of part <em>D′</em>.</td>
<td>82</td>
</tr>
<tr>
<td>6.7</td>
<td>After repositioning the slot, the correct design variant is shown in the middle.</td>
<td>85</td>
</tr>
<tr>
<td>6.8</td>
<td>Left: edge <em>e</em> is named in the design. After editing, the edge is subdivided by colliding features. Segments <em>e</em> and <em>e</em> match.</td>
<td>87</td>
</tr>
<tr>
<td>6.9</td>
<td>Different representatives of a merged face produce different results after feature editing.</td>
<td>89</td>
</tr>
<tr>
<td>7.1</td>
<td>ACIS as the geometry bus advocated by Spatial Technology Inc.</td>
<td>96</td>
</tr>
<tr>
<td>7.2</td>
<td>Impact of a high-level design representation</td>
<td>97</td>
</tr>
</tbody>
</table>
ABSTRACT

Chen, Xiangping. Ph.D., Purdue University, May 1995. Representation, Evaluation and Editing of Feature-Based and Constraint-Based Design. Major Professor: Christoph M. Hoffmann.

This thesis investigates a general and systematic approach to feature-based and constraint-based design. We combine feature-based design and constraint-based design by globally decomposing a design into a sequence of feature attachments and locally defining and positioning each feature by constraints. Analogous to the concept of high-level programming languages, we formalize a layered design model that eliminates the dependency of a design representation on a solid modeler. With this design model, design intent, such as feature descriptions and constraints, is stored in an unevaluated, modeler-independent design representation while the geometry to which it corresponds is stored in an evaluated, modeler-dependent design representation. The separation essentially relies on a naming and matching schema that converts between a geometric reference and a generic name, and a design compiler that automatically instantiates the unevaluated design representation to an evaluated design representation with respect to a solid modeler.

The geometric references for defining feature attributes and constraints are recorded with their generic names in the unevaluated design representation. We propose several techniques for naming geometric entities unambiguously.

The design compilation or instantiation involves remapping a generic name back to a geometric reference in the selected geometric modeler, solving constraints and implementing feature operations or attachments. Instead of developing a constraint solver for this design compiler, we use an independent and general solver. Feature attachment operations are different from classical Boolean operations in solid modeling. However, we provide a semantics for them that is based on existing operations in solid modeling.

The layered design model allows users to edit archived conceptual designs to derive new designs quickly. We investigate the coordination of later features in the unevaluated and modeler-independent representation when a feature is edited and provide a method for editing feature-based and constraint-based design.

We also discuss how to extend this work to a commercial feature-based and constraint-based CAD system.
1. INTRODUCTION

Engineering design is the process of translating customer requirements into a comprehensive description of a manufacturable product. The emergence of computer-aided design (CAD) is driven by the goal of improving the efficiency of this design process in every way possible and lowering the costs of product development.

CAD is one of many applications which stimulated the development of solid modeling [28, 48, 52]. A geometric model is part of a computerized design representation. However the widely used solid modeling technology has been developed quite independently from engineering design. Although existing modeling techniques are able to construct complicated shapes, these techniques are often used at a low level of conceptualization. Design intent, however, is expressed at a higher level of abstraction. The translation from design requirements and intent to geometric operations suggests feature-based design or design by features and constraint-based design.

Feature-based design decomposes a CAD process into procedural application-oriented features that are intuitive to engineers while constraint-based design allows users to express design freely with algebraic and geometric constraints that are automatically evaluated. Features, such as holes, slots and rounds, and constraints, such as dimensions, alignments and relations, are part of a high-level vocabulary of engineering design. The recent success of Pro/ENGINEER [44], a feature-based and constraint-based CAD system developed by Parametric Technology Corp., has strongly influenced the CAD/CAM (computer-aided design/computer-aided manufacturing) industry.

While almost every CAD/CAM vendor is ready to pursue feature-based and constraint-based design, some complex technical issues have to be addressed to do
so. This thesis investigates a general and systematic approach and develops the necessary details for a feature-based and constraint-based CAD system. This research is motivated by the following factors.

1. Exchange and reuse of design history

Current CAD systems allow the exchange of only completed designs with each other. Although design history may be reused and modified in the same system in which the original design has been made, the exchange of design history among current CAD systems is not possible. CAD end-users often have difficulty deciding which and how many CAD systems to use. Choosing a single CAD system involves a risk that the selected CAD might not remain competitive in the future. On the other hand, the use of several CAD systems complicates data management and increases data redundancy, especially when the exchange of designs created by different CAD systems remains at a low level.

2. Modeler-independence of conceptual design

We believe that conceptual design is abstract knowledge that is independent of specific software. However, current CAD systems have used specific modeler-dependent representations to archive designs, a major barrier to exchange designs created by different CAD systems.

3. Conversion from solid modelers to CAD systems

CAD is only one of many applications of solid modeling. Through many years of development, commercial solid modelers serve various applications. The operations of solid modeling are well-defined, but unfortunately solid modeling does a poor job if it is directly used as a CAD system that involves various design operations. Instead of starting from scratch, we would like to use well-defined solid modeling operations to carry out design operations.

4. Success of high-level programming languages
High-level programming languages have been very successful in abstracting conceptual computations from various computer architectures. The abstraction is realized by different compilers that translate the abstract computations into different machine instruction sets.

Analogously, we want to express conceptual design by a high-level representation and to target design compilers to different solid modelers. Based on the benefits of high-level programming languages, we expect that a high-level design representation will improve CAD technology significantly.

1.1 Thesis Statement

In this thesis we deal with three aspects of feature-based and constraint-based design: representation, evaluation and editing. We combine feature-based design and constraint-based design by globally decomposing a design into a sequence of feature attachments and locally defining and positioning each feature by constraints. At present, a typical CAD system is tightly coupled with a core solid modeler and it allows the exchange with other CAD systems of only completed design models, which in general are represented by some augmented boundary representation. The design concept, however, cannot be shared by different CAD systems, nor can the design be changed easily when standard exchange representations are used. Therefore, creative design knowledge has to be understood and edited in the particular CAD system in which the initial design was made.

We remove this limitation by proposing a layered design model that contains an unevaluated, modeler-independent representation and an evaluated, modeler-dependent representation. Design intent is stored in the unevaluated representation, independent from any core solid modeler supplied by a vendor, while the geometry to which it corresponds is stored in the evaluated representation, typically a boundary representation of a solid model. The separation essentially relies on a naming and matching
system that converts between a geometric reference and its generic name, and a design compiler that automatically interprets the unevaluated representation to obtain the evaluated representation with respect to a solid modeler.

The geometric references for defining feature attributes and constraints are recorded with their generic names in the unevaluated design representation. We propose several techniques for naming geometric entities unambiguously [6].

The design compilation or instantiation [14] involves remapping a generic name back to a geometric reference in the selected geometric modeler, solving constraints and implementing feature operations or attachments. Instead of developing a constraint solver for this design compiler, we use an independent and general solver [5]. Feature attachment operations are different from classical Boolean operations in solid modeling. However, we provide a semantics for them that is based on existing operations in solid modeling [13].

The layered design model allows users to edit archived conceptual designs to derive new designs quickly. We investigate the coordination of later features in the unevaluated and modeler-independent representation when an earlier feature is edited and provide a method for editing feature-based and constraint-based design [12].

We also discuss the issues involved in extending our prototype to a commercial feature-based and constraint-based CAD system.

1.2 Thesis Outline

In Chapter 2 we review the current CAD paradigms, including solid modeling, feature-based design and constraint-based design. We characterize a layered design model in Chapter 3 with emphasis on an unevaluated and modeler-independent design representation. We also discuss the architectural issues of a feature-based and constraint-based CAD system. In Chapters 4 and 5, we describe the two vital issues in the layered design model. In Chapter 4, we investigates the generic naming schema that labels geometric references of a specific model with generic names. In Chapter 5,
we show how to compile the unevaluated and modeler-independent design representation to obtain its geometric representation with respect a core modeler. We explain the semantics of feature attachment based on mixed Boolean operations. To show the benefits of the layered design model, we explore in Chapter 6 the editability of feature-based and constraint-based design. We discuss in Chapter 7 how to extend our prototype of feature-based and constraint-based design to a commercial design system.
2. CURRENT CAD PARADIGMS

In this chapter we review and evaluate current CAD paradigms including solid modeling, feature-based design and constraint-based design. We do not separately investigate parametric design, a commercial term which appears to refer to the combined use of feature-based design and constraint-based design. The emphasis of this chapter is on feature-based design. Solid modeling and constraint-based design are briefly summerized in terms of CAD. Extensive literature reviews can be traced from the author’s previous two Masters theses [10, 11] in solid modeling and Fudos’ Ph.D dissertation in constraint-based design.

2.1 Solid Modeling

Solid modeling has become a well-established field after its evolution for more than a quarter of a century [16, 28, 39, 48, 52, 49, 51]. Solid modeling deals with geometric representations and manipulations of three-dimensional objects. The fundamental representation schemas for solids are Constructive Solid Geometry (CSG), Boundary Representation (BRep) and spatial decomposition. The classical operations for solid construction are Boolean operations and boundary modifications. In this section, we give a brief overview of each of these topics and study whether solid modeling is suitable for CAD.

2.1.1 Constructive Solid Geometry

Constructive solid geometry creates geometric models by providing a set of primitives and a set of operations including (regularized) Boolean operations and rigid-body motion. A solid is represented as a tree whose leaves are primitives and whose internal nodes are operations. A CSG model is both a representation and a construction
method. Primitives normally are limited to cubes or blocks, cones, cylinders, spheres and tori. In some systems, they are joined with certain halfspaces such as halfplanes and, sometimes, algebraic halfspaces.

2.1.2 Boundary Representation

Boundary representation describes a solid with its adjacent boundaries and boundary orientations. The topology of a solid in BRep is hierarchically described by shells, faces, edges, vertices and their adjacencies, and the geometry of a solid is represented by surface and curve equations and coordinates of points. Different data structures have been proposed and used in existing solid modeling systems. Typical structures are winged-edge [4], half-edge [10, 39, 72, 73] and radial-edge data structures [70].

Operations on BRep can be performed at a very low level such as local geometric modifications. The algorithms from CSG to BRep conversion [38, 49, 61] and the direct Boolean operations [28] on two BRep solids make BRep as powerful as CSG representation. The Boolean operations on two BRep solids implies that extruded and revolved solids are easily incorporated in BRep systems.

2.1.3 Decomposition

Regular decompositions approximate a solid by a set of disjoint volumetric primitives. Such primitives can be cubes of the same size, cubes of different sizes (such as octree), or simplicies. It is possible to have an accurate representation by storing additional information in approximate decompositions [7]. Other decompositions have also been proposed such as BSP (binary space partition) tree [68] and convex decomposition [71].

2.1.4 Boolean Operations

Solid Boolean operations refer to union, intersection and difference of two solids. They are used in CSG to BRep conversion and operations of BRep-based systems. Regularized Boolean operations are the solution for eliminating dangling lower-dimensional
structures [38, 49]. Although the mathematical theories of regularized Boolean operations are well developed, a fully robust implementation of such algorithms, due to finite precision arithmetic, remains difficult [11, 31]. Robustness has been studied in the past few years in order to make such operations correct and reliable.

2.1.5 Boundary Modifications

Euler-operations [40] are performed to adjust local geometry and topology of a solid model. The validity of modifications can be based on the Euler-Poincare formula. Operations have been extended to local operations [15, 16] that include rounding, local sweeping and gluing. The results from these operations cannot be guaranteed from Euler-Poincare formula alone and problems such as self-intersections must be avoided using additional tools.

2.1.6 Deficiencies of Solid Modeling

While solid modeling has been studied for decades, researchers and users have realized that the direct use of solid modeling for CAD/CAM has serious limitations [32, 51, 54, 59, 66]. Solid modeling seems to have developed too independently to fit CAD/CAM applications naturally. As a complement, feature-based and constraint-based design has been prevailing in the past years and has gained significant attention. As a CAD tool, solid modeling has the following problems.

1. Design intent is expressed by high-level specifications such as features and constraints while solid modeling deals with low-level geometry such as BRep, CSG primitives and coordinate transformations. The low-level geometry is not convenient for design or redesign and it requires CAD users to convert design intent to geometric details explicitly.

2. In solid modeling, variational geometry can be obtained by reevaluating a modified creation history, for example a CSG tree. But even to edit a dimension of a simple design may involve explicit and very low-level coordinate transformations
and shape modifications of primitives. Typically, a small shape adjustment at a node in a CSG tree may affect all tree nodes descendent from this node.

3. A geometric model alone is far from sufficient for a design representation. Information other than geometry such as tolerances and material properties is not considered in solid modeling systems.

4. The geometric representation in most solid modeling systems cannot be directly used in analysis, process planning, manufacturing activities and other applications. A more interesting and active role has been played by features which current solid modeling systems are unable to support. With the help of feature extraction or feature recognition, it is possible to extract a limited range of geometric features. However, other design features such as tolerances cannot be extracted from geometric models by such techniques.

2.2 Feature-Based Design

_Feature_ is a broad engineering term that names a meaningful entity, attribute or property of a part. The feature in feature-based design confines the attribute to design, such as functional details, material properties and tolerances. In terms of design, features can be classified into two types: _form features_ and _specification features_. A form feature is a set of geometric elements that contributes to a meaningful functional detail. Examples of form features are holes, slots and rounds. Specification features represent more detailed requirements of form features and the whole design. Tolerances, surface finish and material condition are specification features. We show in Figure 2.1 a simple example of feature-based design.

It is generally agreed that design processes are complex and their conceptualization is difficult. Feature-based design is considered an intuitive approach which creates a geometric model with feature attachment operations [23, 45, 53, 51, 54, 59]. Feature-based design is important for the following two reasons:
1. Features best represent design intent. In particular, form features provide a concise and high-level abstraction of part characteristics, and specification features supply additional design requirements.

2. Features in design and manufacturing are sometimes similar, such as holes and slots. Therefore, design features can be recorded and may be converted to features in analysis, process planning, manufacturing and other applications. Examples are [8, 60].

2.2.1 Form Features

Features were initially recognized important in the areas of process planning and manufacturing. At that time, solid models did not have any feature information and integration of design and manufacturing encouraged feature recognition and feature extraction [1, 19, 26, 57, 71]. The recognized features are classified into surface
features and volume features. Surface features are not convenient in applications such as process planning and manufacturing. Algorithms of converting surface features to volume features are available such as [20], where closure of surfaces is added for constructing a volume feature.

Feature extraction for process planning and manufacturing influenced CAD developers to consider design by features directly. The CAM-I project supported many research activities in feature-based design and manufacturing [23, 45, 47]. In [45] Pratt advocated the importance of design by features although features may vary in different application domains. Pratt and Wilson [47] laid down the functional requirements of support of form features in a solid modeling environment. Faux in [23] showed the importance of both form features and tolerances in design and manufacturing. Later, Pratt [46] concluded that volumetric BRep be used for features and feature models be integrated with internal solid modelers.

Following the CAM-I project, various research systems in feature-based design have been reported. At the University of Massachusetts, three exploratory systems have been developed in the domains of extrusion, injection and casting [17, 18, 22, 37]. Instead of supplying a feature set which meets the requirements of some specific application, a standard primitive feature set is provided upon which application-dependent features can be defined. Each primitive feature is defined as an object class with a set of attributes and user-defined primitive feature classes are allowed with user specified methods.

An inter-departmental effort at Purdue brought out an integrated feature-based design, manufacturing and inspection system, the quick turnaround cell (QTC), for prismatic parts [9, 65]. QTC is a manufacturing oriented system which supports features in process planning. It initially dealt with subtractive features only and its later version, QTC-II, allows also additive features [8]. With feature refinement, QTC design features can be automatically converted to features in process planning. The QTC system has hard-coded reference-handles in each feature prior to attachment for reference structures to position and orient later features.
The research group at Arizona State University has reported several results in feature-based design and feature recognition [59, 58, 60]. In their feature-based design system, two shells play important roles: a feature modeling shell and a feature mapping shell. A feature library is organized with feature hierarchies and new features can be added to this library. The validity of features after operations is checked with a set of rules. Interactive feature identification and feature recognition are also supported in their system. The mapping shell was developed to convert design features to manufacturing features.

Other similar systems include Miner’s MS thesis [41] using a CSG approach to design features, FSMT [21] reported from China, and EXTDdesign [35] from Finland.

These feature-based design systems directly extend the CSG paradigm [62] in terms of representations and solid modeling operations, typically, classical Boolean operations. Parametrization was introduced to specify form features, but constraints were less often involved. The primary difference of these systems from solid modelers is that the feature-based design systems have eliminated two deficiencies of solid modeling by expressing design intent in engineering vocabulary and by incorporating specification features. However, many research issues that already existed in classical solid modeling have not yet been solved. The major issues of these feature-based design systems are as follows:

1. Representation

Both high-level features and low-level geometry are still used side-by-side for design representation in these systems. Although features belong to the engineering vocabulary, their positions are expressed by low-level geometry such as coordinate values.
2. Operations

These feature-based design systems use union and subtraction in solid modeling to implement additive and subtractive features. The semantics of feature attachment operations, as indicated in [13, 29, 54], is different from the operations in solid modeling.

3. Editability

Because low-level geometry is used in design representation, editing feature-based design in these systems requires users to perform explicit translations.

4. Modeler independence

These feature-based design systems tightly integrate the feature model and its associated shape model. Tight integration is undesirable because it encourages nonstandard design representations that mix problem-specific generic information with software-specific instance information. An integrated implementation is difficult to adapt to new application domains and is cumbersome to change. Moreover, it impedes neutral product data exchange in environments where design in progress is to be exchanged, for example in distributed, collaborative manufacturing.

Some of these issues, such as semantics of feature attachment and persistent references to further eliminate the use of low-level geometry, were addressed by Rossignac [54]. With constraints, he and his colleagues illustrated some of their ideas in a 2D design system [55]. However, these ideas are difficult to generalize to 3D design.

Based on the research attempt of feature-based design, commercialized feature-based and constraint-based design systems, such as Pro/ENGINEER [44], are rapidly reshaping the CAD industry. In Pro/ENGINEER, features are represented and edited in terms of engineering vocabulary thus design and redesign are carried out at a conceptual level. Unfortunately, the technology underlying these commercial systems is proprietary and not known.
Despite the commercial success of these systems and their leadership in feature-based design, deficiencies have been reported, especially in the semantics of feature attachment, persistent reference structures, design editability and modeler independence.

Hoffmann [29] investigated the semantic deficiencies for defining feature attachment operations of the leading commercial feature-based system, Pro/ENGINEER. In a follow-up research effort, Chen and Hoffmann [13] formalized the semantics of two generic features, protrusion and cut under extrusion and revolution, and implemented the feature attachment operations with this semantics using a commercial solid modeling system, ACIS [63]. It appears that Pro/ENGINEER has a persistent or generic naming schema so that low-level geometry is eliminated in feature descriptions. This naming schema has been found to be unreliable [6]. Furthermore, this commercial systems provides limited editability for archived designs supported. Examples are shown in [12, 29, 30]. There is no standard design representation so that interchange of conceptual designs between these systems is not possible. We believe a complete and formal study of feature-based design is needed to guide the development of advanced CAD systems.

2.2.2 Specification Features

Specification features, in our definition, are tolerances, surface finish and material condition which give additional information when form features are to be manufactured. They are part of design information that a CAD system needs to deal with. Tolerances are the variational limits of acceptable part geometry. Surface finish is the specification of surface quality that constrains specific machine tools to be used. Process planning and manufacturing can be greatly affected by the specification features. Processes for a part with nominal dimensions but different tolerance values can be totally different because each process has certain accuracy. The technical issues involved in specification features are explained in Chapter 4. In Chapter 7,
we give a more detailed discussion on how specification features can be added in our feature-based and constraint-based design framework.

Regardless of which standard [2, 3] of specification features to follow, a CAD system needs to provide a convenient way for application users to attach and edit them. Specification features can be equivalently represented as dimensions in form features. Basically they are attached to geometric entities, i.e. surfaces, edges and vertices. A tolerance of flatness, for example, is attached to a face of a given geometric model.

Most feature-based design systems reviewed earlier handle specification features such as tolerances by defining a set of tolerancing types and recording each as either an association list or an object in a database. Examples are [21, 50, 56, 60]. However, since low-level geometry is used in these systems, their specification features are difficult to edit and may be easily destroyed when form features are modified.

To avoid this problem, in the work of Requicha and Chan [50], they restrict to face names only when specifying geometric tolerances and other attributes. In CSG operations, all new faces are either merged or split from existing faces, and so all faces in the resulting geometry have inherited names, which can be marked generically. This raises two new questions: What if a user has to reference edges or vertices and what if the two faces split from face subdivision have to have different specifications? This naming problem is partially solved in some commercial systems such as Pro/ENGINEER in which a persistent reference schema is employed.

We solve this referential issue in Chapter 4 with a generic naming system.

2.3 Constraint-Based Design

Constraint-based design permits flexible design by varying the values associated with constraints. Algebraic constraints, such as algebraic equations, and geometric constraints, such as perpendicularity and alignment, are commonly used to express design intent. A typical constraint-based design system allows a user to sketch inaccurate geometry, to specify a set of constraints, and to change the values associated
with constraints. The system then automatically solves all the constraints to derive the accurate geometry expected by the user.

![Diagram](image.png)

\[ l_1 = 2 l_2 \]

Figure 2.2 An example of constraint-based design

Figure 2.2 shows a simple 2D example of constraint-based design. A user sketches the left of Figure 2.2, then adds constraints to the sketch, and finally gives the needed values for some constraints. Note that the segments marked // are to be parallel, as are the segments marked \[]. Constraint-based design then produces the right shape of Figure 2.2.

Constraint-based design has been widely used in 2D systems, but not so successfully in 3D. Constraints can be represented by association lists or objects where geometric references and relations are recorded. Representation of geometric references leads to the development of a labeling or naming system which we will fully describe in Chapter 4. Research of constraint-based design has mainly focused on constraint solvers which derive geometry from constraints. Currently there are three classes of constraint solvers: simultaneous solver, incremental solver and constructive or mixed solver. We briefly explain each of these classes and then evaluate constraint-based design.
2.3.1 Simultaneous Solver

In a simultaneous constraint solver, all constraints are converted into a system of algebraic equations and this system is solved by either a symbolic method or a numerical method\cite{36, 42, 67}. Generally speaking, symbolic methods can find all solutions but are of limited efficiency for large systems of equations. Most numerical solvers use iterative methods, typically Newton iteration, to obtain one solution with an initial guess.

Simultaneous solvers are not suitable for large systems or interactive design. A symbolic solver may produce too many solutions for large systems. When values of some constraints are changed and a new system is formed, the solution of the old system may not be a good iteration seed for the new system. A numerical solver does not have control over which solution is produced because it depends on an initial guess.

2.3.2 Incremental Solver

In an incremental solver, geometries are solved stepwise and intermediate results are propagated to solve other geometries. The incremental approach requires that the geometries under given constraints can be ordered in a sequence that is solvable with propagation of previous results. This restriction limits the generality of incremental constraint solvers. A common problem is that such a solver cannot handle the cyclic chaining in a set of constraints.

Incremental solvers, however, allow users to control an intended solution. The solving sequence of a constraint system is commonly not unique. Users may change the sequence and add rules of solution selection for a particular sequence step to identify which of the several solutions is wanted. Because there are only few solutions at each step, the rules of solution selection are relatively simple to find and record.
2.3.3 Constructive or Mixed Solver

Naturally, simultaneous and incremental techniques can be combined to extend the solvable constraints. Constructive or mixed solvers employ incremental techniques globally and simultaneous techniques recursively. In particular, the geometries in a cyclic chain are solved simultaneously.

Constructive solvers retain the advantages of incremental solvers. Users may add rules of solution selection in the ambiguous steps to determine a specific solution for a constraint system. These rules are recorded along with a solving sequence so that initial design intent is properly archived. When constraints are edited, a new solution can be automatically obtained by reexecuting the recorded sequence and the attached rules of solution selection.

The number of geometries solved by simultaneous techniques is normally small so that the needed rules to select one among multiple solutions are manageable. Such rules are difficult to find and tend to be not persistent when many geometries are involved. Constructive solvers have been reported in [5, 24, 43, 64]. In this thesis, we use a constructive solver reported in [5].

2.3.4 Deficiencies of Constraint-Based Design

The most significant advantage of constraint-based design is that editing a design is simpler, faster and more generally intuitive. Almost every recently developed CAD system uses constraints. In spite of the success of constraints in 2D geometric design, the extension of solving 3D constraints remains difficult. Specific open problems include:

1. Constraints alone cannot characterize complex design because they specify design intent flatly. Most 2D geometric design falls into this category. But design intent involving 3D geometry is best expressed incrementally.

2. While 2D constraint solving techniques have been studied extensively, solving constraints in 3D remains difficult at present.
3. Constraints can only solve the geometric representation of a design. Other design requirements such as form features and specification features need separate techniques to model.

As a final remark, we believe that feature-based design and constraint-based design are complementary for expressing design intent. Although solid modeling by itself is not suitable for CAD, the geometry of a particular design is best represented by geometric modelers. In the next chapter, we describe our approach to feature-based and constraint-based design by using solid modeling independently from feature and constraint mechanisms.
3. A LAYERED DESIGN MODEL

We have concluded in the previous chapter that both features and constraints represent design intent. To combine feature-based design and constraint-based design, we globally decompose a design into a sequence of feature attachments and locally define and position features by constraints. Figure 3.1 shows a step-by-step example of our feature-based and constraint-based design.

It is essential for a feature-based and constraint-based design system to allow users to freely add, delete and modify features and constraints. The resulting design specifications should then be processed automatically, to derive a particular shape that satisfies the constraints, and analyze it with manufacturability, performance, etc.. Common approaches to feature-based design with or without constraints tightly integrate the feature model and the associated shape model as summarized in the previous chapter. While affirming this interrelationship, we believe it advantageous to replace tight integration with design compilation, raising the level of abstraction, to achieve modeler independence of a design representation.

The consequence of using a modeler-independent representation is that this representation must be reevaluated when a feature is edited. The value of this concept lies in its ability to avoid dependence on the specific capabilities of a core modeling system, and the great flexibility it offers in adapting to new application domains. Moreover, the concept serves demands for a neutral product data representation. However, several important and challenging technical issues, such as generic or persistent naming of feature collisions, must be solved to support this representation. We study a modeler-independent design representation in this chapter, but leave the generic naming issue to the next chapter because a naming schema is also an independent and useful tool for other applications.
Figure 3.1 An example of feature-based and constraint-based design.
3.1 A Layered Design Model

Based on the high-level representation proposed by Hoffmann and Juan [32], we formalize a layered design model which contains an unevauated, modeler-independent design representation and an evaluated, modeler-dependent design representation. Design intent, such as features and constraints, is stored in the unevauated representation, called *editable representation* (Erep) [32]. The geometry to which an Erep model corresponds, for specific constraint values, is stored in the evaluated representation, for instance as a particular boundary representation.

![Diagram](image_url)

**Figure 3.2** A design model is layered into an unevauated, modeler-independent design representation and an evaluated, modeler-dependent design representation.

Figure 3.2 shows this design model. A user constructs features serially, based on variational constraints and attachment attributes. The geometry of a design in progress is referenced by the user through the design interface to express constraints and feature attributes that specify the unevauated representation of the next feature. The design interface consists of a graphical user interface (GUI) and a code generator for the Erep language (see appendix). When the next feature is properly defined and, consequently, its unevauated design representation is generated and recorded,
this representation is then automatically translated into an evaluated representation including new geometry that, in turn, will be referenced for expressing the next feature.

It is desirable that this interaction between the model representations and the design interface be carefully controlled so as not to create any dependency of the unevaluated model construction on the particulars of the core modeling system and its native representation. The independence is achieved by mediating the interaction between the design interface and the evaluated design representation, in particular, boundary representation, with a generic or persistent naming schema and a design compiler. A generic naming schema [6] replaces all geometric references by generic names to describe features and constraints in the unevaluated representation while a design compiler [14] translates or instantiates an unevaluated and modeler-independent representation into its evaluated and modeler-dependent representation with respect to a core solid modeler. We will show later (also see [12]) that a design can be edited through the design interface and its updated unevaluated design can be identically reinterpreted by this design compiler. The functional organization of the architecture is shown in Figure 3.3.

Figure 3.3 The modeler independence is achieved by a generic naming schema and a design compiler.
In the following section we describe the unevaluated design representation, with our example Erepy system. The evaluated design representation mainly contains the geometric representation of a design and instantiated feature information derived from the unevaluated feature information. The only difference between the instantiated feature information and unevaluated feature information is that the former uses geometric references as feature attributes while the latter uses generic names. Because the geometric representation has been extensively covered in a rich literature, we do not separately discuss the evaluated representation.

3.2 Unevaluated and Modeler-Independent Design

The unevaluated and modeler-independent design of our system, or Erepy language, was first proposed by Hoffmann and Juan in [32], where the main benefits of organizing a CAD system around this language were articulated. The Erepy specification given in that paper omits a number of technical issues that concern the abstraction of design gestures into the generic Erepy level and the details of recording those abstractions. In this thesis we study Erepy as a whole system.

The feature operations in Erepy can be categorized into three broad groups: generated features, modifying features and datum features. While listing the detailed syntax of Erepy language in the appendix, we explain the semantics of these features as follows.

3.2.1 Generated Features

We consider generated features that are based on a planar profile, swept into a three-dimensional shape. To simplify matters further, we concentrate on extrusions and revolutions only. Such a sweep is to be modified by feature attributes that govern the exact interpretation of the sweep operation and determine how the existing geometry will be changed.
3.2.1.1 Extrusions and Revolutions

A profile $C$ is defined in a sketching plane $P$. The sketching plane can be the support of a planar face or a datum plane defined separately. The profile must be a set of closed curves defining interior and exterior. The profile interior is finite and is used to define the interior and exterior of the sweep.

The definition of $C$ is based on variational constraints that are solved when the feature is created. The constraints define both the intrinsic shape of $C$ as well as its position relative to the existing geometry. More precisely, the constraints position the profile $C$ with respect to the projection of the existing geometry onto the plane $P$. The representation of 2D constraints for defining and positioning the profile $P$ is omitted here but detailed in another thesis [24].

Let $C$ be a closed profile in the plane $P$. The extrusion of $C$ is defined to be a solid obtained as follows: The profile $C$ is swept perpendicular to $P$, up to a parallel plane $P'$, resulting in one or more tubular surfaces. The interior of the contour in $P$ and in $P'$ defines planar faces that together with the tubular surfaces bounds a solid volume. The solid volume so obtained is the extrusion of $C$. See also Figure 3.4.

![Figure 3.4 Extrusion of profile C](image)

Let $C$ be a closed profile in the plane $P$, $A$ be an oriented line in $P$ that does not intersect $C$, except, possibly, in finitely many isolated points. Assume that no part of
$C$ is to the left of $A$. The toroidal revolution of $C$ is the solid obtained by revolving $C$, and its interior, about the axis $A$, by a positive angle not greater than 360 degrees. In case the revolution is by 360 degrees, the resulting surfaces are topologically tori. As before, planar end faces are required to complete a solid surface in the case of an incomplete revolution. See also Figure 3.5. If $C$ intersects $A$, the intersection points in general become nonmanifold points on the surface of the resulting solid.

![Figure 3.5 Toroidal revolution of profile $C$](image)

Let $C$ be a closed profile in the plane $P$, $A$ be an oriented line in $P$ that intersects $C$ in finitely many segments. Assume that no segment of $C$ is to the left of $A$, and that every component of $C$ intersects $A$ in at least one segment. The spherical revolution of $C$ is the solid obtained by revolving $C$, and its interior, about the axis $A$, by a positive angle not greater than 360 degrees. See also Figure 3.6. In case the revolution is by 360 degrees, the segments on $A$ are interior to the solid and resulting topology is a collection of spheres. If $C$ intersects $A$, the intersection points in general become nonmanifold points on the surface of the resulting solid.

In the following, we consider revolutions in which both spherical and toroidal topologies are generated from full revolutions.
3.2.1.2 Shape Attributes

Extrusions and revolutions are generated based on shape attributes. The simplest case is a blind extrusion or revolution:

1. *One-Sided and Blind, or Bi-Sided and Blind*

A blind extrusion is determined from an explicit dimension \( d \) specifying the depth of the extrusion. If the extrusion is one-sided, the solid is on one side of the plane \( P \). For a positive value of \( d \), this side is in the direction of the plane normal; for a negative value, the solid is on the opposite side. A bi-sided extrusion is one in which the extruded solid is bisected by the plane \( P \), with the solid extending by \(|d/2|\) to both sides of \( P \). The sign of \( d \) is irrelevant in this case.

A blind revolution is one in which the contour is revolved about an oriented line \( A \) by an angle \( \alpha \). For one-sided revolutions, a positive angle \( \alpha \) is counterclockwise about \( A \) as seen from the direction in which \( A \) points. Thus, for positive angles less than 180 degrees, the revolved solid lies on the side away from the normal of \( P \).
2. **From-To**

Intuitively, the *from-to* operation is a sweep that begins at a face or face plane designated as *from*, and ends at a face or face plane designated *to*.

3. **From-Next, Previous-To**

The *from* or *to* face or face plane is explicitly designated and is called the *explicit* face. The *previous* face is the face preceding the explicit face in the direction of the sweep, the *next* face is the one following the explicit face in the direction of the sweep.\(^1\) The operations are now like the *from-to* operations using a combination of explicit and implicit faces.

4. **FromAll-To, From-ThroughAll, FromAll-ThroughAll**

Here, *from all* means that all volumes preceding the *to* face or face plane are used, and *through all* means that all volumes following the *from* face or face plane are used, in the direction of the sweep. These two operations make sense only for extrusions. The meaning of *from all-through all* is means all volumes following the extruding direction or about the revolving axis.

Figure 3.7(a) illustrates a design in progress where a semi-circle has been sketched on a sketching face and constrained with the existing geometry to cut straight along the normal of the profile. Using "*from f to g*" as one of the example of above attributes, we express a feature operation that results in the geometry of Figure 3.7(b). The details of interpreting these attributes depend on the feature being a cut or a protrusion and belong to the subject of design evaluation in Chapter 5. We will revisit this example there.

A generated feature creates a new shape by sweeping a profile and relating the swept volume to the prior existing geometry. We describe the profile, the trajectory, and the attributes that define feature attachment. Clearly, most of this can be described independently of the core modeling system. Although perhaps not apparent at first, the cross section description must include how to display constraints, and

---

\(^1\)Strictly speaking, there need not be a single *previous* or *next* face, as discussed later.
must include an identification of which of the many possible solutions of the geometric constraints defining the profile. The profile description is therefore analogous to specifying a 2D drawing, and there are neutral standards that can be used for this purpose.

In addition to profile and trajectory descriptions, we need to identify specific shape elements of the prior geometry. These include a specification of the plane in which the profile must be constructed, faces that might limit the extent of the sweep, as well as projected faces, edges and vertices referenced for the purpose of defining the constraints on the profile and its relative position. Here, a good naming schema is required that identifies the shape elements without using the boundary representation; [6]. Moreover, the naming schema must be assessed in view of the editing operations allowed; [12].

3.2.2 Modifying Features

Adding rounds and chamfers are operations that modify the existing geometry in a way that is understood from a few parameters. For a round, a user selects a list of edges and vertices in the existing geometry to be rounded with a chosen radius. The user may have to specify also end-conditions that determine how the round
should terminate. Chamfers similarly require some shape parameters, in particular the parameters to define the chamfering triangle, plus a list of edges and vertices. Again, the unevaluated representation must record generic names identifying the selected edges and vertices.

3.2.3 Datum Features

Datums are auxiliary geometric entities that serve to define sketching planes, aid placing features and define sweep extents. Our system uses points, lines and planes that are defined, one at a time, by a set of 3D geometric constraints. These constraints reference prior existing geometry. Generic names substitute the referenced geometric entities.

3.2.3.1 Datum Points

We allow to define a datum point by the following constraints:

1. on a vertex
2. at the center of an arc
3. intersection of two datum axes, an edge and a datum axis or two edges
4. intersection of a datum axis with a plane, a datum axis with a face, an edge with a plane or an edge with a face

3.2.3.2 Datum Axes

A datum axis is defined by a root point and a direction vector. A root point can be moved along the direction. We allow to specify a datum axis by one or more of the following constraints:

1. through an edge
2. through a datum point or a vertex
3. perpendicular to a face or a datum plane (parallel to a datum axis or an edge)

4. parallel to a face or a datum plane (perpendicular to a datum axis or an edge)

Note that we exclude to use the intersection of two planes because possible inconsistent orientation of the defined axis may arise from the positional change of the two planes. If we define the orientation of such a datum axis by the rule of cross-product of two normals of the planes, the orientation of the datum axis will be flipped when the angle between the two normals moves across 180°. We have found the misuse in some of existing commercial systems.

A datum axis through an edge is consistently oriented because every edge is oriented in Erep. We discuss the orientations of edges in Chapter 4.

3.2.3.3 Datum Planes

A datum plane is defined by a root point and a normal. We allow to specify a datum plane by one or more of the following constraints:

1. through a face

2. offset from a face or a datum plane

3. through a datum point or a vertex

4. through a datum axis or an edge

5. parallel to a face or a datum plane (perpendicular to a datum axis or an edge)

6. perpendicular to a face or a datum plane (parallel to a datum axis or an edge)

7. angle to a face or a datum plane

As an example, \( P/25 \), a datum plane in Figure 3.8, is defined through an edge connecting \( V_1 \) and \( V_2 \), and by an angle of \( -30° \) from face \( f \).
Figure 3.8  A datum plane defined in Erep.
4. GENERIC NAMING SCHEMA

In feature-based and constraint-based design, users graphically select geometric entities of a graphical representation of a design in progress in order to specify operations that have generic intent. A generic representation of these geometric entities makes a design representation modeler-independent. Moreover, the significance of a robust naming schema extends beyond the purely geometric problem of automatically regenerating a design variant correctly. In particular, tolerancing, annotating parts of a design with boundary conditions for engineering analysis, and recording surface finish, are some of the many activities arising in manufacturing applications that also depend on a generic naming schema.

In this chapter we discuss techniques for naming algorithmically and generically the identified geometric instance under feature attachment operations.

4.1 Background

New CAD systems such as Pro/ENGINEER have become available in which geometric and dimensional constraints can be defined and solved. Such systems allow the user to instantiate generically defined models from user-supplied dimension values and constraints, and permit easy editing of the design in order to derive design variants, for example by changing dimension values. But the automatic generation of designs from constraints causes many semantic problems, [29, 30, 61]. Specifically, the user executes some design gestures that graphically interact with the currently instantiated geometric design. The system is then expected to abstract those gestures and produce from them a generic description of the geometric elements the user referred to.
For obvious reasons, design interfaces are centered around visual design gestures. While visual design gestures interact with images that are representations of instances of a generic model, their intent is to modify the generic model itself. However, geometric elements identified visually need not correspond explicitly to design gestures ever made, and to capture design intent on the basis of the explicit gestures is a profound challenge. Consider for example Figure 4.1 showing a design and its variant prepared with Pro/ENGINEER Version 12.0. The shape shown there has been designed in three steps:

1. A block has been created by drawing a rectangular profile and extruding it by a specified depth.
2. A round slot has been cut by extruding a circular profile across the block.
3. An edge round of constant radius has been defined by visually identifying one edge and specifying a radius for the round.

After so defining the model on the left, the dimension value locating the center of the slot profile is changed. On regeneration, the edge round “jumps” to a different edge.

![Figure 4.1 Persistent naming error in design variant.](image)

The basic problem is to identify the correct edge. The edge that was identified on the left corresponds implicitly to the intersection of the slot surface with the top
of the block. Both surfaces, in turn, are the trajectory of explicitly drawn geometric elements, and so can be represented generically. However, the edge on the right, to which the blend jumps after relocating the slot center, is also the intersection of the two surfaces. No generic way is immediately evident that could distinguish between the two edges. Since regeneration of the slot must precede the existence of the two edges in the new shape instance, any annotation of the previous model instance in which the edge was selected has been lost.

Prior work [29, 30, 61] has articulated the problem clearly. Others have hinted at the issue without putting it into sharp focus. For instance, [55] observes that to record a picked vertex in the log file, the mouse position is inadequate because, if the log file is edited later, the position may become meaningless. The paper proposes a naming schema derived from an underlying CSG expression, but does not give precise rules for how these expressions are to be constructed.

In [69], Várady proposes a syntax for defining topological expressions that serves to identify geometric elements and features in parts and assemblies. However, the method uses directives such as left, top, front, etc., which must rely on coordinate frames that are understood and remain invariant. This aspect makes the method unsuitable for constraint-based design in which no explicit coordinate system should be assumed.

Solano and Brunet propose in [62] a CSG-based Erep with constraint-based object instantiation. However, as their system description does not include operations that need to reference geometric elements which would arise from feature collisions, their description of the underlying language has no linguistic elements that address the persistent naming problem in the needed generality.

Kripac [34] gives a naming schema that is characterized by the immediate adjacency of an entity, its orientation in the boundary representation, and a graph recording the history of face merging and splitting. The naming schema is used for dimensional changes, and the matching algorithms are sophisticated. However, the
presentation omits a number of details, most notably under which circumstances exact matches are required and when approximate matches may be acceptable.

In the collection [33] a number of commercial CAD vendors have published papers that acknowledge the importance of the naming problem but give no details on the particular algorithms their respective systems implement. Existing problems such as the one illustrated in Figure 4.1 urge us to study a naming schema formally in order to build a feature-based and constraint-based design system.

In the following sections we define three naming mechanisms that we use in our experimental feature-based and constraint-based design system. These naming mechanisms are meant to be generic under feature attachment operations. Neither of these mechanisms is complete. We show in Section 4.4 how to combine them into a comprehensive naming schema. In a later chapter we describe in detail the matching algorithms for persistent names and their use when editing generic design.

4.2 Topology-Based Naming

All three feature classes in the Erep system require, for their definition, references to existing geometry, and those references must be made generic, so that, upon editing the design variables, the new design variant can be constructed completely automatically.

The Brep entities that need to be named persistently fall into two categories. One class of entities corresponds to geometry created explicitly as part of a feature operation. This includes virtually all faces and some edges and vertices. The second class includes entities that comes about through feature collision: Such entities are, essentially, edges and vertices in which different feature elements intersect. For example, the intersection edges of a slot and a block are in this category (see Figure 4.1).

Entities in the first class are uniquely associated with a single feature, whereas entities in the second class are associated with several features and are transient in the sense that, after editing, such entities could disappear or new ones could be created. Some examples of such changes are discussed in Chapter 6.
A suitable naming schema for geometric elements directly created by design gestures is not complicated. We describe such a schema first. Then, we concentrate on naming geometric elements that arise implicitly from feature collision, giving a topological naming schema.

4.2.1 Naming Created Geometric Elements

Generated Feature Elements

Generated features create geometry by sweeping a standardized or sketched cross section. The elements of the cross section can be named as follows: First, $v_1, v_2, \ldots$ will be points of the sketch. In a subsequent extrusion operation, these points will become vertices and edges. Next, $e_1, e_2, \ldots$ will be segments and arcs of the sketch. They become edges and faces in the extrusion. Finally, $f$ will be the area enclosed by the loops of the sketch. It will become two faces of the extrusion. The situation is depicted schematically in Figure 4.2. We have two instances of each $v_i, e_k$, and of $f$, for the front and the back face. Moreover, the side faces are named $f(e_k)$ and the side edges are named $e(v_i)$. The two instances can be ordered by the direction of the sweep and are thereby distinguished from each other.

![Figure 4.2](image)

Figure 4.2 Generic naming in an extrusion. Left: names assigned by sketcher.

Right: names for extrusion.
For revolute sweeps the same naming scheme is adopted, except that a full revolu-

tion has neither front nor back face, so that only side faces and edges are named in

that case.

Since the names $v_i$, $e_k$ and $f$ are generated by the sketch, they are made unique

in the global context by adding the feature name as qualification. For instance, $e_k$ of

feature $X$ is referenced as $X.e_k$.

*Merging Named Entities*

When generating the first feature, the vertices, edges and faces of the created

shape can be named by this scheme, and the names are unique for valid extrusions

and revolutions. Subsequent protrusions and cuts introduce geometry collision. The
detailed attachment semantics is explained in Chapter 5. For a generated feature, we
first create a proto feature which is a sufficiently large extrusion or a full revolu-

tion. The proto feature is then analyzed and trimmed before being added to or subtracted
from the existing geometry. The proto feature is named in the manner defined before.
The attachment steps result in new intersected, split and merged geometric entities.

We postpone the discussion of naming intersected and split entities. When en-

tities are merged it is ambiguous how to name the merged geometric elements. We
uniformly name the merged entity from the geometry that exists prior to attaching
the feature: When two vertices are coincident, the new vertex is named from the
existing geometry. When two edges overlap and are merged into one, the new edge is
named from the existing geometry. When two faces overlap and are merged, the new
face is named from the existing geometry. In all other cases, the geometric elements
of the new feature are logically distinguishable from existing geometry and so keep
the name of the proto feature. It is also possible to name the merged entity by all
the merging entities. We will have more discuss on the choice in Chapter 6.
Datum and Modifying Features

Datum features are named serially in the order of creation. If the intersection of two datum features is to be referenced, then an explicit feature creation operation will be needed [32] and a name will have been associated with the feature. The same holds for the intersection of a datum with existing geometry.

Chamfers and rounds replace selected edges and vertices with new faces. We label the face replacing edge \( e \) or edges \( l_e \) with \( c(e) \) or \( c(l_e) \) in the case of a chamfer and with \( r(e) \) or \( r(l_e) \) in the case of a round. The bounding edges and vertices are named exactly in the same way as intersecting edges and intersecting vertices.

4.2.2 Topological Context Computation

A geometric model instance named in the manner described has a number of elements that have the same name. These include not only intersected elements, such as intersection edges, \( I_e \), and intersection vertices \( I_v \), but also the edges and faces that have been subdivided by feature collisions. To distinguish between them, we determine their topological context.

The adjacencies of the geometric elements form a context graph. By convention, a face is adjacent to edges, an edge adjacent to vertices and faces, and a vertex to edges. In many cases the context graph is essentially the adjacency graph of the Brep\(^1\). But, in some cases, several faces of the Brep are grouped into a single logical face and then the context graph is a reduction of the adjacency graph. Note that the context graph is a labeled graph. For example, consider the design of Figure 4.3. The context graph is shown in Figure 4.4. The types of the graph nodes — vertex, edge or face — are indicated by the enclosure: a face node has a rectangle, an edge node a circle enclosing it. Vertex nodes are not enclosed.

We define the immediate context of a graph node to be its star; i.e., the nodes immediately adjacent. For example, the immediate context of \( f \) consists of the six edge nodes labeled \( e_1, e_2, e_3, e_5, e_3, \) and \( e_4 \). Note that \( e_3 \) occurs twice. There are two

\(^1\)Face to vertex adjacency is ignored.
Figure 4.3  Labeling of object with two features. For simplicity, the names omit the feature qualifier.

Figure 4.4  Context graph for object of Figure 4.3. Faces indicated by boxes, edges by circles.
edges of the object of Figure 4.3 labeled $I_e$. Their immediate context consists of the faces $f(e_3)$ and $f(e_5)$ and two vertices labeled $I_v$. It is clear that the edges cannot be distinguished by their immediate context. However, the immediate context of the two adjacent faces differs. One of the $f(e_3)$ faces is adjacent to $e(v_4)$, whereas the other one is adjacent to $e(v_3)$. Consequently, the two occurrences of $f(e_3)$ can be distinguished, and therefore also the two $I_e$ edges.

We compute the extended context of a geometric element by a breadth-first graph search. The algorithm has the following structure:

1. The extended context of level 1 is the immediate context.

2. The extended context of level $k + 1$ is the extended context of level $k$ plus those elements of the immediate context of any node at level $k$ that are not yet in the extended context.

After determining the extended context to some depth, we will be able to differentiate some of the geometric elements of the same type that have the same name. In particular, all elements of the example in Figure 4.3 can be distinguished by contexts of depth 2. The algorithm is:

*Full Topological Context Description*

*Input:* Context graph $G$ and a node $r$ of the graph; depth $d$.

*Output:* Context description of the node.

*Method:*

1. Let $R$ be the set of nodes that have the same label as $r$.
2. If $R$ contains only $r$ then stop; $r$ is unique without context.
3. Otherwise, for every $v$ in $R$ do the following:
4. By breadth-first search, compute the subgraph reachable from $v$ in up to $d$ steps.
5. Label each subgraph node $u$ by its distance $d_v(u)$ from $v$.
6. Partition the nodes in the subgraph into equivalence classes, where two nodes are equivalent if they have the same label
and the same distance from $r$.

7. The description of a node $v$ in $R$ is the class structure so computed.

Note that a node can be distinguished from $r$ if it has a different equivalence class structure. The context description can be textually encoded as follows: Each equivalence class is represented by a triple $[d, l, n]$, where $d$ is the distance from $r$, $l$ is the label, and $n$ is the number of nodes in the class. For instance, the two edges labeled $I_e$ in Figure 4.3 have the following descriptions:

Left edge:

$[0, I_e, 1],$

$[1, I_e, 2], [1, f(e_3), 1], [1, f(e_5), 1],$

$[2, e(v_4), 1], [2, e_3, 1], [2, e_5, 1], [2, I_e, 1], [2, e_5, 1], [2, e_3, 1]$

Right edge:

$[0, I_e, 1],$

$[1, I_e, 2], [1, f(e_3), 1], [1, f(e_5), 1],$

$[2, e(v_3), 1], [2, e_3, 1], [2, e_5, 1], [2, I_e, 1], [2, e_5, 1], [2, e_3, 1]$

The two edges can be distinguished because of the two different classes $[2, e(v_3), 1]$ and $[2, e(v_4), 1]$.

Rather than storing the full context description, we could delete from it those classes that are not distinguishing and that do not contain nodes that are in the context graph on the path from $r$ to a node belonging to a distinguishing class. We call this the reduced context. For example, the reduced context of the two edges is

Left edge:  

$[0, I_e, 1],$

$[1, f(e_3), 1],$

$[2, e(v_4), 1]$

Right edge:  

$[0, I_e, 1],$

$[1, f(e_3), 1],$

$[2, e(v_3), 1]$
Note that the reduced context sacrifices some resolution when matching in design variants.

4.2.3 Remarks

Since features are evaluated in a fixed sequence, the context of a referenced element is evaluated for the geometry that exists at the time the new feature is created. Thus, the context graph of feature \( k \) is based on the geometry created by features \( 1 \) through \( k \) only. Later features may well obliterate geometric elements used to construct earlier features, yet they can be referenced at the earlier times.

The algorithm for distinguishing geometric elements by their topological context is essentially a spectral graph isomorphism algorithm (see [27]). The context computation determines a spectrum, and the reduced context is a certificate. Graph nodes with distinct certificates can then be distinguished.

It is well known that spectral algorithms do not solve the graph isomorphism problem and that they fail on highly regular graphs; [27]. It is easy to construct an example in which no topological context can distinguish some geometric features: Consider Figure 4.5. A toroidal hole has been cut into a block. The two circular edges bounding the cut cannot be distinguished, because the context graph has a structural symmetry that exchanges the two edges.

4.3 Orientation Information

The topological context information discussed above essentially encodes the adjacency structure of feature collision. In addition, we will consider orientation information that is based on the way geometry has been created and the way features lie with respect to each other geometrically. The directional information derived from the feature creation process is called feature orientation, and the orientation information locally derived from the geometry of the boundary elements is called local orientation.
Figure 4.5 Block with toroidal hole. The edges of the cut cannot be distinguished from the context graph.

4.3.1 Local Orientation

Local orientation information is based on the direction in which an adjacent face uses an edge, and on the face orientation in the solid boundary. By convention, a face is oriented so that the surface normal locally points to the solid exterior, and an edge is used in a direction such that locally, when viewed from the exterior of the solid, the face’s interior is to the left of the edge; e.g., [28].

Distinguishing Edges with Local Orientation

An edge use by an adjacent face orders the vertices of the edge, assuming that the edge is not closed. The vertices can be designated explicitly by their names, so that the edge use orientation can be encoded by a triple consisting of the face name and the names of the two vertices in order. In Figure 4.6 (left), the edge $e$ is used by $f_1$.
Figure 4.6 Computation of local orientation of an edge.

from $v_1$ to $v_2$, and by $f_2$ from $v_2$ to $v_1$. The edge use can be encoded by the triples

$$L_e = [f_1, v_1, v_2] \quad \text{or} \quad L_e = [f_2, v_2, v_1]$$

In most cases, we may use equivalently terminating faces, i.e.,

$$L_e = [f_1, f_3, f_4] \quad \text{or} \quad L_e = [f_2, f_4, f_3]$$

Now consider Figure 4.6 (right). Edge $e_1$ is used by the adjacent face $f_2$ in the direction $f_4$ to $f_3$, whereas $e_2$ is used in the opposite direction:

$$L_{e_1} = [f_2, f_4, f_3] \quad L_{e_2} = [f_2, f_3, f_4]$$

Therefore, $e_1$ and $e_2$ can be distinguished by the local edge orientation with respect to $f_2$. Of course, the edges can also be distinguished by the local edge orientation with respect to $f_1$.

Note that the orientation of closed edges cannot be recorded by triples. For those edges an orientation comparison is needed based on comparing the edge use direction with an intrinsic orientation induced by the feature creation, as explained later. See also Figure 4.11 below. Local orientation cannot distinguish the intersection edges of Figure 4.5.

**Distinguishing Vertices with Local Orientation**

Consider the vertex $v$ shown in Figure 4.7 (left). Generically, a manifold vertex is adjacent to three faces which are the direct context of the vertex. Ordering the faces
cyclically, with $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ the surface normals at the vertex, we may define the \textit{local orientation at $v$} as the sign of the mixed product of the normals in that order:

$$S_v = \text{sign}((\mathbf{n}_1 \times \mathbf{n}_2)\mathbf{n}_3)$$

Then $S_v$ is positive if the three vectors form a right-handed coordinate system, and negative if they form a left-handed one. If two surfaces meet with tangent continuity at the vertex, then $S_v$ is zero.

![Figure 4.7 Computation of local orientation of a vertex.](image)

The quantity $S_v$ essentially expresses the local geometry at a vertex where three adjacent faces meet transversally. This schema can be extended to vertices with more than three adjacent faces. However, it is not clear whether this geometric information should be expected to remain invariant under a broad class of feature editing operations, so we do not use it. Instead, we record whether a vertex is a manifold or a nonmanifold vertex, and if it is a manifold vertex, we describe it locally by a list of incident faces, ordered cyclically counter-clockwise, as seen from the vertex exterior. Thus, the four vertices in Figure 4.7 (right) have the descriptions

$$L_{v_1} = [f_1, f_3, f_2] \quad L_{v_2} = [f_1, f_2, f_3] \quad L_{v_3} = [f_1, f_2, f_4] \quad L_{v_4} = [f_1, f_4, f_2]$$

They can be distinguished if $f_3 \neq f_4$. However, if $f_3 = f_4$, then $L_{v_1} = L_{v_4}$ and $L_{v_2} = L_{v_3}$. An example of this situation is shown in Figure 4.8. To distinguish these four vertices, other information must be used as explained later.
4.3.2 Feature Orientation

In the example of Figure 4.5, the cut was created by revolving a circular profile. The direction of revolution defines an orientation of the surface that can be derived unambiguously from the design gestures and can be thought of as a vector field that is tangent to the surface and perpendicular to the axis of rotation. See also Figure 4.9. We call this orientation the feature orientation since it is defined by the feature creation, and not by the geometric characteristics of the instantiated feature.

Feature orientation information can be defined based on the direction of extrusion or the spin of revolution. It amounts essentially to observing the (local) motion of the sweep of the 2D cross section as the 3D proto feature is created. In the case of extrusions, the extrusion direction is the feature orientation. In the case of revolution, the feature orientation at any point is defined by the instantaneous rotation vector at that point. At singular points, where the rotation axis intersects the revolved surface, the axis orientation can be used instead. Note that the axis orientation and the direction of the revolution are by convention related by a right-hand rule as illustrated in Figure 4.9.
Distinguishing Edges with Feature Orientation

Feature orientation induces a direction of adjacent edges. For edges such as the straight boundaries of the slot in Figure 4.8, the orientation directs them from $v_1$ to $v_3$ and from $v_2$ to $v_4$, assuming the slot was an extrusion of a circular profile from the front to the back. Loosely speaking, one could call such an edge *aligned* with the orientation.

More precisely, consider an extrusion or a partial revolution, as seen in Figure 4.10. Its topology consists of cylinders that could be pinched along edges that are the

Figure 4.9 Orientation of torus surface created by revolving a circular profile.

Figure 4.10 Curve types on a cylindrical topology.
trajectory of nonmanifold vertices of the swept profile. In the case of such nonmanifold topology, we cut the profile at nonmanifold vertices and consider the resulting cylinders separately. Three types of curves must be considered:

1. Curves that cut the cylinder such that we can unroll the surface into a rectangle.

2. Curves that cut the cylinder into two separate cylinders.

3. Curves that separate the cylinder into a cylinder with a hole and the hole interior which is essentially a disk.

Type 1 is an aligned edge, and can be directed by the orientation of the cylinder surface. See Figure 4.8 for an example. Note that edges of Type 2 must occur in pairs, each source edge having a corresponding sink edge. The correspondence is established by the direction of the sweep. Type 2 is an edge where the adjacent cylindrical surface, used in the feature defined, either has a source or a sink at the edge. See Figure 4.5 for an example. Type 3 is an edge where the adjacent surface has both a source and a sink on the edge. If the axis of the toroidal cut were lifted in Figure 4.5, the two delimiting edges would merge into a single edge of type 3. The same types can be defined analogously for full revolutions with spherical or toroidal topologies.

Using feature orientation information, we can distinguish the two circular edges of the Figure 4.5. Both are type 2, and at one of them the orientation of the toroidal face has a source, whereas at the other one it has a sink. If more than two such edges are present, they can be sorted by the direction of the feature orientation and distinguished positionally in the order in which they lie with respect to the feature orientation. Note that for toroidal topologies the ordering is cyclical, whereas for partial revolutions or extrusions a total order is imposed.

In Figure 4.11, edges $e_1$ and $e_2$ are both of type 1 and are oriented in the same way by the direction of revolution. They can only be distinguished by observing that face $f$ uses $e_2$ in the feature orientation direction, whereas $e_1$ is used by $f$ in the opposite direction.
Figure 4.11 Edges $e_1$ and $e_2$ are not distinguished by feature orientation alone.

Two closed edges of type 3 cannot be distinguished using feature orientation except in rather special cases where a cutting plane can be defined that induces a separating curve of type 2. This separating curve allows us to sort edges of type 3. However, type 3 edges arise from feature collisions where the other, intersecting surface usually allows differentiation.

The feature orientation of an edge is recorded by a triple $[F, f, c]$. $F$ is the feature that defines the feature orientation; $f$ is a face of the feature that is adjacent to the edge. If the edge is of type 1, then $c = \pm$ according to whether the edge is used by $f$ in the direction of the feature orientation or in the opposite orientation. If the edge is of type 2, then $c = \text{in/out}$, where ‘in’ means that the feature orientation flows from the face exterior across the edge into the face interior, whereas for ‘out’ the orientation flows from the interior of $f$ to the exterior. If the edge is of type 3, then $c = 0$ and no information is obtained.

The feature orientation information for edges of type 2 can be refined further by ordering all edges of type 2 in the direction of the feature orientation. For extrusions, this is a total ordering, but for revolutions with a toroidal topology the ordering is only cyclical. In such a schema, an expression of the form $[F, f, m, n, q]$ could be used, where $F$ and $f$ are as before. Moreover, the edge is the $m^{th}$ of $n$ edges of type 2, and $q$ designates whether the ordering is cyclical or total.
Distinguishing Vertices with Feature Orientation

Feature orientation does not add a separate naming mechanism for vertices. However, the vertex \( v \) may be incident to an edge \( e \) that has a feature orientation. If \( e \) is of type 1, we observe whether \( e \) begins or ends at \( v \). The name of the vertex is then \([e, +]\) or \([e, -]\).

For edges of type 2, this approach does not apply, but we can derive similar information by a slightly more complicated computation. Instead of using the edge tangent, we can use the feature direction vector at the vertex. For edges of type 1 we have conceptually observed whether the inner product of this vector with the edge tangent is positive or negative. Instead of using the edge tangent, we use the normal of an incident face not adjacent to the edge. Consequently, the vertex feature orientation encoding can be uniformly expressed by \([F, f', \pm]\), where \( F \) is the feature whose orientation we use, \( f' \) is an incident face that is not adjacent to an intersection edge of the feature \( F \), and \( \pm \) records the sign of the inner product. Note that \( f' \) must be selected such that the inner product does not vanish.

For example, consider vertex \( v_1 \) of Figure 4.8 right. Assuming that the round slot is generated as feature \( F \) with an extrusion from the front to the rear, \( v_1 \) is labeled \([F, f_3, -]\).

4.3.3 Combining Local and Feature Orientation

Intersection features are named by a combination of local and global orientations, because both types of orientation provide only limited resolution. In addition, the immediate topological context is added. The combined information is recorded in a formal expression has the following properties:

1. The expression is canonical. If one entity name uses one face, another entity uses a different face when in both the same face could have been used, then an ambiguity could go unnoticed. We avoid this by exploiting the serial creation of features and by using set expressions.
2. The expressions are not circular. If a name for edge \(e\) uses a vertex \(v\), then the name of \(v\) does not use edge \(e\). We avoid this by using only face names in formal expressions.

3. Where reasonable, we use the generated names explained in Section 4.2.1.

The naming expressions that combine this information are composed of three subexpressions, the ordered, immediate context, the local orientation information, and the feature information. Naming expressions may be nested.

**Naming Intersection Edges**

The name is an expression of the form

\[ E_e = [C_e, L_e, F_e] \]

where \(C_e = [f_1, f_2, \ldots]\) are the names of all faces adjacent to the edge, cyclically ordered. There is no direction in which the cycle is understood. For example, \([f_1, f_2, f_3, f_4]\) and \([f_4, f_3, f_2, f_1]\) are not distinguished. This is because not every edge has an intrinsic direction that can be defined generically. If edges have an intrinsic orientation derived from the feature orientation, then the cycle direction can be inferred from the feature orientation subexpression.

\(L_e\) is the local orientation, which has the form \([V, W, s]\) if the edge is open. For closed edges we have \(L_e = [\].\) Here \(V\) and \(W\) identify the two vertices bounding the edge. If \(s = +1\) then \(f_1\) uses the edge from \(V\) to \(W\), and if \(s = -1\), then \(f_1\) uses the edge from \(W\) to \(V\). Since the \(F_i\) are cyclically ordered, it follows that the faces \(f_1, f_2, \ldots\) use the edge in the orientation \([s, -s, s, -s, \ldots]\). If the vertex identified by \(V\) is a created vertex, then \(V\) is the name generated for that vertex as described in Section 4.2.1. Otherwise, \(V\) is an expression identifying an intersection vertex as described later.

\(F_e\) is the feature orientation of the edge. It has the form \(F_e = [F, H]\) where \(F\) is a feature and \(H\) is a set of feature orientation expressions. \(F\) is the latest feature one
of whose faces is adjacent to the edge. Let \( h_1, h_2, \ldots \) be all faces belonging to \( F \) that are adjacent to the edge. Then \( H = \{ [h_1, c_1], [h_2, c_2], \ldots \} \), where \([F, h_i, c_i]\) is a feature orientation triple of the edge.

**Naming Intersection Vertices**

The name of a vertex is an expression

\[
E_v = [C_v, L_v, F_v]
\]

Here \( C_v = \{ f_1, f_2, f_3, \ldots \} \) is the set of incident faces. It is not ordered if \( v \) is a nonmanifold vertex. For manifold vertices, there is a cyclical ordering of the \( f_i \) that is counterclockwise as explained before. If \( L_v = M \), then the vertex is manifold, if \( L_v = N \), then the vertex is nonmanifold.

The expression \( F_v = [F, H] \) is the feature orientation information consisting of a feature name \( F \) and a set \( H \) of orientation informations. For every face \( f \) incident to the vertex but not part of the feature \( F \), \( H \) contains the expression \([f, s]\), where \([F, f, s]\) is the feature orientation as explained before. Note that \( s = 0 \) is allowed.

### 4.4 Implementation

Both the context graph and the orientation information eliminate ambiguities, but neither technique is complete. Therefore, we have implemented a combination of the two methods. In particular, intersection elements are first encoded using the formal expressions derived in Section 4.3.3. These expressions include the immediate context. We add to this information the reduced topological context, to some predefined maximum depth. Because both the orientation information and reduced context is canonically organized, the derived naming expressions remain canonical.

We note that ambiguities are not always resolvable by our method. Structural symmetries can be constructed that foil our naming schema. However, such highly
symmetric situations seem to occur rather rarely in practice. When they arise, moreover, choosing arbitrarily between several entities with the same name is often acceptable. In particular, when a straight edge is referenced as a boundary of a distance dimension, any edge that has the same underlying line equation meets this reference need. In such cases, certain ambiguities are allowed.

Remaining ambiguities for different classes of entities, if existing, are uniformly handled as follows. If the operation is a modifying feature, such as a chamfer or blend, then the operation is applied to all geometric entities with the same name. If the operation requires exact identification in a dimensioning schema, then the user has to change the dimensioning schema.

4.4.1 Vertices

Vertices that are not the result of feature collision are named as explained in Section 4.2.1. Evidently, such names are unique. Intersecting vertices, on the other hand, are named first with the expression $E_v$ explained before. If a vertex is not uniquely named by the expression, then the reduced context is computed.

4.4.2 Edges

Edges are also classified as intersecting edges and non-intersecting edges. Intersecting edges are first named with the expression $E_e$ explained in Section 4.3.3. If ambiguities remain for identifying an edge, the reduced context is computed.

A nonintersecting edge is named similarly, except that the immediate context of a non-intersecting edge is replaced by the inherited edge name of the parent edge in the proto feature.

4.4.3 Faces

All faces are generated as faces of proto features or modifying features, or are a subdivision of such faces. Furthermore, different faces may be merged if they are on the same underlying geometric surface and overlap. A face name is then an expression
that contains a part $H_0$ describing the origin of the face. If the face is from a proto feature, $H_0$ is one of the following

$$(\text{start}_f \ \text{feature}.f2D), (\text{end}_f \ \text{feature}.f2D), (\text{side}_f \ \text{feature}.f2D)$$

If the face is from a chamfer, then $H_0 = c(l_e)$, where $l_e$ is a description of the edges chamfered. If the face is from a round, then $H_0 = r(l_e)$. When faces are merged, then this name is inherited from the face of the earliest feature involved in the merge. In addition, the face name contains an expression naming every adjacent edge of the face.
5. DESIGN COMPILATION

The unevaluated design representation, formalized and justified in Chapter 3, does not contain a geometric representation which is needed in intermediate design process and as a final design. A design compiler serves to evaluate this modeler-independent design representation to a concrete geometric model. In this chapter we describe design evaluation or compilation and the implementation is based on the commercial solid modeler ACIS [63].

To compile an Erep description with respect to a core modeler, we interpret each feature definition by a sequence of modeling operations or auxiliary semantic computations. The modeling operations are done by a core modeling system, in our case ACIS. Auxiliary computations include maintaining tables and solving constraint problems. They are done by the design compiler itself. Datum features are managed by the compiler, but could be instead managed by the core modeler. Communication between the design interface and the design compiler is through a textual description, ordinarily a temporary file. Although this impacts speed, it does ensure a complete formalization of the information flow.

An Erep segment for a feature is automatically written by the design interface when a user completes his/her feature specifications including constraints and feature attachment attributes explained in Chapter 3. Such Erep segments are serially accumulated and archived for a design. Our design compiler evaluates features in the one-by-one manner, primarily because the geometry after interpreting a feature operation is needed to specify and evaluate the features attached later. Generally speaking, three distinct steps are carried out to evaluate the modeler-independent design representation for each of the three feature classes: matching the generic names which refer to geometric references [13, 12]; defining and placing the feature by solving
specified constraints; and attaching the feature based on an unambiguous semantics [13]. All three steps are required for evaluating a generated feature but only two steps are needed for evaluating a modifying feature or a datum feature. A modifying feature does not use constraints while a datum feature does not change or interact with the geometric representation of a design. Table 5.1 summarizes which of these steps are needed for each class of features. If a new feature class is introduced in this feature-based and constraint-based design paradigm, these three steps apply for its evaluation. We describe these three steps in the following sections emphasizing the attachment operations for generated features.

<table>
<thead>
<tr>
<th></th>
<th>generated feature</th>
<th>datum feature</th>
<th>modifying feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>matching names</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>constraint solving</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>attachment</td>
<td>×</td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

Table 5.1 Compilation steps involved in three classes of features

5.1 Matching Generic Names

We have described in Chapter 4 how to derive generic names of referenced geometric entities under feature attachment operations. In order to evaluate the modeler-independent design representation, we reverse the naming process and find corresponding geometric entities in the core modeler for these generic names. For instance, a rounding feature in Erep records a generically named edge and radius. We must find the edge in the ACIS modeler before performing the rounding operation.

Conceptually, we can carry out the naming schema described in Chapter 4 on an ACIS model and compare the generic names between the feature specifications and
the existing ACIS model. Clearly, such process is inefficient. Instead, with the help of the ACIS attribute mechanism, we procedurally interpret a generic name constructed with the naming schema described in Chapter 4.

Matching after editing features can be quite complicated because even simple changes of dimension values can have complex shape implications. We postpone the detailed matching caused by editing to Chapter 6 where different aspects of editing designs are studied.

5.2 Solving Constraints

Profiles are instantiated by solving geometric constraints. There is an extensive literature on geometric constraint solvers, and we use the method described in [5]. Finding a neutral representation of the constraint problem is rather straightforward. However, recording the correct solution to the problem is difficult.

Mathematically, a geometric constraint problem is equivalent to a system of non-linear algebraic equations and has, therefore, more than one solution. Commercial constraint solvers apply proprietary heuristics that are not always good, but since the heuristics are proprietary, it is difficult to characterize the domain in which the rules succeed. Academic research has not addressed the issue as far as we know.

As explained in [5], we select the solution obtained by applying a few simple design rules. In view of the difficulty to anticipate the user's intention, especially when editing designs, we give the user the option of overruling the solution selection. Since our solver finds solutions by carrying out a sequence of simple construction steps, the particular solution is expressed in terms of the construction sequence and the choices at each construction step. A theoretical analysis of the appropriateness of this approach is given in [25].

This style of recording solutions leaves open the important problem of translating this record so that other constraint solvers working on different principles can find the same solution. In principle, one could enumerate all solutions. However, since the number of solutions of a constraint problem is exponential in the number of
constraints, this approach is unacceptable in practice. More research is needed before this problem can be addressed effectively.

The constraints for defining datum features represent a small set of 3D parametric constraints. We solve them inside the compiler with algebraic equations. The orientation of new datum axes and planes are carefully constructed based on the rules defined in 3.2.3.

5.3 Performing Feature Attachment

Generated features and modifying features require a feature attachment step. For a modifying feature, attachment is directly converted to a Brep modification operation. Rounding and chamfering operations have become common in most commercial solid modelers available today. We focus in this section on the semantics of attachment operations of generated features.

We define the feature operations of cut and protrusion, paying close attention to the possibility that the conceptual view of the designer, formed by a visual design interface, does not necessarily match the technical view a system implementor has of them.

It is convenient to think of the two feature operations as being synonymous with regularized Boolean operations, and we explain their semantics using this vocabulary. Roughly speaking, a cut is a regularized volume subtraction from existing geometry and a protrusion is a regularized union. However, we note that the operations need not be so implemented, and that the semantic properties to be defined encourage a mix of Booleans and boundary-based operations instead.

Conceptually, generated feature operations can be thought to have two phases (see Figure 5.1). In the first phase, a blind extrusion of sufficient extent is computed, thereby obtaining a proto feature. The proto feature is intersected with the existing geometry. The result is a set of volumes \( S_i \) and their relative (regularized) complements \( S_c \) with respect to the proto feature. From these volumes the final operation is defined. For instance, if the feature is a from-to feature, then we select those volumes
(a) existing geometry to be cut

(b) proto feature

(c) intersection with existing geometry; complement volumes not shown.

(d) subtraction from the existing geometry with the selected volumes

Figure 5.1 Two phases of feature attachment.
or their complements that include the ones bounded in part by these faces and those that lie “in-between.” The semantics of “in-between” has to be defined with care and depends on the geometry of the selected faces and on the nature of the feature operation. In Figure 5.1(a), we specify the curved face to be the from face and the rightmost face to be the to face. Then the Figure 5.1(d) is the geometry after the cut. By decomposing volumes we make the semantic definitions unambiguous in generated features where an intuitive interpretation of the attachment attributes is not immediately obvious. This point is taken up again later.

5.3.1 Explicitly Bounded Features

5.3.1.1 Blind Features

Blind features are semantically straightforward. In essence, they are not different from the customary CSG design vocabulary. Blind cuts and protrusions are semantically the corresponding Boolean operations using the explicitly dimensioned extrusions or revolutions as defined before.

5.3.1.2 From-To Features

The extent of the sweep is implied by the designated from and to faces or face planes. In either case, the direction of an extrusion must be known explicitly and determines how the from and the to faces or planes are used.

*Plane Delimiters*

In the case of extrusions bounded by datum or face planes, we require that neither the from-plane nor the to-plane is orthogonal to the sketching plane. If the two planes are not parallel, then they bound four wedges of space. One of the wedges is candidate for defining the precise feature extent, and is determined by the following rules. See also Figure 5.2. In this figure, the wedge used is the lower one because the direction of the extrusion is left-to-right.
1. Consider the two-sided infinite extrusion of the profile. The two wedges whose intersections with this extrusion are infinite are not used.

2. Of the remaining wedges, use the one whose from-plane boundary precedes its to-plane boundary in the extrusion direction.

The feature semantics for the case of from-to planes is defined as follows: The proto feature is the intersection of the wedge so identified with the infinite, two-sided extrusion of the profile. A cut is the regularized subtraction and a protrusion is the regularized union of the proto feature with the existing geometry.

In the case of a revolved feature with from and to planes we require that neither the from-plane nor the to-plane is orthogonal to the sketching plane. Consider the revolution of any point not on the axis of rotation about \( A \) in the designated orientation. Then the trajectory is a circle that is divided, in general, into four arcs by the two planes. The arcs are oriented, and two of them start at an intersection with the from-plane and end at an intersection with the to-plane. See also Figure 5.3. The two wedges in which these arcs lie, intersected with the full revolution of the profile define the proto feature.
Figure 5.3 The two wedges used for revolved features delimited by \textit{from} and \textit{to} planes are shown shaded. Revolution seen in a plane perpendicular to axis $A$ and from the direction in which $A$ points. $Q$ is the intersection of the \textit{from} and \textit{to} planes, $p$ a generic point revolved about $A$.

A cut is now the regularized subtraction and a protrusion is the regularized union of the proto feature with the existing geometry.

\textit{From-To Face}

In contrast to \textit{from-to} plane feature definitions, face-based feature delimiters are defined based on the volumes in $S_i$ and $S_c$, where the intersection volumes in $S_i$ are used for cuts, and the complement volumes in $S_c$ are used for protrusions. The different conceptualization becomes necessary in view of the difficulties to define how to extend curved faces when the \textit{from} or the \textit{to} face do not completely intersect the proto feature. This will be further discussed later.

We explain the semantics of the feature operations for extrusions and revolutions assuming that the profile $C$ has a connected interior. If $C$ bounds an interior that has several components then every component is considered separately using these rules.

Let $S_f = S_i$ in the case of cuts, and let $S_f = S_c$ in the case of protrusions. We define the semantics of the feature operation using the set $S_f$ of volumes. The direction of sweeping must be explicitly designated by the user.
Let $S_{\text{from}}$ be the set of volumes in $S_f$ whose boundary contains a nonzero area of the \emph{from} face.\footnote{In the case of protrusions, we require that such volumes be finite.} We consider volumes as separate if their interior is not connected, and require that the set $S_{\text{from}}$ be a singleton. Similarly, let $S_{\text{to}}$ be the set of volumes in $S_f$ bounded in part by a nonzero area of the \emph{to} face.\footnote{In the case of protrusions, such volumes also must be finite.} This set also must be a singleton. We partition the set $S_f$ into the following:

1. $S_{\text{from}}$, containing the \emph{from} volume.
2. $S_{\text{to}}$, containing only the \emph{to} volume.
3. $S_{\text{in}}$, containing volumes that are “in-between” the \emph{from} and the \emph{to} volumes.
4. $S_{\text{out}}$, containing all remaining volumes.

The set $S_{\text{in}}$ is defined differently depending on whether the bounding faces completely intersect the proto feature or not. Figure 5.4 illustrates the intuition of a complete intersection.

![Image](from face)

Figure 5.4 From face completely intersects swept volume.

More precisely, if the trajectory of every point of the contour $C$, and its interior, intersects a bounding face, then the bounding face completely intersects the proto feature. For complete intersecting faces it is straightforward to define the volumes $S_{\text{in}}$. Let $V$ be a volume in $S_f - S_{\text{from}} - S_{\text{to}}$, $p$ a point in $V$. If $p$ is inside the volume bounded by \emph{from} face, \emph{to} face and their trimmed swept boundary faces, then $V$ is in...
$S_{in}$; otherwise $V$ is in $S_{out}$. Note this is the typical point classification with respect to a volume. A simple solution is to fire a ray, count the number of intersections and give the result based on whether the number is odd or even, duly considering degenerate intersections. See also Figure 5.5 left. We note that if one point of $p$ satisfies the condition, then all must because the bounding faces intersect the proto feature completely.

![Diagram](image.png)

**Figure 5.5** Definition of a volume in set $S_{in}$. Left: completely intersecting bounding faces. Right: partially intersecting bounding faces.

In the case of extrusions where one or both of the bounding faces have partial intersections with the proto feature, the semantic definitions are more technical. As before, we require that the sets $S_{from}$ and $S_{to}$ be singletons. Let $B$ and $E$ be two planes perpendicular to the direction of extrusion that box the area a bounding face. That is, that part of a bounding face is boxed that is on one of the bounding volumes. See also Figure 5.5 right. Then a volume is in $S_{in}$ if all its interior points $p$ are preceded by the $E$ bound of the from face and precede, in turn, the $B$ bound of the to face.

The semantics of extrusions is now as follows: Let the feature volume set $V_F$ be $V_F = S_{from} \cup S_{in} \cup S_{to}$. Then a cut is the regularized difference and a protrusion the regularized union of the existing geometry with $V_F$. For instance, the cut defined as shown in Figure 5.2 will extend through the central half cylinder.

In the case of revolutions, the semantics requires replacing the notion of preceding and succeeding by the corresponding ordering along a circular trajectory in the
orientation of revolution. Furthermore, the boxing planes $B$ and $E$ are half planes that are bounded by the axis of revolution $A$.

5.3.2 Implicitly Bounded Features

The attributes from all, through all, previous, and next are implicit ways to define the extent of a sweep. Their exact meaning depends on the existing geometry, on the type of the feature operation, and on the direction/orientation of the sweep. Implicit bound designations must be paired with explicitly named faces or planes. We impose a number of restrictions to limit degeneracies.

As before, we explain the semantics of the feature operations in terms of the volumes in the sets $S_i$ and $S_c$. Again, the set $S_f$ is either $S_i$ for cuts or is the set $S_c$ in the case of protrusions.

**Previous and Next**

*Previous* implicitly defines the from face of a feature extent and must be paired with an explicit to face. We require that the to face intersects the proto feature completely, and, as before, that the set $S_{to}$ is a singleton. The implicit from face need not intersect the proto feature completely. Possible ambiguities are resolved by the requirement that $S_{to}$ is a singleton. The set $V_F$ is then defined by

$$S_{from} = S_{to} \quad S_{in} = \emptyset$$

Thus, in the previous-to combination only one volume in $S_f$ defines the feature.\(^3\) The semantics is now as in the explicit from to case.

*Next* is symmetric to previous and designates implicitly the to face. Here, the from face must be designated explicitly, and must intersect the proto feature completely. Again, the implicit to face need not intersect the proto feature completely, and the feature volume set $V_F$ is defined by

$$S_{from} = S_{to} \quad S_{in} = \emptyset$$

\(^3\)Recall that each component of the proto feature is considered separately.
For an example of the operation see Figure 5.6. Note that the protrusion, proceeding from right to left, terminates at a combination of different faces.

![Figure 5.6 Creating a protrusion via from face to next. The protrusion is from right to left.](image)

*FromAll or ThroughAll*

Due to the circular topology, the interpretation of the *from all* and *through all* designations is not meaningful for revolved features. For extrusions, *from all* must be paired with an explicit *to* face, and *through all* must be paired with an explicit *from* face. The explicit faces must intersect the proto feature completely.

In contrast to previous, *from all* requires that all volumes preceding the *to* volume in \( S_f \) be in the set \( S_{in} \), in addition to the *to* volume. Moreover, the set \( S_{from} \) is empty. This defines the feature volume set \( V_F \), and with it the semantics of the operations. Similarly, *through all* requires that all volumes following the *from* face are in \( V_F \), and that \( S_{to} \) is empty.

*5.3.3 Discussion*

The use of double wedges when delimiting revolved features with planes seems counter-intuitive. It would appear that the user has only one wedge in mind, particularly when the intersection of the two planes is the axis of revolution. If the planes
are considered oriented, or if we work with half planes, then it is possible to define a single wedge in space whose interior limits the revolved feature. We did not do so because neither the definition of half planes, nor the orientation of datum planes, appears to be natural. While very familiar to implementors, it is not clear to us that users would think in such terms. One could find a middle ground: The user determines graphically which wedge is meant, the system internally orients planes and records design intent in terms of this internal orientation.

The rules for features with face boundaries negotiate several difficulties. The central problem is that for most curved faces there is no clear-cut mathematical definition of how to extend the face. If there were, one could extend a partially intersecting face to a completely intersecting surface and so determine what is “enclosed” between from and to face. To illustrate this point, consider Figure 5.7 in which we assume that $F$ is a curved face that happens to be part of a larger surface that extends as shown in (b). Intuitively, the designated from face in an extrusion should eliminate all parts of the proto feature that “precede” the from face. In the situation shown as (a) in the upper left of the figure, we see a volume $V$ whose classification is unclear. If the surface $F$ is extended as mathematical surface, then $V$ would precede $F$, as shown in (b). If the surface $F$ is extended by its tangents at the planar edge of $F$, then $V$ is partially intersected, (c), and no clear decision can be made. If the surface $F$ is extended by a ruled surface whose generators are perpendicular to the extrusion direction and connects at the boundary of $F,$ then $V$ might follow $F,$ (d). Moreover, in each case additional conventions are needed to define the extension mechanism unambiguously, and the conventions would not be very intuitive. Note that the ruled surface extension generalizes the bounding box idea we have used.

The concept of boxing planes in the case of partially intersecting bounding faces reduces the test whether a volume of $S_f$ is in $S_{in}$ to a bounding box computation. Variants of our definition could be considered. For example, our definition excludes volume $V$ in Figure 5.8(a). Here, the concave from face has a bounding plane $E$ that partially intersects $V$. Intuitively, $V$ should be in $S_{in}$. If we use the plane $B$ instead,
Figure 5.7 Alternatives for incompletely intersecting from/to faces: (a) from face $F$ incompletely intersects circular proto feature; (b) $F$ is extended as mathematical surface, (c) $F$ is extended by tangent directions at the boundary, (d) $F$ is extended by a ruling of the boundary perpendicular to the direction of the sweep.

then the interpretation of Figure 5.8(a) is as expected. However, in that case the volume $V$ in Figure 5.8(b) would also be considered to be in the set $S_{in}$ which is counter intuitive.

Ultimately, the notion of “in-between” rests on a concept of separation that is unambiguous only for completely intersecting bounding faces. Partially intersecting boundaries are a necessity unless we allow open profiles. But open profiles have more difficult semantic problems; [29, 30]. A useful device, therefore, might be to allow users to define datum surfaces for the purpose of separating volumes in ambiguous positions.

Our semantic definitions have been given in terms of regularized Boolean operations. This was done so as to define unambiguously what each feature operation means. It also implies that the feature operations can be implemented literally using Booleans. This could be attractive in legacy systems in which Boolean operations are a prominent aspect of the system architecture. However, the manner in which
Figure 5.8 Determining in-between volumes from intervals: (a) use the minimum value of the bounding box to include $V$, (b) use the maximum value of the bounding box to exclude $V$.

the features have been defined implies a locality that should be exploited in any implementation. For example, it is clear that the definition of a contour for extrusion already reduces face-intersection candidates: Faces whose projections do not intersect the contour clearly could not intersect the proto feature in 3-space.
6. EDITING FEATURE-BASED AND CONSTRAINT-BASED DESIGN

In this chapter, we present techniques needed for editing feature-based and constraint-based design based on the layered design model. When a feature is edited, all features attached later are reevaluated to satisfy the required constraints and shape references in the initial design. To accommodate feature editing we modify the unevaluated design representation only. When the geometry of a new design is requested, we employ the identical design compiler to instantiate its geometry. The design compiler actually adapts to update the unevaluated design representation as needed when a feature is edited.

We describe name matching techniques that support the design reevaluation procedure. The algorithms account for failed or multiple matches. The matching uses a naming schema based on historical, topological, and geometric design information that has been described in Chapter 4.

In feature-based design, editing operations can be classified into the following broad categories:

(E1) inserting or deleting an entire feature;

(E2) changing feature attributes, e.g. from a blind hole to a through hole;

(E3) modifying dimension values defining and/or placing a feature;

(E4) changing the dimensioning schema;

(E5) changing the feature shape definition, e.g. changing the cross section.

Here, we concentrate on editing operations (E1) through (E4), and on the name matching algorithms that are needed to support them. Operation (E5) requires certain mapping techniques in addition.
Editing operations such as insertion, deletion and modification in general have been addressed by Pratt [46] and alternative approaches are discussed in that paper. No naming mechanism is suggested that would be appropriate in a constraint-based design environment. Rossignac in [54] has explained the difficulty of feature editing. He and his colleagues implemented some feature editing operations in a 2D system [55].

Current feature-based design systems support many of the editing operations (E1–E5). However, because of limitations their generic naming schemas have with disambiguating feature interactions, many restrictions are placed on the editing operations, and apparent errors are made. In [29, 30], some of these errors have been characterized using Pro/ENGINEER as example. However, an explanation of the precise methods implemented in Pro/ENGINEER cannot be given because they are guarded as proprietary information. Note that (E4) and (E5) fundamentally affect the way the design has been conceptualized. These two editing functions are not usually supported in current systems.

6.1 Feature Editing

6.1.1 Editing Operations

From the five editing operations, we extract common procedural steps from which they can be composed. This elucidates in turn how the name matching algorithms should be structured and how exact a match is required.

(E1) Feature Deletion and Insertion

Deletion of a feature $F$ can be done as follows:

1. The geometry is rolled back to the prior geometry as if we were creating the feature to be deleted.

2. The feature definition of $F$ is deleted from the unevaluated model.
3. The subsequent features are reevaluated from the unevaluated model. Due to
match mediation, the unevaluated description of the subsequent features may
change.

Insertion of a feature is similar, except that a new feature definition is inserted in
Step 2. Thus, editing operation (E1) requires only a naming schema and mediation
when matching names. Furthermore, we must account for the possibility that a later
feature references geometric elements that were created before by the now deleted
feature.

(E2, E3, E4) Changing Attributes, Modifying Dimension Values, Changing the Di-
mension Schema

When editing dimension values, the dimension schema, or the attributes of a
generated feature \( F \), the following steps are executed:

1. The part geometry is rolled back to the state of the prior geometry that existed
when \( F \) was defined.

2. The cross section on which \( F \) is based is reconstructed in the required projection.

3. Attribute changes, dimensional changes, and changes in the constraint schema
are defined by the editing process. The cross section is then regenerated for a
generated feature.

4. The proto feature is constructed, and the attachment of \( F \) proceeds as if \( F \) was
defined for the first time.

5. The subsequent features are reevaluated from the unevaluated model. Due to
match mediation, the unevaluated description of the subsequent features may
change.

If \( F \) is a modifying feature or a datum feature, the work is simplified in that no
cross section has to be modified and reevaluated. However, in that case the match
mediation has to be more rigorous, as we will detail in later sections.
(E5) Changing the Cross Section

Finally, if the cross section is redrawn in part or in whole, a remapping step is required in addition, in which the new names of geometric elements of the cross section are mapped, where possible, to names of the old cross section.

6.1.2 Editing Semantics

In feature-based and constraint-based design, even changes to dimension values can entail substantial topological changes of the instance model, and those changes must follow a predictable pattern that makes sense to the user. For example, consider Figure 6.1. By changing the position of the slot in part A, the rounded edge ought to disappear in part A': It seems unnatural to us to round the second, elongated edge, because we conceptualize the slot repositioning as a continuous motion of the feature from the position in part A to the position in part A'. In this motion, the length of the rounded edge in part A diminishes to zero. Yet some commercial systems may round the other edge, as shown in the figure.

On the other hand, when modifying the length of the slot in Figure 6.2, the round of part B would sensibly be inherited by both segments of the divided edge in part B', yet some commercial systems signal an error.

We learn from these two examples that the semantics of editing feature-based design deserves attention. We have adopted as guiding principle to consider the intended meaning to be derived from considering the editing operation as a continuous process in which specific events such as edge subdivision are interpreted using rules of positional inheritance. Of course, other interpretations are possible, and our interpretation does not claim to be universally compelling.

In a number of cases, different interpretations of the intended effect of the editing operation seem equally sensible. In those situations, we adhere to the principle that the effect of the modification should not depend on the prior design modification history. Because prior history is no longer is visually available, we feel that using it to decide between different interpretations would confuse the user. For example,
Figure 6.1 When the slot (second feature of part $A$) is moved to the right and becomes a step, rounding the edge in part $A'$ should be considered an error.
Figure 6.2 When the slot (second feature of part $B$) is moved to the right and becomes a step, rounding the edge in part $B'$ should be considered an error.

consider Figure 6.3. Part $C'$ was obtained from part $C$ by contracting the length of the slot, and the blend was spread to the larger edge of $C'$ that “contains” the previously rounded edge. If we now lengthen the slot to divide again the rounded edge, by editing part $C'$, we would derive the shape of part $B'$ — not the shape of part $C$ — because reconstructing $C$ depends on older design history. Note that none of these rules preclude adding an explicit history mechanism that checkpoints designs and allows backing up the design/editing process to earlier variants.

It is clear that the topological changes entailed by such simple editing operations must be supported by a robust generic naming schema, along with sensible matching
Figure 6.3 When the through slot (second feature of part C) is changed to a blind slot, the entire edge of part C' should be rounded. For subsequent editing parts B and C' should be equivalent.

algorithms that give predictable, meaningful results. The purpose of this work is to propose such matching algorithms.

Matching does not always succeed. Even simple changes of dimension values can imply design variants in which elements referenced in the construction of subsequent features no longer exist. In some cases, plausible substitutes can be constructed, but in other cases matching, and with it reconstruction, can get stuck. In those situations different philosophies could be adopted. We could require some substitution to be made without exception, and accept substitutions that are wrong from the designer’s point of view. This strategy makes sense if the editing mechanism has been activated
under program control, for instance in an automated design optimization loop. We could also require user intervention in such situations. That would be appropriate if a user initiates design editing. In our example Erep system, we adopt the latter approach.

6.1.3 Editing Examples

We demonstrate the editing concepts with a sequence of editing examples done with our Erep system. In Figure 3.1, we create a part with four features: a block, a protrusion, a slot and a round, created in this sequence.

The way in which the part has been created implies some feature dependency. The definition of the slot in the example of Figure 3.1 makes reference to the protrusion, for placement purposes. On the other hand, the round does not make a reference to the protrusion or the slot. So, if we delete the protrusion feature, the round remains while the slot feature would be deleted.

If the slot feature should be kept, we would have to modify the dimension schema so that the slot is independent of the protrusion. This may be done by editing the slot first: The system brings up the cross section of the slot as shown in Figure 6.4(left). We may now delete the distance constraint that places the slot with respect to the protrusion edge, and add a new distance constraint that references equivalently the top edge of the block; Figure 6.4(right). Reevaluation with the new dimensioning schema will not alter the design form, but the slot’s definition no longer references the protrusion and so the protrusion could be deleted next — without affecting the slot.

Consider now the part of Figure 3.1 with the original dimensioning schema shown in Figure 6.5(left). To enlarge the radius of the protrusion, we edit by modifying the dimension values of the cross-section; Figure 6.5(top middle). After reevaluation, the part variant of Figure 6.5(top right) is produced, maintaining the constraints that specified the slot and its position relative to the protrusion. Note that the slot has moved. This would not be the case with the dimensioning schema of Figure 6.4(right).
Figure 6.4 The slot dimensioning schema is edited. The new schema shown to the
right no longer references the protrusion, so the slot position is now independent of
the protrusion feature and its variations.

In the bottom of Figure 6.5 we show a cross section change of the slot. The round
end of the slot has been replaced with a square end; Figure 6.5(bottom middle). The
result is shown in Figure 6.5(bottom right).

The editing semantics that we have illustrated is intuitive for feature-based design. However, both research and commercial systems are limited in their support of the
feature editability, due to the technical demands constraint-based design imposes.

6.2 Feature Reevaluation

Since features are created sequentially, later feature can depend only on earlier
features. When a feature has been edited, all later features may be affected. Therefore, reevaluation has to account for the possibility that features that have not been
edited directly may have to change nevertheless. The changes could be deleting a
marginalized feature, or modifying the names referenced by the feature.

A feature becomes marginalized when it no longer contributes to the geometric
shape of the solid at the moment of attachment; for example, when a hole is dimen-
sioned such that it no longer subtracts volume from prior geometry. Such features
are deleted. A feature becomes orphaned when the names used in defining the feature
and attaching it refer to geometric elements that no longer exist. For instance, if the
position of a hole depends on having a given distance from a boss and the boss is deleted, the hole can no longer be positioned. Orphaned feature can be reattached as described later.

A generated feature refers to elements of the existing geometry, using a generic naming schema as explained in [13]. In some cases, the editing change implies that the name of the referenced entity changes. In such cases, the referenced names must be adjusted by feature correction. The possibilities are summarized as follows:

Let

\[ S = (F_1, F_2, \ldots, F_{i-1}, F_i, F_{i+1}, \ldots, F_n), \]
be the unevaluated model, and let $F'_i$ be the description of the modified feature. Then

$$S' = (F_1, F_2, ..., F_{i-1}, F'_i, F'_{i+1}, ..., F'_m)$$

where $m \leq n$ and $F'_j$, $i \leq j \leq m$, are the regenerated features following and including $F'_i$.

The rewriting of the later features $F'_{i+1}, F'_{i+2}, ...$, and deletion of the marginalized features, limits the dependency of editing on prior design history.

A marginalized feature is easily detected: When reevaluating the feature, the feature attachment procedure detects that the prior geometry remains unchanged. In this case, the feature description is deleted. An orphaned feature arises when, in the course of reevaluation, the cross section is not fully determined or cannot be placed. Both situations can be due to unresolved names for dimensioning, but also to loss of the sketching plane for generated features, or loss of an entity for modifying features. Orphans can be reattached if we initiate feature editing and let the user supply new targets for the unresolved names.

In all cases, the crucial aspect is a matching algorithm for generic names that has to mediate newly arising ambiguities and has to recognize failure to match. These algorithms differ depending on the use of the name. For dimensioning, ambiguities are acceptable as long as the target entities project to the same point, line or plane. For sketching planes, multiple coplanar faces are acceptable. For modifying features, ambiguous references must be of the same type and are considered collectively. In all other cases, mediation and error recovery is needed.

6.3 Matching a Vertex

Vertices are matched to define constraints, to construct datum features, to be used by modifying features, or to be used as a subexpression in another name (also see Chapter 4). In the dimensioning case, we need the vertex in projection, otherwise we
need to identify the vertex in the three-dimensional space. Moreover, only in the case of modifying features can we accept multiple matched vertices with the same name.

A vertex can be lost when editing because of changes in a feature collision. An example is shown in Figure 6.6. In some cases, lost vertices can be uniquely reconstructed on basis of their names. However, we limit reconstruction so as to limit dependency on the design history. Moreover, the vertices used in modifying features should not be recovered from the design history because a modifying feature operates on the geometry after attaching its previous feature. Therefore, if the vertex cannot be found or if reconstruction is not desirable, we need to interact with the user to find a suitable alternate.

![Figure 6.6](image)

Figure 6.6 The vertex $V$ of part $D$ is not found in part $D'$. As a proto feature vertex, it can be reconstructed from the first feature of part $D'$.

A vertex name certifies whether the vertex is a vertex of a proto feature. If so, the vertex can always be recovered from the proto feature, because the name identifies the vertex uniquely and the proto feature can be constructed explicitly. Thus, $V$ in Figure 6.6 can be localized even after it has been obliterated in the design variant, or
is lost as result of merging with a prior vertex. In the former case, the name of the vertex is not changed in the feature correction, but in the latter case the name of the coincident, merged vertex is substituted.

In the case of an intersection vertex, there will be three or more incident faces that define the vertex. If no vertex can be found that matches the description, we construct an approximate match as follows by parsing the name expression generated in Chapter 4.

1. Let $E_v$ be the vertex name we seek to match (see [6]). Let $F$ be the feature used in the feature orientation part of $E_v$. We consider a set $V$ of vertices that are intersection vertices and part of the feature $F$.

2. Let $J = \{f_1, f_2, ..., f_m\}$ be the incident faces listed in the first part of $E_v$. We partition $V$ into subsets $V_1, V_2, ..., V_m$, where $V_k$ consists of the vertices in $V$ that have $k$ incident faces in $J$.

3. We examine $V_k$, for $k = 1...m$. Each vertex $w$ in $V_k$ is assigned a grade $G_w$ as follows: Initially, $G_w = 3k$. If $w$ is manifold, and $E_v$ requires a manifold vertex, then we add 1 to $G_w$. We add 2 if the order in which the incident faces are around the vertex agrees with the order of $J$ for manifold vertices (note that $k \geq 3$ in that case). If $w$ is nonmanifold and we require a nonmanifold vertex, then 1 is added to $G_w$. For every expression $[f, s]$ in the feature orientation, we add 1 to $G_w$ if the corresponding computation for $w$ would also produce $[f, s]$ as feature orientation element.

4. The set of vertices approximately matching $E_v$ is the set of vertices $w$ with the maximum grade $G_w$. If $G_w < T$, where $T$ is a preset threshold, then the match is assumed to have failed.

Assume that a set of vertices has the same maximum grade $G_w$ and passes the threshold criterion. For dimensioning use, such a multiple match is acceptable when all matched vertices project to the same point. If this is not the case, the name $E_w$
of every matched vertex can be computed. This partitions the match set into classes of vertices that are equivalent in the new design variant. The user is then asked to identify which equivalence class of the matched vertices is meant. The set so identified becomes the accepted match.

6.4 Matching an Edge

Edges are matched for defining constraints, constructing datum features and for being used by rounds and chamfers. In the case of constraints and datum features, ambiguous edges can be tolerated if they are collinear line segments with consistent orientation. In the case of 2D constraint definitions, ambiguities are acceptable if the edges project onto the same line, again with consistent orientation.

Edges of a generated feature have unique names. As in the vertex case, an obliterated edge can be reconstructed from the design history. However, note that such an edge may have been subdivided, and only some of the resulting segments may be meant.

We structure the matching algorithm by parsing the naming expression of an edge in Chapter 4. First, a preliminary set of candidate edges is identified, based on matching adjacent faces. Unless two or more adjacent faces can be matched, the edge cannot be matched and user intervention is required.

Next, the preliminary set of edges is narrowed using the incident vertex specification and feature orientation. If the vertices can be identified, then we can also determine which subdivision segments are to be matched. If no vertex information is given (for closed edges), a subdivision test may still be applicable. Note that this step is not needed for an edge used in constraint definitions. Edge names are further analyzed based on feature information, especially when the vertex match is unsuccessful or ambiguous.
Figure 6.7 After repositioning the slot, the correct design variant is shown in the middle.

Preliminary Edge Set

A preliminary edge set is found using immediate context. Let $E_c$ be the edge name we are to match. If the edge is (a subdivision of) an edge of a proto feature, then the preliminary set consists of all edges that are adjacent to the faces of the proto feature. Such edges must lie on a common space curve that can always be reconstructed. Note that the set may be empty.

If we are matching intersection edges, the preliminary edge set consists of all edges that have a maximum number of adjacent faces that lie in the cycle of faces $C_e$ identified by the first part of the edge expression $E_e$. At least two adjacent faces must be matched, otherwise the preliminary set is empty.

If the preliminary edge set is empty the match has failed. However, even if the preliminary edge set is a singleton, the match is not successful unless this edge has the equivalent description of adjacent vertices (for a modifying feature), and feature orientation information to $E_e$. For example, consider the part in Figure 6.7 left. Assume that we created the round slot by an extrusion from the front to the rear, and rounded the right edge of the slot. By changing dimension values, the slot can be repositioned so that it becomes a step. The middle variant correctly recognizes that the singleton preliminary edge set does not constitute a match on basis of the incident vertices or feature orientation information, whereas the right design variant is an error.
Incidence Based Narrowing

Preliminary edge identification does not account for the possibilities that an edge has been subdivided, or that two different edges have the same immediate context. Examples are shown in Figures 6.7 and 6.8. An effective distinction between the two types in mathematical terms is complicated, because the distinction should conform with user intuition and then must be based on dividing space curves into real components, a complicated undertaking. For instance, the two near-circular edges of a cylindrical through hole in a sphere would be considered two distinct edges with the same immediate context, even when the axis of the hole does not contain the center of the sphere. However, classical algebraic geometry would consider both curves part of a single space curve connected in complex projective three-space. Therefore, we do not differentiate between the two cases.

The edge name $E_e$ has the form $[C_e, L_e, F_e]$ (see [6]). The expression $L_e$ identifies the incident vertices and is now matched. If $L_e = [\ ]$, the edge was closed when the name was constructed. In this case, no vertex matching is possible and we continue with feature orientation based narrowing.

If $L_e = [V, W, s]$, then we match the vertices described by $V$ and by $W$. If the vertices can be matched uniquely, or can be uniquely reconstructed, or if one edge and a direction can be established on the basis of $s$ and the immediate context specification, then the matching process continues with a subdivision identification. Otherwise, we continue with feature orientation based narrowing.

If we have uniquely identified or reconstructed two vertices $v$ and $w$ described in $E_e$, then we retain all edge segments that lie between the two vertices. See also Figure 6.8. From this reduced edge set, every candidate edge is deleted that has an inconsistent feature orientation.

If we can match only one vertex, say $V$, from the name $E_e$, then all edges not incident to $V$ are removed from further consideration. This implements the semantics discussed in conjunction with Figure 6.1. The remaining edges must have consistent feature orientation to constitute a match.
Figure 6.8 Left: edge $e_1$ is named in the design. After editing, the edge is subdivided by colliding features. Segments $e_{11}$, $e_{12}$, and $e_{13}$ match.

Feature Orientation Based Narrowing

In general, an edge $E_w$ is considered an exact match if the edge is in the preliminary edge set and has two vertices that are uniquely matched from the description in $E_e$. However, there are other cases requiring further analysis, for example edges of which one or no adjacent vertex has been matched and closed edges that do not have adjacent vertices. We narrow as follows.

Examine every candidate edge $e'$. Let $E_e = [C_e, L_e, F_e]$, and let $F$ be the feature name occurring in $F_e = [F, [f_1, s_1], ..., [f_k, s_k]]$. We require that at least one of the faces $f_1, ..., f_k$ be adjacent to $e'$. Moreover, for every subfield $[f, s]$ where $f$ is adjacent to $e'$ we require that the orientation $s$ agree with the orientation of $e'$. Only edges satisfying these two criteria remain in the match set.

Grading Multiple Matches

If the narrowing does not produce a unique match, we give each remaining candidate edge $e'$ a grade based on how closely the vertices of $e'$ match the vertex descriptions $V$ and $W$ in $E_e$. This is done by evaluating the vertices of $e'$ as described in Section 6.3. The edges that have the highest grade are the approximate match of $E_e$. 
Match Mediation

When the matching algorithm has failed, then the user is asked to reidentify the edge. A new edge name is computed, and regeneration continues.

When matching has produced several edges, additional information may be needed depending on the purpose of the match. In the case of rounds and chamfers, multiple matches are always accepted. All edges matched are used in the feature operation. The names of the matched edges are recomputed from the current design variant and are stored for future reference. In the case of 2D dimensioning, multiple matches are acceptable only when every matched edge projects to the same circle or line with identical orientation. Otherwise, the user is asked to differentiate which edge is to be referred to. In the case of datum plane definitions, multiple matches are accepted when the edges are collinear line segments. For datum axis definitions, the collinear line segments must be consistently oriented.

6.5 Matching a Face

Faces are matched to determine the limits of feature attachment operations, defining draft operations, identifying a sketching plane, or constraining a datum definition. The degree of exactness with which a face is to be matched varies with the operation. For faces used in defining constraints and sketching planes, the match only requires matching the internal geometry of a named face, not a particular subdivision. As before, an obliterated face used in those cases can always be reconstructed from the design history, from the corresponding proto feature or modifying feature. In the other cases, however, a more specific match is required.

All faces of a solid must be subdivisions of faces created in proto features and modifying features, with the possible exception that two faces may be merged so that one of them loses its identity. Faces on the proto feature of a generated feature have unique names that are assigned when the feature is created. Faces of chamfers and rounds are named after the vertices and edges they chamfer or round.
When a face has been subdivided, the internal geometry of this face does not change, therefore all subfaces inherit the same name of their parent face. When faces are merged, every face in the merging list can be the match representative. However, because of feature editing, different face representatives may lead to different interpretations. In Figure 6.9, suppose the merged face of $f_1$ and $f_2$ are initially used for the sketching plane of the third feature. After the second feature is edited, using $f_1$ and $f_2$ as the representatives of the merged face leads to protrusions with different heights.

Figure 6.9 Different representatives of a merged face produce different results after feature editing.
When merging faces, there is no clear preference of how to name the resulting face; either name could be used. In our implementation, the result has the name of the face that belongs to the earliest feature among all merging components.

**Preliminary Set**

As explained in [6], every face name starts with one of the following five expressions:

\[
\begin{align*}
(\text{start}_f \text{ feature}_f \text{ f2D}),
(\text{end}_f \text{ feature}_f \text{ f2D}),
(\text{side}_f \text{ feature}_f \text{ f2D}),
(\text{r}(l_e)) \text{ or } (\text{c}(l_e))
\end{align*}
\]

Here, faces of type \(c(l_e)\) are from chamfers and faces of type \(r(l_e)\) are from rounds. These expressions are used to select the preliminary set of faces. If only the geometry of a face is required, the faces in this set will typically lie on the same surface, and so any face in the set can serve as a match. Moreover, a face can be reconstructed if the match set is empty. If faces in the preliminary match set have different underlying surfaces, for instance when the set contains chamfers of several edges, then further narrowing is needed.

**Narrowing Multiple Matches**

When a specific subdivision of a face is required, we process the adjacent edges and compare them to the required edges. Edge matching proceeds for each individual bounding edge as described before in Section 6.4. The grade for each candidate face in the preliminary matching set is the number of matched boundary edges. This number has to exceed a threshold which is set as a fraction of all adjacent edges. As an example, assume that a face is to have five adjacent edges with prescribed names. We match each edge name against the edges of a candidate face. With a threshold set at 3/4, the match succeeds if at least four out of the five edges can be matched, and fails if three or fewer edges are matched.
6.6 Remarks

The matching mechanism is an inverse processes of generic naming. Naming and matching are prerequisites for flexible, constraint-based feature editing. Without the ability to do graded matches, however, only exact unambiguous matches would be allowed, and we would seriously restrict the flexibility of feature editing. Therefore, we have introduced rankings of partial matches by assigning numerical grades.

The support of feature editing is only one application of our naming and matching techniques. The fundamental significance of these techniques is that a separation is achieved between generic design and modeler-specific design. The naming and matching systems are key components of the design compiler that translates the generic design that is modeler-independent to a specific model instance that is constructed with the infrastructure of a particular geometric modeler.

Our naming and matching algorithms are a contribution to defining a modeler-independent semantics of feature-based and constraint-based editing, just as Chapter 5 gives a semantics for feature attachment. Where possible, our semantics depends on the conceptualization that the result of a changed dimension ought to be derived by considering it a "deformation" process governed by gradually varying the dimension from the initial to the final value.

In our implementation of the editing semantics, we have concentrated on this semantics aspect, experimenting with the mechanisms described here and in Chapter 4. Thus, no attempt has been made to devise efficient searching algorithms for locating, in the Brep instance, which boundary element is referred to by a particular name. Thus, finding the possible referenced entities takes linear time in the number of boundary elements. Significant heuristic and asymptotic improvements are easy to make. For instance, when a design is evaluated, searching structures can be built that index subsets of boundary elements by the feature(s) they belong to. This immediately results in sublinear performance. Coupled with efficient searching techniques very attractive speed-ups would be obtained. The details are routine.
7. PERSPECTIVE

We have described in the previous chapters a layered design model and investigated associated issues to support this model. Essentially we deal with the representation, evaluation and editing of feature-based and constraint-based design. The focus of this work has been on solid design in generic approaches although many technical solutions are applicable to non-geometric design. However, solid design is not sufficient to model the design process even for parts. In this chapter we discuss the extension of our current design system, particularly, specification features, instance features and assembly. These aspects should be completed before this CAD research prototype becomes directly beneficial in industry. We also propose some future work related to feature-based and constraint-based design.

7.1 Specification Features

Specification features, in particular, tolerances, surface finish and material condition, have not been implemented in the experimental Erep system. The un evaluated representation of these features, similar to form features, have geometric references. Accordingly, we explain the representation, evaluation and editing of these features.

- Representation

Specification features can be generically represented in the same way as constraints. Again, the geometric references used to store specification features should be replaced by the generic names as explained in Chapter 4.

Specification features are not necessarily the last features in a design. They may be followed by other form features. When the geometric entities referenced by specification features are split or merged, rules should be formulated to regulate
how to propagate the specification features. For instance, a rule governs that the surface finish attached to a face is inherited when the face is split.

- Evaluation

  The evaluation of specification features does not connect with constraints, nor attachment operations. Matching generic names is the only step needed to evaluate specification features.

- Editing

  Editing specification features are equivalent to editing constraints as we have described in Chapter 6. Note that any modification of a specification feature does not change the geometric representation of a design. Therefore, reevaluation of the geometry is not necessary when a specification feature is edited.

7.2 Instance Features

The generated features that we have described in this thesis consist of two generic feature classes: extrusion and revolution. However, instead of generic extrusion and revolution, specific shapes of extrusion and revolution are frequently used in engineering design, such as holes, slots, bosses and pockets. Referring to QTC features, we have derived some specific features from the two generic features, including holes, linear slots, circular slots and pockets.

Strictly speaking, instance features have more concrete semantics, besides what we have shown in section 5.3 for generated features. For instance, a hole means a cylindrical cut that goes through the interior of the existing geometry whereas a slot means to cut some boundary of the existing geometry. Each instance feature thus needs additional semantic guards to protect its specific shape. The difference between a cut and a hole may be distinguished by the genus of the resulting geometry. These semantic guards are also effective for editing features.
The way of instantiating generic feature classes varies, depending on the application domain. When features are derived towards manufacturing, for example, QTC features, we can convert the designed features into manufacturing features.

7.3 Assembly

Conceptually, engineering design is a top-down process that lays out the functional parts of a product first and then details each part. The exact shapes of the functional parts in the layout phase are normally not clear but are made in the part design phase. These parts relate each other to satisfy the requirements of the product. By assembling the parts following these relations, we obtain a complete product design.

To describe assembly procedure, both flat or procedural approaches are useful. A flat representation requires solving 3D constraints. As we have pointed out early that currently solving 3D constraints remains difficult, we advocate using a procedural approach globally and a small set of 3D constraints in each assembly step.

A generic representation for the procedural assembly can be similarly formulated to replace the geometric references by generic names. Assembly compilation or evaluation follows the same process for datum features where the feature attachment step is no longer needed. Editing an existing assembly involves less complication because the generic names are matched in the one-by-one manner when parts to be assembled are not edited.

7.4 Future Work

These three issues that we describe above are important and required for completing an feature-based and constraint-based design system. To transfer this research prototype to a commercial CAD system, consolidating the existing techniques is needed, besides solving these three issues. For instance, constraint-solving capability should be enhanced to better serve generated features and datum features.
At present, we solve 2D constraints for specifying the cross-sections or profiles of generated features and restrictive 3D constraints for datum features.

We expect that this research work will influence the development and enhancement of new and existing CAD systems. In particular, we anticipate the trends of CAD research and development in following aspects.

1. Converting solid modelers to CAD systems

   Instead of keeping up in-house solid modelers, CAD developers such as AutoDesk Inc. may adopt commercial solid modelers. The adoption of commercial solid modelers will certainly alleviate the burden from CAD developers. However, solid modeling is not dedicated to CAD only and solid modeling faces the challenge of robustness. It may take some time before CAD developers to give up their in-house modelers.

2. Exchange and reuse of design history

   Current CAD systems exchange with each other their completed designs, typically, in IGES format. Spatial Tecknology Inc. advertises ACTIS [63] as the geometric standard to unify and exchange designs and other applications of solid modeling (also see Figure 7.1). Most CAD companies are still using their own solid modelers and some companies write translators that convert the final designs created by various CAD systems. However, it is difficult to reuse a design in one CAD system if the original design is created by another CAD system. A high-level design representation that records generic design intent and history will be overwhelmingly welcome by CAD end-users for the purposes of exchange and reuse of design history. Moreover, it is reasonable for these end-users to advocate some design exchange standard which CAD developers should follow.
3. Various design compilers

When design history is exchangeable and reusable, the task of developing new CAD systems can be decomposed into two modules: a design interface and a design compiler. A typical design interface captures the design intent of engineers and writes the generic design representation, whereas a typical design compiler instantiates a generic design with respect to a solid modeler into a geometric representation and explicit design information attached to geometric entities. The number of design compilers depends on the number of solid modelers (see Figure 7.2).

The interoperability of different CAD systems achieved by various design compilers faces the challenge of possibly incompatible constraint solvers that are consulted by these compilers. Different constraint solvers may favor different solution for a constraint system. To obtain the interoperability at a high-level design representation, we may restrict that the recorded geometries in a design representation are defined and placed accurately. Therefore, a compiler does
not need to consult with its constraint solver when it first loads an existing design created by another CAD system.

There are some important research issues that are beyond the feature-based and constraint-based design, but within the CAD territory. Although feature-based and constraint-based design has moved one big step forward from solid modeling in the CAD technology, it is not certain that this is the ultimate approach, especially the way we combine features and constraints. On the other hand, a new CAD paradigm should be incorporated with other technologies and integrated with analysis, process planning and manufacturing. We show more details as follows.

1. Conceptualization of design intent.

In this thesis we use constraints and procedural features to substitute design intent. At present this substitution seems to be effective given that Pro/ENGINEER
has gained the favor from a large number of engineering designers. However, this substitution is not conclusive. Design intent should also include other functions such as kinematic and dynamical behaviors, and strength requirements. Additional approaches should be studied to model these aspects.

2. Advanced design interface

Currently we use mouse and graphical user interface for the design interface. Virtual reality, if successfully developed, should be a good replacement for the design interface.

3. Integration of design, analysis, process planning and manufacturing

After a design is made, it may be analyzed that improvement is possible based on performance, costs and manufacturability. The feedback at present is handled by users through editing in CAD systems. If an optimal design is desired, editing may take place several times. Integrate design, analysis, process planning and manufacturing should reduce user interactions.
BIBLIOGRAPHY


APPENDIX
EREP SPECIFICATIONS

The following is the extended BNF (EBNF) specifications of the Erep language. To be noted are a few notations: integer and number are obvious except that number stands for a real number, id stands for a string but string for a quoted string.

A.1 Parts and Features

\[
\begin{align*}
\text{<part>} & \quad ::= \ \text{PART} \ \text{<name>} \\
& \quad \{ \ \text{<feature>} \ \} + \\
& \quad \text{END\_PART} \\
\text{<feature>} & \quad ::= \ \text{<d\_feature>} \\
& \quad | \ \text{<g\_feature>} \\
& \quad | \ \text{<m\_feature>} \\
\text{<d\_feature>} & \quad ::= \ \text{<datum\_plane>} \\
& \quad | \ \text{<datum\_axis>} \\
& \quad | \ \text{<datum\_point>} \\
\text{<g\_feature>} & \quad ::= \ \text{<e\_feature>} \\
& \quad | \ \text{<r\_feature>} \\
\text{<m\_feature>} & \quad ::= \ \text{<c\_feature>} \\
& \quad | \ \text{<o\_feature>}
\end{align*}
\]

A.2 Generated Features

\[
\begin{align*}
\text{<e\_feature>} & \quad ::= \ \text{FEATURE} \ \text{<name>} \ \text{EXTRUDED} \\
& \quad \text{<references>} \\
& \quad \text{<cross\_section>}
\end{align*}
\]
<e_description>
<display>
END_FEATURE
<references> ::= REFERENCES
    { <vert_decl> | <edge_decl> | <face_decl> }*
END_REFERENCES
<e_description> ::= DESCRIPTION
    <volume_type> ','
    <sketch_plane> ','
    <e_trajectory> ','
    <e_extent> ','
END_DESCRIPTION
<display> ::= DISPLAY_OF_CONSTRAINTS
    to_be_added
END_DISPLAY
<face_decl> ::= FACE id '=' <face> ','
<face_ref> ::= FACE id
<edge_decl> ::= EDGE id '=' <edge> ','
<edge_ref> ::= EDGE id
<vert_decl> ::= VERTEX id '=' <vertex> ','
<vert_ref> ::= VERTEX id
<volume_type> ::= ( PROTRUSION | CUT ) ','
<sketch_plane> ::= SKETCHING_PLANE ( <dplane_ref> | <face_ref> ) ','
<e_trajectory> ::= TRAJECTORY ( NORMAL | <daxis_ref> ) ','
<e_extent> ::= <e_from_spec> <e_to_spec> ','
<e_from_spec> ::= FROM ( OFFSET number | ALL | <face_ref> |
    <dplane_ref> | PREVIOUS )
<e_to_spec>  ::= TO  ( OFFSET number | ALL | <face_ref> | <dplane_ref> )
                  | THROUGH NEXT
<r_feature>  ::= FEATURE <name> REVOLVED
                  <references>
                  <cross-section>
                  <r_description>
                  <display>
                  END_FEATURE
<r_decription> ::= DESCRIPTION
                  <volumetric_type> ';'
                  <sketching_plane> ';'
                  <r_trajectory> ';'
                  <r_extent> ';'
                  <pattern> ';'
                  END_DESCRIPTION
<r_trajectory> ::= ABOUT <daxis_ref> ';'
<r_extent>    ::= <r_from_spec> <r_to_spec> ';
                  | FULL ';
<r_from_spec> ::= FROM ( ANGLE number | <face_ref> | <dplane_ref> | PREVIOUS )
<r_to_spec>   ::= TO  ( ANGLE number | <face_ref> | <dplane_ref> )
                  | THROUGH NEXT

A.3 Datum Features

<dplane_ref> ::= DATUM_PLANE ( DEFAULT | id )
<daxis_ref>  ::= DATUM_AXIS ( DEFAULT | id )
<dpoint_ref> ::= DATUM_POINT ( DEFAULT | id )
<datum_plane> ::= DATUM_PLANE <name> CREATED

    <references> <p_description>

END_DATUM

<p_description> ::= DESCRIPTION

    { <subdef_dplane> }+

END_DESCRIPTION

<subdef_dplane> ::= OFFSET ( <face_ref> | <dplane_ref> ) BY number ';'

    | TANGENT TO ( <face_ref> | <dplane_ref> ) ';'

    | PERPENDICULAR TO ( <face_ref> | <dplane_ref> |

        <edge_ref> | <daxis_ref> ) ';'

    | PARALLEL TO ( <face_ref> | <dplane_ref> ) ';'

    | THROUGH ( <face_ref> | <edge_ref> | <vert_ref>

        | <daxis_ref> | <dpoint_ref> ) ';'

    | ANGLE ( <face_ref> | <dplane_ref> ) BY number ';'

<datum_axis> ::= DATUM_AXIS <name> CREATED

    <references> <a_description>

END_DATUM

<a_description> ::= DESCRIPTION

    { <subdef_daxis> }+

END_DESCRIPTION

<subdef_daxis> ::= THROUGH <edge_ref> ';'

    | THROUGH ( <vert_ref> | <dpoint_ref> ) ';'

    | INTERSECTION OF ( <face_ref> | <dplane_ref> )';'

    | PERPENDICULAR TO ( <face_ref> | <dplane_ref> ) ';'

<datum_point> ::= DATUM_POINT <name> CREATED

    <references> <v_description>

END_DATUM
\[<v\_description> ::= DESCRIPTION
\
   \{ <subdef\_dpoint> \}+
   END\_DATUM
\]

\[<define\_dpoint> ::= ON <vert\_ref> ',';
   | AT CENTER OF <edge\_ref> ',';
   | INTERSECTION OF ( <face\_ref> | <dplane\_ref> |
   | <edge\_ref> | <daxis\_ref> ) ',';
\]

A.4 Modifying Features

\[<c\_feature> ::= FEATURE <name> CHAMFERED
   <references>
   <c\_description>
   <display>
   END\_FEATURE
\]

\[<c\_description> ::= DESCRIPTION
   <edgelist> ',';
   <chamf\_spec> ',';
   <end\_cond> ',';
   END\_DESCRIPTION
\]

\[<o\_feature> ::= FEATURE <name> ROUNDED
   <references>
   <o\_description>
   <display>
   END\_FEATURE
\]

\[<o\_description> ::= DESCRIPTION
   <edgelist> ',';
   RADIUS number ',';
   END\_DESCRIPTION
\]

\[<edgelist> ::= EDGE id { ',' EDGE id }*
\]
<chamf_spec> ::= ANGLE number WIDTH number
   | WIDTH1 number WIDTH2 number
<end_cond> ::= <face_ref>
   | <dplane_ref>

A.5 Naming Schema

<face> ::= <inherit_face> [ <f_imm_extent> ]
   | [ <ext_context> ]
<inherit_face> ::= START_F id DOT id DOT id
   | END_F id DOT id DOT id
   | SIDE_F id DOT id DOT id
   | CHAMFER_F ( ' <edge> ' )
   | ROUND_F ( ' <edge> { ', ' <edge> }* ' )
<f_imm_extent> ::= EDGES ( ' <edge> { ', ' <edge> }* ' )
<edge> ::= <e_imm_context>
   | [ <from_to> ]
   | [ <e_orient> ]
   | [ <ext_context> ]
<e_imm_context> ::= START_E id DOT id DOT id
   | END_E id DOT id DOT id
   | SIDE_E id DOT id DOT id
   | INT_E ( ' <face_list> ' )
<from_to> ::= FROM <vert> TO <vert>
<e_orient> ::= FLOW NONE
   | FLOW ( ' ( '+' | '-' | 0 ) FEATURE id ' )
<vert> ::= START_V id DOT id DOT id
   | END_V id DOT id DOT id
   | INT_V ( ' <face_list> ' ) <v_orient>
   | [ <ext_context> ]
<v_orient> ::= FLOW NONE
               | FLOW '(' (' ' | '-' | 0 ) FEATURE id FACE id
               | { (' ' | '-' | 0 ) FEATURE id FACE id }+
                     ')',
<ext_context> ::= ADJACENT TO <ith_context> { ',' <ith_context> }
<ith_context> ::= '(' integer { '(' ( <inherit_face> | 
                       <e_imm_context> ) integer ')' }+
                    ')',
<face_list> ::= FACE id { ',', FACE id }*

A.6 Cross Sections

<x_cross_section> ::= SECTION id { <geometry> | 
                      <components> | 
                      <auxiliaries> | 
                      <constraints> | 
                      <projection> | 
                      <const_disp> | 
                      <root_info> | 
                      <objects>
                    }*

                    END_SECTION
<geometry> ::= GEOMETRY
              { <geom_decl> }*

              END_GEOMETRY
<geom_decl> ::= <point_decl>
               | <line_decl>
               | <circle_decl>
<p_point> ::= POINT id '=' '(' number ',', number ')' ';'
<line_decl> ::= LINE id '=' '(' number ',', number ',', number ')';
<circle_decl> ::= CIRCLE id '=' '(' number ',', number ',', number ')';
<components> ::= COMPONENTS
  {
    <comp_decl> *
  } END_COMPONENTS
<comp_decl> ::= <vert2d_decl>
  | <segment_decl>
  | <arc_decl>
  | <top_line_decl>
  | <top_cir_decl>
<vert2d_decl> ::= VERTEX id ON id <fix_option> ';
<segment_decl> ::= SEGMENT id ON id FROM id TO id <fix_option> ';
<arc_decl> ::= ARC id ON id FROM id TO id <fix_option> ';
<top_line_decl> ::= TLINE id ON id <fix_option> ';
<top_cir_decl> ::= TCIRCLE id ON id <fix_option> ';
<fix_option> ::= FIXED
  | NONFIXED
<auxiliaries> ::= AUX_COMPONENTS
  {
    <aux_decl> *
  } END_AUX_COMPONENTS
<aux_decl> ::= <vertex2D_decl>
  | <top_line_decl>
  | <top_cir_decl>
<constraints> ::= GEOMETRY_CONSTRAINTS
  {
    <const_decl> *
  } END_CONSTRAINTS
<const_decl>   ::= <d_const_decl>
|   <g_const_decl>
<d_const_decl> ::= ( DISTANCE id | ANGLE id ) <geom_pair> '='
|                   '(' id ',' string ',' id ')' ''
|       RADIUS id '(' id ')' '='
|                   '(' id ',' string ',' id ')' ''
<g_const_decl> ::= ( TANGENT       | PARALLEL       | PERPENDICULAR |
|                   CONCENTRIC    | ALIGN         | ON   |
|                   ) <geom_pair> ';'
<geom_pair> ::= '(' id ',' id ')''
<const_disp> ::= id '(' number ',' number ',' number ')' '';
<projection> ::= PROJECTION
|                   { <vert_decl> | <edge_decl> | <face_decl> }*
|                   { <vert_proj> | <edge_proj> }*
|                   END_PROJECTION
<vert_proj> ::= VERTEX id PROJECTED FROM ( <vert_ref>     | 
|                   <dpoint_ref> | 
|                   <edge_ref>    | 
|                   <daxis_ref>  ) ','''
<edge_proj> ::= EDGE id PROJECTED FROM ( <edge_ref>     | 
|                   <daxis_ref>  | 
|                   <face_ref>    | 
|                   <dplane_ref> ) ','''
<root_info> ::= ROOT_SELECTION
|                   <level_num>
<cur_level>
{ <cluster> }*
END_ROOT_SELECTION

<level_num> ::= NUMBER OF LEVEL IS integer ';
<cur_level> ::= CURRENT LEVEL is ( ALL | integer ) ';
<cluster> ::= BEGIN CLUSTER id [ '(' id { ',' id }* ')' ] ';
               LEVEL integer <int_pair> id { ',' id }* ';
               END_CLUSTER ';
<int_pair> ::= '(' integer ', integer ')'
<objects> ::= OBJECTS
            { <loop2d> }+
            <face2d>
            END_OBJECTS
<loop2d> ::= LOOP id of id { ',' id }* ';
<face2d> ::= FACE id INSIDE is [ OUTSIDE id { ',' id }* ] ';

VITA
VITA

Xiangping Chen was born on October 12, 1965 in Zhejiang, China. He received a B.S. and an M.S. in Mechanical Engineering from Zhejiang University, Hangzhou, China, in 1985 and 1988 respectively. He received his second M.S. in 1991 in Computer Science from Louisiana State University, Baton Rouge, Louisiana. He earned his Ph.D. in 1995 in Computer Sciences from Purdue University, West Lafayette, Indiana.

His previous research work includes geometric and solid modeling, computer graphics and applications, CAD/CAM, feature-based and constraint-based design.