

Geometric Reasoning for Process Planning

Jonathan Charles Salmon

Ph.D.

The University of Edinburgh

1997



Abstract

A Feature Oriented Detail Design System (FODDS2) is described that allows design of 2½D components using machining features. The geometric reasoning required to generate anteriority constraints for subsequent process planning is described with accompanying problems, experiments and proposed solutions. Many of the test components are from other institutions, and the success of the reasoning confirms the robustness of the approach.

The geometric algorithms currently unavailable in commercial solid modellers, but required for the system, are described. These are particularly the Minkowski sum and the medial axis operator. Some implementation work is shown.

All features in the system are described in a consistent manner in terms of a 'tool profile' and 'cutter path' allowing new features to be added simply and easily and ensuring that geometric reasoning can still be performed.

Novel work in the area anteriority checking and of proximity checking of feature-based designs is presented.

Acknowledgements

I would like to acknowledge the support of a number of individuals and groups.

Angela Noble for long-suffering support and tolerance as I slaved long into the nights, my parents for their continuous support over the years for my studies and Frank Mill, my supervisor over the years.

To the other members of the Manufacturing Planning Group for their help and at times extremely frank criticism, namely Stephen Warrington, Dominique Jacquel, Andrew Sherlock and Jane Naish and to former members of the group, Gordon Little, and Ser Chong Chia.

To the department in particular the tolerance of the two heads of department during the long wait for this volume, George Alder and Bill Easson (in chronological order), as well as the staff, researchers, postgraduates, secretarial and technical staff.

To Roy Jones and Doug Clarke (and all those in the Cosmic Noodle, Wan Chai) to whom I promised a bottle of whisky if I failed to get this in on time.

To the Djinn consortium, Sir John Leslie and the Diggers, Carsten Friedrich, Sascha Kuessner and Uwe Holznagel.

The errors, however, are all my own.

Jonathan Salmon

Edinburgh, December 1997.

Table of Contents

Abstract i

Declaration ii

Acknowledgements iii

1 Introduction 1

1.1 Background 1

1.2 Scope and Problem..... 1

1.3 Aims and Objectives 2

1.3.1 *Aims*..... 2

1.3.2 *Objectives*..... 3

1.4 Organisation of Thesis 4

2 Literature Review 6

2.1 Shah’s Summary of Feature Based Design Approaches 6

2.1.1 *Human Assisted Feature Definition* 7

2.1.2 *Automatic Feature Recognition*..... 7

2.1.3 *Design by Features*..... 9

2.1.4 *Parametric Geometric Modelling*..... 11

2.2 Wilson and Pratt..... 12

2.3 Arikan’s Design Using Machining Operations 13

2.4 Material Removal Shape Element Volumes 13

2.4.1 *How Does a Feature Relate to a Machining Operation?*..... 14

2.4.2 *Relating Features to Workpiece Shape* 14

2.4.3 *Carrying Machining Information in Features*..... 15

2.4.4 *Fixed library vs. Define as Needed* 15

2.4.5 *Complete Definition vs. Ambiguity*..... 16

2.4.6 *Machining Applications Supported*..... 16

2.4.7 *Accessibility for Machining*..... 16

2.5 Gupta’s Review of Manufacturability Analysis..... 16

2.6 Bidarra’s Feature Interaction Detection 18

2.7 Vandenbrande’s Spatial Reasoning..... 19

2.8 PART Process Planning System..... 19

2.9 GARI: An Expert System for Process Planning..... 20

2.10 Summary 21

3 Background 22

3.1 Manufacturing Planning Group, Edinburgh..... 22

3.1.1	<i>The HAPPI Process Planner.....</i>	25
3.1.2	<i>The SESAME Project</i>	30
3.2	Computer Integrated Manufacturing (CIM).....	32
3.3	Solid Modelling.....	35
3.3.1	<i>Spatial Occupancy Methods.....</i>	38
3.3.2	<i>Octree modelling.....</i>	39
3.3.3	<i>Depth maps.....</i>	40
3.3.4	<i>Constructive Solid Geometry.....</i>	41
3.3.5	<i>Boundary Representation.....</i>	44
3.4	Geometric Algorithms.....	46
3.4.1	<i>Minkowski Sum.....</i>	46
3.4.2	<i>Medial Axis.....</i>	52
3.4.3	<i>Definition of 2½D.....</i>	53
3.5	Features	54
3.5.1	<i>Applications.....</i>	57
3.5.2	<i>Feature Definitions</i>	58
3.5.3	<i>Feature Attributes</i>	61
3.5.4	<i>Manufacturing Feature Justification.....</i>	63
3.6	Summary	64
4	Geometric Reasoning for Process Planning	65
4.1	Definition of a Component.....	65
4.2	The Feature Library.....	67
4.2.1	<i>Holes</i>	68
4.2.2	<i>Slots.....</i>	69
4.2.3	<i>Pockets</i>	69
4.3	A Grounding for Manufacturing Features.....	69
4.3.1	<i>Cutters with Limited Access</i>	72
4.3.2	<i>Tolerancing</i>	73
4.4	Generating Feature Volumes for Manufacturability Analysis	74
4.5	The Geometric Reasoning Algorithms.....	78
4.5.1	<i>Void Recognition.....</i>	80
4.5.2	<i>Feature Presence.....</i>	83
4.5.3	<i>Access Problem Detection.....</i>	84
4.5.4	<i>Through Feature Detection.....</i>	96
4.5.5	<i>Proximity Detection.....</i>	97
4.5.6	<i>Intersection Detection</i>	99
4.5.7	<i>Hole Interference Detection.....</i>	100
4.5.8	<i>Alternate Access Direction.....</i>	102
4.6	Summary	102

5	Feature Oriented Detail Design System.....	104
5.1	User Requirements.....	104
5.1.1	<i>Requirements of the 'Front-End' User.....</i>	<i>104</i>
5.1.2	<i>Requirements for Downstream Applications.....</i>	<i>105</i>
5.2	Shah's Feature Based Design System Characteristics.....	106
5.2.1	<i>Representation of Feature Definitions</i>	<i>106</i>
5.2.2	<i>Level of Support of User-defined Features.....</i>	<i>107</i>
5.2.3	<i>Type of Linkage with a Geometric Modelling System</i>	<i>108</i>
5.2.4	<i>Application Context.....</i>	<i>108</i>
5.2.5	<i>Support for Feature Validation.....</i>	<i>109</i>
5.3	The Design Features in FODDS2.....	109
5.3.1	<i>The Positive Workpiece Construction Features</i>	<i>109</i>
5.3.2	<i>The Primitive Material Removal (Negative) Features.</i>	<i>110</i>
5.3.3	<i>Tolerancing and Surface Finish.....</i>	<i>115</i>
6	Experiments	117
6.1	Focussed Experiments.....	117
6.1.1	<i>Feature Variety Test.....</i>	<i>117</i>
6.1.2	<i>Access test for Crossed Slots with Hole.....</i>	<i>118</i>
6.1.3	<i>Thin Wall Demonstration</i>	<i>121</i>
6.2	Test Components.....	123
6.2.1	<i>The HAPPI component.....</i>	<i>123</i>
6.2.2	<i>The Edinburgh Composite Component (TECC).....</i>	<i>124</i>
6.2.3	<i>The Heriot-Watt 2 'MacTaggart Scott' Component.....</i>	<i>126</i>
6.2.4	<i>The Heriot-Watt 'Teddy Bear'</i>	<i>127</i>
6.2.5	<i>The Han1 Component.....</i>	<i>129</i>
6.2.6	<i>The Gadh2 Component.....</i>	<i>130</i>
6.2.7	<i>The Regli component.....</i>	<i>132</i>
6.3	Summary	133
7	Conclusions	134
7.1	Summary of Conclusions	135
7.2	Further Research	135
7.2.1	<i>Addition of More Powerful Features.....</i>	<i>136</i>
7.2.2	<i>Additional Feature Validation.....</i>	<i>136</i>
7.2.3	<i>Feature Relaxation Techniques.....</i>	<i>136</i>
7.2.4	<i>Alternate Feature Views.....</i>	<i>137</i>
7.2.5	<i>Agent Based Approaches to Speedup the Overall System.</i>	<i>137</i>
7.2.6	<i>Improved Tolerancing Mechanism.....</i>	<i>137</i>
7.2.7	<i>Parametric and Constraint Modelling</i>	<i>137</i>
7.2.8	<i>Geometric Algorithms</i>	<i>137</i>

References.....	138
Appendix A. FODDS2 Implementation Details	150
System Architecture	151
<i>The FODDS2 Graphical User Interface</i>	<i>153</i>
FODDS2 Data Structure	154
Implementation of the Method/Feature type call	158
<i>Tree Building, Editing and Interrogation.....</i>	<i>161</i>
<i>Building Features.....</i>	<i>162</i>
<i>Operator or Branch Node Generation Functions</i>	<i>163</i>
<i>The Requirements of a Feature Oriented Design System.....</i>	<i>164</i>
Appendix B. Relationship Lists	167
Han1 Relationships	167
Gadh2 Relationships	171
HAPPI Relationships	171

Table of Figures

Figure 1 Human Interactive Feature Definition 7

Figure 2 Automatic Feature Recognition..... 7

Figure 3 Design by Features 9

Figure 4 Pratt & Wilson’s Feature Taxonomy..... 13

Figure 5 Slabs and Steps 15

Figure 6 Manufacturability Analysis According to Gupta..... 17

Figure 7 A Schematic Representation of the PART System (after Hout89)..... 20

Figure 8 Summary of Feature Based Research in the MPG 24

Figure 9 Architecture of HAPPI Process Planner 25

Figure 10 Extract from HAPPI Component Model 26

Figure 11 Architecture of the Comparison Module of the HAPPI Planner 28

Figure 12 A HAPPI Example Component..... 29

Figure 13 Relationships in an Example HAPPI Database 29

Figure 14 CEW System Overview..... 30

Figure 15 Lead Times vs. Engineering Methodology..... 32

Figure 16 A 2D Drafting Package (EasyCAD)..... 36

Figure 17 Impossible Objects 37

Figure 18 An Example Object and Octant Numbering..... 39

Figure 19 Octree Representation of Example Object 39

Figure 20 Original and Reconstructed Golf Club Head..... 41

Figure 21 Diagram of a Planar Halfspace 42

Figure 22 A Box and a Cylinder in terms of their Constituent Halfspaces..... 42

Figure 23 A Simple CSG Tree and Model..... 43

Figure 24 A B-Rep Hierarchy (ACIS) 44

Figure 25 A Non-Manifold body 45

Figure 26 Minkowski Sum of Two Polygons M and N 46

Figure 27 Minkowski Sum Combines the Shape Characteristics of its Arguments..... 47

Figure 28 Navigating an Object (R) amongst Obstacles (O1, O2) using Interference Detection 48

Figure 29 Navigating an Object (R) amongst Obstacles (O1,O2) using Minkowski Sums 48

Figure 30 Machining Volume Generation 49

Figure 31 Problems with Minkowski Blends (after [Midd88])..... 51

Figure 32 Dilation using a Minkowski Sum 51

Figure 33 An Example of an Object created using CTS 52

Figure 34 A 2D Region and its Medial Axis..... 53

Figure 35 A Body Formed from Three Intersecting Cylinders..... 55

Figure 36 Blank, Features, Component and Delta-Volume..... 67

Figure 37 Feature Volumes of Cutters with Limited Access..... 72

Figure 38 'Tool' Profiles in FODDS2..... 75

Figure 39 Void Detection 81

Figure 40 Void Detection Algorithm..... 83

Figure 41 Feature Presence Algorithm 83

Figure 42 Feature Presence Detection 84

Figure 43 Possible Access Directions for Slot and Hole 85

Figure 44 Problems using Access Vectors..... 86

Figure 45 Evolution of Access Body Types 87

Figure 46 A Problem in Recognising an Access Problem 88

Figure 47 Problems with a Simple Grown Access Body on an Angled Surface..... 88

Figure 48 Access body e) is less prone to Problems of Angle than Type d)..... 89

Figure 49 Nested Pockets with a Coplanar Sidewall 90

Figure 50 Access Problem Algorithm..... 92

Figure 51 Access Bodies and Anteriority Constraints for the Triply Nested Slot 93

Figure 52 The Slot and Hole case..... 93

Figure 53 Crossed Slots with Hole 94

Figure 54 Improved Anteriority Algorithm 96

Figure 55 Through Feature Detection..... 96

Figure 56 Thin Wall Detection 98

Figure 57 Thin Walls Algorithm..... 99

Figure 58 Intersection Detection Algorithm 100

Figure 59 Hole Access Interaction Algorithm 101

Figure 60 Problems with Hole Access Interference..... 102

Figure 61 Example of Hole Types 113

Figure 62 The Simple 'Ell' Component..... 116

Figure 63 The Feature Library Demonstration Part..... 117

Figure 64 Crossed Slots with a Hole..... 119

Figure 65 Offset Hole in Crossed Slots 120

Figure 66 Four Crossed Slots with Hole..... 121

Figure 67 Thin Wall Demonstration Component..... 121

Figure 68 The HAPPI Test Component..... 123

Figure 69 The HAPPI Test Component with Features and Access Bodies 124

Figure 70 The TECC Component..... 124

Figure 71 The Heriot-Watt 2 Component..... 126

Figure 72The Heriot-Watt 'Teddy Bear' 127

Figure 73 Example of the 3D Geometry of a Complex Pocket.....	128
Figure 74 The Han1 Test Component in the FODDS2 GUI.....	129
Figure 75 The Gadh2 Test Component.....	130
Figure 76 The Regli Component in FODDS2.....	132

Table of Tables

Table 1 Geometric Reasoning Body Methods and their Equations 78

Table 2 The Primitive Material Removal Features 112

Table 3 The HAPPI Test Component Access List..... 123

Table 4 Results of Anteriority Checks on Han1 Component..... 130

Table 5 Relationships of the Regli Component 133

1 Introduction

1.1 Background

Traditional manufacturing techniques distinguish between the sequential processes of designing, process planning and manufacturing. Frequently as the design proceeds through these stages it is rejected and earlier stages must be reiterated. The earlier mistakes can be caught the cheaper they are to remedy.

Computer Aided Process Planning systems developed in the 1980s were intended to achieve two main goals. Firstly, to speed up the process planning task, and secondly to improve the quality and cost of resultant process plans.

In order to ensure that a near optimal process plan has been discovered, a great many alternative process plans must be explored. This is a drawback to both classical rule-based approaches and even to evolutionary algorithm approaches.

To reduce the size of the search space for the process planner, it is beneficial if clearly unmanufacturable process plans are pruned from the search space at the earliest possible juncture.

Many CAPP systems, though receiving information from CAD systems and feeding CAM systems, do not contain a solid modeller.

1.2 Scope and Problem

If geometric algorithms can be developed that can be performed on a design in order to infer manufacturability problems at an early stage, this can dramatically prune the search space of process plans. To ensure this reasoning is performed as early as possible, it is sensible to link this manufacturability analysis stage to the design system and perform the manufacturability analysis either immediately on finishing a design, or concurrently with design changes.

This would allow many manufacturing problems to be identified by the system and flagged to the designer at an early stage and thus at a near insignificant cost to the finished product. It is not necessary to decide on particular machine/tool/setup

combinations, and by avoiding these process selections, the search space is pruned to a greater degree and at less expense than by making the more detailed decisions. (The tree is pruned by lopping off a few large branches rather than many small twigs.)

Fundamental to many CAPP and CAM systems is the use of features, though different schools differ on their definition of a feature. In order to integrate the manufacturability analysis into the CAD system, a CAD system where components are designed in terms of manufacturing features shall be used.

The design by manufacturing features approach is often considered a significant drawback of feature-based design systems, but it seems reasonable that an additional front-end with either a feature recognition or feature transformation system, producing suitable manufacturing features as its output, would silence many of these objections. Such a feature transformation system is briefly discussed in Chapter 3.

Perhaps the chief problem for CAPP systems, and a problem that is not readily amenable to CAPP systems without a solid modeller, is that of machining accessibility leading to machining precedence constraints between features. Other manufacturability constraints are also of interest.

1.3 Aims and Objectives

1.3.1 Aims

This thesis has the following aims:

1. To show that anteriority inferencing algorithms are necessary for automated process planning from a feature based design system
2. To develop such algorithms to satisfy Aim 1 and prove that these algorithms indeed satisfy Aim 1.
3. To show through the development of a Feature Based Design system (FODDS2) that with these algorithms, it is possible to design a significant proportion of mechanical components and that the inferencing algorithms do indeed detect anteriority errors, and that NC code can be generated to automatically produce a selection of these components.

4. To show that the modelling of negative features as a Minkowski sum of tool and toolpath is a powerful method of ensuring ease of extensibility of the feature set, and provides a common robust mechanism for adding geometric algorithms for manufacturability analysis of feature based designs.

1.3.2 Objectives

The objectives of the thesis are more extensive.

1. A geometric algorithm to elicit ordering constraints in the manufacturing of features shall be developed.
2. In order to demonstrate these algorithms it will be necessary to build a feature based design system that shall be known as Feature Oriented Detail Design System version 2 (FODDS2).
3. FODDS2 shall have a sufficiently rich, powerful and flexible feature library to allow the modelling of significant real world components.
4. In order to prove the generality and intuitiveness of the design metaphor used in FODDS2, and in order to show that this design methodology allows real designs to be created in reasonable timescales it will be necessary to have a system that allows any competent user to easily create component instances with a minimum of training. Thus, a system with a near professional user interface and operation will be required.
5. The features meta-model and the FODDS2 system shall allow new features to be added with relative ease. An important property of this meta-model and the geometric reasoning algorithms is that all algorithms must still operate correctly as new features are added. Thus, it is important that the features meta-model is general enough to allow addition of more features, but sufficiently constrained that reasoning algorithms can operate on any feature type generated with this features meta-model.
6. The geometric algorithms for anteriority checking require a set of tools that enable other useful algorithms for process planning to be developed quite easily. This set of algorithms shall be demonstrated.

A set of components falling into two main categories shall be tested. That is, they will be built using the FODDS2 system and the geometric algorithms run on those components to identify machining problems and ordering constraints on feature manufacturing. For some components, NC code shall be produced and the components manufactured on a BridgeportII NC mill in order to prove the system in reality. The first set of components will be components with a limited number of features whose purpose is directed towards illustrating a particular problem. The second set will consist of real and test components from companies and research groups throughout the world in order to show the general applicability of the system.

1.4 Organisation of Thesis

Chapter 1 introduces the thesis and contents and contains the aims and objectives of the thesis as well as a brief summary of the remainder of the thesis.

Chapter 2 contains a Literature Review of those groups and individuals throughout the world undertaking work in the area of manufacturability analysis and related areas. These areas include feature based design, computer aided process planning. Additionally the chapter contains a brief review of solid modelling techniques concluding with a review of particular useful geometric algorithms, the Minkowski sum and the medial axis.

Chapter 3 gives the background to the thesis, in particular prior work within the Manufacturing Planning Group of the Department of Mechanical Engineering at The University of Edinburgh (MPG) in order to place the work in this thesis in context. This chapter also reviews solid modelling, certain geometric algorithms and features.

Chapter 4 details the geometric reasoning algorithms in FODDS2 necessary to infer anteriority constraints as a necessary precursor to automated process planning as well as additional algorithms of importance in validating feature based designs for subsequent process planning.

Chapter 5 covers aspects of the Feature Oriented Detail Design System 2, as are relevant to the thesis.

Chapter 6 details experiments carried out to test the system. These fall into two groups; focussed experiments to show particular aspects of the system and more general experiments with a selection of parts from various research groups and industrial components to show the robustness of the system. Automatic production of a machined part from the system is shown.

Chapter 7 concludes the thesis with a summary of the work, conclusions drawn, an evaluation of the original content of the thesis and pointers towards further work.

2 Literature Review

This Literature Review chapter collects the work undertaken in other research groups that is of particular relevance to this thesis. Thus it covers work in the areas of features, feature based design and feature recognition and computer aided process planning as well as some geometric algorithms.

In the area of features, Shah's summary of approaches to Feature Based Design is discussed. Kramer's work on Material Removal Shape Element Volumes [Kram92] whilst at NIST is discussed.

In the areas of manufacturability analysis, the work of Gupta [Gupt95] and Nau from the University of Maryland, and Bidarra [Bida97] from Delft, along with Vandenbrande [Vand93] and Requicha from the University of Rochester are discussed.

The PART process planning system from the University of Twente [DeJo94] is reviewed.

A review of the work of the Manufacturing Planning Group of The University of Edinburgh including the work on the HAPPI Process Planner and the SESAME Simultaneous Engineering System, and a background section on solid modelling and the geometric algorithms of Minkowski Sums and medial axes can be found in Chapter 3.

2.1 Shah's Summary of Feature Based Design

Approaches

Shah [Shah91b] surveyed CAD/feature-based process planning and NC programming techniques in 1991. He classifies these systems into generic categories. Shah identified four different methods of incorporating features into geometric models

- Human assisted feature definition using geometric models.
- Automatic (machine) recognition of features from geometric models.

- Design by features
- Parametric geometric modelling

2.1.1 Human Assisted Feature Definition

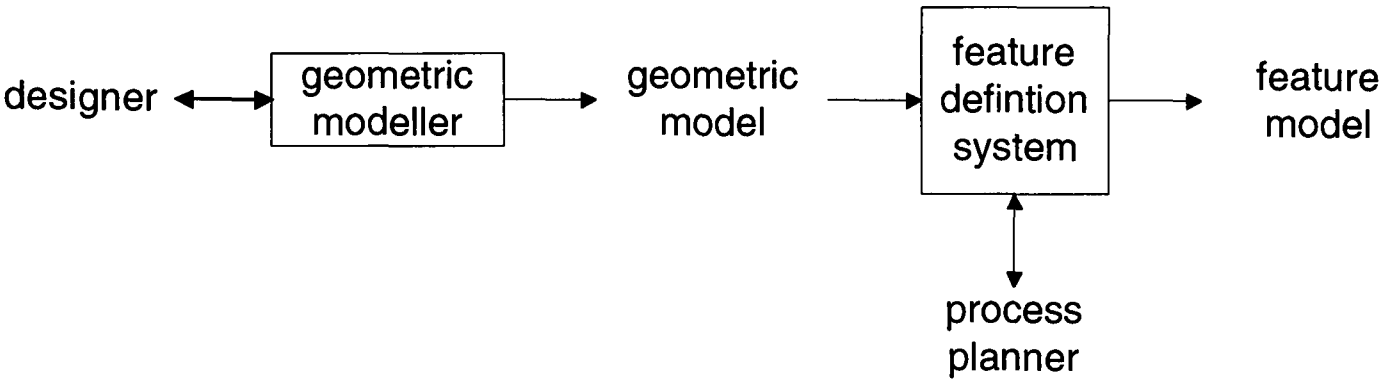


Figure 1 Human Interactive Feature Definition

The Human Interactive Feature Definition normally involves a designer first generating a solid model of a component and then subsequently, the process planner (or other user requiring 'features') embellishes the solid model with feature tags (see Figure 1). Usually the designer is required to label every surface in the solid model. This is both demanding in time and effort, and requires the process planner to perform mentally much of the process planning task prior to automated process planning. In addition the feature tags tend to be attached to the surfaces and edges of the B-rep model of the component and not to the material removal volumes. Lastly, there is no provision for tagging intermediate features that are required during machining but play no physical part in either the stock or the finished component (though this disadvantage can be levelled at most of the other methods).

2.1.2 Automatic Feature Recognition

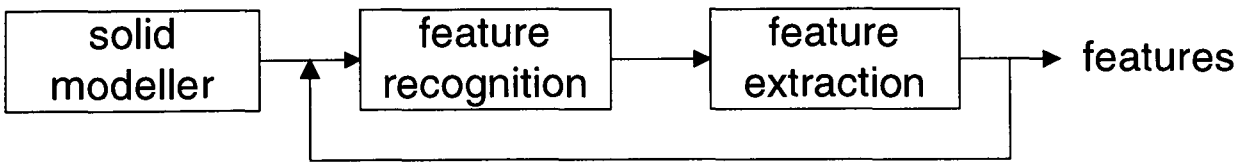


Figure 2 Automatic Feature Recognition

Automatic Feature Recognition attempts to automate the above process. Again a solid model of the desired part is fed into the system and the system attempts to automatically extract features of interest to downstream processes from the solid model (see Figure 2). Many groups are working on automatic feature recognition

including Corney and Clark [Corn90][Tutt97], Regli [Regl95], Joneja [Yang97] and Jared [Jare89]. Many of the disadvantages of this approach are similar to the previous approach.

The system is trying to extract features from a solid model that has already been designed with a particular set of design features in mind to satisfy the functional requirements of the component. It seems reasonable that a good design system should not immediately throw this functional information away, but retain it for subsequent downstream processing.

Secondly, a solid model only contains information about nominal geometry, it contains no information regarding tolerances or surface finishes. This information is crucial for subsequent process planning.

Thirdly, current feature recognition systems can only recognise a proportion of the features actually in existence on a component, requiring interactive definition of certain features. Here, it should be emphasised that any system that can automatically recognise 90% of the features on a component will certainly save a process planner a great deal of drudgery. The process planner would be quite pleased to only have to deal with the more complex and interesting features.

The work on Feature Recognition undertaken in PART [Hout89][DeJo94] and many of the FR systems of the early 80s uses shape grammars to recognise features within a (typically B-rep) solid model. This approach, whilst working adequately for isolated features on a component, quickly falls down when features intersect. It is frequently these interactions between features that define a component's function, and so inability to recognise interacting features is a severe problem.

2.1.3 Design by Features

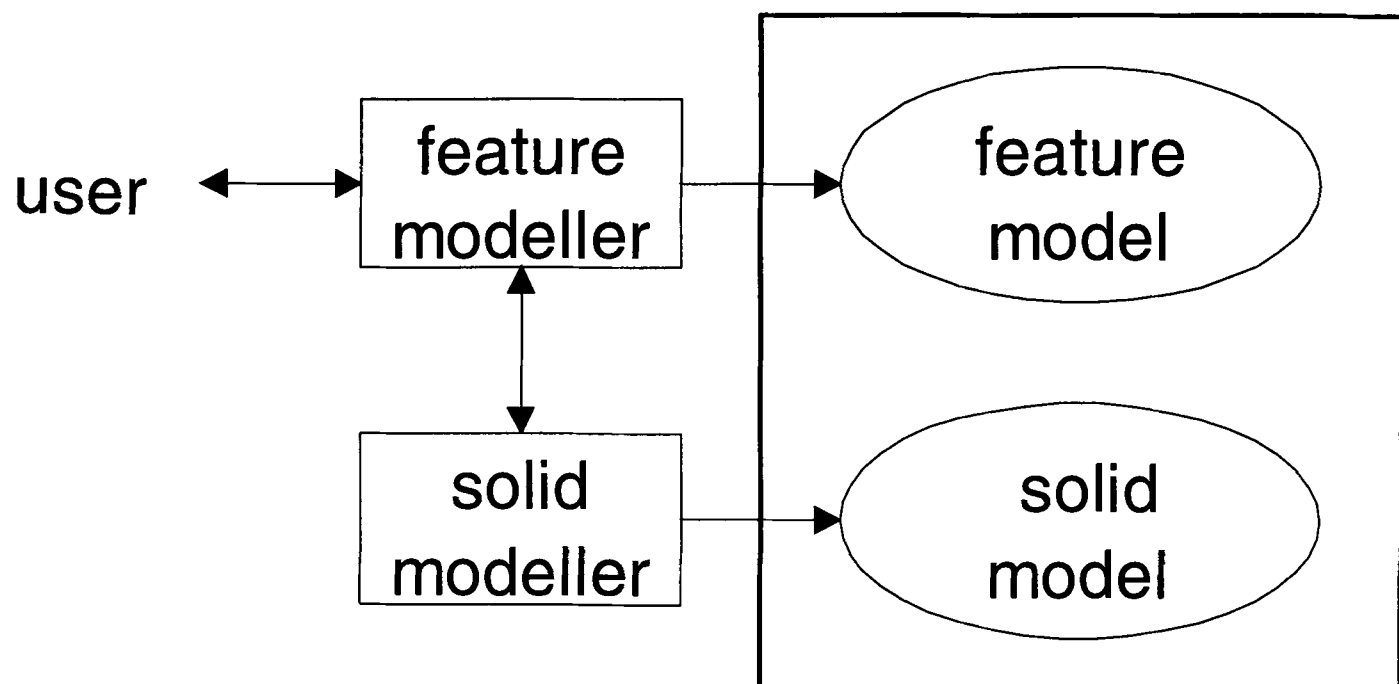


Figure 3 Design by Features

It is possible that the reason for some features being unrecognisable is that they are also unmanufacturable. It is equally possible that some features that are recognised are not immediately manufacturable in the form in which they are recognised. This is especially true of systems that recognise protrusions as features. Little [Litt97] and others [Cham93] have worked on systems for taking feature based descriptions of models containing positive features and turning them into a feature-based description containing only negative features. This is but one class of feature transformation that is frequently necessary.

Particular problems in feature recognition according to Corney and Clark [Corn90][Tutt97] are dealing with small scale shape variation such as chamfers and fillets, where the topology of the B-rep model is significantly more complex, but the complexity is only due to small scale features that could be neglected. Some work has been carried out in suppressing these small scale variations. Techniques such as medial axis transforms can be used [Tam91][Arms94]. The Corney and Clark approach in particular uses a graph-based method based on face-traversal rather than the more common edge traversal. A useful abstraction that is made is identifying the aspect vector of largely 2½D models and using this approach direction to simplify the quantity of information to be subsequently processed in the feature finder. This can be done as most components, particularly in the domain Corney and Clark are

interested in are predominantly 2½D with minor excursions into 3D such as side holes and pockets in the ‘base’ as well as the top.

2.1.3.1 Design by Features versus Feature Recognition

These two distinct approaches to features, that of Design by Features and Feature Recognition are often perceived as being in conflict. Both approaches produce feature based models of a component, so if either approach were widely adopted in industry the other approach might want for support. In fact, the two areas are complementary. Feature Recognition attempts to produce a feature-based description of a component from its basic geometry. This has the advantage that the source component design can (in theory) be taken from any appropriate 3D CAD package, an appealing idea, especially given the current trend towards Open Systems. However Feature Recognition systems working from a pure solid model will only be able to extract geometrical features, and are unable to infer other non-geometrical information such as tolerances and material type. Feature-Based Design Systems take a somewhat more pragmatic approach. If the intention is to use a feature based process planning system then the input to the CAPP system must be some sort of feature-based design. Either the designer initially produces designs in terms of features or an additional feature recognition step is required. As Pratt has remarked [Prat84](taken from [Wils89]):

“Finally, the feature recognizer informs the user that it has detected the presence of a cylindrical hole in the part. But this is information which the designer was fully aware of when he created the model; it was lost when the system reduced all input to low-level details of topology and geometry.”

Protagonists of Feature Recognition answer this by saying that even if the design is in terms of features, the designer will wish to use functional features, as the designer is currently attempting to generate a functional design, and that the manufacturing features required for feature based process planning are not suitable for this task. An oft quoted example of this is the case where the designer requires a set of ribs for strengthening a component, and is required to design in terms of slots or pockets, the empty space between the ribs.

There are two responses to this argument.

Firstly, the system must ultimately be capable of manufacturing a component; so by constraining the designer to use manufacturing features, the designer is aided in this requirement.

The second counter argument is that the input to the process planning system must still be in terms of manufacturing features, so a system based on manufacturing features is ultimately required. If subsequent to the introduction of a manufacturing feature design system, a development allows transforming functional design features into manufacturing features, there is no reason not to allow this. Indeed functional design features can then be added to the manufacturing oriented system. The converse, where design using functional features is used, but no transform mechanism exists cannot lead to the goal of producing an automated design to manufacture system. Design by manufacturing features can then be regarded as a pragmatic sub goal on the road to automation of the design to manufacture path. The differing views of features is a stumbling block for newcomers into the area and is not limited just to functional versus manufacturing features. In as much as a design passes through various departments in a traditional manufacturing environment, so the views on what constitutes a feature change. Features differ according to the point of view, whether it is coordinate measurement, finite element analysis, design or manufacture.

2.1.4 Parametric Geometric Modelling

In the parametric approach, users build features with standard solid primitives instead of features. The construction procedures and geometry parameters are retained, and the model can be modified by changing the parameters used during the construction. This approach whilst aiding reuse of geometry and easing the editing of well-designed components, does not directly aid CAPP, but is a technology that should be incorporated in the design by features approach.

Parametric design leads naturally to constraint systems. This allows in some way, the nominal dimensions of components to be described in terms of formulae, and that these formulae may be circular and a system of constraints must be solved before the solid model can be generated.

Not all constraint-based solid modelling is necessarily feature based. The 2D drafting and 3D parametric modelling in Solid Edge, for instance, cannot truly be described as feature based. True 3D constraint satisfaction and the resultant generation of valid solids is a difficult research area.

Brunetti et al. [Brun95] propose a feature based model that allows algebraically formulated relationships between feature entities (not just the features themselves, but also subfeatures) to be expressed. This structure is called a FERG (Feature Entity Relation Graph). The real features in this graph are represented as implicit algebraic equations that build up the halfspace decomposition of the form feature.

Shapiro [Shap95] shows how in many commercial systems, parametric and variational modelling may not be robustly supported because the meaning of a “parametric family” is not always well-defined. Shapiro gives examples of how small changes in parametric dimensions can result in huge changes in component topology.

2.2 Wilson and Pratt

Wilson and Pratt in 1989 [Wils89] said:

A feature is a region of interest in a part model

Wilson and Pratt divided representations of features into two major classes:

EXPLICIT: All the geometric details of the feature are fully defined.

IMPLICIT: Sufficient information is supplied to define the feature, but the full geometric details have to be calculated when required.

Pratt and Wilson’s Feature Taxonomy can be seen in Figure 4. From their definition, any feature-based modeller that carries a solid model representation of each feature around with the data structure is said to be *explicit*, whereas any feature-based modeller that merely carries attributes around to be reconstructed when required is an *implicit* modeller. This corresponds to the ideas of an evaluated and unevaluated modeller. This distinction is weak. Many solid modellers perform lazy evaluation, where externally the model always appears evaluated, although internally the model is only evaluated when necessary. Pratt [Wils89] gives examples such as chamfers as

examples of implicit features. This contrasts with ACIS [Mart96][Corn97] where a chamfer can either be implicit or explicit, and whilst for compactness the chamfer data is tagged to an edge, the chamfer can be explicitly created on demand. Pratt and Wilson's distinction seems to be based on the practicalities of modellers in the late eighties rather than a fundamental requirement, and the distinction has become less important as computers and modellers become more powerful.

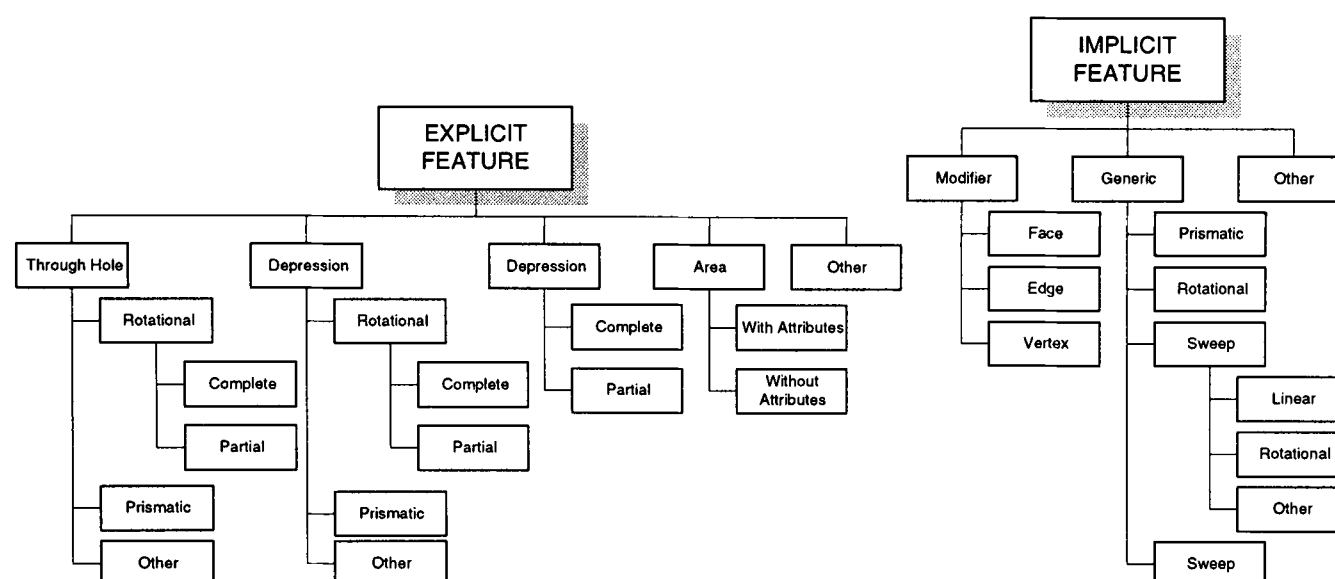


Figure 4 Pratt & Wilson's Feature Taxonomy

2.3 Arikan's Design Using Machining Operations

An extreme example of the design-by-features paradigm is CADP-NC (Computer Aided Design, Process Planning and NC-Programming) of Arikan and Totuk [Arik92]. Their paper is entitled *Design by Using Machining Operations*. It describes a system that allows a feature-based design of a part to be described using detailed machining features including *centre drilling*, *countersinking* and *spot facing* in addition to some more 'traditional' features such as *steps*, *pockets* and *holes*. Though functional, this is perhaps at too low a level of detail and entails the designer knowing more about process planning than is typically the case.

2.4 Material Removal Shape Element Volumes

The work undertaken by Kramer [Kram91] [Kram92] and others [Gupt95] on Machine Removal Shape Element Volumes (MRSEVs) is particularly relevant to the manufacturing features standpoint taken in this thesis.

2.4.1 How Does a Feature Relate to a Machining Operation?

Kramer requires that the volume described by a MRSEV should have no material in it when machining is complete and the operation associated with an MRSEV should remove no material outside the MRSEV. This allows the MRSEV to represent the swept volume of the tool. MRSEVs are allowed to partially machine empty air if required.

Kramer allows disjoint volumes in an MRSEV. This is a distinction between the MRSEV and the traditional negative feature view. It is more usual to disallow disjoint features, but to allow collections of features, such as a pitch circle diameter of holes.

2.4.2 Relating Features to Workpiece Shape

The case Kramer discusses here can be best explained by means of an example. Many feature taxonomies including those of Gindy [Gind89] and Wilson and Pratt [Wils89], differentiate between volumetric features depending on their relationship with the workpiece. Notably a slot is distinct from a step. Gindy differentiates these using external access directions (EADs), checking the access of a feature in six orthonormal approach directions. Kramer does not distinguish in this way. The distinction is not embodied in the MRSEV (or feature) description, but in the machining operation used to remove the material from the workpiece.

This is illustrated in Figure 5. Figure 5a) shows a block from which a thin slab is to be milled from the entire top portion of the block. In each of the figures b-d, a volume of the same dimensions is to be machined. In b), this volume can readily be called a step. In c), though slab milling might be used, the cylindrical protrusion would have to be avoided. In d), slab milling is inappropriate. In each of these cases, the machining process is dependent not on the shape of the volume to be machined but on the accessibility of the feature. If a feature in a feature based design system defines a volume, but not a machining strategy then any feature with the required volume can be used. A single MRSEV could be used in all four cases a-d and subsequent accessibility checks would identify possible machining strategies.

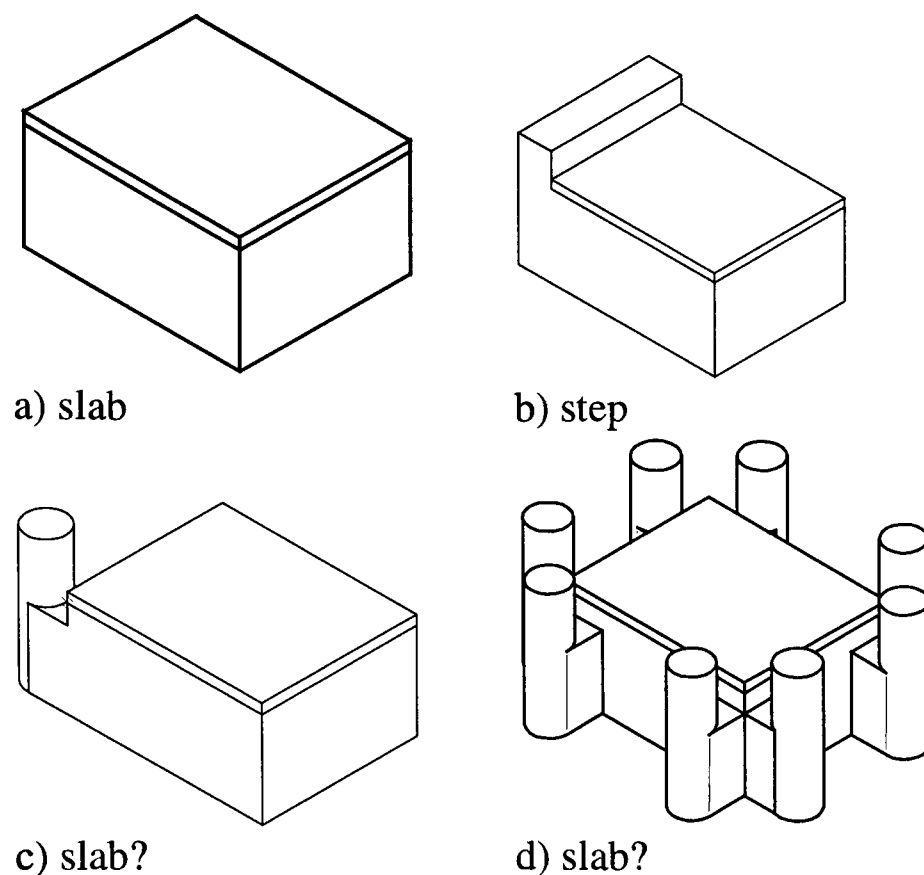


Figure 5 Slabs and Steps

2.4.3 Carrying Machining Information in Features

Kramer assumes that the process planner is the one defining the MRSEVs and as such there is no advantage to be gained by attaching machining information to them. Rather, it is assumed that libraries of machining operations and MRSEVs will be matched appropriately, but independently.

2.4.4 Fixed library vs. Define as Needed

The advantage of a fixed library (says Kramer) is that it is feasible to write computer-executable algorithms which will automatically generate toolpaths for cutting out the volumes in a fixed library. The algorithms are parametric and use feature data fairly directly.

Kramer argues that if a 'define as needed' approach is taken then the most straightforward way that will permit almost any shape to be expressed is to use a boundary representation (B-rep) of the model. Automatic generation of toolpaths from a general B-rep shape is much more difficult.

2.4.5 Complete Definition vs. Ambiguity

Mantyla [Mant89] discusses the problem of premature commitment, where because a single choice of feature geometry (or indeed of feature orientation) is made at an early stage, a commitment has unwittingly been made to a particular manufacturing solution, such as enforcing a certain set up. Mantyla proposes a method of feature relaxation that avoids early commitment. For functional reasons, it is not possible to allow all features to relax. This method is complex.

2.4.6 Machining Applications Supported

The complexity of the MRSEV library increases as milling moves from 2½D through 3, 4 and 5 axis milling. Kramer's proposed library is intended to support 3 axis milling and act as the core for 4 and 5 axis milling.

2.4.7 Accessibility for Machining

Kramer discusses whether accessibility for machining is an attribute that should be carried with a feature and concludes that this is the domain of process planning.

2.5 Gupta's Review of Manufacturability Analysis

Gupta [Gupt97] provides a review of manufacturability analysis. Given a computerised representation of the design and a set of manufacturing resources, Gupta defines the automated manufacturability analysis problem as follows:

1. Determine whether the design attributes (e.g. shape, dimensions, tolerances, surface finishes) can be achieved.
2. If the design is found manufacturable, determine a *manufacturability rating*, to reflect the ease (or difficulty) with which the design can be manufactured.
3. If the design is not manufacturable, then identify the design attributes that pose manufacturability problems.

A flow chart showing this view of manufacturability analysis can be seen in Figure 6.

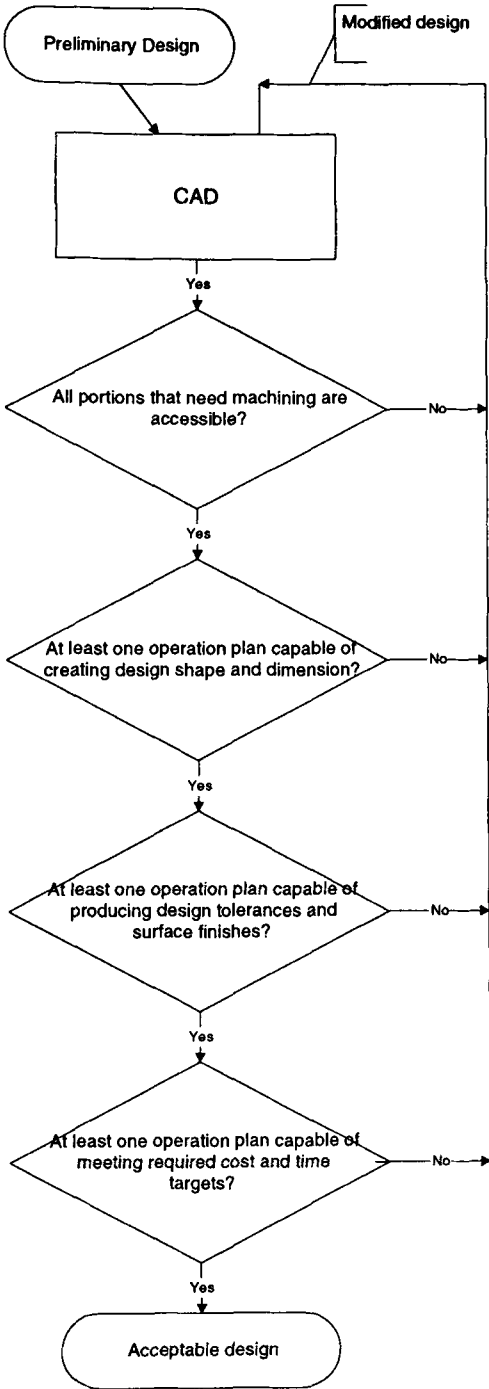


Figure 6 Manufacturability Analysis According to Gupta

[Gupt95] takes a wider view of manufacturability analysis than is taken in this thesis. In addition to accessibility and other geometric constraints on manufacturability, Gupta evaluates the design by considering all manufacturing processes that can manufacture any feature in a part and quantifies these to produce a component’s manufacturability rating. Gupta explicitly produces many operation plans capable of machining a part in order to come up with a finished manufacturability rating. Models in this methodology are created using the MRSEVs of Kramer [Kram92]. Algorithms are presented for modifying the effective feature volume (that volume of a feature required to be machined) depending on a feature’s position in a process plan.

The algorithms are implemented in the IMACS system *IMACS, A System for Computer-Aided Manufacturability Analysis*

Gupta [Gupt95] performs automatic manufacturability analysis for machined parts. Gupta identifies four classes of manufacturing feature formed by different machining processes: drilling, end-milling (closed and open pockets) face milling and side milling.

Gupta searches for thin walls by first faceting the component and then finding the separation distance of close non-adjacent facets. This technique depends on the faceting being a reasonable approximation to the original surfaces.

Gupta also performs some tolerance consistency and redundancy checking.

Gupta's system identifies features using Regli's approach [Regl95]. The system identifies certain precedence constraints between features by reasoning about accessibility, datum-dependency and approachability.

Gupta then defines a machining plan as a set of machining operations and precedence constraints.

2.6 Bidarra's Feature Interaction Detection

Bidarra from Delft University of Technology believes a significant lack in current feature-based systems is the failure of the systems to maintain effective feature validity throughout the design process [Bida97]. That is interactions between features can cause features to change properties, e.g. an addition of a slot might change a blind hole to a through hole, or to change validity, a hole might be completely subsumed in a larger pocket and so not contribute to the final design at all. Bidarra has implemented a feature interaction detection mechanism for eight interaction classes within the SPIFF modelling system, a prototype multiple-view feature-based modeller.

The classes of interaction defined are:

splitting, disconnection, boundary clearance, volume clearance, closure, absorption, geometric and transmutation.

Though Bidarra looks at many feature interactions, he is not concerned directly with identifying machining precedence through accessibility analysis, though the tests for boundary clearance and volume clearance provide some of the necessary information.

2.7 Vandenbrande's Spatial Reasoning

Vandenbrande [Vand91][Vand93] at the University of Rochester performs automatic feature recognition and subsequently performs some spatial reasoning tests to test feature validity and accessibility. Interactions are represented by segmenting the feature into "required" and "optional" volumes. Feature validity tests include nonintrusion, presence and accessibility. Accessibility is further subdivided in to local, partial and semi-infinite accessibility. Vandenbrande mentions an approach for thin wall detection, but does not implement such an algorithm. Vandenbrande's system uses OPS-5 production rules and the PADL-2 solid modeller.

2.8 PART Process Planning System

Perhaps the most successful process planning system containing a solid modeller is PART [Hout89][Hout91][DeJo94][Erve88].

PART is now a commercial process planning system from C3 in the Netherlands, though PART was previously the result of a 40 man year research project. PART and its successors PART-S and FROOM (Features and Relations in Object Oriented Modelling) address many research topics including CAPP software architectures [Jonk92], tool management, [Boog94], constraint satisfaction in feature-based design [Salo95], and process and production planning integration [Lend94]. A schematic representation of the PART system is shown in Figure 7.

PART uses a Feature Description Language and scours the product model to find groups of faces that meet the shape definition of a particular feature. Taking this approach means PART differentiates between pockets and steps, but the features are more specific. The commercial system contains more than forty features, though these features are more restrictive than a general MRSEV.

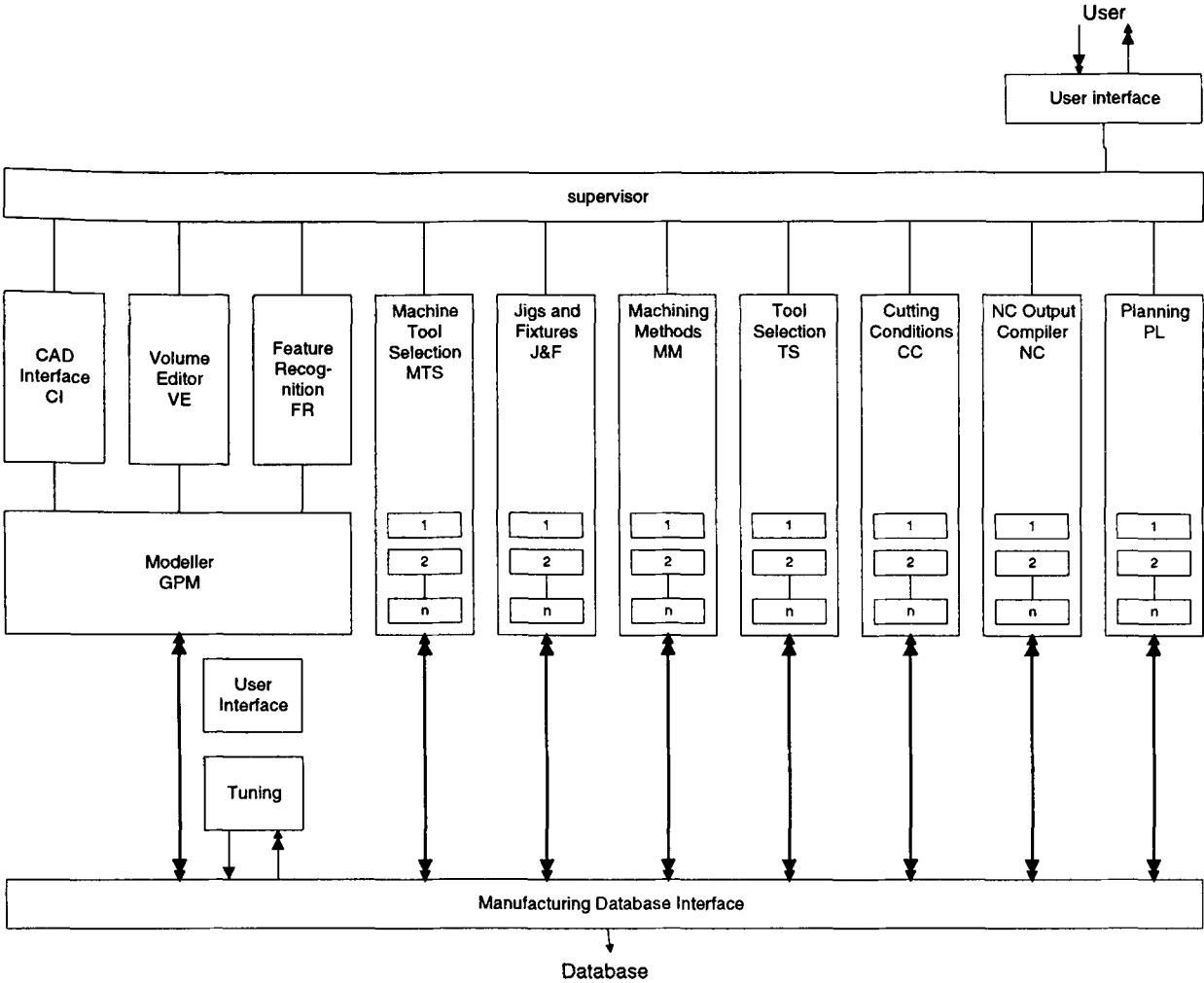


Figure 7 A Schematic Representation of the PART System (after Hout89)

The PART system represents features such as manufacturing features, tolerance features and work holding features. The ‘atomic’ manufacturing features used are based on the CAM-I feature set [Butt86], and includes features such as holes, pockets, slots and corner notches.

Though PART is aware when a feature has more than one possible machining direction, it is not clear what accessibility checking is performed. The accessibility checking may be implicit in the feature recognition process. Precedence problems are considered from a tolerance satisfaction view rather than a geometry view.

2.9 GARI: An Expert System for Process Planning

Descotte and Latombe [Desc81] present a feature based process planning system known as GARI. It consisted of an expert knowledge base composed of manufacturing rules and a planner. GARI did not contain a solid modeller. A part is described to GARI as a series of features and a series of dimensions between features. The dimensions also carry tolerance information. This view bears similarities to a CSG model.

Some features are described in terms of faces, leading to a B-rep type model. As with HAPPI (described in Chapter 3) the description of features with relationships such as “starting-from” and “opening-into” defines machining precedence relationships in the supplied model, so these relationships do not have to be inferred.

The expert system would cause rules to fire depending on information in the part model, which would provide weighted machining and planning information. Some hints are contradictory and the weightings would help resolve these conflicts. A process plan would result.

The types of part modelled in GARI are strictly orthomorphic prismatic parts.

2.10 Summary

Early process planning systems did not contain geometric modelling systems and so accessibility problems were implicit in the component description. Later work particularly has been coupled to feature recognition and subsequent manufacturability analysis and process planning. Some authors have concentrated on spatial reasoning for feature validity maintenance whilst others have tackled the manufacturability analysis problem. There still remains much work to be undertaken in this area.

The interpretation of the features concept differs greatly between researchers.

Bronsvoort and Jansen review the areas of feature modelling and conversion in their paper of 1993 [Bron93].

Hounsell and Case are aiming to better capture Designer’s Intents in [Houn97].

Other reviews of features can be found in a number of papers [Shah91b] [Salo93] [Maro95] [Shah95][Case93].

Other work in this area can be found in [Ande90], [Jone93], [Case94], [Case97], [Chan85], [Dowl94], [Maro95a], [Maro95b], [Mant89], [Opas94], [Salo95], [Laak96], [Cutm91], [Requ89], [Yang97] and [Mill94].

3 Background

The subjects covered in this chapter lay down the groundwork for the subsequent core of the thesis that is manufacturability analysis for process planning through geometric reasoning.

This chapter covers the following areas:

1. A review of the work undertaken in the Manufacturing Planning Group of The University of Edinburgh with special attention to the HAPPI process planner, in order to place the thesis in context.
2. An introduction to the area of concurrent engineering.
3. A review of solid modelling to set the scene for subsequent sections.
4. An overview of two geometric algorithms, those of Minkowski sums and medial axes that are important for the subsequent reasoning algorithms as well as a definition of 2½D.
5. A review of features as a higher level of abstraction than pure geometry.

3.1 Manufacturing Planning Group, Edinburgh

Work on feature based methods began in 1985 with the start of a project entitled Representation, Reasoning and Decision Making in Process Planning with Complex Components [Husb90]. Ending in 1989, this project resulted in the development of a novel prototype CAPP system that was capable of generating full cutting plans for 2½D prismatic parts. The system, known as HAPPI, operates in two stages. The first stage entails the generation of all possible machining methods that can be used as a result of feature interactions arising from geometric tolerances or those where anteriority must be considered. The second stage involves setting the manufacturing methods to be used: an optimisation problem which is NP complete and which was solved by the early use of Genetic Algorithms. The success of this early work has led the group to consider related areas of interest, in particular the investigation of possible methods for automatically generating product representations that would be suitable for process planning purposes.

In 1989, a second major project started entitled Feature Oriented Design [Mill93]. A feature based detail design system was created, capable of assisting a designer to model engineering components with a high level user interface. Many parts from several companies have been modelled using the system which accepted feature descriptions, dimensional and geometrical tolerances and complex blank shapes as would be used with cast stock material. Furthermore, the system, named FODDS, could perform some preliminary geometric reasoning functions which are used to detect intersections, proximities, (thin walls), and tool access. FODDS' output is in the form of a COmponent Description Language, (CODL), file which describes all the information generated and includes a full geometric part description in the form of an ACIS solid model.

The HAPPI CAPP planning system became the core planner in the EC funded SESAME (Simultaneous Engineering System for Applications in Mechanical Engineering) project [Mill94]. Furthermore the CODL language was adopted as a standard for component description transfer by the SESAME Consortium, (e.g. the Straessle GmbH feature based modeller, FeatureM). The SESAME research allowed investigation into issues such as concurrent engineering, machine tool modelling and NC code generation and verification.

The following figure illustrates the feature based work recently undertaken in the group. Highlighted are those areas central to this thesis on which work has been undertaken exclusively by the author. Additionally some areas have been covered by undergraduate and Erasmus exchange students working under the direction of the author.

Areas in which research is being undertaken, but have not been covered in detail in this thesis are tolerancing, fixture planning, and alternate feature views. Fixture planning is covered in Chia's PhD thesis [Chia97a][Chia97b].

The important area of multiple feature views has been undertaken by Little [Litt97]. His work focuses on transforming a feature based description of a component containing both positive and negative features (bosses and protrusions), into a feature based description containing only a blank and negative features.

The tool and cutter selection work of Naish [Nais97][Nais98] generates plan spaces after the manufacturability analysis without knowledge of tools and machines described in this thesis.

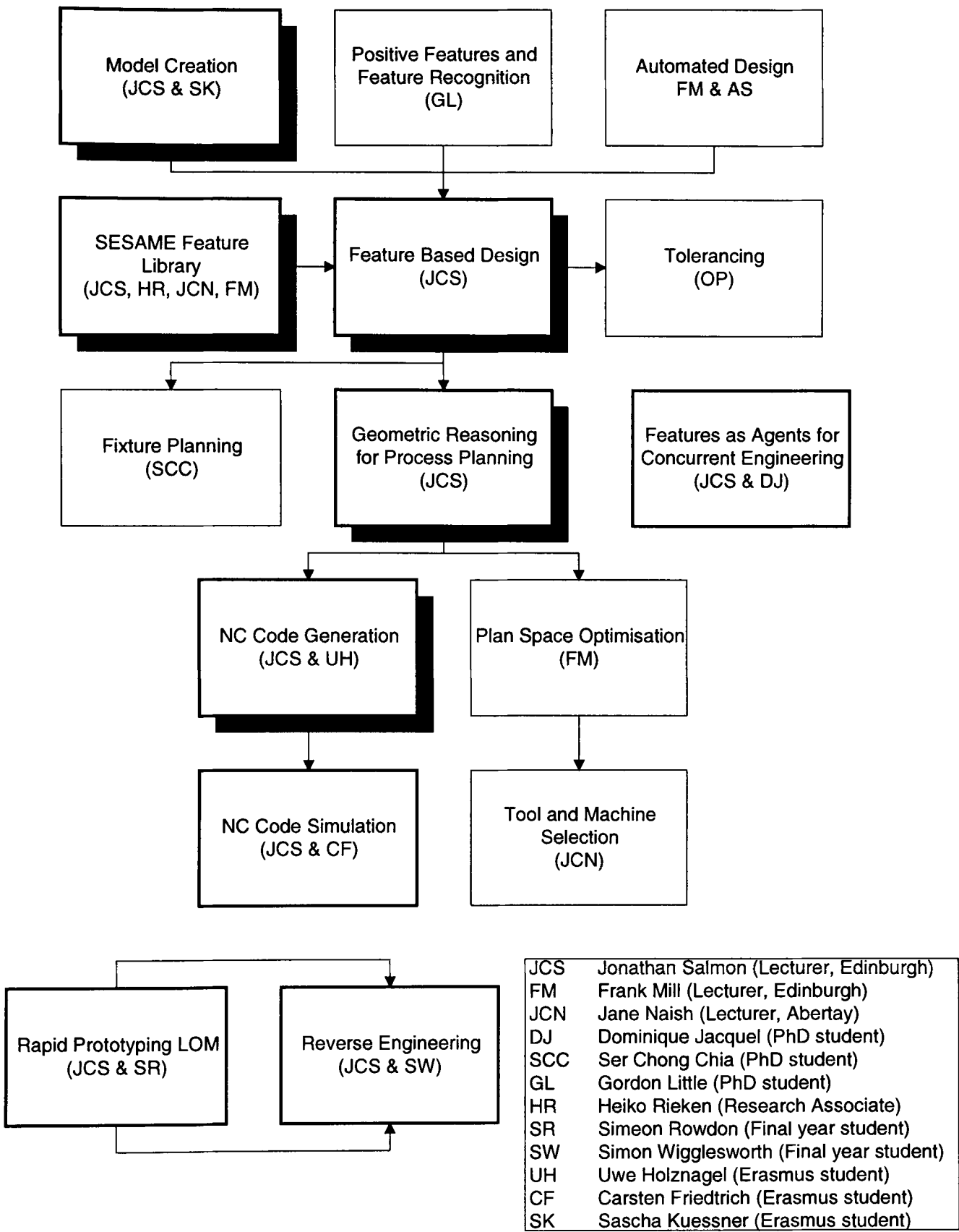


Figure 8 Summary of Feature Based Research in the MPG

The Plan Space Optimisation box in Figure 8 refers particularly to the HAPPI process planner developed in the 1980s. Providing correct feature based designs and manufacturing precedence information for this planner provided the motivation for the work in this thesis.

3.1.1 The HAPPI Process Planner

In 1989, the Manufacturing Planning Group in the Department of Mechanical Engineering at The University of Edinburgh completed a research project on optimisation of process plans entitled *Representation, Reasoning and Decision Making in Process Planning with Complex Components* (SERC/ACME GR/D 63101). This project successfully developed an automatic process planning system (called HAPPI) that would produce a near optimal process plan given a design of a component [Husb88][Husb89][Husb90].

The architecture of the HAPPI Process Planner is shown in Figure 9. From this, it can be seen that at a high level of abstraction the planner would compare the desired component with the blank and produce a search space of all valid process plans. Genetic Algorithms were then used to find a near optimal process plan in this search space.

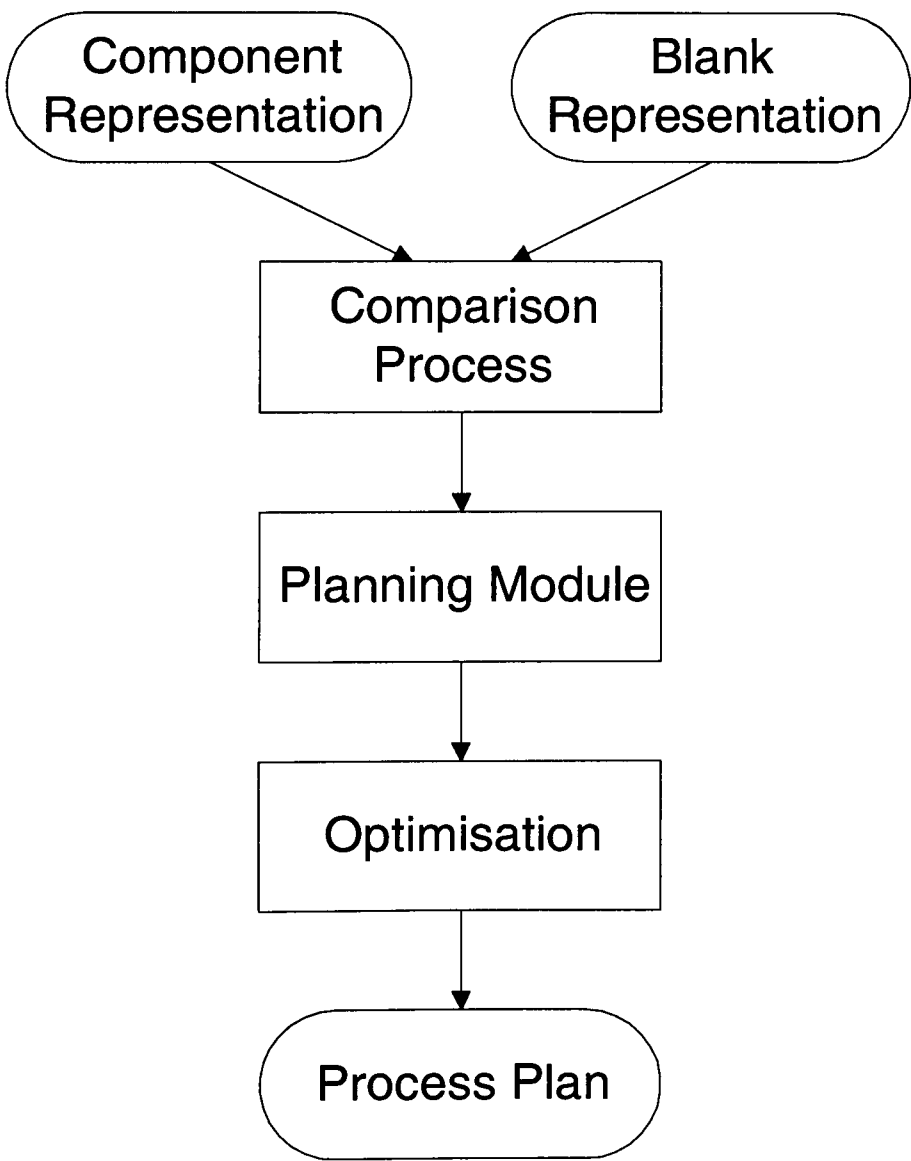


Figure 9 Architecture of HAPPI Process Planner

The component model, however, was a particularly tortuous representation consisting of triples of information written in PROLOG, forming a semantic net (see Figure 13). A specification for even a visually simple component would run to many hundreds of lines of code [Husb88]. Much of the geometric reasoning about the component was effectively performed by the ‘programmer’ who hand-coded the component. A brief extract is shown in Figure 10 describing a portion of the part in Figure 12.

```
p1 isa plane.  
th6 isa thru_hole.  
tap carries_out tapping.  
p4 positioned to p6 withtol 0.125.  
h2 isa blind_hole.  
p5 vexedges p6.  
p7 hasfeat pk1.  
e17 edges p6.  
s1 comprises p10.  
p5 para p8 withtol 0.005.
```

Figure 10 Extract from HAPPI Component Model

For their component representation, Husbands et al., eschewed the linear approach to feature-based design, i.e., a list of features each with a short list of attributes, and instead adopted a semantic net approach allowing much richer relationships to be described. Husbands and Mill in [Husb89] suggest more than 40 relationship types between entities. Entities could include solid features, components, and surface features. An example of a typical representation network can be seen in

Figure 13. The richness of this data structure led in turn to its own problems. In particular, in the HAPPI system, component descriptions were generated by hand and frequently took several days to code. A combination of hand-coding and the semantic net approach meant a model might contain much redundant information. Major redundancy was a property that could be neglected in the prototype, however if the data structure were to be created automatically in a real-time design system, and were to be capable of being saved and restored in that system, issues regarding redundant information would become important, particularly under modification of a component design, where searching for all mentions of an entity that may have been modified becomes complex. As Sabin comments (in [Husb89]), “...*redundancy leads to potential inconsistency*”. This was not a problem in the prototype HAPPI,

but inconsistency is avoided in FODDS2 through a deliberately more limiting data structure. It is important to note that HAPPI did not contain a solid model of the component, though some experiments were performed with the NONAME modeller from Leeds University. To quote Mill:

“Interestingly, the feature representation used developed into a Solid Modeller, (a boundary representation modeller). Not a very good one though.” [Husb89]

The first process undertaken by the HAPPI process planner is a comparison of the Blank Representation with the Component Representation (see Figure 9) in order to discover which features in the component require machining (in other words, do not exist in the required form in the blank). Chiefly, it is apparent that the component representation in HAPPI is a hybrid solid-surface representation and that there is often room for confusion between the surface and solid representation. Neither representation is complete, and decisions on representation have been made pragmatically on an *ad hoc* basis.

The planning module then generates (implicitly) the complete space of possible process plans, which is then searched for a near optimal solution during the optimisation stage. Though now an established technique, the use of genetic algorithms for process planning optimisation was novel.

The HAPPI test component is also used as a FODDS2 test component in later chapters. The entire HAPPI system was implemented in Edinburgh Prolog.

The comparison module (see Figure 11) is of particular interest in this thesis as it is here that manufacturability decisions were originally made, but as there is no geometric modeller in HAPPI this information is now provided by FODDS2.

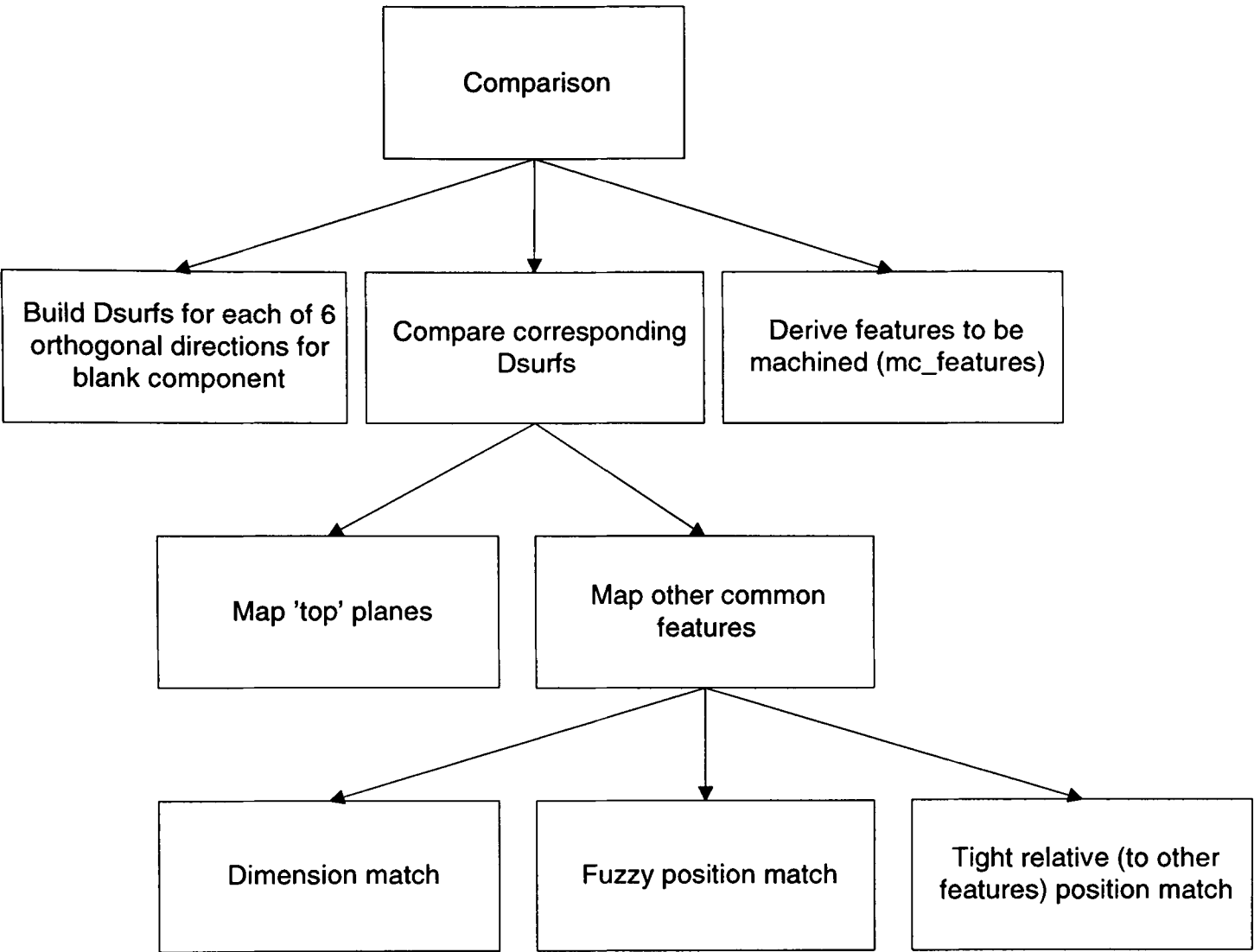


Figure 11 Architecture of the Comparison Module of the HAPPI Planner

The HAPPI system chiefly represented features as being primarily attached to planes of the original blank, or nested within existing features. This is represented through use of the *hasfeat* (or ‘has_as_child_feature’) relationships. Thus, problems can be envisaged regarding steps (open to more than one face), or indeed slots, machined primarily from one direction but often intersecting with three faces of the blank. In the event that two crossed slots have a hole in the base of the intersection, it is unclear to which slot the hole belongs, and so complicates the *hasfeat* relationships. The *hasfeat* relationship codes the feature tree in such a way that the *blank* has 6 orthogonal surfaces (*Dsurfs* (see Figure 11)) and volumetric material removal features are attached through the *hasfeat* relationships to one of the 6 *Dsurfs*. Subsequent material removal features may be attached to others to allow chaining of features such as the nested slots, or hole in a pocket of Figure 12. The *Dsurfs* help define the setup constraints for each feature. As, these also help define a setup for subsequent machining operations. The dependency on the surface planes can be clearly seen. Though the *hasfeat* relationship is one of the most important in HAPPI,

other relationships define geometric and dimensional tolerance relationships. Some of these can be seen in the example of a HAPPI semantic network in Figure 13, which shows the portion of the network relating to the large hole in Figure 12. This diagram also shows the relationship between the component datastructure on the left and the machine and tool database, of which a portion is shown on the right.

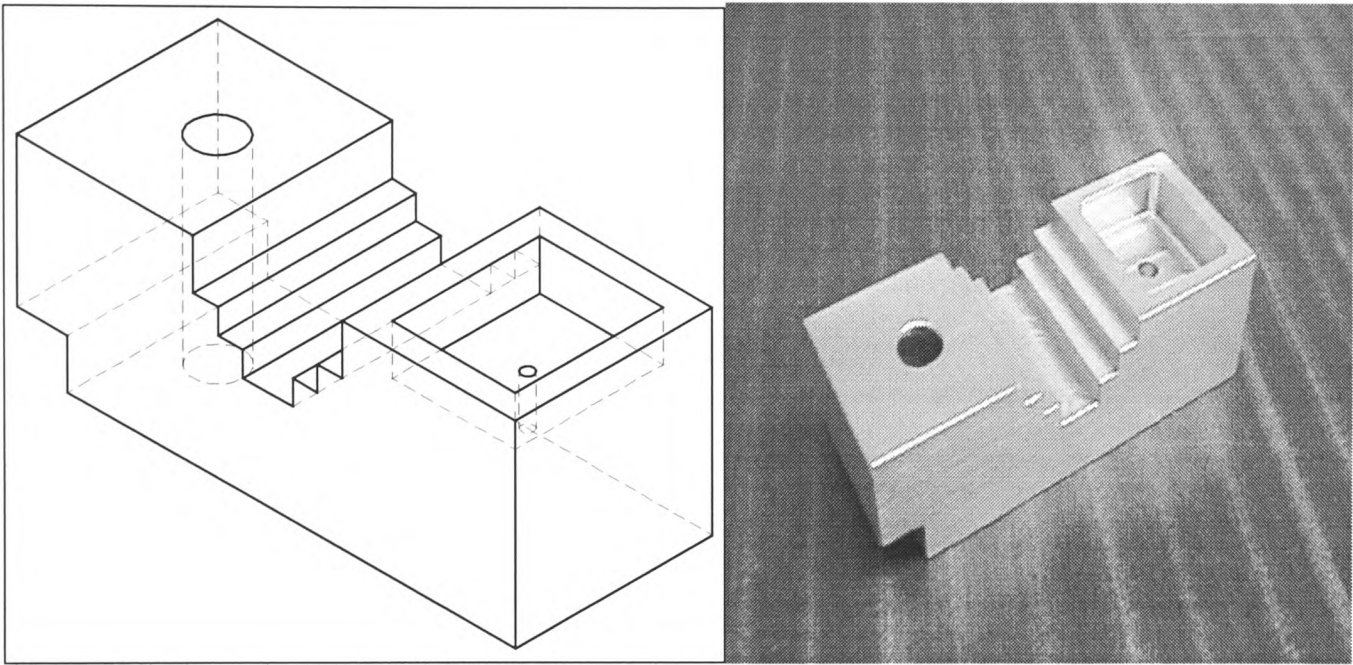


Figure 12 A HAPPI Example Component

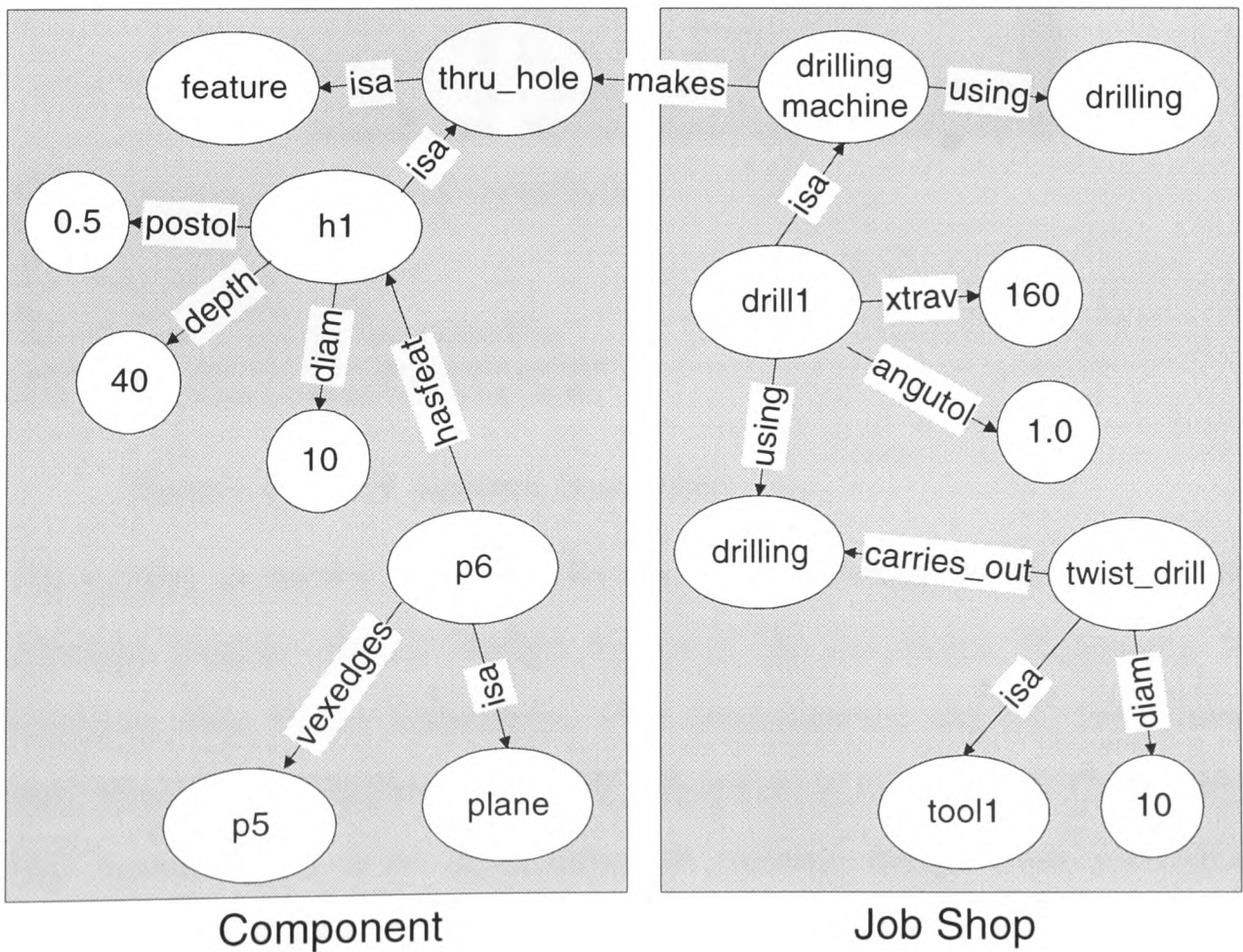
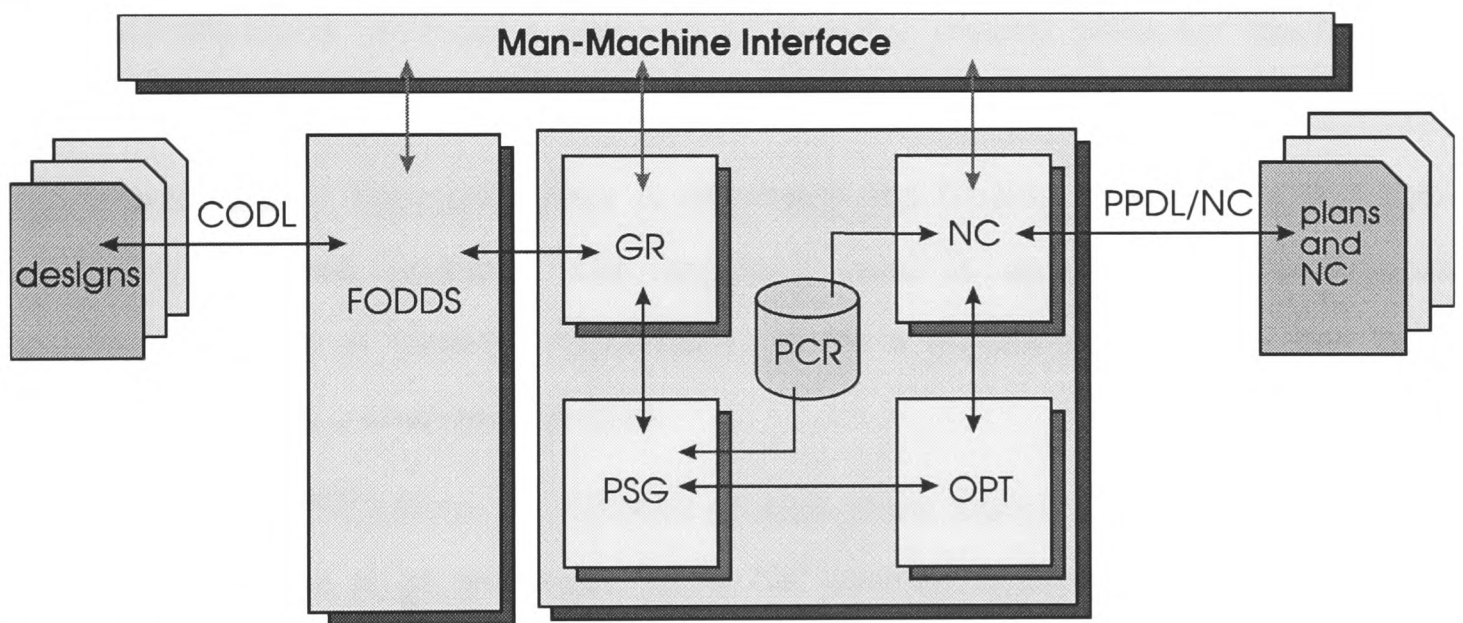


Figure 13 Relationships in an Example HAPPI Database

3.1.2 The SESAME Project

Between 1990 and 1995, the Manufacturing Planning Group was involved in two feature based projects. Firstly, the national project SERC/ACME GR/F92312 Feature Oriented Design (FOD) and secondly the European Brite/EuRam project Simultaneous Engineering System for Applications in Mechanical Engineering. It was during this period that the first version of FODDS was developed by the author, along with the Geometric Reasoner (GR) (see Figure 14) [Mill93][Mill94].

The FODDS system itself is but one component of an entire Design for Manufacture System, the Concurrent Engineering Workstation (CEW). An overview of the entire system is shown in the figure below.



FODDS - Feature Oriented Detail Design System
GR - Geometric Reasoner
PSG - Plan Space Generator
OPT - Plan optimiser
NC - NC Code Generator
PCR - Process Capability Representation
CODL - COmponent Description Language
PPDL - Process Plan Description Language

Figure 14 CEW System Overview

The system as shown above can be considered without feedback as a serial system allowing Feature Based Design followed by Geometric Reasoning for Process Planning, Plan Space Generation, Plan Optimisation and NC Generation. Feeding both Plan Space Generator and NC generator is the Process Capability Database. This system takes as its input either an existing design from a FODDL (Feature Oriented Detail Design Language) File or a new design via the Man-Machine

Interface direct from the designer. The final output is one or more process plans and accompanying NC data. Each intervening module is briefly described below.

Feature Oriented Detail Design System (FODDS) This allows either graphical based design on screen or input from file. Allows additional information to be added to the system by the designer (Explicit Feature Information)

Geometric Reasoner (GR) This module performs geometric reasoning on a Feature Based Design of a Component to generate information necessary for process planning (Implicit Feature Interactions)

Plan Space Generator (PSG) Using the features and relationships produced by FODDS and a variety of data and rule bases including tool and machine databases, the PSG implicitly produces a space of all possible process plans for machining a component.

Optimiser (OPT) This plan space is enormous and finding a good plan is a non-trivial optimisation problem. The approach used is an Artificial Intelligence technique known as Genetic Algorithms, where a population of plans are 'bred' together to evolve a near optimal plan.

NC Generator (NC) From the finished process plans passed from the optimiser, all operations that are to be performed on an NC machine have NC data generated for them.

Output A process plan suitable for sending down to the machine shop. This is a text based process plan with accompanying diagrams and NC data.

A goal for this system is a concurrent system where any system component can be run at any time and contribute to a 'blackboard' style database of information. However, the prototype is a largely serial system with feedback. All these modules however can be run on a single workstation, and the design process can be iterative, giving some concurrence and a common look and feel to all modules improving learning curves and hence designer productivity.

The Feature Based Design System (FODDS in Figure 14) and the Geometric Reasoner (GR) have been combined and subsequently augmented to form the new system FODDS2.

3.2 Computer Integrated Manufacturing (CIM)

The Computer Integrated Manufacturing (CIM) concept emerged in the seventies as a response to a changing marketplace. There was a shift from the large batch sizes and few product lines. In the face of increased competition and more specific demands from clients, the move had to be made to shorter production runs, smaller batches and lower lead times.

Smaller batches meant that an increasing amount of manufacturing information is needed on the shop floor.

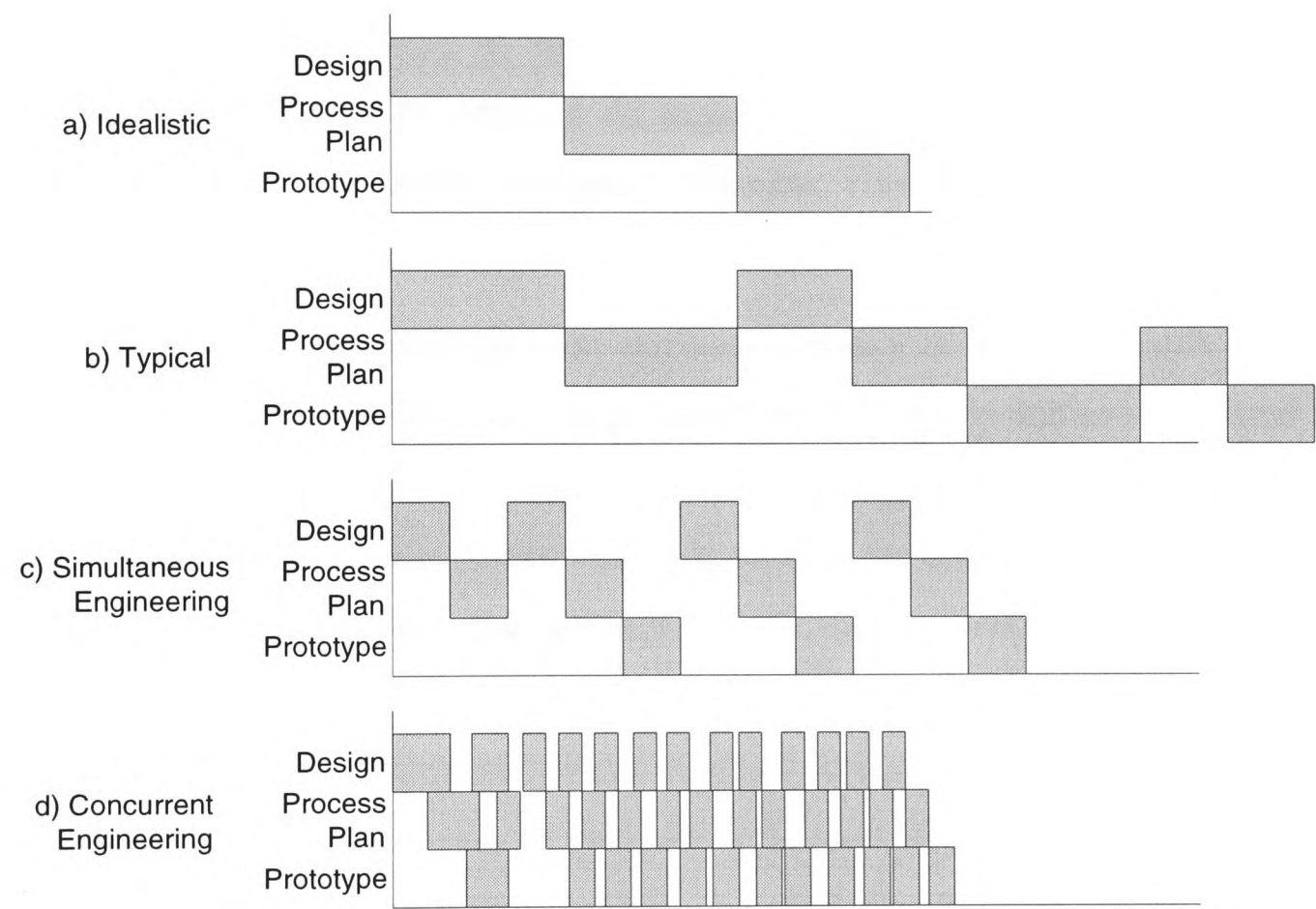


Figure 15 Lead Times vs. Engineering Methodology

Traditionally, engineering has split design, process planning and latterly NC generation. Each is performed by different people, often in different offices, and possibly on different sites.

Designs are drawn up as two-dimensional technical drawings and not until a design is complete is it passed to the process planning team (see Figure 15a). From the drawings a process plan is created and those elements of the process plan that require NC machining then have the NC programs created for them. In each of these distinct stages the opportunity for feedback to the previous stage is weak. If a process-

planning problem is so great that a process plan cannot be created, then great effort must be expended passing the design back up the design chain (Figure 15 b). Redesign means that much of the process plan may have to be rewritten, which again may mean any NC code so far generated has to be discarded.

In the work undertaken in the Manufacturing Planning Group in Edinburgh, *simultaneous* engineering and *concurrent* engineering are differentiated in the following way.

A *simultaneous engineering system* allows a single engineer to work with a set of packages on a single workstation in order to take a product from design to NC code production (Figure 15 c). In simultaneous engineering the feedback is primarily accomplished through the brain of the engineer using a set of programs linked such that the output of one forms the input of the next. This view is akin to the Unix view of programs as filters; the output of any filter can be piped into another filter.

In *concurrent engineering*, the same engineer can take an integrated suite of software tools again at a single seat on a single platform in order to achieve the same result. The difference lies in the level of feedback to previous stages. In concurrent engineering, in addition to a feedback loop through the engineer's brain, the software is sufficiently integrated that automatic tools allow feedback from *downstream* activities all the way back up to design if need be. This model is more complex, and the system design draws on object-oriented and agent-based techniques (using SWARM [Burk97]) and more recently the Microsoft Object Linking and Embedding for Design and Manufacture (OLE for D&M). In concurrent engineering, 'downstream' tools may start working on a design whilst the designer continues to modify that design, thus saving overall design time at the expense of higher CPU time and occasional unused partial solutions (Figure 15 d).

Again in concurrent engineering, multiple concerns are addressed (e.g. function, geometry and manufacturing) as a design evolves rather than waiting until the completion of the design geometry. Cutkosky [Cutk91] takes this approach in the prototype concurrent design system NextCUT (a development of FirstCUT), through the use of coarse-grained agents including a geometry agent, a process planning agent, and a fixturing agent. The open architecture model allows the relatively

painless addition of other agents. First-Cut [Cutk88] on the other hand is a process oriented system enforcing design-for-manufacturability and simultaneously constraining process planning through the ordering of the design process. Though FODDS2 encourages design in terms of manufacturing features, the geometric reasoning allows the process planning optimisation function to be independent of ordering in design and relatively free of unnecessary design constraints. This freedom from ordering at the design stage is an important characteristic of FODDS2 that allows the automatic process planner to make decisions based on cost and genuine ordering constraints rather than arbitrary user based constraints.

Design is a lengthy process of which this thesis only looks at one of the final stages, that of detailed design. In principle, the process planning and NC code generation phases of product development can be regarded as part of the design phase, in that, if either of these stages fails then it will necessarily lead to redesign, and so impacts the finished design.

The designer of a product is trying to achieve a specific function. Sometimes that function will require some aesthetic considerations, particularly if it is a consumer product. Invariably there will be financial constraints on the design, both the time and resources to generate the design and the eventual cost of the product. In this thesis only detailed design, process planning and manufacture are considered, however, along with these considerations, there is a continuous awareness of the following list of design concerns, collectively known as *Design for Whole Life Cycle*.

Primarily:

- Design for Function
- Design for Manufacture

Secondarily:

Design for Assembly; Design for Maintenance; Design for Disassembly;
Design for Recycling; Design for Machining; Design for Test

Any designer must think about all these aspects of design simultaneously. The role of a concurrent engineering system is to aid the designer in as many of these areas as

possible. This enables him to concentrate on those important aspects that the system is unable to deal with such as specific function and aesthetics.

The design by manufacturing features approach taken as the primary form of design in this thesis is not capable by itself of dealing with all the approaches to design listed above. Developments within the Manufacturing Planning Group built upon the ideas developed in FODDS and FODDS2 are being used to allow design in terms of alternate feature sets such as positive features [Litt97] and other aspects of Design for Manufacture such as fixture design [Chia97a].

Computer Integrated Manufacture is a large and complex topic. The Manufacturing Planning Group have addressed many issues in this area over the years, and this thesis focuses on the particular area of geometric reasoning for process planning through manufacturability analysis.

3.3 Solid Modelling

This section introduces 2D CAD, 3D solid modelling techniques of Constructive Solid Geometry, Boundary Representation and Spatial Decomposition techniques as well as mentioning some of the modellers and applications of these techniques. It shows how the emphasis has been on geometry and point sets. It sets the scene for the higher level of abstraction that is features.

Feature-based methods have emerged over the last twenty years in response to industry's requirements for an integrated solution to design and production, in turn required in order to reduce product lead time. Features enable this reduction in lead-time by supplying the designer with a set of tools at a higher level of abstraction than those of a typical computer-aided design system.

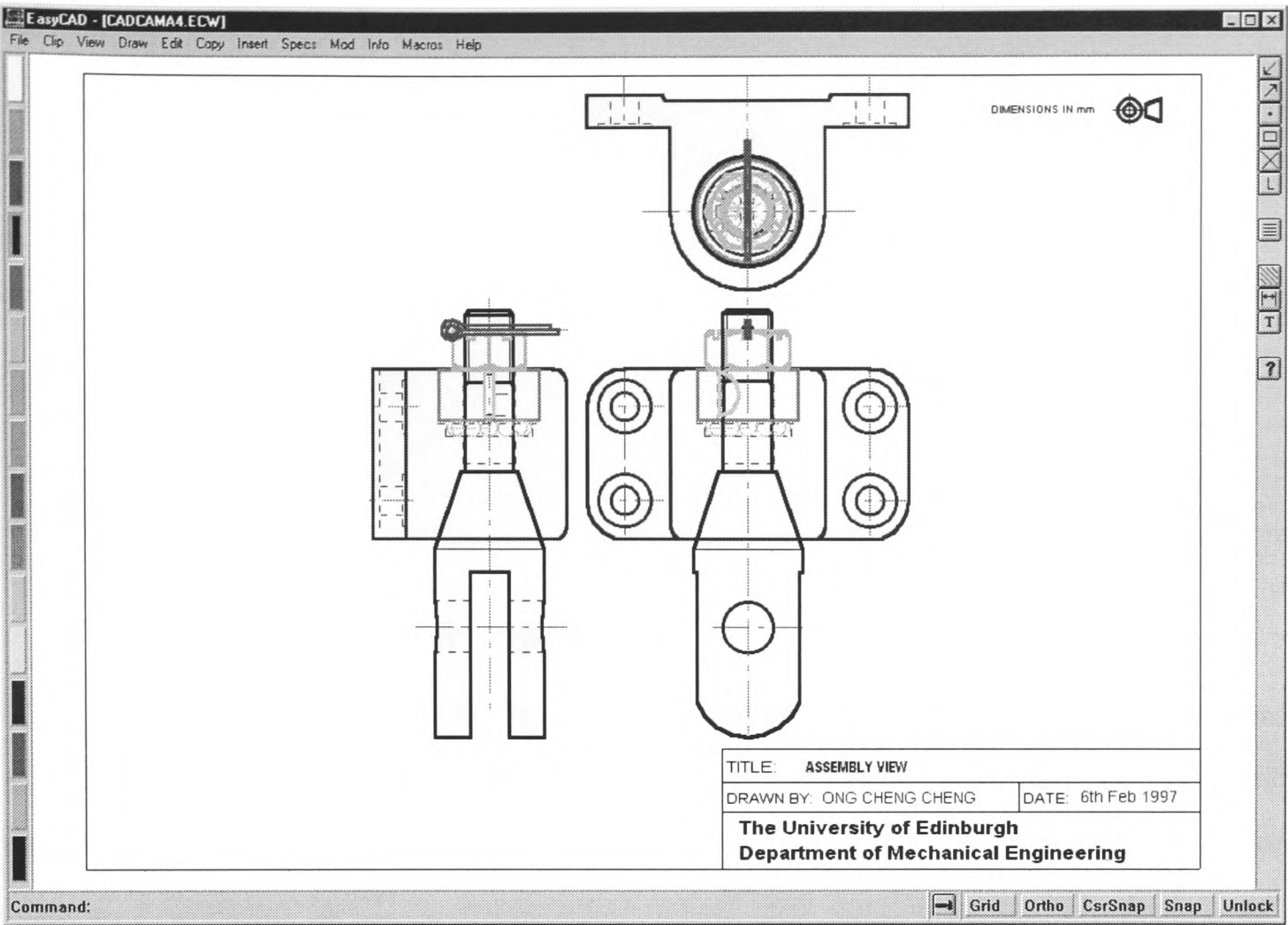


Figure 16 A 2D Drafting Package (EasyCAD)

A typical 2D system (EasyCAD from Evolution Computing (see Figure 16) is such a system) is only able to design in terms of lines, arcs and circles, admittedly annotated with dimensions where appropriate. There is no concept of part within the system. There is no knowledge of the three-dimensional geometry of the system, and indeed there is little knowledge of the two-dimensional geometry of the system particularly where on the drawing represents solid material and where represents empty space. The closest the system comes to this is the ability to ‘area fill’ parts of the drawing at will. This can clearly be shown by the ability of these drafting packages to produce drawings of impossible objects such as those in Figure 17 (prepared in EasyCAD).

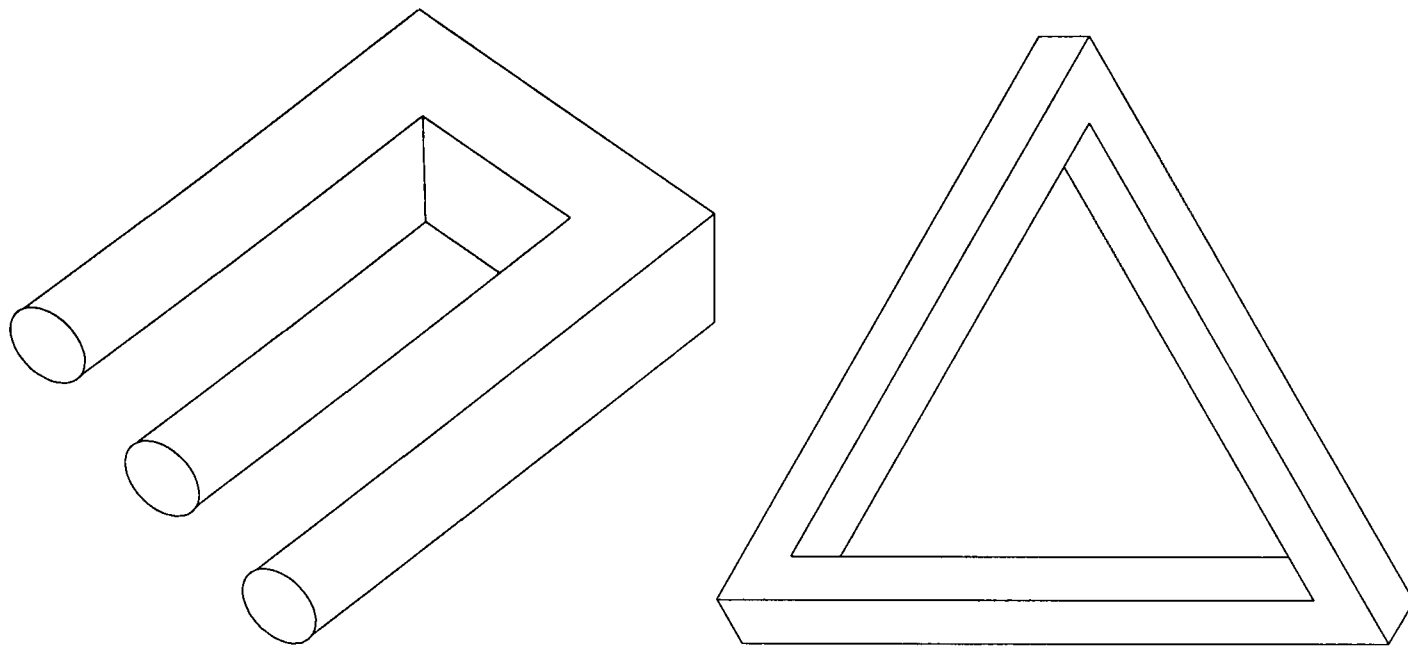


Figure 17 Impossible Objects

The trend to 3D systems, with the advent of increasingly powerful computers and software of ever growing sophistication, is fraught with similar limitations. The first three-dimensional systems were so-called wire frame modellers. In much the same way as a drawing in 2D is constructed out of line segments, in 3D, wire frame modellers allowed models to be constructed of 3D line segments. Though 3D models could now be produced, these models are inherently ambiguous. It is not possible to categorically infer from a wire frame model where particular surfaces lie or indeed whether they exist. Perhaps the simplest example of this is that a wire frame modeller has the same representation for both a solid cube and an empty box.

To resolve this problem, ways of representing solids were required. Solid modellers can be characterised as representing a point set in Euclidean three-space. Typically further restrictions are placed on the point set to allow the set to be represented to some degree of satisfaction within a finite computer system and in such a way as to allow useful manipulation and interrogation of this point set. [Bowdy95] gives background information on areas of solid modelling still requiring research. The currently successful solid modelling methods tend to fall into three major categories:

- Spatial Occupancy methods
- Constructive Solid Geometry
- Boundary Representation

The work of the Djinn project [Arms97] is attempting to define an Application Programmer's Interface (API) that hides the underlying modeller and also the type of modeller from the applications developer. [Shah97] is also looking at a modeller independent API but is currently restricted to two B-Rep modellers.

3.3.1 Spatial Occupancy Methods

Spatial Occupancy Methods are often less able to model exact geometry, but can be more compact than other methods. This allows octrees in particular to be used in cases where space is at a premium, such as in fine-grained agent models [Jacq98].

Spatial Occupancy methods come in 4 main forms

- Exhaustive Enumeration
- Cellular Decomposition
- Space Subdivision
- Depth Maps

3.3.1.1 Exhaustive Enumeration

Exhaustive enumeration is little used but represents a volume as a number of volume cells or *voxels*. Exhaustive enumeration is extremely expensive in memory terms, so space subdivision methods such as octree methods are more frequently used as they can result in a huge space saving at a small increase in algorithmic complexity.

Consider the TECC component (see Figure 70, later in the thesis and [Husb91]), roughly 200mm long by 100mm wide by 200mm high. Modelled at an accuracy of only 0.1mm requires $2000 \times 1000 \times 2000$ voxels or $\sim 4 \times 10^9$ voxels. At one bit for each voxel and at eight (8) bits to the byte, approximately 500Mbs of memory are required. Exhaustive enumeration is not compact.

3.3.1.2 Cellular Decomposition

Cellular decomposition where a body is composed of a number of non-overlapping simple, but not necessarily regular, cells joined at common faces. Finite element meshes used for stress analysis and computational fluid dynamics are perhaps the

most common uses of this technique. The other techniques (CSG and B-rep) are often enhanced with cellular decomposition methods, for instance, to provide fast raytracing.

3.3.2 Octree modelling

Octrees are used for many tasks, often in association with CSG or B-rep modellers. Representation methods are described by Yamaguchi [Yama84]. An example object along with an octant numbering system and the resultant octree are shown in Figure 18 and Figure 19.

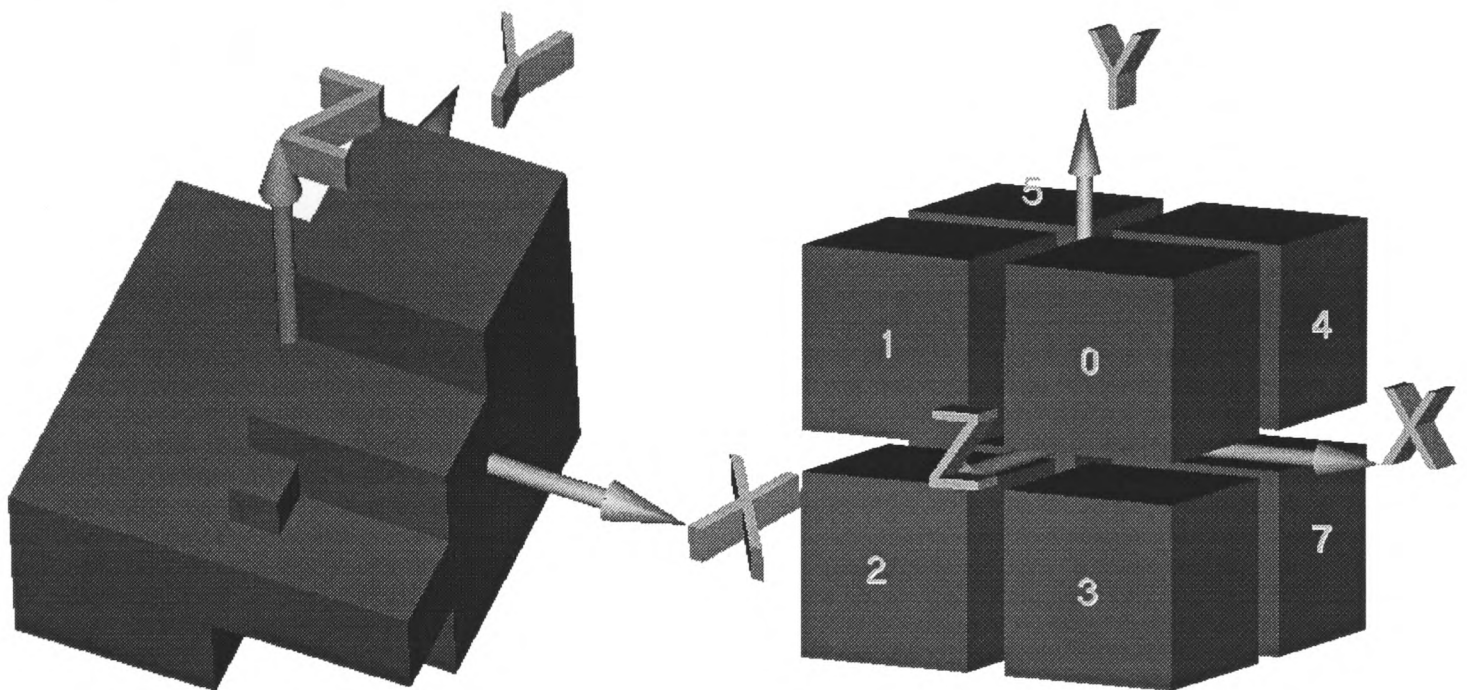


Figure 18 An Example Object and Octant Numbering

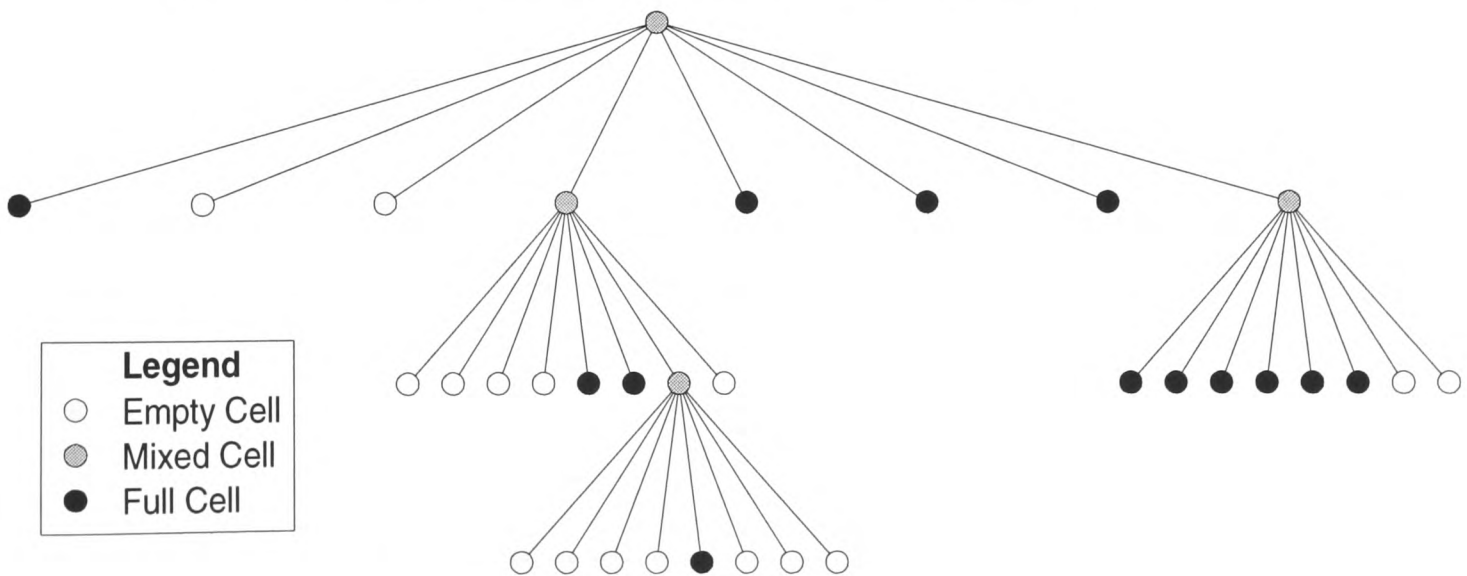


Figure 19 Octree Representation of Example Object

In this octant numbering system, octant 0 is the $x,y,z>0$ octant and the octants are numbered anticlockwise around the z axis, followed by those in the $z<0$ halfspace.

The octree representation then given is equivalent to the object shown. The initial, top level octants are centred on the origin.

Taking the TECC component again (200mm×200mm×100mm), modelled to an accuracy of 1 part in 2000 (or 0.1mm), would require an octree up to 11 levels deep ($2^{12}=2048$). The size in terms of the number of nodes of the resultant octree would result in great space savings with all the areas that were either entirely within the object or entirely without the object. Each node in an octree can have one of three values: *full*, *empty*, or *grey* (i.e. mixed and requiring further decomposition).

One particular application of octrees is for pre-segmenting a B-rep model to allow rapid ray casting of the model by photorealistic renderers. An octree is made of the B-rep model to some suitable resolution and the nodes of the octree are tagged with those faces of the model that can be found in the appropriate region. This allows rapid ray-firing as precise intersections need only be found for those nodes the ray passes through containing interesting surfaces [Glas84].

3.3.3 Depth maps

A common, but limited Spatial Occupancy method that has the advantage of being comparatively compact is the depth-map. Depth maps can only represent components that are single-sided, in machining terms this means that they must be machined only from a single direction. Other single sided objects include terrain maps, making depth maps suitable for Geographic Information Systems (GISs). The output from 3D digitisers, at least in an intermediate form, tends to be as a set of depth maps each from a different viewpoint. A number of 3D depth maps can be knitted together to form a solid 3D object (though not without difficulty and with some limitations). NC simulation programs dealing with purely one-sided 3-axis NC milling can also use a depth map as a suitable representation provided each *pixel* is small enough to provide satisfactory resolution. Representing a component of dimensions described earlier, but using a single sided depth map takes only 2000×1000 pixels. If each depth is modelled using an 8bit fixed point number, giving 256 possible depths, then 2Mb of storage is required, large, but a fraction of the space previously needed. The NC simulation program need only handle two dimensional geometrical problems, setting the depth of all pixels a cutter passes over to the cutter height. Simple rendering of

depth maps is comparatively easy, as pixels in the depth map represent pre-sorted areas in the resultant image.

The following image is a golf club head and its digital reconstruction in ACIS (a solid modeller) from a digitised depth map taken from the laser digitiser of Machine Vision Group of the Artificial Intelligence Department at The University of Edinburgh. Each pixel in the depth map was about 2mm square, and due to specular reflections, some of the data is clearly spurious. The ACIS model reconstruction software constructs a series of slices from the data and unites them. No attempt has been made to smooth the resulting body in any way. The resulting body was then rendered using the LightWorks extension to ACIS to produce the image seen. The original head is courtesy of Ben Sayers Ltd, North Berwick.

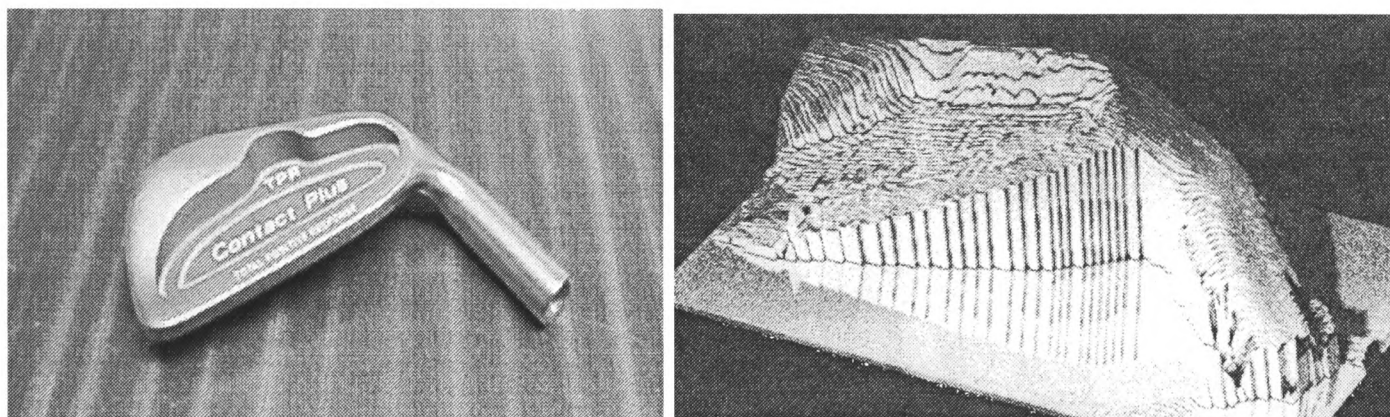


Figure 20 Original and Reconstructed Golf Club Head

3.3.4 Constructive Solid Geometry

Constructive Solid Geometry models (CSG) also known as set-theoretic models are defined as combinations of primitive sets by Boolean operators (chiefly union, subtraction and intersection). The most primitive sets are so-called *half spaces*; these are defined by simple functions that separate the world (or three-dimensional Euclidean space E^3 to be more precise) into *in* and *out*. Half spaces can be defined in terms of any real-valued analytic function $f(P)$, $P=(x, y, z)$ (Certain non-analytic functions cause problems according to Requicha [Requ80]).

Thus for a plane passing through a point $P_0=(x,y,z)$, and with an outward surface normal $V=(x,y,z)$, it is possible to determine whether any point P is inside or outside the halfspace by evaluating the expression

$$(P-P_0) \times V$$

and noting that if the result is less than zero then the point P_0 is inside the half space, if it equals zero it is on the planar surface, and if it is more than zero it is outside the halfspace.

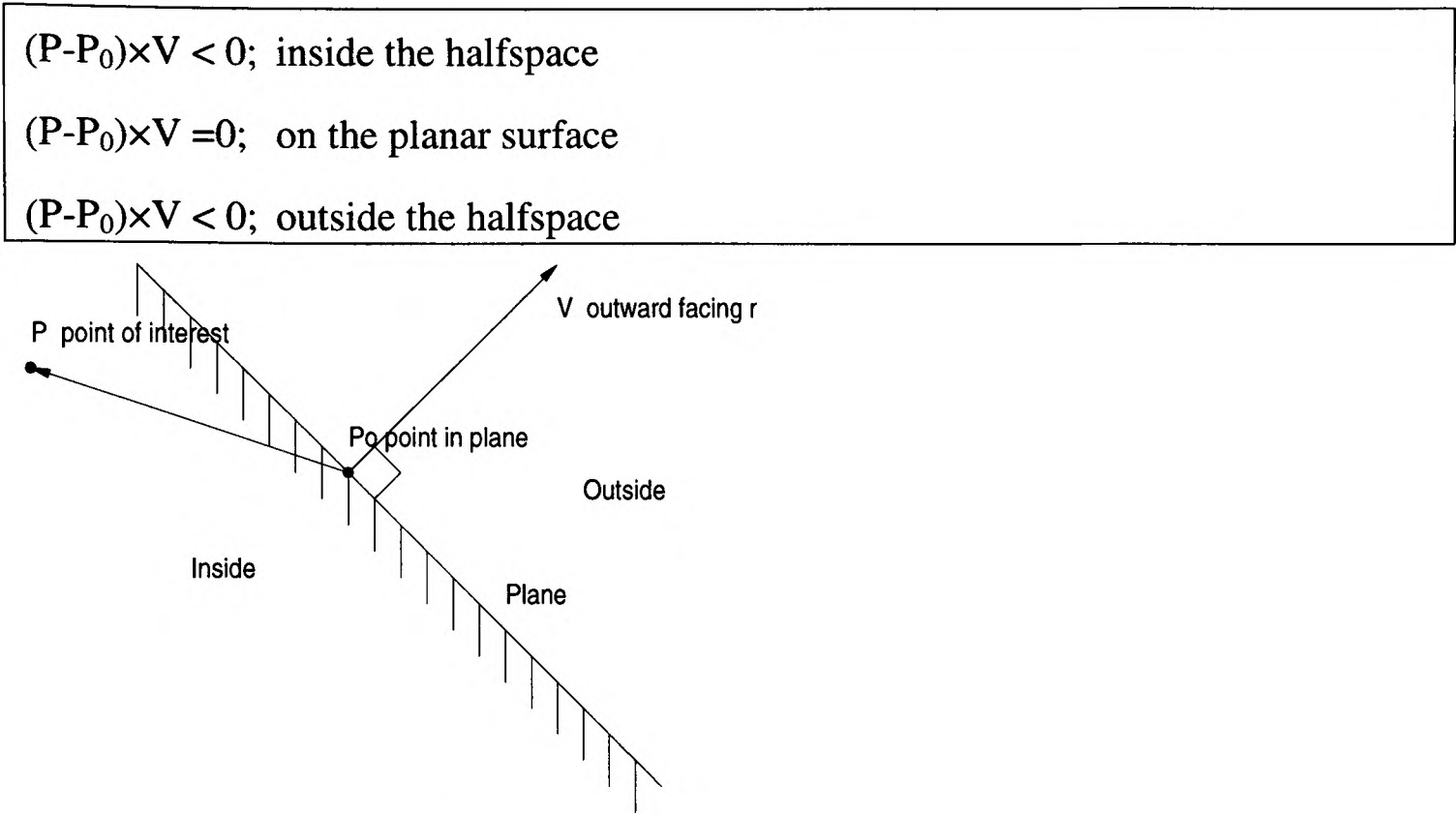


Figure 21 Diagram of a Planar Halfspace

Similarly, an open-ended (infinitely long) cylinder whose axis passes through a point P_0 and is aligned with a vector $V=(x,y,z)$ and with radius r , can be defined with the following equation.

$(P-P_0) \bullet V - r < 0$; inside the cylindrical surface

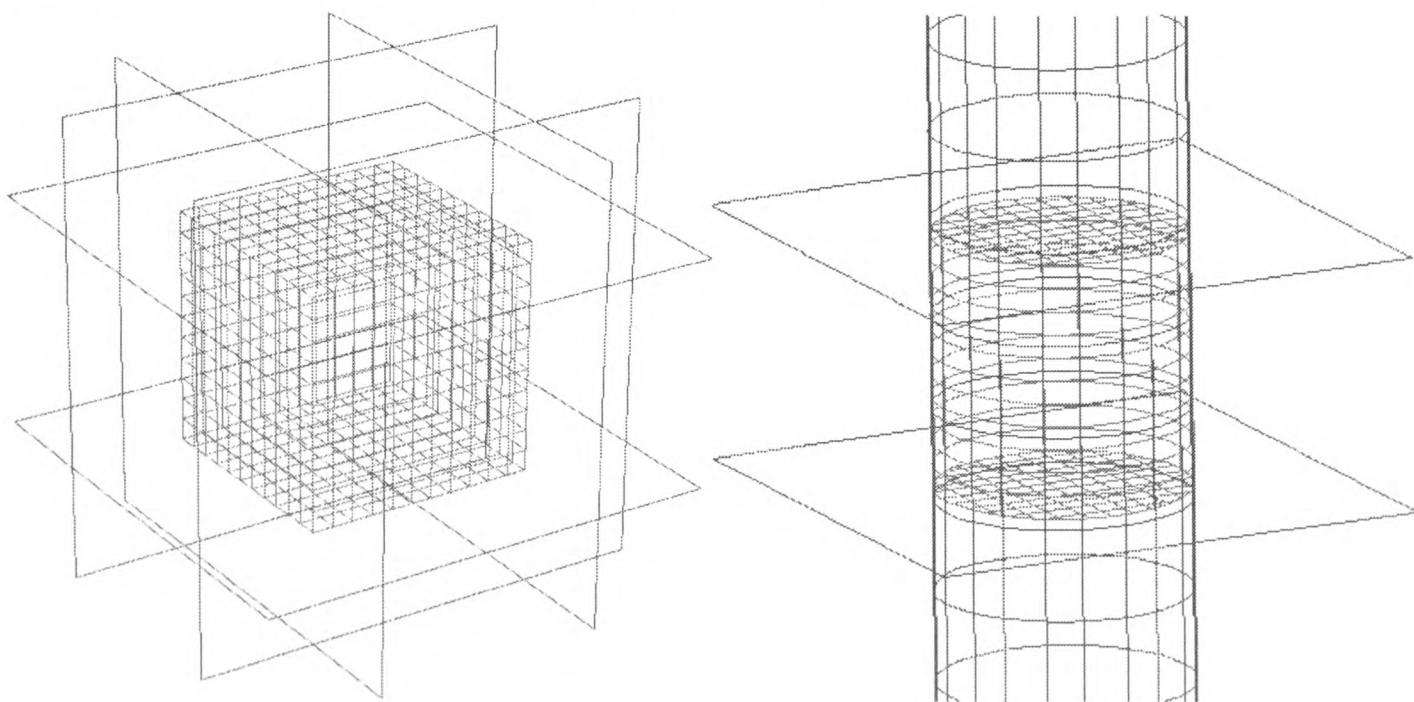


Figure 22 A Box and a Cylinder in terms of their Constituent Halfspaces

A number of half spaces can be combined using set operators such as intersection (\cap), union (\cup) or subtraction ($-$) operators to produce more complex bodies (these operators are also referred to as Booleans). For instance a cylinder, C_1 , of height h , and radius r , centred on the origin (as in Figure 22) can be defined in terms of three half spaces in the following way:

$$H_1: \quad (P-(0,0,h/2)) \times (0,0,1) < 0$$
$$H_2: \quad (P-(0,0,-h/2)) \times (0,0,-1) < 0$$
$$H_3: \quad (P-(0,0,0)) \bullet (0,0,1) < 0$$
$$C_1 = H_1 \cap H_2 \cap H_3$$

Though CSG modellers may allow use of the half spaces, most modellers wrap these up into suitable, more user-friendly primitives, such as blocks, cylinders, spheres, cones and tori. Though a user may be aware that he is developing a CSG tree to describe his component, he may remain unaware of the primitive half spaces at the bottom of the tree and so a simple component may be effectively modelled as in Figure 23.

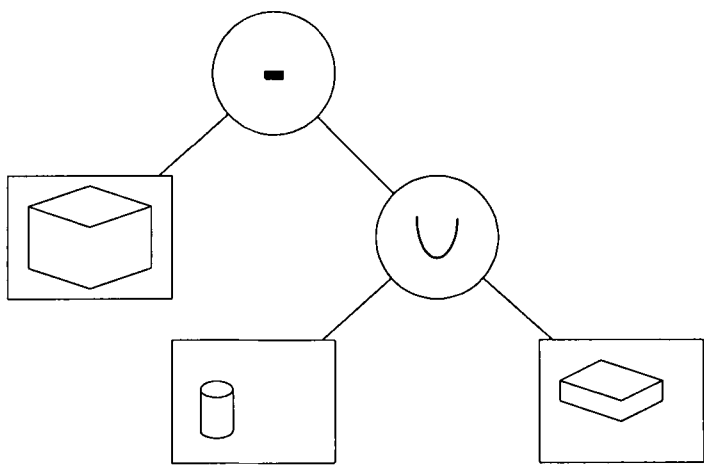


Figure 23 A Simple CSG Tree and Model

3.3.5 Boundary Representation

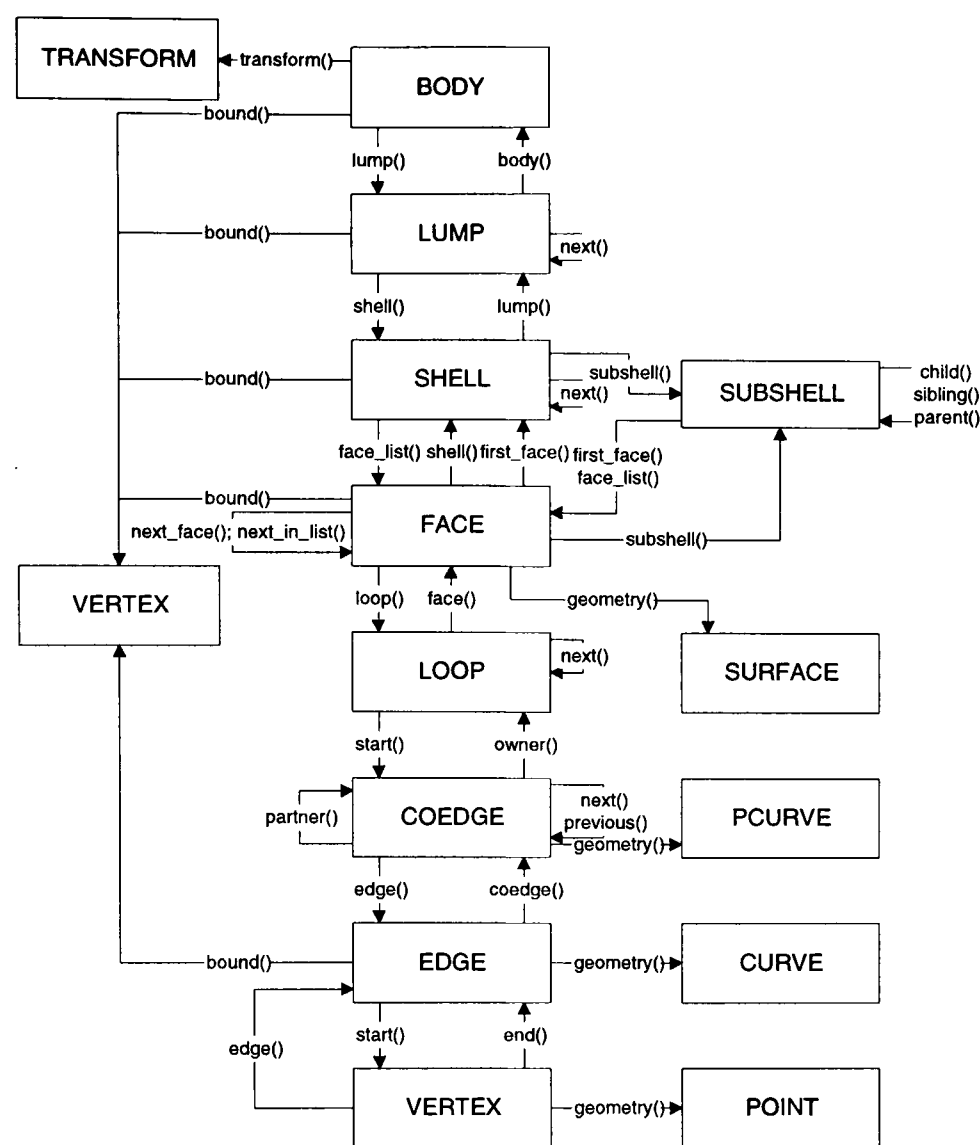


Figure 24 A B-Rep Hierarchy (ACIS)

Boundary representation modellers seem to be in the majority in industry. They are perceived as having a number of minor advantages over their CSG compatriots, namely:

A B-rep model is canonical; there is a unique representation of any solid object, whereas the same solid can (in general) be modelled in an infinity of ways in a CSG modeller. This makes similarity checking very expensive (if not impossible) in a CSG modeller, compared with ‘just expensive’ in a B-rep modeller.

B-rep modellers contain an explicit face and edge list, making visualisation routines cheaper for B-rep modellers than for CSG. The hierarchy of topological elements in ACIS, a commercial B-rep modeller is shown in Figure 24. The hierarchy for Parasolid, the other leading commercial solid modeller is very similar to that of ACIS [Shah97].

These differences are in fact less than might at first appear, as CSG modellers tend to maintain a B-rep model of the object for visualisation, and B-rep modellers tend to allow generation of solids using CSG-like primitives and Boolean operators. Additionally, it is easier in B-reps to associate additional information (attributes) with individual faces or edges as these entities do not exist within the CSG data structure. However, the difference is more in the primary representation and the philosophy of the research/developer.

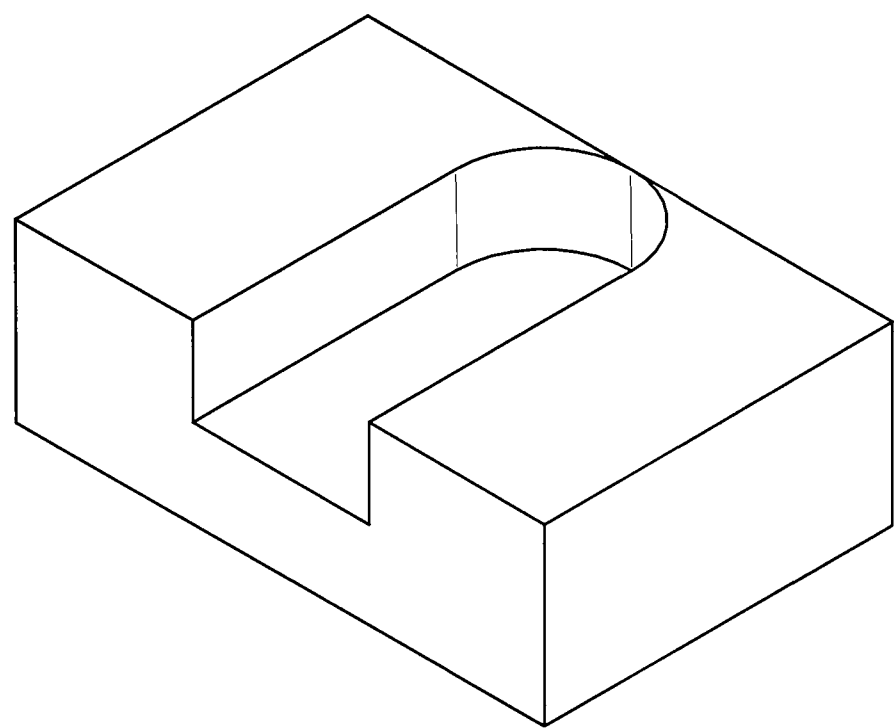


Figure 25 A Non-Manifold body

The body in Figure 25 was generated in a demonstration of an early version of FODDS (ACIS based) and crashed the system. It is an example of a non-manifold body. Non-manifold bodies are objects where, for instance, more than two surfaces meet at a single edge. These have been a problem for B-rep modellers for a number of years, as it is quite easy to try to create them. It has been difficult until recent years to represent them and in particular to perform subsequent operations on the resulting solid model. ACIS 1.3 would allow the above model to be generated, but subsequent Boolean modelling operations on any portion of the model would cause the modeller to crash. A physical interpretation of non-manifold bodies is difficult. It is not clear from the model whether the portion of space along the rear edge is a very thin wall, or a very thin crack.

3.4 Geometric Algorithms

For the FODDS2 system to work effectively, certain geometric algorithms not readily available in the current generation of commercial solid modellers are required.

The Minkowski sum is a form of sweep operator suitable for describing the swept volume that a tool passes through given a model of that tool and the toolpath.

The medial axis recovers the ‘skeleton’ of a body from the body itself and can be used to form part of a thin wall detection algorithm.

3.4.1 Minkowski Sum

[Kaul92] provides a comprehensive survey of Minkowski sums. De Berg [Berg97] and Middleditch [Midd88] also describe suitable algorithms.

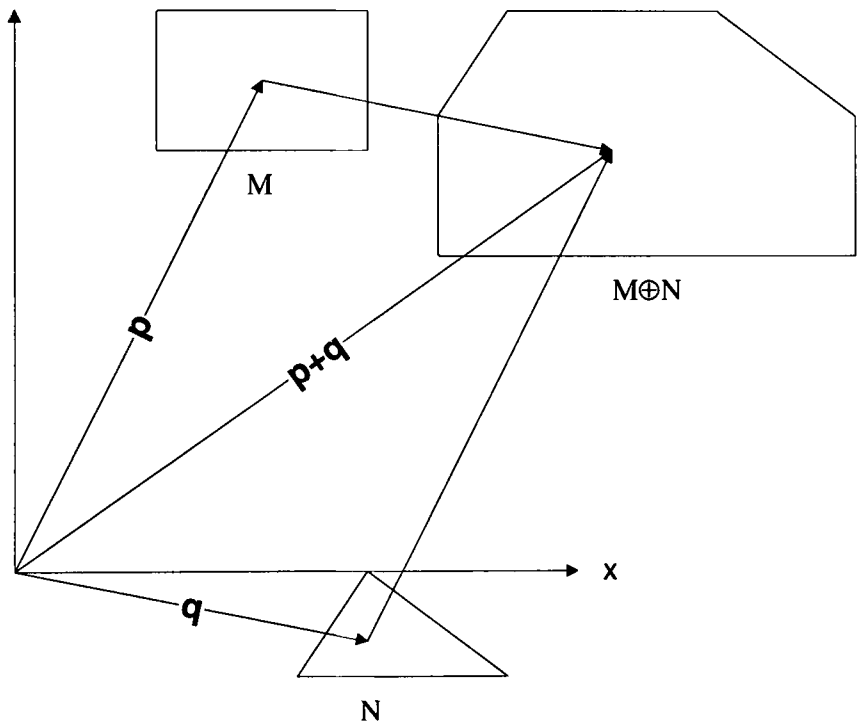


Figure 26 Minkowski Sum of Two Polygons M and N

Intuitively, the Minkowski sum of two closed regions can be considered as a dilation process where one region is expanded by the other. In more formal terms, the Minkowski sum of two sets M and $N \subset \mathcal{R}^d$, denoted $M \oplus N$, is defined as the set

$$M \oplus N = \bigcup_{\substack{p \in M \\ q \in N}} \{p + q\}$$

Defining $X_h = \{x : x - h \in X\}$, allows the definition of Minkowski sums to be rewritten as $M \oplus N = \cup_{p \in m} N_p$. This equality is obtained by changing the order of the union in the Minkowski sum by keeping p fixed and having point q run all over N . An example of the Minkowski sum of two polygons is shown in Figure 26.

An operation analogous to Minkowski summation can be termed the Minkowski difference (\ominus). It can be intuitively looked upon as the erosion of one set by another. More formally, it can be defined as:

$$M \ominus N = \bigcup_{\substack{p \in M \\ q \in N}} \{p - q\}$$

or

$$M \ominus N = \overline{\overline{M} \oplus N}$$

where \overline{M} corresponds to the complement of M .

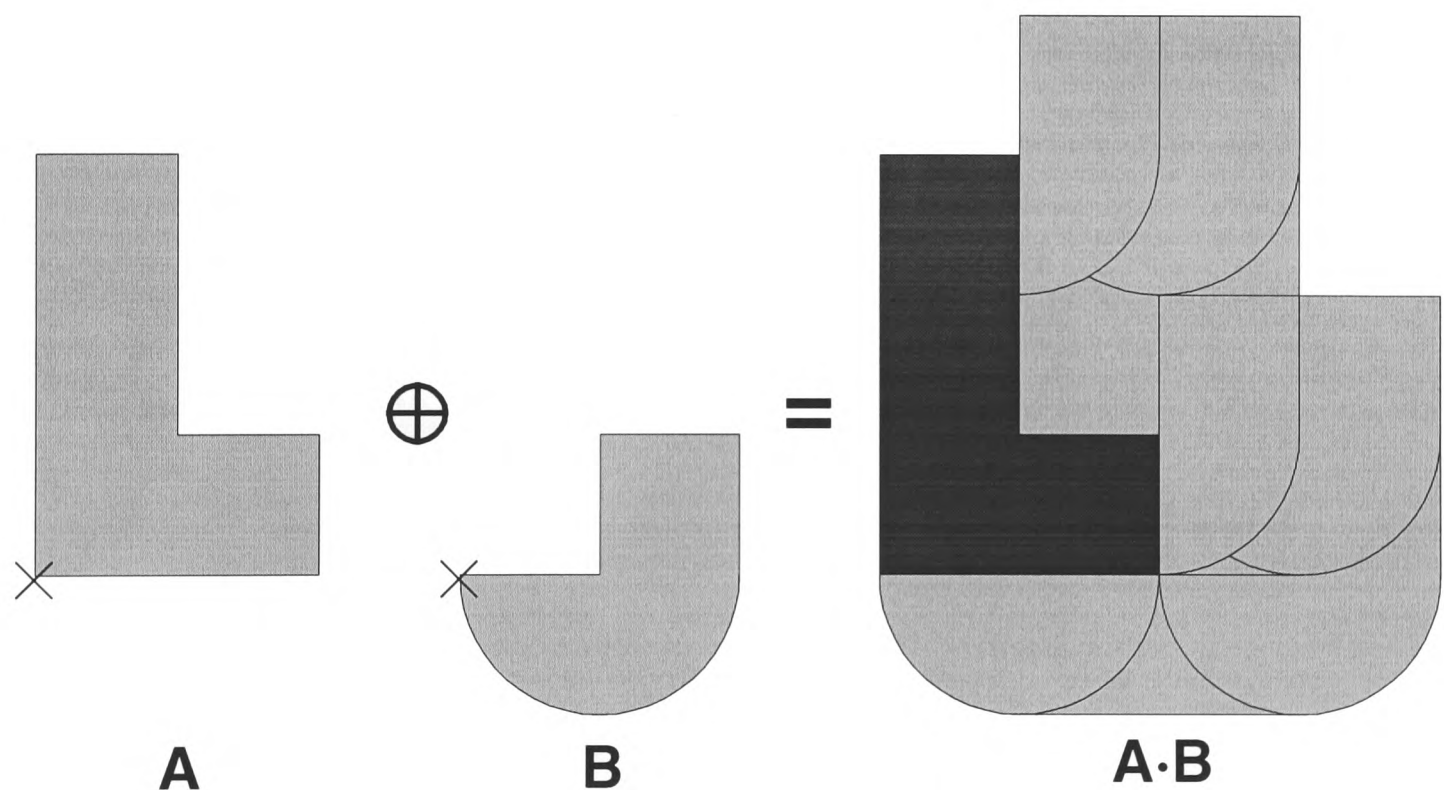


Figure 27 Minkowski Sum Combines the Shape Characteristics of its Arguments.

Minkowski sums have found various applications in the field of CAD/CAM. Some of the better known applications are robot path planning, creation of machining volumes, rounding and filleting and shape design.

3.4.1.1 Robot path planning

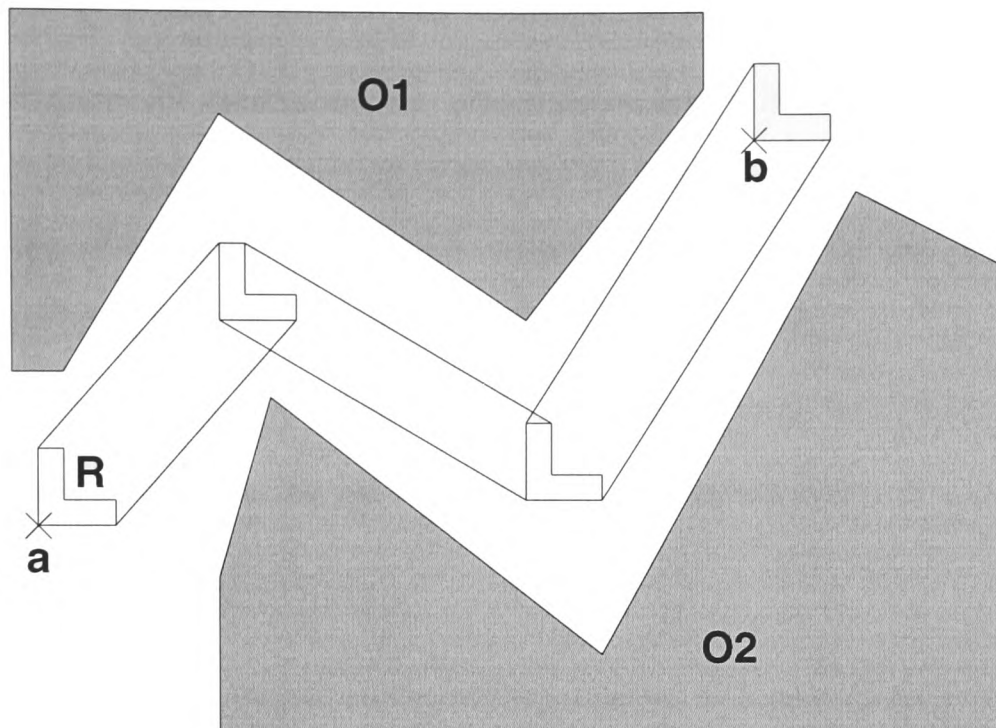


Figure 28 Navigating an Object (R) amongst Obstacles (O1, O2) using Interference Detection

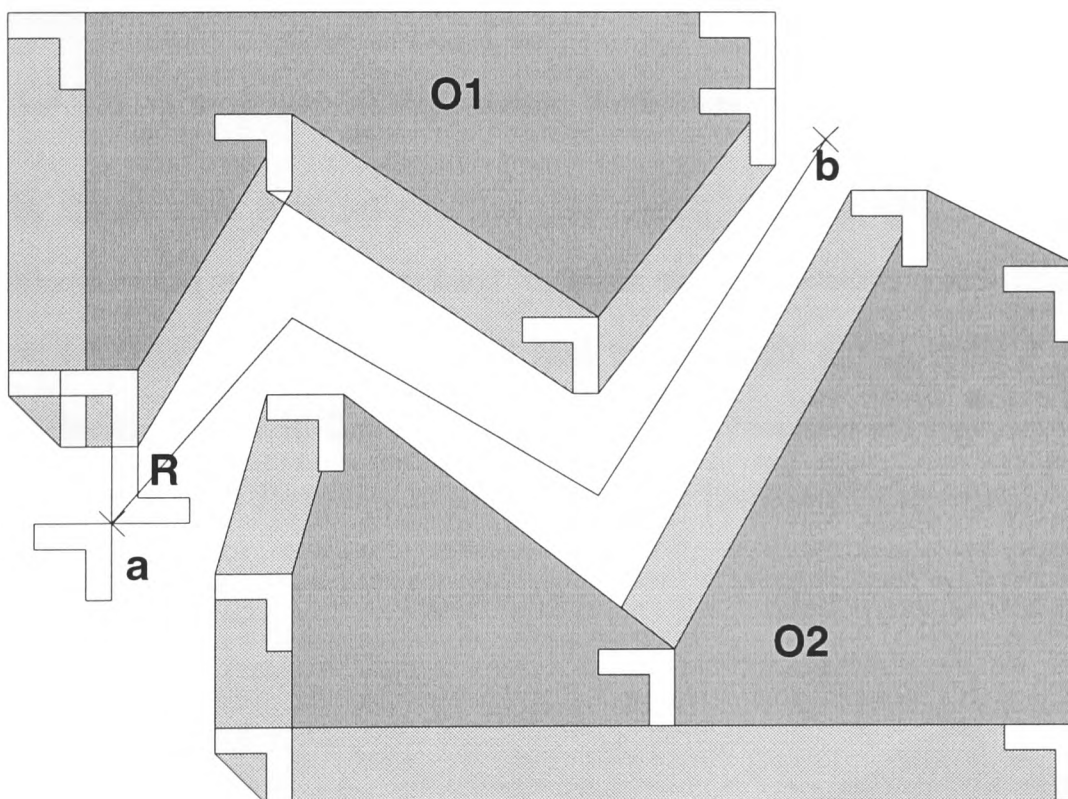


Figure 29 Navigating an Object (R) amongst Obstacles (O1,O2) using Minkowski Sums

Minkowski Sums have been used extensively in robot path planning [Berg97]. The problem should first be restricted to translations of the robot object among obstacles. A naïve approach would be to consider alternate paths by sweeping the robot object along a candidate path and then checking for interference between each swept path and the object (see Figure 28). The bottleneck in this approach is the complexity

of repeatedly detecting interference of the swept paths with the obstacles. Generating swept paths is not a trivial operation either.

Figure 29 describes an alternate scheme where Minkowski sums are utilised to compute *forbidden zones* around each obstacle. The forbidden zone corresponding to the obstacle is that region where a translation of the robot object centred about its reference point would collide with the obstacle.

If the reflection of the robot object R , about its reference point is called $-R$. The forbidden zone S_i , for an obstacle O_i , is given by:

$$S_i = O_i \oplus (-R)$$

Using this method the problem can be reduced to that of navigating a point among forbidden regions which is much simpler and can be solved using ray casting techniques. This method must be extended to cope with non-cylindrical robots that change orientation during a move.

3.4.1.2 Creation of Machining Volumes

In three axis machining, the machining volume can usefully be modelled as the Minkowski sum of the tool volume and the cutter path [Sung86]. In Figure 30 a wireframe model of a blank and cutter path along with a wireframe profile of a cylindrical tool in the home position. The Minkowski sum of the tool and tool path is then removed from the blank to give the finished component.

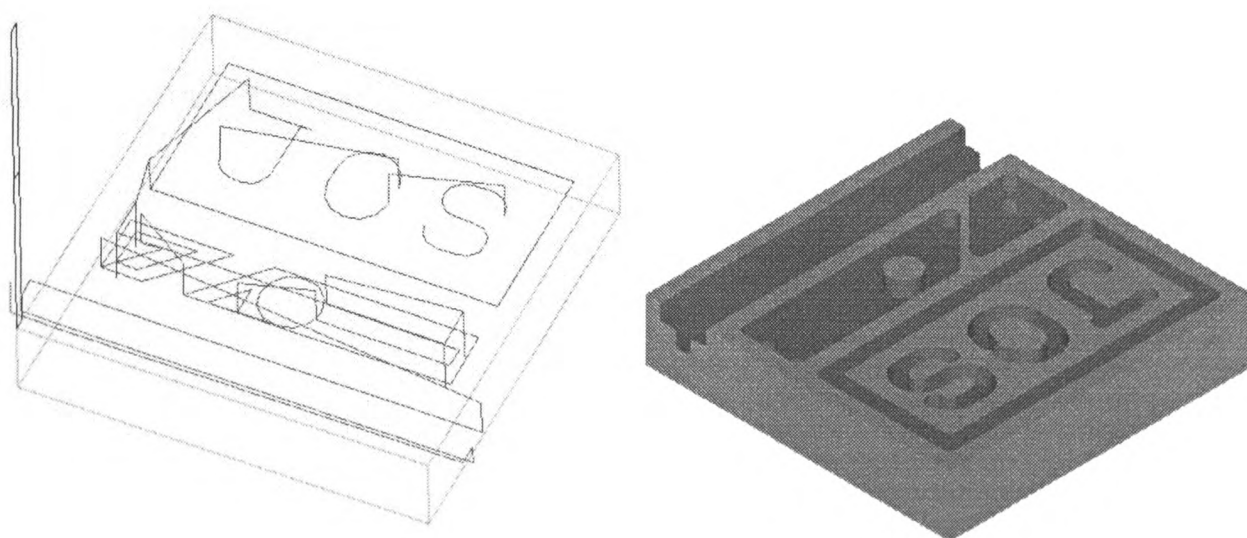


Figure 30 Machining Volume Generation

3.4.1.3 Rounding and Filleting

Rounding and filleting are commonly used operations in geometric modelling. The construction of geometry in most modellers allows users to build their objects using primitives that generally have sharp corners although these are rarely desired in the final shape.

Consider a shape G generated by a solid modeller. A global rounding of radius r can be generated by specifying a sphere S with the same radius and computing $\mathbb{R} = (((G \oplus S) \ominus 2 * S) \oplus S)$. That is, first by dilating the object all over by a given thickness, then by eroding by twice that thickness and then by dilating by the thickness again. The net effect is to round both convex and concave edges [Midd88].

Filleting alone can be achieved by $(G \oplus S) \ominus S$.

Whereas external rounding alone is achieved by $(G \ominus S) \oplus S$.

This technique of Minkowski blending does not always result in the intuitive shape, particularly for shapes with thin passes or causeways (see Figure 31). The result depends on whether dilation or erosion is performed first.

Minkowski blends, though well-defined, are of limited use in the real world. In particular it is difficult to know how to limit the extent over which they operate, and they can produce some counter-intuitive solutions in confined spaces or over small peninsula. The result of the global rounding operation can then depend on whether the filleting occurs before the rounding or vice versa. The rounding can eliminate object protrusions and/or small voids.

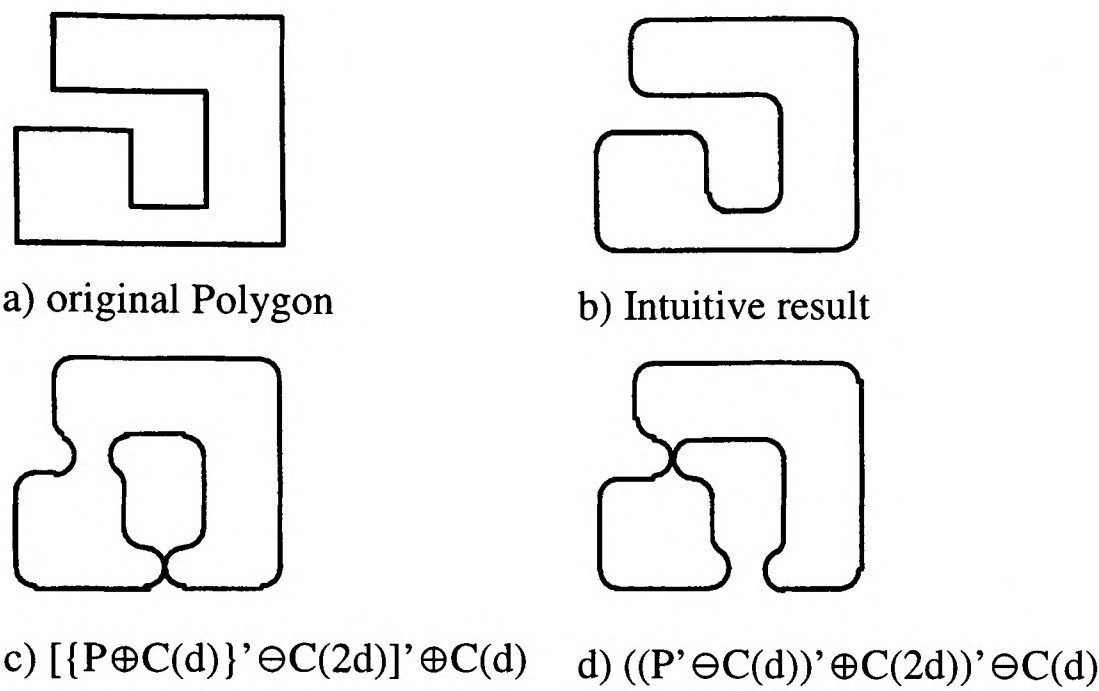


Figure 31 Problems with Minkowski Blends (after [Midd88])

Figure 32 shows the Minkowski sum of a small sphere and a simple object resulting in a dilated object. This is the output of an ACIS program by the author that can perform the dilation of any polyhedron. Because the resulting object always has curved faces, the process cannot be reapplied without first facetting the model.

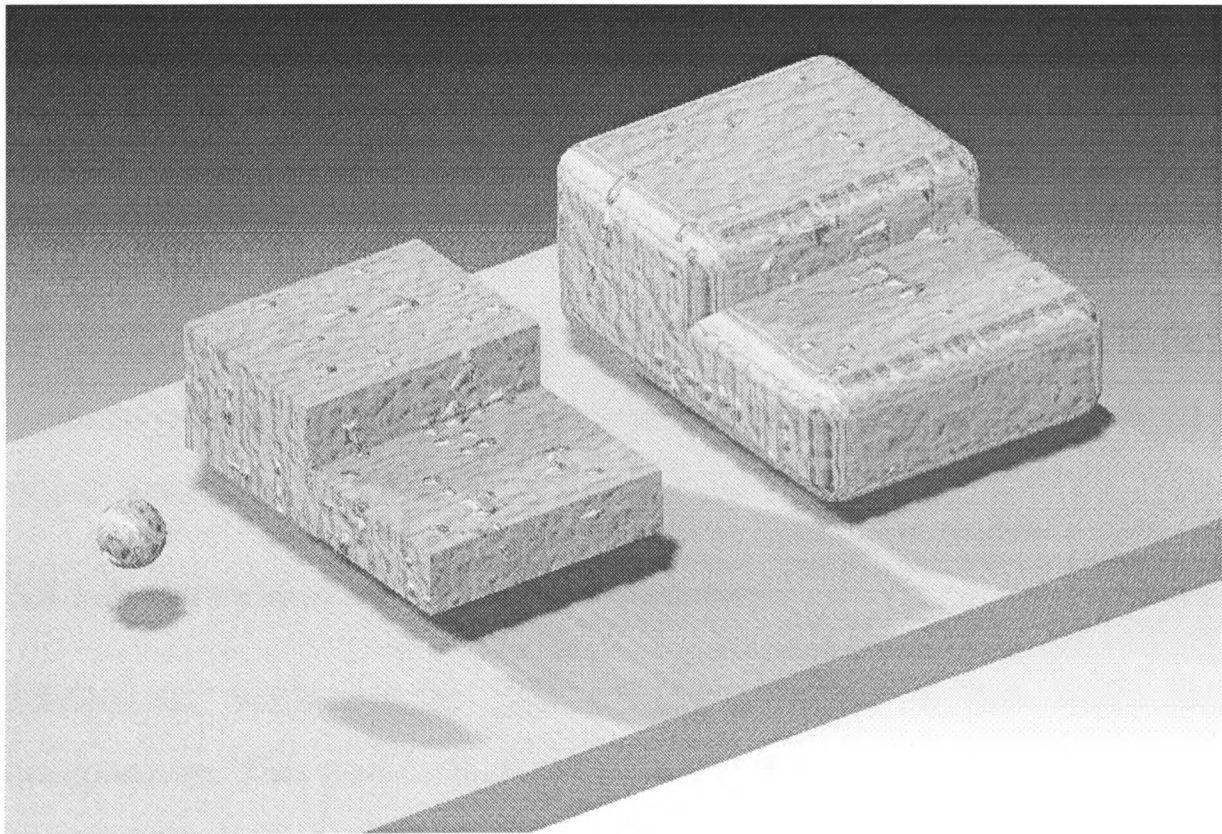


Figure 32 Dilation using a Minkowski Sum

3.4.1.4 Shape Design

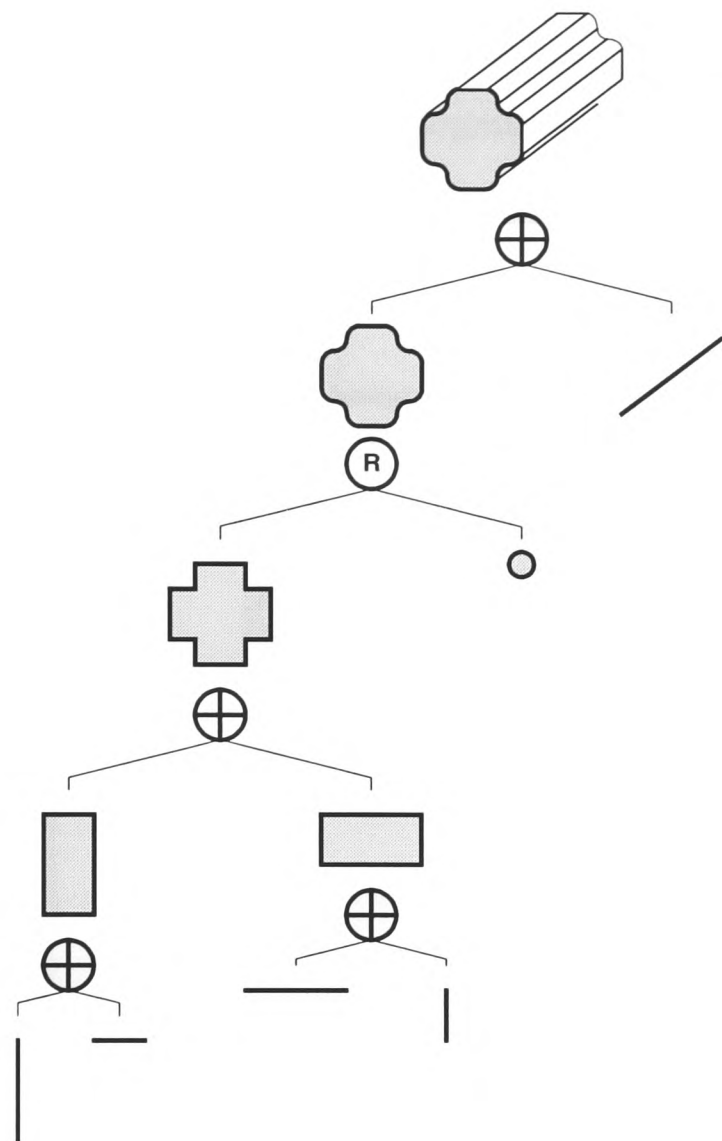


Figure 33 An Example of an Object created using CTS

In addition to solid modelling techniques such as CSG and B-Rep, Cumulative Translational Sweeps (CTS) have been proposed. Figure 33 shows an example of an object generated by CTS, Booleans and rounding.

3.4.1.5 Proximity or Thin Wall Detection

Minkowski Sums can be used for the detection of thin walls in feature based components. This particular problem is discussed in [Mill94] and in Chapter 4.

3.4.2 Medial Axis

A recent addition to the range of possible techniques for the recognition of features of engineering significance in geometric models is the medial-axis transform [Arms94]. The medial axis of a 2D region is the locus of the centre of an inscribed disc of maximal diameter as it rolls around the object interior. The approach has been

extended to 3D solids using the medial surface. Medial axis has potential for thin wall detection, applications in meshing for finite element analysis and the lower dimensionality of the medial axis compared with its parent body is thought to simplify some feature recognition processes.

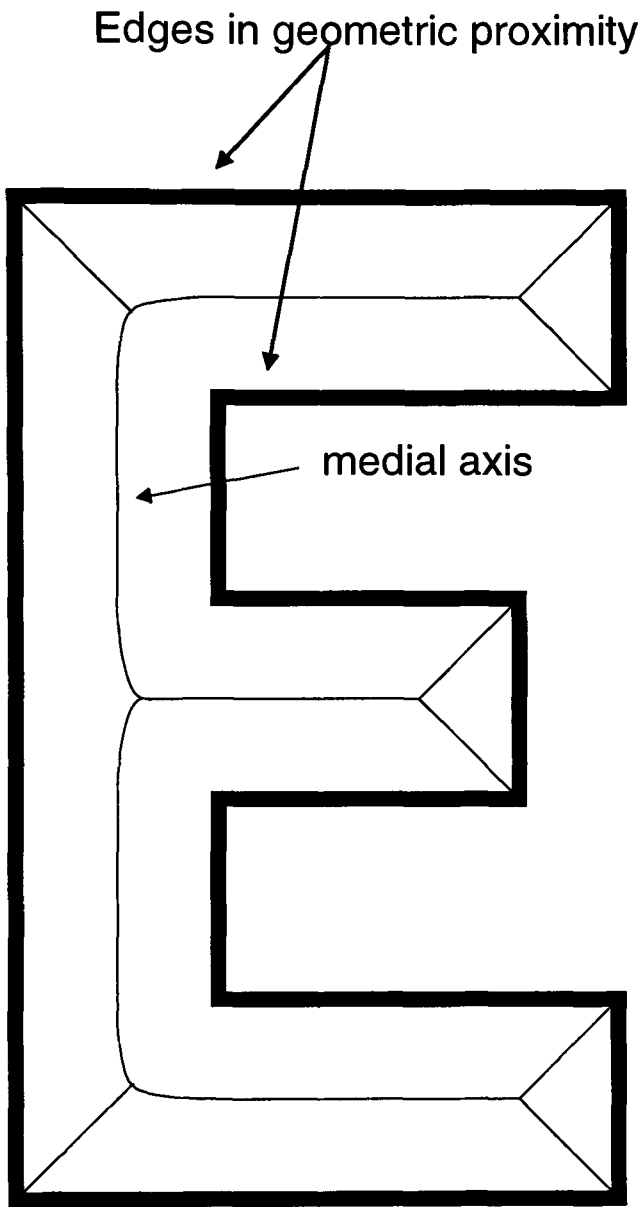


Figure 34 A 2D Region and its Medial Axis

3.4.3 Definition of 2½D

The term used by manufacturing engineers to describe components easily manufactured by three axis NC machine tools without the need for special tooling is 2½D. This informal definition is insufficient for features researchers and a more rigorous geometric definition is required. The following definition is due to Corney [Corn90].

A “strictly 2½D” object might be defined as having any of the following properties:

- Contains only planar and cylindrical faces.

- Has only step changes in height.
- Is prismatic. That is, it is one sided and when viewed from that side contains no undercuts or overhangs.
- Not multisided.
- Contains no small-scale shape variations such as chamfers and edge or fillet radii.

Mill regards a 2½D component as being any component that can be machined in a small number of setups on a 3-axis mill with standard tooling, and so would allow T-slots.

The author's strict definition of 2½D is a component for which there is a (one or more) planar face (called the 'back face') such that the surface normal of any point on the surface of the component, but not on the back face makes an angle of greater than 90° with the surface normal of the back face. This definition can be expressed formally and is equivalent to saying that a 2½D component can be fully modelled with a (arbitrarily exact) depth map. This definition does allow freeform surfaces however, something that is normally not considered 2½D in a manufacturing environment.

Any system capable of dealing with 2½D components in a reasonably complete way would be of great value to manufacturing engineers. True 3D components including sculptured surfaces and multiple approach directions represent only a small (but highly lucrative) portion of the machined component market. The FODDS2 system, whilst allowing excursions into true 3D, has clearly evolved to primarily handle 2½D components.

3.5 Features

As mentioned in section 3.3, the first generation of CAD tools were little more than glorified 2-dimensional drawing packages. It should be appreciated that even a correct set of 2D drawings of a 3-dimensional component drawn to an appropriate standard, (e.g. BS308), can still be an ambiguous or more often hard to interpret representation of the intended 3-dimensional component. Ensuring consistency

between multiple views of a component without having some common underlying model is also extremely difficult to achieve on all but the simplest components.

An example of an object whose plan, elevation and side views are difficult to interpret is that object made by the intersection of three cylinders of equal diameters aligned along the x , y and z axes. This problem is shown in Figure 35 below. A rendered image is shown on the left, with a front, plan, side and angled view on the right of the same object.

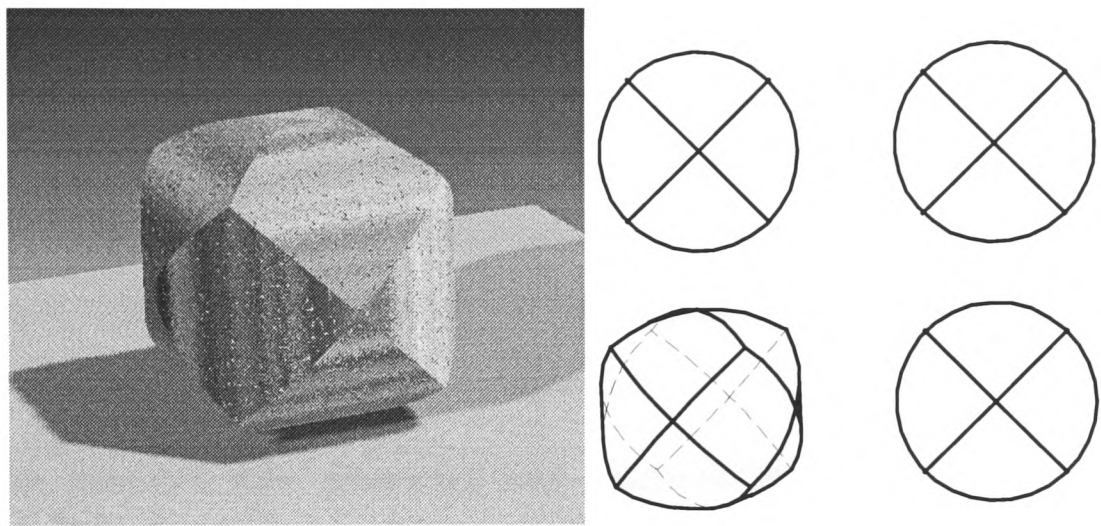


Figure 35 A Body Formed from Three Intersecting Cylinders

As a result of this deficit of 2D CAD systems, and with the advent of sufficiently powerful computers, there has been a move towards 3D CAD systems. In the first instance this was in the form of limited 3D functionality added to an existing 2D system. Such a system is the 3D addition to FastCAD that allows generation of isometric images from 2D part drawings. This necessitates a fair amount of extra work for the user ensuring validity of the 3D drawings so produced. This approach is nothing more than a visualisation tool.

However, the alternate approach is for systems containing true 3D solid modellers. Perhaps the best known of these is AutoCAD, particularly AutoCAD Mechanical Desktop. Initially a 2D system and now with a true 3D husk (incidentally built on ACIS) and strong links back to the original 2D systems. Other 3D systems include and EDS Unigraphics' IDEAS and SolidEdge, Strässle's Konsys [Denz93], HP's SolidModeler, Parametric Technology's ProEngineer, IBM's CATIA, Computervision's CADD5 and SolidWorks.

Interestingly, these products invariably contain 2D sketchpads, as even now this is frequently the fastest way of getting the bare bones of a design into the system. It is interesting to speculate whether gradual improvement in immersive virtual reality systems will result in the emergence of true 3D sculpting systems, or whether (particularly in the mechanical design industry), the 2D abstraction is of fundamental importance.

The majority of 3D design tools now have a powerful underlying solid modeller, of which perhaps Parasolid and ACIS from Spatial Technology are the leading commercial modellers. Systems not containing one of these commercial modellers contain a closed proprietary modeller. All too frequently, however the resulting 3D CAD system has powerful mechanisms for describing the geometry of the component that is designed, but poor mechanisms for extracting suitable data from this geometry to manufacture the component.

This gap between raw geometry and manufacturing requirements is frequently known as The Great Divide [Shah95].

A fundamental technology for bridging this divide is that of features [Mill96]. This concept has emerged over the past decade as a central technology for a number of academic CIM products notably PART [Hout91][Lend94]. It is taking longer to infiltrate into the commercial sector though certain systems can claim to have feature based modules, namely Parametric Technology's ProEngineer and Strässle's Konsys system.

It is useful also to consider the philosophies behind the various tasks to be merged. Particularly those of conceptual design, detailed design and automatic process planning.

Features are a natural way of thinking about a design at both the conceptual phase and the detailed design stage. What is considered to be a feature in these design phases and what is considered a feature for process planning may not (or indeed will not) be the same.

3.5.1 Applications

The most popular areas for potential ‘featurisation’ appear to be in detail design and process planning. Although many British companies have been able to make use of CAD systems in order to help gain significant improvements in handling design data, relatively few have benefited from Computer-Aided Process Planning (CAPP) systems. Several reasons have been proposed for this, including the high cost of implementation (due to company specific methods), lack of basic research (e.g. solids modellers and expert systems do not fully support CAPP research), few tangible advancements made in CAPP research and slow technology transfer (advancements that have been made are more likely to be exploited abroad).

Consultants from several countries who have assessed the potential for manufacturing planning software have invariably predicted potentially very large markets although the performance of present CAPP systems is not sufficient to satisfy demand from a technical viewpoint.

The majority of those working in CAPP world-wide have adopted feature oriented methods and there is, in this area at least, some consensus that this is one way forward, even though there is little agreement on what features are and how they should be used. Groups at Bath, Bristol, Brunel, Cambridge, Cranfield, Edinburgh, Heriot-Watt, Leeds, Loughborough, Newcastle, Southampton and Sussex as well as others have all been active in work using ‘features’ in the last few years. Successes in research have been confined to narrow areas of industrial application. Broader based work has also been attempted, but its success to date has been confined to making contributions to the research community as a whole, usually by showing how novel approaches might be used (e.g. in using solid modellers, IKBS techniques, Genetic Algorithms etc.).

UK research in feature oriented applications has been going on for some considerable time [Husb90][Gind92]. Features have generally been regarded by their supporters as providing high level and inherently meaningful data structures [Ovtc92] which allow for better user interfaces, improved part model transfer, increased automation of design and planning tasks, easier analysis, and improved

control over models that would encourage design for manufacture and parametric design.

3.5.2 Feature Definitions

An initial feature definition might be:

Any geometric or non-geometric attribute of a discrete part whose presence or dimensions are relevant to the product or part function, manufacturing engineering analysis, use, and so on, or whose availability as a primitive or operation facilitates the design process and manufacturing activities. [DeFa93]

There are three aspects of interest in this definition

A feature is a part of a larger entity.

The part has some properties that distinguish it from the whole.

Features can occur in all kind of entities and have a specific meaning for each such entity.

The shape, behaviour and engineering significance of a feature needs to be encoded in its definition. For this reason, and because the feature concept was initially linked with process planning, the feature definition originally implied *form features*. Therefore an early definition specific to geometric modelling was:

A specific geometric configuration formed on the surface, edge or corner of a workpiece [author unknown].

Another broader definition of a process planning related form feature was given by Wingard [Wing91]:

A generic shape that carries some engineering meaning.

Through such work, it has emerged that features are also relevant to other application domains such as engineering analysis, and that features do not necessarily relate to form. The pure geometric modelling definition does not include the reason for a feature's existence or usefulness. As features encode the engineering significance of the geometry, the definition must be extended to include the purpose for which a feature is used. The previous definitions have meaning only for manufacturing and

not necessarily for other applications. These definitions are considered as *manufacturing form features*, or simply *manufacturing features*.

One of the most recent and most comprehensive definitions of manufacturing features is:

Form features are form elements with some function or meaning, used to model functional information about the way some part of the object is manufactured or assembled, so not only the geometric description of form features is of importance, but also the functional information [Bron93].

It is important to underline that a form feature is considered to be a distinctive characteristic of the topology, rather than the entire shape of the part.

As already mentioned the main motivation for the development of the feature concept has been in the area of process planning, where features have been used as a research tool for several years. In this field, features can identify areas in a product that can be manufactured in one operation with one type of machine, e.g. a hole that can be sunk with a particular type of drill, or a slot that can be milled with a particular type of milling machine.

Since applications in other areas are emerging now, it is important to discover an all purpose definition of a feature. Examples of these more general definitions are:

A recurring pattern of information related to a part description [Shah91a].

A semantic grouping used to describe a part and its assembly. It groups in a relevant manner functional, design and manufacturing information [Giac90].

An element used in generating analysing or evaluating design [Wils90].

A functional shape aspect for design and manufacturing [Vane90].

A semantic data set that can be attached to product parts [Shah91c].

In the last series of definitions, other information is mentioned in addition to the shape or form of parts. As Shah has defined, features represent the engineering meaning of the geometry of a part or assembly; thus, the requirements that a feature should at least fulfil are that it has:

to be a physical constituent of a part

to be mappable to a generic shape

to have engineering significance

to have predictable properties

It is possible to continue listing different definitions, as there are almost as many definitions of features as there are researchers in features. At the moment it is difficult to decide which one, among all these definitions proposed, can be considered the most significant. However, considering the intrinsic nature of features in manufacturing industry, the definition of features should not focus on a particular process. The greatest difficulty and challenge is to define features that are meaningful to all life cycle issues. Furthermore, it is only recently that researchers have found unified definitions for any feature applications, as shown in Table 1.

Design Feature	A discrete piece of information fulfilling a function on the component and that is made available for the designer’s use.
Process Planning Feature	A distinctive or characteristic part of a workpiece, defining a geometric shape, which is either specific to machining processes or can be used for fixturing or measuring purpose.
Manufacturing Feature	A parameterised geometric object that corresponds to a manufacturing operation.
Machining Feature	A subclass of manufacturing feature. A prismatic or cylindrical volume that has primitive machining operations associated with it.
Assembly Feature	A feature that defines relationships between different parts in an assembly.
Feature in Solid Modelling	A volume whose properties include translation, rotation and scaling

Table 1 Feature Applications

For completeness it has also been necessary to introduce the concept of *abstract features*, defined by Shah [Shah91a] as:

Entities that cannot be evaluated or physically realised until all variables have been specified or derived from the model.

It is necessary to introduce this idea as the complete definition of a shape requires the specification of all dimensions and location parameters, but not all these parameters are available, or even important, until the final stages in design. Artefacts evolve

progressively, with partial or sketchy definitions of the product. Geometric evaluations of such features need to be postponed. Many types of reasoning, both automatic and manual, can be performed on incomplete or abstract feature instances.

Table 2 gives examples of some common features classified according to the definitions given above:

Design features	Blind Hole, Through Hole, Slot, T-Slot, Curved Slot, Through Slot, Pocket, Rectangular Pocket, Step, Blend, Boss, Fillet.
Process Planning Features	Cylindrical Hole, Slot, Others.
Manufacturing Features	Hole, Slot, Shoulder, Rectangular Pocket, Chamfer, Undercut, Pocket, Boss, Block, Island, Fillet, Ring, Slice, Bearing Seat Circular Pattern, Array Pattern, Elastic Ring Seat, Internal Centring Surface, External Centring Surface, Screw hole, T-Slot, Curved Slot, Pin, Step, Others.
Assembly Features	Screw Hole, Spline, Hole, Datum (Plane, Axis), Screw, Pin, Slot.

Table 2 Common Feature Examples

Much attention has been given in the research community to the classification of features. Examples of these taxonomies include Wilson [Wils89], Case and Gindy [Case94], STEP [Shah91c], Ovtcharova [Ovtc92], Shah [Shah91a], and Dohem-Bronsvoot [DeKr95].

3.5.3 Feature Attributes

The complexity of design and manufacturing usually determines the number and types of features required to represent a part of an object. Additionally features should represent a set of design attributes or specifications that design attempts to pursue in a part or product.

Attributes associated with the features are abstract entities that provide a specific part or product description. The two terms, features and attributes, have often been regarded as synonymous, however, while a feature is something that goes to make up something else, an attribute is a characteristic or quality of a thing.

Attributes can be considered as a way CAE tools to transfer the non-geometric technical information that is needed for downstream applications in the product life

cycle, onto the CAD model. They are used to represent a wide variety of information, from identifier labels to complex geometric relationships. Attributes help in reducing ambiguity and non uniqueness for feature manipulation methods. The same feature may be represented by a different set of attributes for a differing design or manufacturing application.

To define attributes is almost as difficult as defining the feature concept, but the Pratt-Devries definition is:

An attribute is a characteristic quality or property which associates meaning to an entity, significant to a particular stage in the life cycle of a product. [Subr95].

Examples of attributes can be considered colour of a face, type of thread or relationships between two faces. An attribute associates meaning to an entity or to a relationship between entities.

It is possible to say that attributes are characteristics of features, as well as features being constituents of parts that can, in turn, be constituents of assemblies. Attributes can be applied at any level of feature, a collection of features or to a whole part. Examples of attributes are given in Table 3 below.

Dimensions	Diameter, Length, Space requirements, Depth, Width, Corner radii, Chamfer angle, Chamfer depth, Thread pitch, Height.
Positions	Location, Orientation/Axis, Direction, Entry/Exit, Boundaries, Centre, Centreline, Origin, Feed direction, Position handle.
Geometric Tolerances	Form: <i>Straightness, Flatness, Roundness, Cylindricity, Profile of a line, Profile of a surface.</i> Position: <i>Parallelism, Perpendicularity, Angularity, Concentricity, Symmetry, True position, Circularity, Runout, Total runout.</i> Maximum material condition, Minimum material condition.
Surface finish	Roughness.
Material properties	Deformation, Hardness, Elasticity, Rigidity, Stiffness.
Properties	Functional, Performance parameters

Table 3 Attribute Examples

3.5.4 Manufacturing Feature Justification

This subsection justifies the use of manufacturing features in a feature based design system that is intended as the front end for a computer aided process planning system.

- A *design feature* is anything that is of interest to a *designer*.
- A *manufacturing feature* is anything that is of interest to the *manufacturer*.

It is a truism that anything that can be manufactured can be designed. It is not necessarily true, however, that everything that can be designed can be manufactured. It is important in order to reduce design rework and manufacturing costs that designers are encouraged to design manufacturable objects.

If some subset of the features that the designer is provided with are manufacturing features then use of those features in a design will increase the likelihood that the entire design is manufacturable. One way of accomplishing this is to provide the designer with all manufacturing features in the system, that is, manufacturing features are a subset of the set of possible design features.

Manufacturing Features \subset Design Features

In practice, the implemented features are in fact a subset of all the design features that could be offered, and so not all manufacturing features may be implemented in the design system.

Implemented Features \subset Design Features

The FODDS2 system is a front end to an automatic CAPP and CAM system. It would seem to be a sensible compromise to offer, in the first instance, a set of manufacturing features to the designer as this will help ensure that all designs are manufacturable.

Constraining the designer to using only manufacturing features does not guarantee manufacturability however; manufacturing features can be placed in a part in such a way that manufacture is not possible. This can be due either to an access problem, where the tool is unable to machine the feature volume due to a potential intersection of the tool or machine by the current state of the workpiece or fixtures, or due to

more interesting feature interactions where a particular order of feature manufacture is required.

3.6 Summary

This chapter has reviewed a number of background areas that lay the foundation for the work in the remainder of the thesis.

1. The ongoing work of the Manufacturing Planning Group at The University of Edinburgh has been introduced to place this thesis in context. In particular, the HAPPI process planner has been discussed in more detail.
2. A discussion of the area of Computer Integrated Manufacture has been included with the definition of the concepts of concurrent and simultaneous engineering to place the work described in the remainder of the thesis.
3. An introduction to solid modelling, still fundamental to feature based design systems and now regarded as essential for CAPP systems.
4. An introduction to Minkowski sums leading in the next chapter to their use as a mechanism for describing both manufacturing features and related volumes. A brief introduction to Medial Axes, which will be discussed briefly in the next chapter as an alternative method for thin wall detection.
5. An introduction to features, building on the previous review of Shah's approach to features, and justifying a manufacturing feature view for a prototype feature based design system.

4 Geometric Reasoning for Process Planning

This chapter covers the theory behind the FODDS2 Feature Based Design and Manufacturability Analysis System.

The chapter first defines the definition of a feature based component within the system. It follows this with a description of the major feature types supported within the system and the mechanism by which these features are generated. The particular mechanism allows automatic generation of the various volumes required for the subsequent reasoning regardless of the individual feature type, provided that feature type is described in the manner detailed. Though this restricts the range of features, it guarantees that the geometric reasoning will work for all features described in this way.

The second half of the chapter details the geometric algorithms that the system uses for manufacturability analysis prior to process planning.

4.1 Definition of a Component

Throughout this thesis a manufacturing viewpoint of components is taken, and more specifically a traditional and CNC machining viewpoint, so a finished component is a blank (or casting) from which material has been removed to result in some desired geometry (and tolerance conditions). The mechanism for material removal is normally some form of metal cutting (though there is no particular reason to rule out EDM or ultrasonic machining for example).

A manufacturing feature (again from the viewpoint of 2½D milling) is a volume of material that is to be removed from the blank and which can (usually) be removed by some standard set of machining operations. Typical examples are slots, holes and pockets. The way in which the geometry of these manufacturing features can be developed is described in the next section.

Having decided on some set of parameterised features, a particular component can be described in the following terms.

If V_i is the i th volumetric feature found in a description of the component, and B is the blank before any machining has been performed, then the finished component C can be described by the following equation:

$$C = B -^* \bigcup_i^* V_i$$

That is the component is the blank from which is subtracted the regularised union over all i of the feature volumes V_i .

The volume to be removed from the blank Δ (the delta-volume) can be expressed as

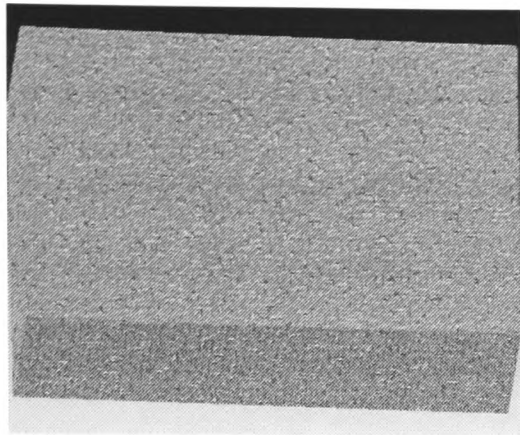
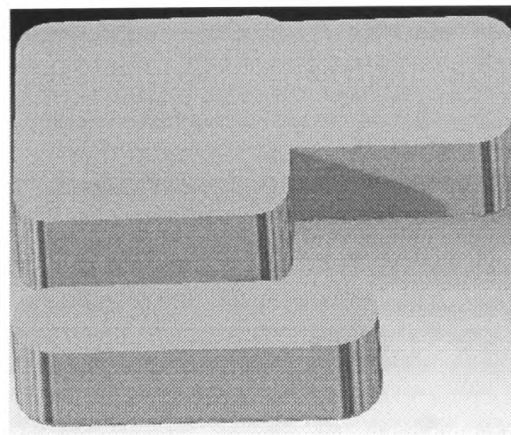
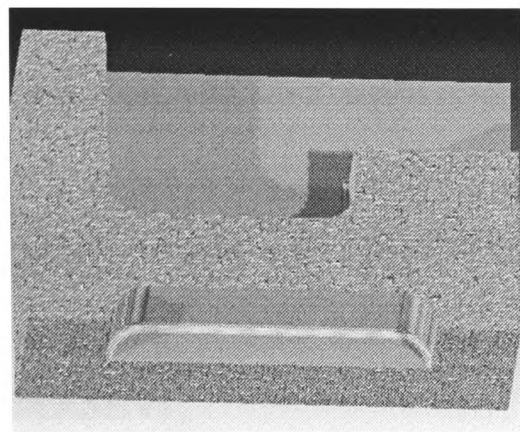
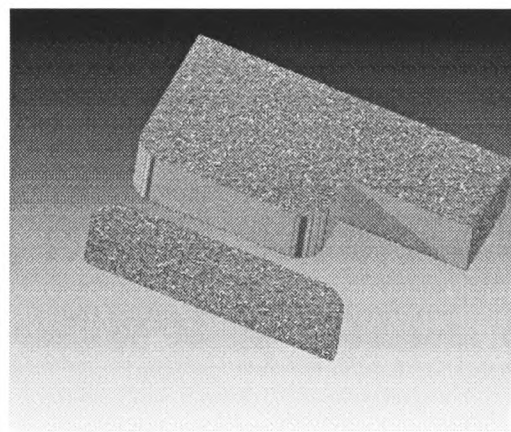
$$\Delta = \bigcup_i^* V_i \cap^* B$$

and therefore,

$$C = B - \Delta$$

$-^*$, \cap^* and \cup^* denote the regularised set operators for subtraction, intersection and union respectively [Requ80]¹. The above equations are illustrated in the example shown in Figure 36 below. The delta-volume may be smaller than the union of the features as parts of some of the features may lie beyond the blank. This represents volume that would have been machined had that volume been part of the blank. Usually the cutter will pass through this extra volume because of tool geometry and motion. Issues arise here regarding fixturing and clamping. The figure is purely illustrative and the use of two rectangular pockets (that intersect) and a slot in this manner are not intended to suggest any particular component.

¹ The $*$ is dropped throughout the rest of this thesis, and unregularised Booleans will be mentioned specifically if needed

**a) blank****b) features****c) finished component****d) delta-volume****Figure 36 Blank, Features, Component and Delta-Volume**

4.2 The Feature Library

Having specified how a component is produced given a list of features (and the blank), it is now necessary to define the features themselves. The features will necessarily be material removal manufacturing features for the reasons given in section 3.5.

Firstly, a feature definition was established. Following this, features from other researchers systems and other entities that might be considered features could be classified for inclusion in the FODDS2 system. It was established that there is a differentiation between design and manufacturing features. Further consideration resulted in the following (somewhat glib) definitions.

In the chosen domain, the principal manufacturing features can be summarised primarily as *holes*, *slots* and *pockets*. These are broken down into subtypes and augmented with a set of features from a particular machine tool manufacturer, Mandelli of Piacenza, Italy, the enduser involved in the SESAME project.

These three classes of features embody the vast majority of geometric shapes involved in 2½D milling.

Some feature systems offer surface features, a class of feature that is not offered in FODDS2. This omission is sufficiently important to warrant explanation.

In a machining context, there is no such thing as a surface feature. All features are material removal features. Often the term feature is used in order to change the surface finish of a face. In the context of FODDS2, this is anathema. In order to change the surface finish of a face using a machining process, material must be removed, and so a depth of material must be specified.

If a depth is specified, then the pocket feature is available to perform this task, and a surface finish can be specified on the pocket. A pocket feature is used rather than inventing a slab feature for the reasons outlined in the discussion of Kramer’s MRSEVs.

However, what is meant by the simple class names is not so clear. A brief discussion of the *hole*, *slot* and *pocket* classes follows:

4.2.1 Holes

No problem here surely. Everyone knows what a hole is.

hole (hōl) *n.* [OE. hol] a small, dingy, squalid place. [Collin’s Concise]

A first definition:

A hole is a rotationally symmetric material removal feature.

This covers many examples of holes, but fails to distinguish between a hole, a circular groove, and in some cases a circular boss, where a boss is machined by removing a ring of material from around the boss which is also rotationally symmetric.

An alternate definition:

Simple holes are material removal features characterised by their chief parameters of diameter and depth. This neglects other attributes of holes such as end geometry,

surface finish, chamfers, counterbores and threads, but captures the essence of a simple hole.

Another typical property of simple holes is that they are manufactured by rotating tools translating only along the axis of rotation. The PART process planning system, for instance will take any *hole* that has an aspect ratio such that its *diameter* > 10 * *depth* and recategorise this hole as a pocket [Lend94].

Lastly an important feature of holes is that they are defined from one direction, but the resulting geometry must be analysed in order to discover whether the hole is a through or blind hole and thus whether a hole can be machined from ‘below’ as an alternative to the default approach direction. This property of multiple access directions is also important for other features, but is of prime importance for holes.

4.2.2 Slots

slot (slot) *n.* [Ofr. *esclot*, the hollow between the breasts]

The chief manufacturing characteristic of a slot seems to be that the slot is predominantly machined with a translation of the tool perpendicular to the rotational axis of the tool. This definition still allows all manner of straight and curved slots and allows the use of special slot cutters such as T-Slot and dovetail cutters.

4.2.3 Pockets

pocket (pok’it) *n.* [Fr. dim. of *poque*, a bag]

Pockets differ from holes and slots in their complexity. Whilst holes and slots are formed from a simple linear chain of movement of the cutter, pockets, at least in the sense used in this work are formed by a two dimensional tool path with additional approach and retract paths.

4.3 A Grounding for Manufacturing Features

All the design features used in the present system are based on manufacturing features. Consequently, a solid grounding for the geometry of these manufacturing features based on typical tool profiles and suitable machining techniques is required. The present set of features are restricted to those easily produced by a 3-axis NC

milling machine. The resulting features support the geometry of the vast majority of 2½D components, and are sufficient to design many real 3D components. However, 3D components with many distinct approach directions will lead to fixturing and setup problems. The chief restriction of the feature library shown is that true freeform surface profiles are not permitted. Design using freeform surface features is a current area of research, particularly in the ongoing BRITE/EuRam IMPRESS project [Agos97].

Sungurtekin [Sung86] lists the trajectories used in 3-axis machining as:

1. Linear in z (the spindle direction)
2. Linear in the xy plane
3. Linear in space (simultaneous movement of the three axes)
4. Circular in the xy plane
5. Circular in the xz and yz plane

Only features using trajectory types 1, 2 and 4 are considered. The reasons for this are now outlined.

Traditional 2½D machining generally involves a rotating tool that is then driven along some path. The tool generally follows some approach and retract path at either end of its machining path. In the case of 2½D objects, the path followed during machining is either parallel or perpendicular to the axis of rotation. Also, somewhat pragmatically, the available BridgeportII NC mill is only capable of producing motions of types 1, 2 and 4.

This generally leads to components composed only of analytic surfaces and hence in the more robust and reliable area of the domain of current solid modellers.²

² ANSYS, a leading finite element package, and other analysis packages, don't restrict themselves to analytic surfaces, instead they carry all surfaces internally as spline surfaces. In ANSYS' case, these are NURBS surfaces. Though this approach offers great flexibility, there are efficiency gains using analytic surfaces when dealing with objects composed predominantly of analytic surfaces. There are then a number of special cases of surface-surface interaction, however each can be dealt with more simply than the general NURBS surface – NURBS surface interaction. The tool CADfix from FECS Ltd allows conversion between these differing internal formats (even when the external format such as an IGES file is the same).

Exceptions to the above general rules can be found in various examples of slot cutters, fly cutters and boring bars.

The machining of freeform surfaces, though an area of interest and much present research is outwith the scope of this thesis.

Any rotating tool of fixed geometry can thus be represented by two profiles, that when swept about the tool's rotational axis give volumes and surfaces of interest. The first profile represents the volume of the entire tool and is used to check the access of the tool to the component. The second profile represents the cutting surface of the tool. In Sungurtekin's nomenclature, the first profile is referred to as the total tool profile and the second, the operational tool profile.

Let some complete cutter path consist of an approach path, a cutting path and a retract path. These paths all consist of straight line segments or curve segments. All the curve segments are restricted to circular arcs. The tool does not change orientation during the motion. This is all consistent with a partial process plan for a 2½D feature to be machined on a three axis NC milling machine.

Boring bars provide an additional problem, their access body for the insert and retract phase is equivalent to the tool profile swept around the vertical axis, however, their 'access body' during the machining phase requires the tool profile to be moved such that the tool tip is at the diameter of the required ring.

All these manufacturing features must be derived from at least one of the tools defined in the system, but any feature may be manufacturable in a number of different ways if access problems are neglected. Only when a manufacturing strategy is identified for a feature can valid tool classes be identified. This is accomplished through the construction of an access body based on the feature and the associated tool classes. Even this level of checking is based on the 'perfect' tool for the features parameters, regardless of the actual tools available. Again the geometric reasoning here is pre-planning, and is merely to perform some preliminary checks regarding the ability of the system to manufacture the feature.

4.3.1 Cutters with Limited Access

There is a class of cutters such as T-slot cutters, dovetail cutters and boring bars that are capable of manufacturing features with limited access.

The cutting volume of a T-slot cutter and a dovetail cutter is shown in Figure 37. These cutter volumes show the volume of the feature, but fail to represent the approach and retract volumes, so a design could become unmanufacturable if access of the cutter to the feature is impossible (this is a different case to the access a cutter needs whilst machining a feature. A plausible solution might be to add these approach and retract volumes to the ends of the features, to ensure they are considered during design and pre-planning, but this in turn enforces the position of the approach and retract phase relative to the feature. If a particular component geometry is required that requires the approach and retract vector to be at the midpoint of a slot this solution would fail. Features requiring tools with limited access are not implemented in the current system.

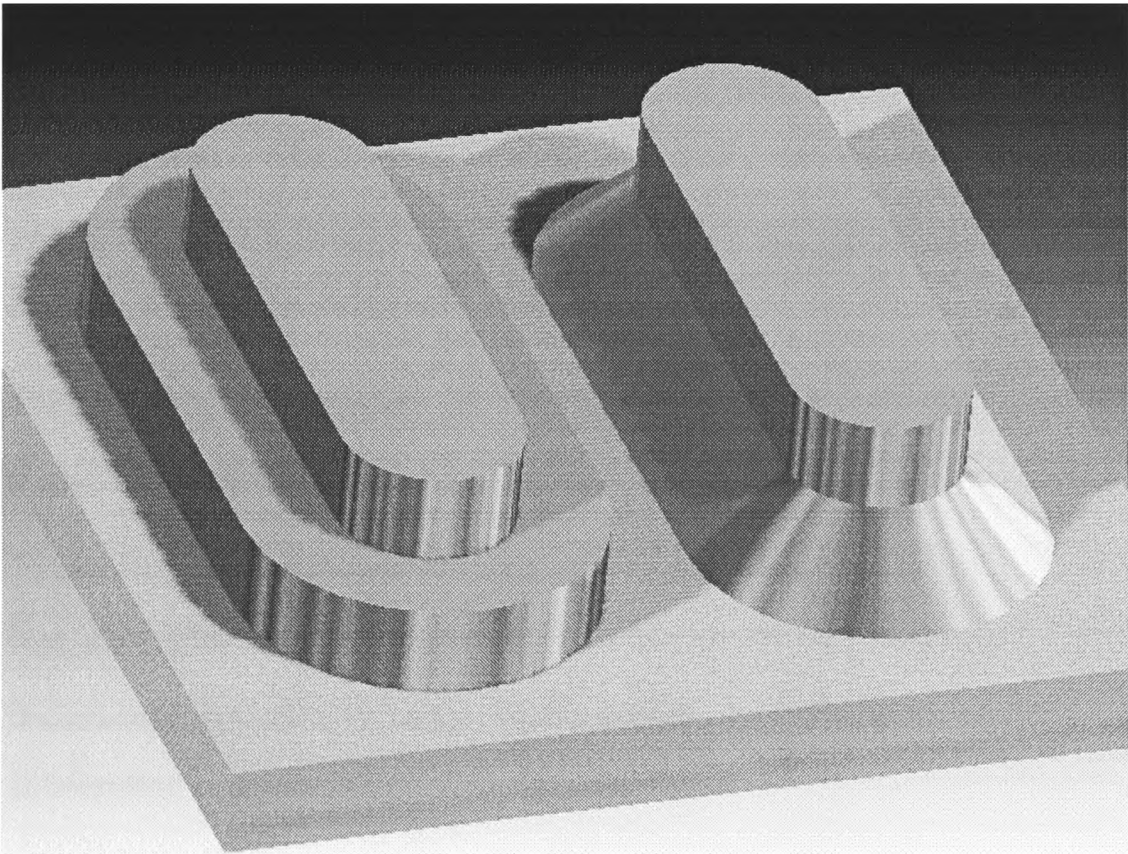


Figure 37 Feature Volumes of Cutters with Limited Access

Boring bars present exceptional problems when trying to develop a sound description for 2½D machining in terms of their cutter profiles and in terms of the Operational Swept Volumes (OSVs) of Sungurtekin. They are the only class of traditional cutter for which the cutter profile can change dynamically during the cut. Vandenbrande

[Vand91] points out that in the limit, a boring bar can cut a groove feature of three times the radius of the hole allowing access. As with the T-slot and dovetail cutters, boring bars are neglected in this thesis.

4.3.2 Tolerancing

Tolerancing presents a problem. It is widely agreed that the tolerances specified on typical engineering drawings through standards such as BS308 map poorly on to tolerances that can be specified in manufacturing.

Tolerances are specified for a number of reasons:

- to ensure that parts will function properly.
- to ensure that mass produced parts will be interchangeable.
- to ensure that parts are manufacturable (at reasonable cost) and assemblable.
- to ensure that the design is robust (critical dimensions have minimum sensitivity to expected variations).

The first two of these reasons can be categorised as *functional* requirements. The third reason is something that may be neglected by the designer in the first case and require an extra iteration of design following initial process planning and is thus a critical reason for Concurrent Engineering.

The fourth encourages the designer to use the loosest tolerance whilst ensuring required functionality.

Thus, *functional* tolerances actually form a network of complex constraints between features that must be resolved into positional and orientational constraints on individual features in order to convert these functional tolerances into manufacturing tolerances.

In order to make the feature based design system both functional and to simplify the coding problems, the full set of geometric tolerances is not presented to the designer, instead the designer must specify tolerances in terms of the position and orientation constraints on each feature relative to its parent feature. This simplifies the problem of tolerance analysis to (in the case of positional tolerances) the summing of tolerances along the principal axes, and provided all angular tolerances are small, the

summing of angular tolerances about principal axes. Thus, the problem of tolerance constraint management is shifted. Tolerances have in fact been the subject another researcher in the group and research in this complex area is expected to continue in the future. According to Voelcker [Voel97], geometric tolerancing is insufficiently well formalised, and an attempt to ‘mathematicise’ tolerancing is under way as part of an ANSI initiative. Though geometric tolerancing has been in use for decades it seems unreasonable to attempt a full implementation whilst a suitable new formulation is under development in addition to being outwith the scope of this thesis.

4.4 Generating Feature Volumes for Manufacturability Analysis

Prior to presenting the algorithms, it is important to set up a framework for discussing the algorithms. This framework is used to describe the geometric models of features and various volumes related to the features and used for the geometric reasoning.

This model for generating the feature volumes has the important property that all feature volumes are generated in exactly the same way given a logical tool profile and logical cutter path. These tool profiles and cutter paths may not correspond to tools subsequently selected by the process planning system, but are representative of the ideal tool for that particular feature volume. This property allows more features to be added with comparative ease, provided they can be described in this way. In the FODDS2 system, ring features have been fully implemented, and pockets described using a polygonal tool path as an extension to the rectangular pockets have been partially implemented. The method by which the features are implemented from the designer’s view in FODDS2 is given in Chapter 5.

Firstly, Figure 38 shows all the various *tool profiles* used to generate the necessary geometric reasoning and modelling bodies in FODDS2. Each solid body associated with the feature can then be described as the Minkowski sum of the solid of revolution of the tool profile and the cutter path of the associated feature type.

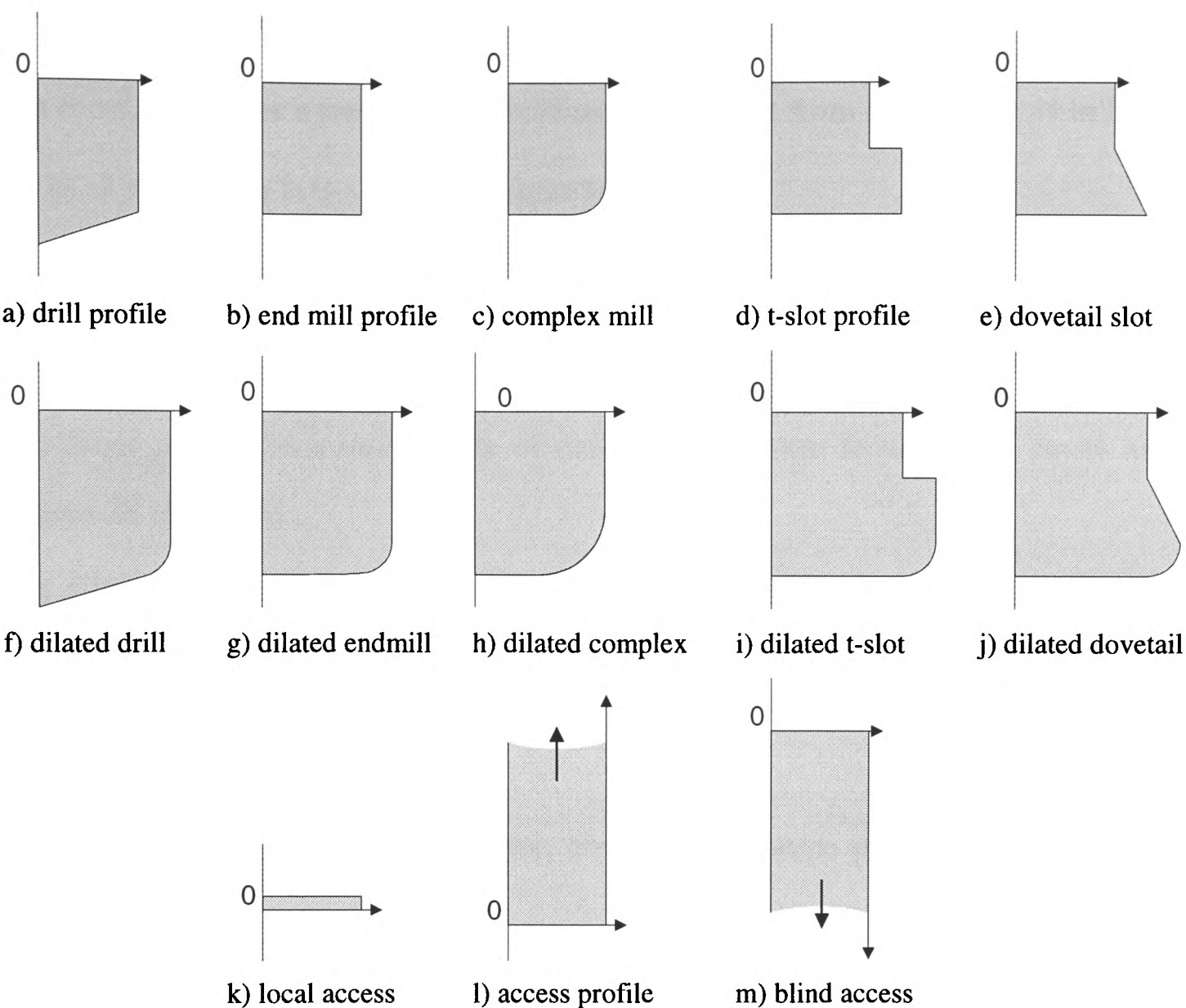


Figure 38 'Tool' Profiles in FODDS2

Firstly, some terms are defined:

Let a component consist of m positive feature volumes and n negative feature volumes.

Here, the positive feature volumes are simply volumes such as cylinders and blocks that can be united to make the majority of stock workpieces found in a workshop. Alternately a solid model of a casting can be imported as a single entity and subsequently used as the stock. Uniting the m positive feature volumes merely produces the stock that is subsequently used.

Let any negative feature volume be specified by a tool path, T_i , and a cutter path C_i .

Where the tool path, T_i , lies in the y - z plane and is entirely in the $+y$ halfspace.

The cutter path, C_i , lies entirely in the x - y plane, and can be a 0D point, a 1D arbitrary wire, or a 2D bounded surface.

Let the maximum radii of the tool path be r_i . ($r_i = \maxrad(T_i)$)

Let $local_prof(r)$ be a tool profile of radius r and extent from $z=0$ to $z=\delta$ as in Figure 38 k) above, where δ is a small but finite length.

Let $access_prof(r)$ be a tool profile of radius r and extent from $z=0$ to $z=+\infty$ as in Figure 38 l) above.

Let $blind_prof(r)$ be a tool profile of radius r and extent from $z=0$ to $z=-\infty$ as in Figure 38 m) above.

Let $dilate_prof(T, x)$ be the Minkowski sum of a profile T and a quarter circle in the $(-y, -z)$ quadrant of radius x . In Figure 38, f)-j) are the dilated profiles of a)-e). That is the dilated profiles are the profiles a-e offset outwards by a distance x , and then trimmed to the $(-y, -z)$ quadrant.

Let $RevZ(t)$ be the solid of revolution created when some profile t is rotated 360° about the z -axis.

Then:

Let P_i ($i=1, m$) be the list of all positive feature volumes

Let N_i ($i=1, n$) be the set of all negative feature volumes,

$$N_i = RevZ(T_i) \oplus C_i$$

Let V_i ($i=1, m+n$) be the list of all feature bodies (whether positive or negative)

Let A_i be the set of access bodies for all the negative feature volumes

$$A_i = RevZ(access_prof(r_i)) \oplus C_i$$

Let D_i be the set of dilated bodies for all the negative feature volumes

$$D_i = RevZ(dilate_prof(T_i)) \oplus C_i$$

Let B_i be the set of blind-access bodies for all negative feature volumes

$$B_i = RevZ(blind_prof(r_i)) \oplus C_i$$

Let L_i be the set of local-access bodies for all negative feature volumes

$$L_i = \text{RevZ}(\text{local_prof}(r_i)) \oplus C_i$$

Let W_i be the set of local-coordinate entities for all the negative feature volumes

(All Boolean operators are regularised unless explicitly mentioned)

The *stock*, P , is the initial state of the workpiece before any machining takes place.

$$P = \bigcup_{i=1,m} P_i$$

Let N be the combined point-set of all the negative features.

$$N = \bigcup_{i=1,m} N_i$$

The *delta-volume* (Δ) is that volume of the *stock* that needs to be removed to form the finished component

$$\Delta = N \cap P$$

The point-set of the finished component C is the difference between the *stock* and the *delta-volume*.

$$C = P - \Delta \text{ or } C = P - N$$

From the above equations, a set of methods can be generated to be implemented in the system. These methods produce sets of bodies depending on the method name, for instance the access method returns a list of all access bodies for all negative features in the component description. These methods are summarised in the table below (Table 1). Subsequent reasoning is performed using these lists as input data.

(The reader should beware of confusion between B , the blank and B_i , the blind access body associated with feature i . A similar confusion may arise between C , the component, and C_i , the cutter path associated with feature i .)

Table 1 Geometric Reasoning Body Methods and their Equations

<i>Method</i>	<i>Equation</i>
“component”	$C = P - N$
“blank”	$\cup^{\dagger}P$
“feature”	$\cup^{\dagger}N$
“access”	$\cup^{\dagger}A$
“local-access”	$\cup^{\dagger}L$
“blind-access”	$\cup^{\dagger}B$
“dilated-feature”	$\cup^{\dagger}D$
“wcs”	$\cup^{\dagger}W$

Note on Table 1: \cup^{\dagger} forms a set, but not a point-set. It forms the set of bodies of the type in the equation, thus the set of bodies is a set in that each body is unique, however the point set of \cup^{\dagger} is not necessarily a set in that features or other bodies may intersect. (An alternative view is that it returns a list of bodies).

4.5 The Geometric Reasoning Algorithms

This section deals with the geometric reasoning that FODDS2 performs. All of these algorithms can come under the heading of manufacturability analysis, but the algorithms have two functions. Firstly, there is the design validation function. Secondly, but more importantly is the geometric reasoning that is done as a necessary prelude to process planning in order to provide the system with specific suggestions regarding manufacturability problems.

The output of the algorithms are referred to as suggestions because at this stage before any planning has been done nothing is known about machine or tool selection and hence for any particular feature the validity of the suggestions provided cannot be immediately confirmed. On the other hand much of this reasoning has to be done at some point and the reasoning performed at this stage can dramatically reduce the search space that the planning system must explore in order to produce acceptable process plans. Additionally, the reasoning performed here provides a sufficient level of clues for the extant planner (HAPPI) to produce process plans. HAPPI does not

contain a solid modeller and hence cannot actually perform any detailed geometric reasoning of its own.

The geometric reasoning assumes that a design exists, whether partial or complete, and each parameter of that design has been instantiated. That is, an evaluated geometry is available. That is, any parametric design or constraint issues have already been resolved.

The algorithms presented in this chapter are:

- Void recognition
- Feature presence
- Access problem detection
- Proximity detection
- Intersection detection
- Through hole detection
- Hole interference detection
- Alternate access direction

The application of the information inferred by these algorithms is discussed.

Void recognition is a validation technique applicable to components expected to be manufactured by 2½D machining. Voids within a body cannot be manufactured by conventional machining techniques.

Feature Presence is also a validation technique that confirms that all features do at least partially intersect with the blank. Features that do not can be ignored for manufacturing purposes as they play no part in the final geometry.

Access problem detection is fundamental to process planning and can be used to infer some anteriority (or ordering) constraints between features.

Proximity detection is used to detect thin walls, both between features and between features and the edge of the workpiece. This detection is performed in order to avoid wall buckling or rupturing during machining.

Intersection detection is comparatively easy to perform, and intersections offer much information about potential problems, but the data obtained can be hard to interpret. Intersections can be used to decide on anteriority constraints, but this often requires intimate knowledge of the tool that is subsequently selected to machine a feature. Intersections can also be used to suggest alternate representations of features to aid process planning.

Through feature detection can apply to any type of feature and is used to detect whether the direction opposite the principal access direction (that specified by the designer) is a possible access direction for that feature. This is particularly important for through holes.

Hole interference detection is used to detect whether features, or indeed the blank, will cause problems for the machining of holes. This algorithm only applies to features machined solely with an approach-retract movement.

Alternate access direction is used to detect whether any direction normal to the principal access direction can be used as a possible machining direction.

The geometric reasoning discussed, in particular proximity detection, would benefit from the ability to dilate a solid body by an arbitrary amount using Minkowski sums. The mechanism for this has been previously discussed.

The algorithms then submit the information they have inferred to a database of ‘facts’ using an *assert* function. A *retract* function exists to remove facts from the database. (This database is implemented as a simple list in Scheme and the concept is borrowed from the *assert/retract* functions available in Prolog).

4.5.1 Void Recognition

If a designer adds a feature to a component and that feature is wholly within the component, then that feature cannot be machined using traditional machining techniques. Other more exotic manufacturing techniques such as Laminated Object Manufacture (LOM) and Stereo Lithography are capable of manufacturing voids.

If a designer adds a further feature that then joins the void to the outside world (thus removing the void) then it may be that the first feature was not a design error.

A question therefore arises; should the designer be warned instantly that a void has been created, or should the test be postponed until the designer signifies that tests should be made?

Consider also a case where the original workpiece from which the machining is to be performed already contains a void (maybe it has been welded together), uncommon but not beyond the bounds of reason. Here it is not desirable to flag an error because of the existing void, but it is desirable to flag an error if either another separate void is added or the feature is added to the existing void changing the geometry of the void (again something not possible with traditional machining techniques). An acceptable modification of a void, however, is if a feature penetrates into the void and that feature is in itself machinable.

A two stage approach is offered to this problem.

During the design stage any feature that is added that increases the number of voids in a component is flagged as such. This catches the majority of problems. In Figure 39 it is clear the designer intended to put the hole in the bottom of the component (a), but placed it incorrectly (b). A drawback of this technique is that if there had already been a pocket with a similar position but in the top of the component (c), then the mistake would not have been detected by void detection. It would however, be detected by access problem detection (see 4.5.3). This facility has proved invaluable in the system catching a number of design errors, particularly in parts designed before interactive graphics and surface rendering were available in the system. It does not detect features that are added to a void after it has first been created, either by being part of the initial workpiece or after a void creating feature has been accepted.

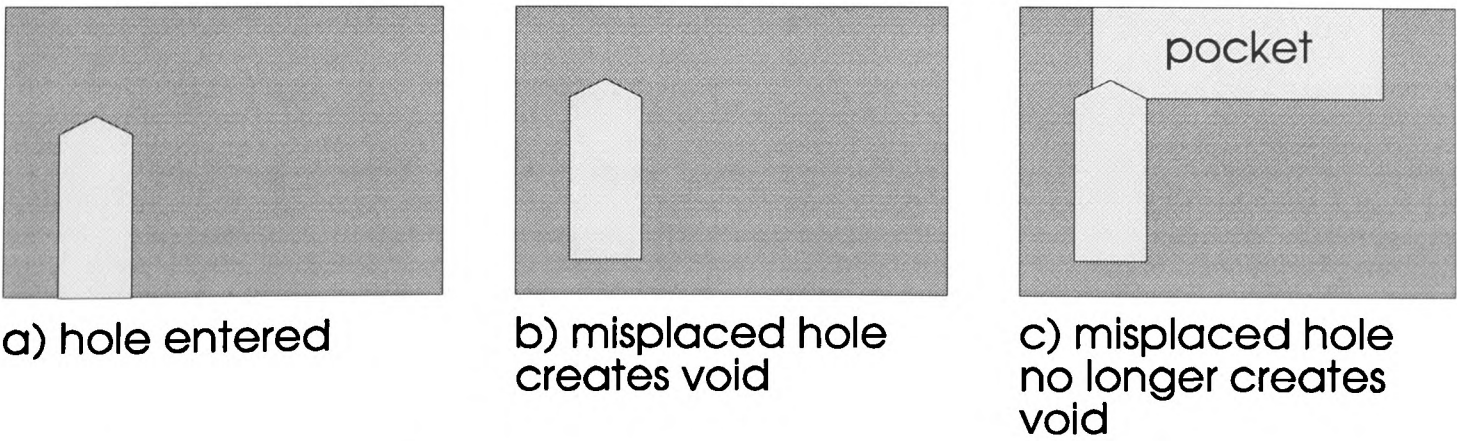


Figure 39 Void Detection

The second approach is a more rigorous void detection approach that is less suitable during incremental design and more suitable for validation of an entire design. The designer could choose to invoke this routine, either at the end of the design process, or at some time during the course of the design when he feels it desirable to check for validation errors.

Again, there are two possible mechanisms. The first involves creating a positive volume exactly filling the void. This is achieved by taking a copy of all shells apart from the outer shell and reversing the sense of all the faces. This produces (with a little manipulation) a solid body exactly filling the void.

Each feature volume can now be intersected in turn with the void volume, and any intersection volumes highlight features causing the void.

An alternative technique relies on the fact that all faces of all features are tagged with the feature ID (in terms of the Node ID) in the feature tree. By examining the attributes of all the faces of shells apart from the outer shell, a list can quickly be formed of all the features touching the shell of the void. In principle some features not touching the shell of the void in the finished component could have been included. By removing the list of features touching the void shell, however, and recomputing the component and then reapplying the test all features causing voids can be removed.

In the rare event that a feature has been included that sits inside the void of a blank component, this would be detected in two other ways. Firstly, it would fail the feature presence test, and secondly, it would certainly fail the component-access test.

The algorithm uses two special functions

numshells(x) returns the number of shells in a solid body *x*.

solid_from_shell(x,n) returns a solid body made by copying shell *n* of solid *x*, (creating a solid that is a lone void) and inverting the sense of all the faces (creating a solid that exactly fills the void). Shell(0) of a body is the outer shell, and all other shells are void shells.

Once the algorithm has made a pass of the body, all void features recognised must be dealt with by the user and then the algorithm must be rerun. It is necessary that the

user decide what action to take for each void creating object. Were it not for this then void creating features could be deleted from the model on each pass and the algorithm automatically rerun until no more voids are discovered.

In practice, it is unlikely that more than one pass would be needed.

```

if numshells(C) > 1
  then
    S = solid_from_shell(C,1)
    for i = 1 to n
      if  $N_i \cap S \neq \emptyset$ 
        then assert( i " creates a void")
      endif
    next i
  endif

```

Figure 40 Void Detection Algorithm

For most of these algorithms purely volume based reasoning has been used, avoiding use of lower dimensional constructs. This has been predominantly to reflect the fact that machining operations are in themselves 3D operations involving material removal. In this void detection algorithm, some 2D inquiries are used.

4.5.2 Feature Presence

Looking for similar sorts of design errors to those discovered with void detection, the accidental misplacement of a feature, Feature Presence checks that all features have some intersection body with the initial workpiece in order to ensure that at least part of each feature requires the component to be machined.

This is somewhat easier to detect than voids. Features that are not present in the component should be flagged to the designer and the requirement that they be planned should be switched off. They may exist as a consequence of the design method in that a pattern of holes could be bored into a component even if the pattern extends beyond the component's boundary so they should not be considered an absolute design error. Feature presence checking is illustrated in Figure 42.

```

for i = 1 to n
  if  $N_i \cap B = \emptyset$ 
    then assert ("Feature " i " fails presence")
  next i

```

Figure 41 Feature Presence Algorithm

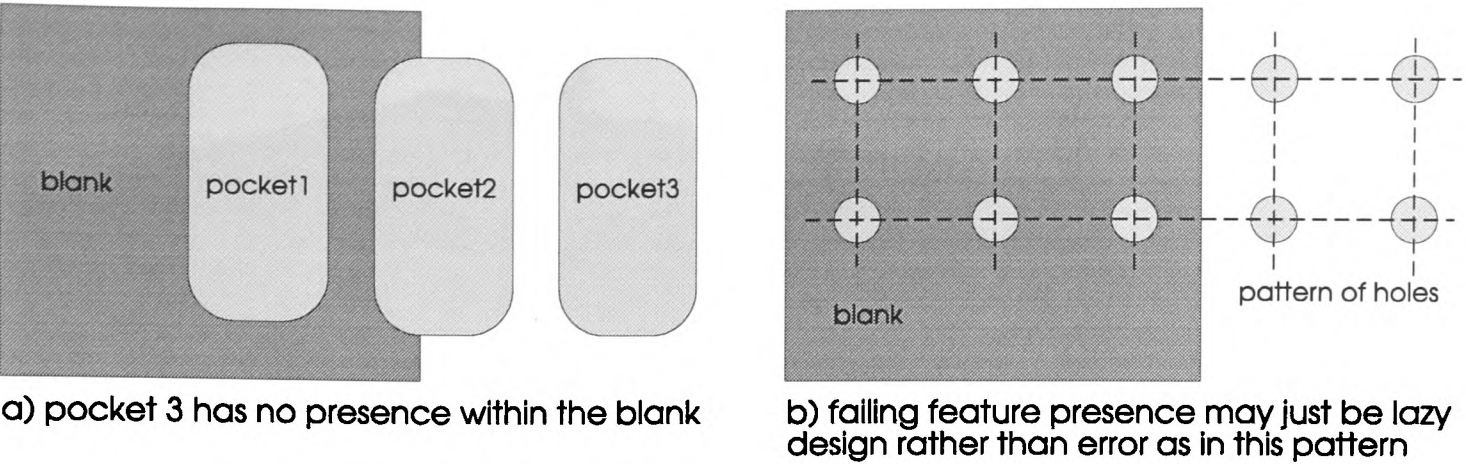


Figure 42 Feature Presence Detection

Any part of a feature that intersects with the blank qualifies that feature as present within the blank. This is again in line with Kramer’s issues regarding MRSEVs [Kram92].

4.5.3 Access Problem Detection

In this section, some assumptions are made about features that do not hold true in the general case. For instance, a Principal Access Direction (PAD) is assumed and performs access checking making the assumption that the feature is to be machined from this access direction. This is often true, for instance, most holes will be machined from above. However, through holes could be machined from either end. Similarly, it is assumed that a slot will be milled from above, but in certain instances, where the placing of a slot results in a step, there may be four access directions, and to be truly certain that all possibilities have been investigated the six orthonormal access directions must be checked. In Figure 43, there are two inaccessible directions for the slot, into the component on the left of the slot and the base of the slot, but there are four possible access directions including the Principal Access Direction vertically upwards.

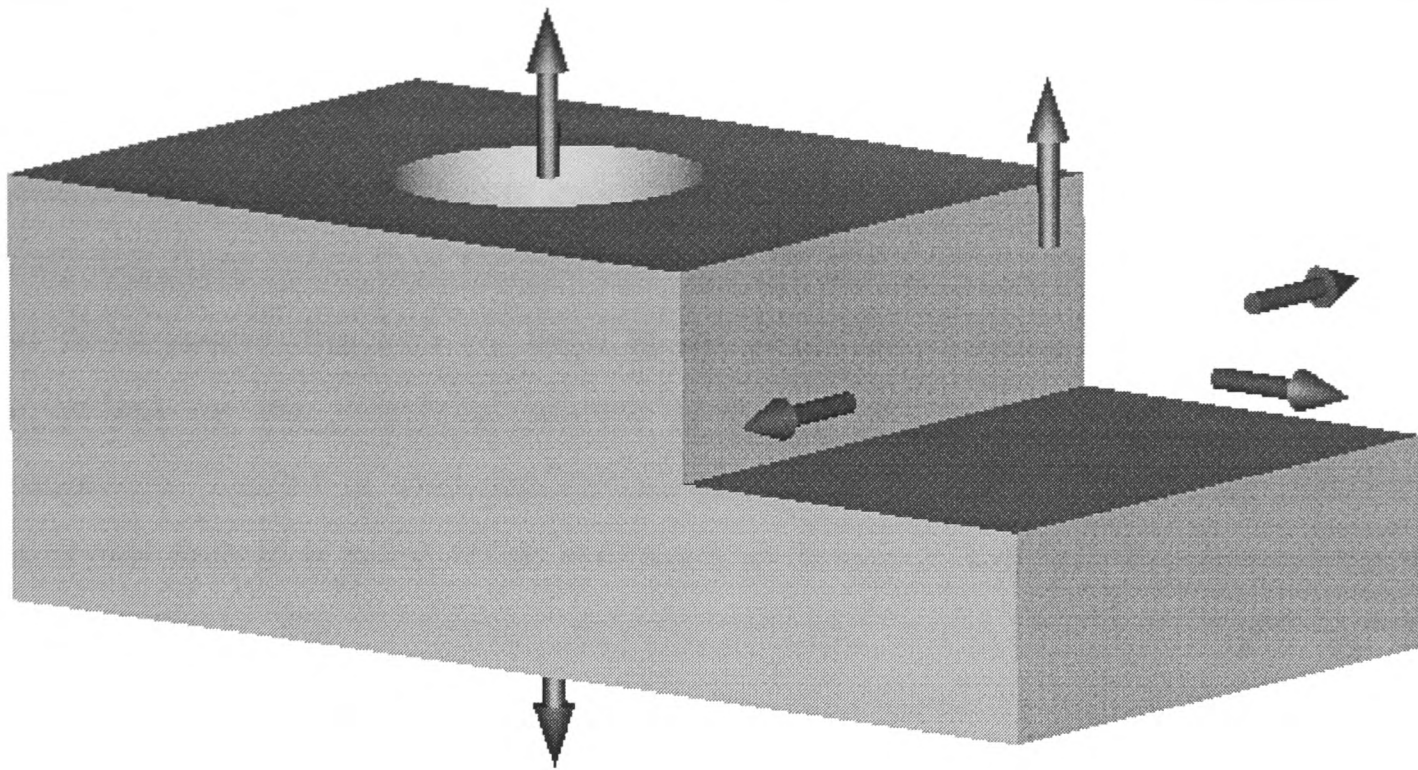


Figure 43 Possible Access Directions for Slot and Hole

Each feature in the model has a principal access direction. This in particular is reflected by the tool-profile and cutter-path model of the features in FODDS2, reflecting the manufacturing feature bias. Many features, however can be accessed from a number of independent directions as illustrated in the case of the step and hole above. Of particular importance is the access direction antiparallel to the principal access direction. This particularly affects through holes, and can change much about how the hole is viewed. A through hole can be machined from two possible setups, and, because it has a simple cylindrical profile, it has no *endtype* associated with it, freeing up more yet more machining possibilities.

Gindy [Gindy89] regards features as having up to six External Access Directions (EADs) but this is only true of components where machining need only occur along the principal axes (whether in the positive or negative directions). In fact there is a 2D space (around the surface of a sphere) of possible approach directions and all these are possible through different setups even when machining is restricted to 2½D.

Examining the step in Figure 43 above, it can be seen that for two of the access directions (+y and -y), a rotating cutter would leave a filleted radius at the juncture of the two machined faces. A slab mill would remedy this. The feature model again

does not allow for slab mills, and so effectively allows only two approach directions, from the +z or from the +x direction.

There is no general (i.e. feature independent) way of telling if machining is possible from directions other than the principal access direction (+z local to the feature) or the blind-access direction (-z local to the feature), all other access directions are dependent on the individual feature geometry and its relationship between the component. Indeed, it also depends at what point during the machining process a particular feature is going to be machined, as it may depend on the current state of the component. For this reason only the principal access direction (PAD) and the blind access direction (BAD) are considered.

Over the years, a number of different ways of detecting access problems have been used. Only recently, now that solid modellers are widely available and robust can true access checking take place. Other process planning systems have relied on access checking to be done manually [Husb90] or for simple vector-based testing to be performed (see Figure 44)[Josh87]. Simple vector based testing produces the correct answer in the majority of cases but cannot be relied upon to always be correct.

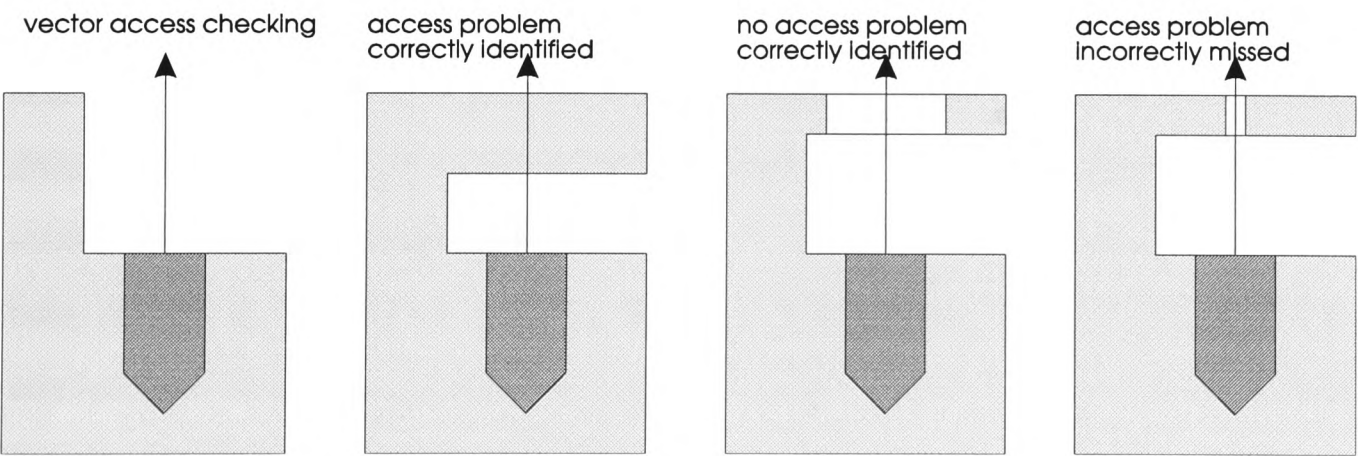


Figure 44 Problems using Access Vectors

FODDS2 uses 'solid' access bodies and shows that a simple extrusion is again insufficient for true access testing. Figure 45 shows how access body types for a simple hole have evolved throughout the project.

Firstly, the vector based checking previously discussed is shown. Many of the problems that vector based checking introduces can be resolved using the simple access body of b). This and all subsequent bodies can be imagined to have a semi-

infinite length, that is, they extend from the surface of the feature away from the feature to infinity. In practice, they have been modelled as bodies whose length is guaranteed to be substantially longer than the maximum distance across a component.

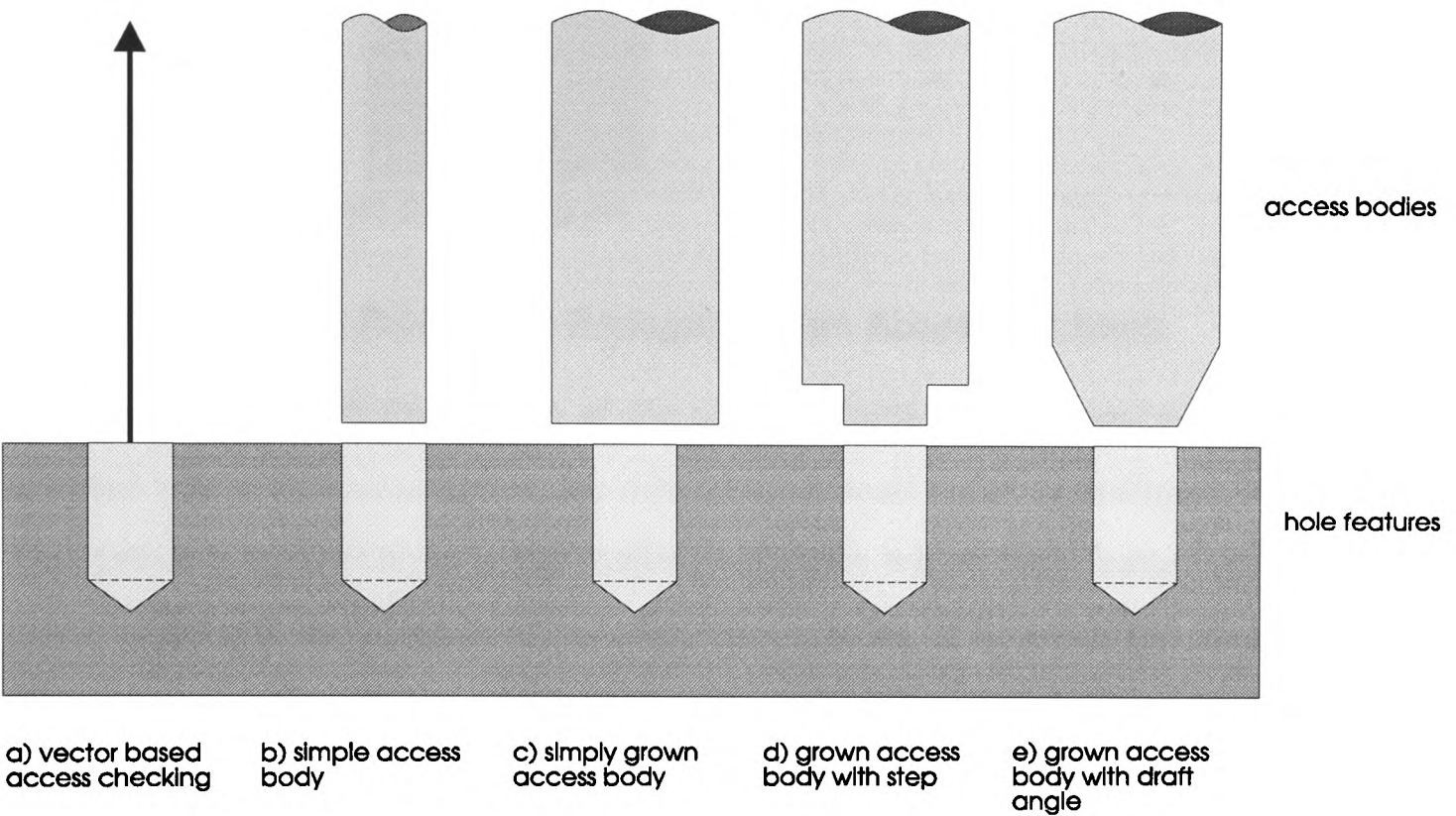


Figure 45 Evolution of Access Body Types

The body in b) does run into some problems however. Consider the case in Figure 46 below (a case similar to Figure 44 discussed previously) where there is an attempt to drill a hole through an existing hole of similar size. If an access body of type b) is to be used and the hole through which it is to be drilled is of equal radius to that of the new hole that is to be drilled then no access problem will be detected, as there is no intersection of the simple access body with the simple hole (in fact, a surface of intersection may result however this has zero volume and regularising the result will dispose of it). In reality, however, this is an access problem because in any real situation there will be tool waver and the surface finish of the upper hole may be detrimentally affected.

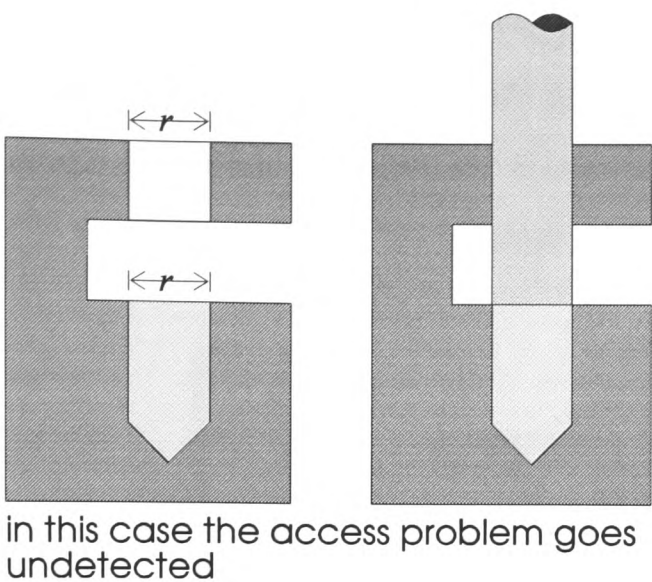


Figure 46 A Problem in Recognising an Access Problem

In order to deal with this failure of the simple access body, a new access body was adopted where all access bodies are dilated by a small value as in Figure 45 c) above. This presents no problems in the majority of cases where both feature and blank are axis aligned and the blank consists of planar surfaces. If however the feature is to be placed at an angle into the surface then this access body produces a spurious access problem (Figure 47 below). There are of course other problems involved with machining into angled surfaces.

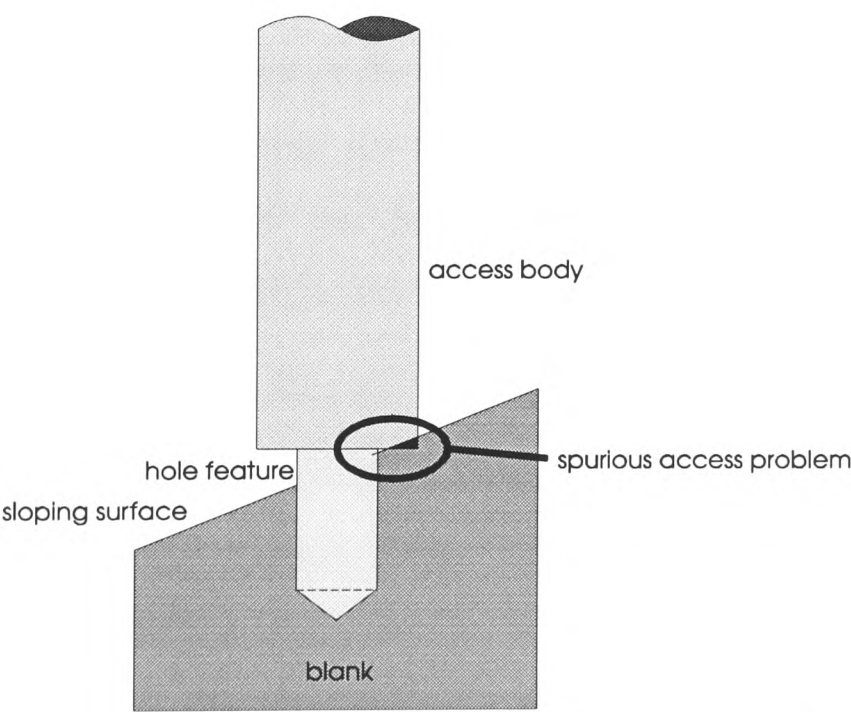


Figure 47 Problems with a Simple Grown Access Body on an Angled Surface

The problem with a sloping surface can be resolved by using the access body of the Figure 45 d). This form will work on slopes of up to 45°. Very few machining operations can exceed this slope without some secondary feature to produce a locally

flat feature. Type e) with a sloped surface to the grown portion and a draft distance of twice vertically the horizontal grow distance is, if nothing else, a smoother elegant shape and leads to a simpler access problem shape (Figure 48 below). It also leads to the problems of local and global access problems.

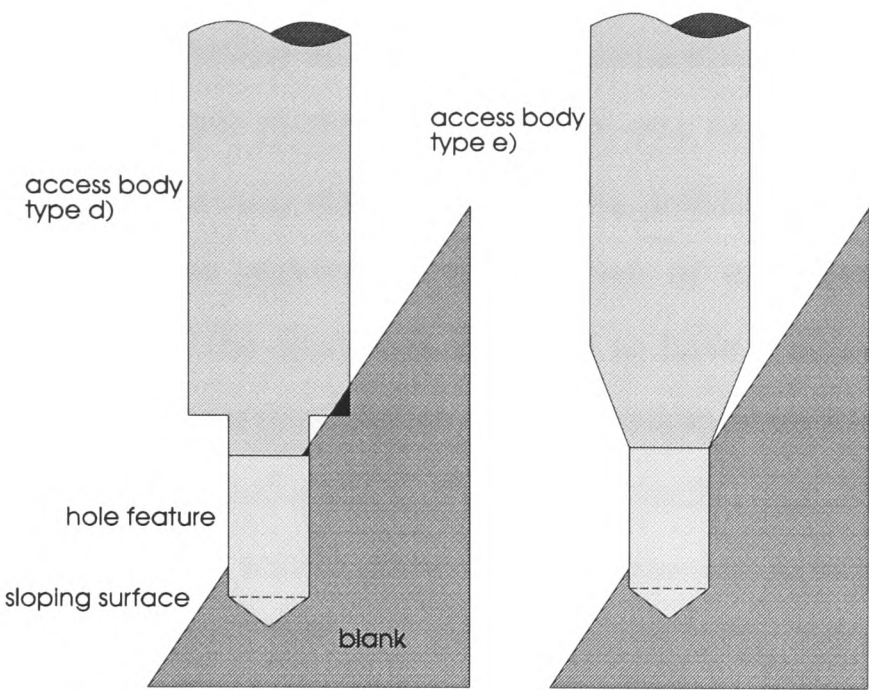


Figure 48 Access body e) is less prone to Problems of Angle than Type d)

The value of δ (delta) must be identified. Delta is the clearance required between the tool and any part of the workpiece that should not be machined. This value is a worst case value, erring on the side of caution, but it is also a value that would vary considerably if the processes and machine/tool combinations were known in advance. Unfortunately this is not the case.

An estimate at the radius of the tool might be obtained, knowing the corner radius of a pocket, the width of the slot or the radius of the hole. Then δ can be specified to be some small fraction of this, such as 5%. But if small holes, drilled using a simple bench drill, are considered then the accuracy might be larger than 5% (consider an M2 hole, 5% would give a required accuracy of 0.04mm, unlikely with a bench drill). So a lower limit of 0.25mm could be introduced. This gives a value of δ given a typical radius r of:

$$\delta = 0.05 \, r \, (\text{mm})$$

if $\delta < 0.25$, $\delta = 0.25$ (mm)

The angle of a feature to the surface in which it lies is an important machining consideration, but it is not a question that can be pursued before some machining process has been selected.

On examination of some real components, a number of cases have been discovered where nested pockets would share one or more sides. With a type d) or e) access body this would result in an access problem. Any access body whose cross section exceeds the maximum cross-section of its parent feature will result in an access problem if the feature is intended to have a sidewall flush with the sidewall of the parent feature (see Figure 49). So, pragmatism has forced a return to an access body of type b). Performing two access checks, one with a type b) body and one with a type e) body would allow further reasoning to be performed in this scenario. If there was no access problem with a type b) body, but an access problem was discovered with a type e) access body, then this relationship would be added between the two features in question thus providing a hint the process planner to manufacture the two features in the same setup.

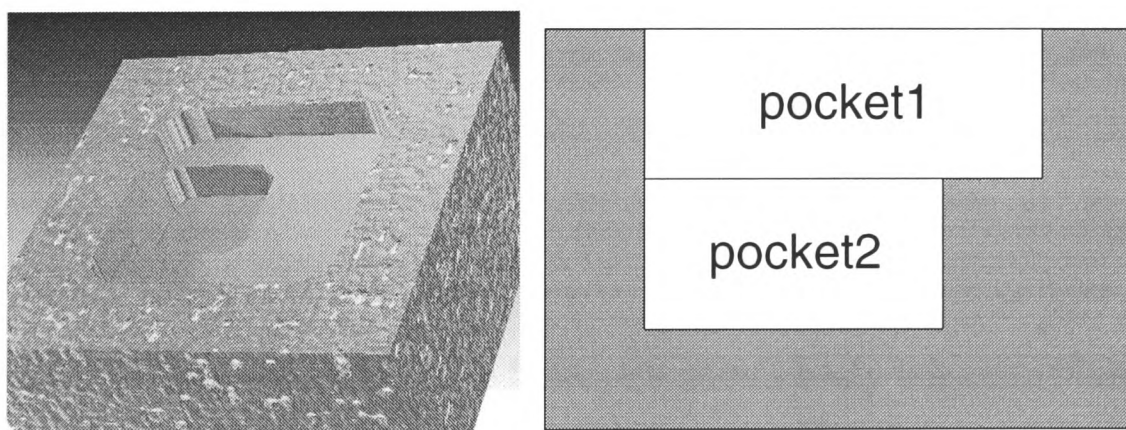


Figure 49 Nested Pockets with a Coplanar Sidewall

Having discussed the geometry of the access body, the uses to which the access body can be put and the information that can be obtained will now be discussed.

The access body represents the volume that the tool would have to travel through in the manufacture of the feature. It does not represent the chuck of the tool and it is assumed that the chuck and the rest of the machine would always be clear of the component. This again represents the path of least commitment. Which machine any particular feature will be machined on is still unknown, so it is unrealistic to

constrain the choice of machine by using a more complex access body. Admittedly a least committed access body generated from an examination of our chosen machines could be used, but the disadvantages in terms of complexities of the resultant bodies outweighs the gain. It does however demonstrate the need for simulation of a process plan after a process plan has been generated to highlight particular problems that this early geometric reasoning cannot hope realistically to pick up.

Access problem detection is performed in three ways. For each feature the following tests are performed:

- Intersection of the feature's access body with the component
- Intersection of the feature's access body with the blank
- Intersection of the feature's access body with all other features

Each of these three are discussed in turn.

4.5.3.1 Intersection of the feature's access body with the component

If the result of the intersection of a feature's access body with the component is not null then the feature cannot be manufactured from that direction. If this test is performed first and the result is not null the other two tests do not need to be performed. This is because the second test is guaranteed to fail as the blank completely encompasses the component and performing the third, most expensive test would be redundant.

4.5.3.2 Intersection of the feature's access body with the blank

This test is far cheaper (computationally) than the third test, and if passed cleanly obviates the need for performing the third test resulting in a considerable saving in computing time. It has been observed subjectively from examination of test components, that more than half of the features on any component are normally machined on the surface of the blank. That is nested features only constitute a fraction of the features to be machined, though often the most interesting fraction. If those parts that lie on the surface can be cheaply identified then the additional test involved ultimately saves processing time. It is not possible to tell whether a feature is on the surface of a component without performing such a test, particularly because

the workpiece may not have simple planar geometry, and a feature is not necessarily aligned with any surface of the workpiece.

In this test, the access body of a feature is intersected with the blank. If the resulting intersection is null then that access to that feature is available regardless of its position in the process plan. Subject to other constraints such as tolerance relations, this feature can be machined at any time. If this is the case then the third test is not required, and substantial processing time is saved.

4.5.3.3 Intersection of the feature's access body with all other features

From the first test, it is known that there is access to the feature provided all other features have been machined. The second test infers that the feature is only accessible after certain features have been removed. The purpose of this third test is to identify which features must be removed prior to the machining of the feature in question. The access body of the feature in question is intersected in turn with all other features and the results of these intersections are added to the database as anteriority constraints. These tests are summarised in the algorithm of Figure 50 below.

```

for i = 1 to n
  if  $A_i \cap C \neq \emptyset$ 
  then assert ("feature " i " is inaccessible")
  else
    if  $A_i \cap B \neq \emptyset$ 
    then // more tests required
      for j = 1 to n
        if j  $\neq$  i
        then
          if  $A_i \cap (N_j \cap B) \neq \emptyset$ 
          then assert(i"has access problem with "j)
          endif
        next j
      endif
    endif
  next i

```

Figure 50 Access Problem Algorithm

The simple case of a triply nested slot is considered (see Figure 51). The slot at the top with global access is slot1 with slots 2 and 3 being machined in the bases of slots

1 and 2 respectively (that is, machining is not to be started from the top surface of the component but from the base of the higher slot).

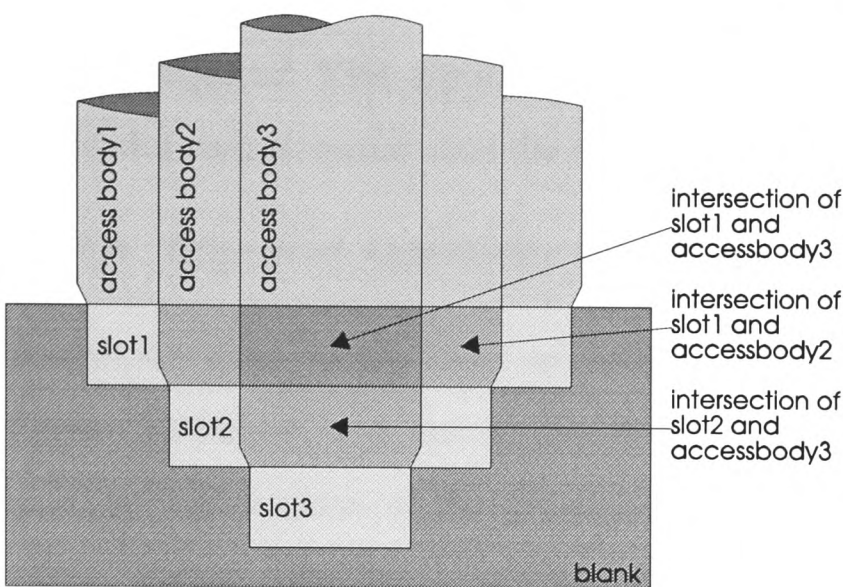


Figure 51 Access Bodies and Anteriority Constraints for the Triply Nested Slot

The anteriority constraints will be successfully inferred in the above case. Consider, however, the case of Figure 52 below. There is a through hole running horizontally through two 'lumps' and passing through the top slot of a nested slot pair. The simple access problem detection algorithm gives the result that the bottom slot must be machined after both the top slot and the hole. This runs contrary to a quick human analysis of the problem where it can clearly be seen that it is only necessary to machine the top slot before the bottom slot, and the hole, though possibly having anteriority constraints of its own does not affect the bottom slot.

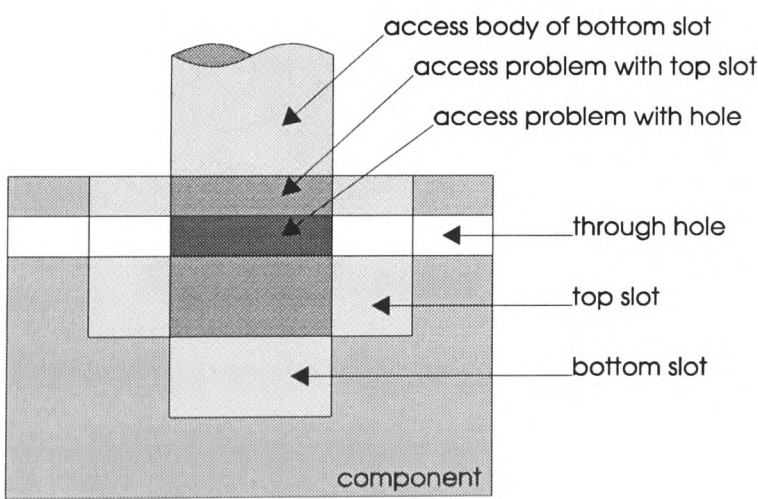


Figure 52 The Slot and Hole case

Considering this case more carefully it is discovered that though an access problem between the bottom slot and the through-hole has been posted to the database, this

access problem body is completely subsumed by the access problem body between the bottom slot and the top slot, so the anteriority constraint between the bottom slot and the hole can be neglected. Nevertheless, an algorithm to check for this sort of case is required. This algorithm must also produce the correct result in the earlier triple slot case to ensure that the two sets of requirements do not conflict.

4.5.3.4 Improved Anteriority Algorithm

A simpler case where such an anteriority check must be performed is shown in Figure 53 below, and indeed the above algorithm does reveal that the hole can be machined after *either* of the two slots has been machined.

Figure 53 shows two crossed slots (slot1 and slot2) and a hole in the base of the intersection of the two slots.

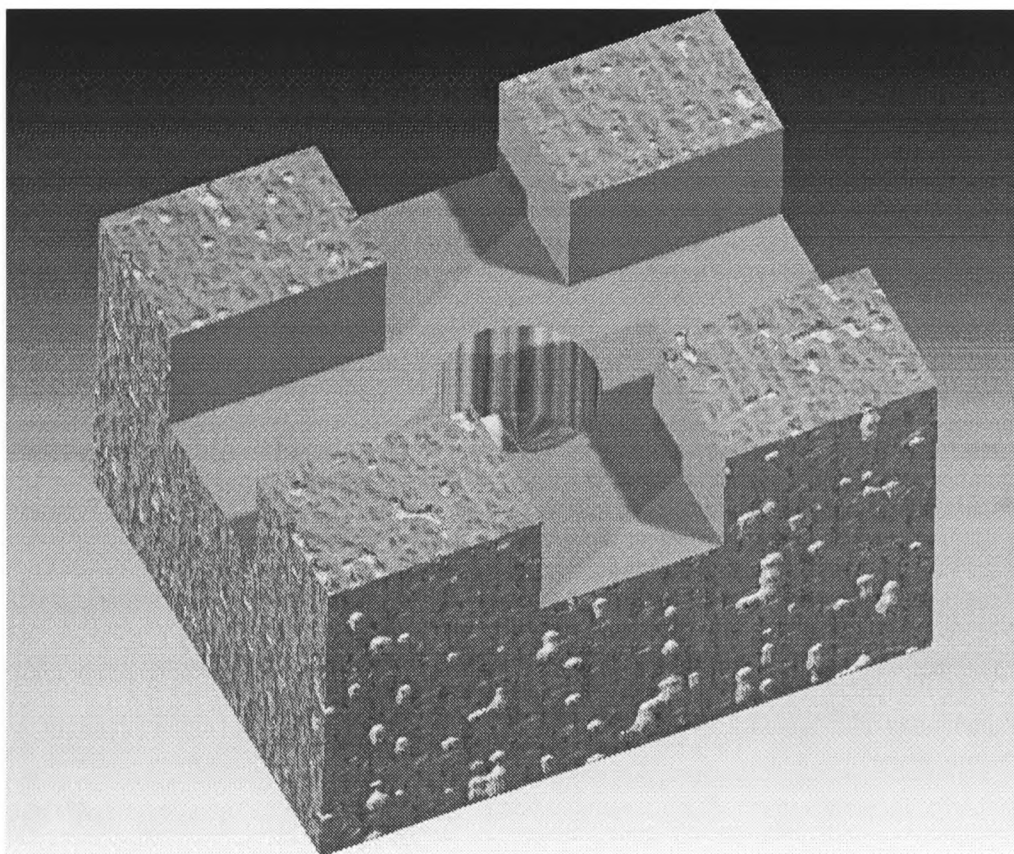


Figure 53 Crossed Slots with Hole

Performing access checks gives the following information:-

```
hole1-access intersects slot1.  
hole1-access intersects slot2.
```

In terms of anteriority constraints this could be reinterpreted as:-


```
hole1 machined_after slot1.
hole1 machined_after slot2.
```

However, it is obvious by inspection that in fact the requirement is weaker than this and is in fact:-

```
hole1 machined_after (slot1 or slot2).
```

The question now arises, what geometrical analysis can be performed to reveal this weaker piece of information. Observing the intersection of the hole1 access body and the blank, shows that this body is entirely contained within the intersection of the two slot features:-

```
hole1_access ∩ blank = (slot1 ∩ slot2) ∩
(hole1_access ∩ blank)
```

The question then remains how a process planning system can model the 'or' constraint. The HAPPI system was not capable of modelling such a constraint.

An algorithm to identify and evaluate these cases follows in Figure 54. Note the existing database of relationships of the type "*has access problem with*" is used, and where necessary the relationship "*machined_after*" is added to the database, where this is taken to mean, "*must be machined after*". Additionally, cases of 0,1 or 2 access problems are considered. In the event of more, all pairwise combinations of partners must be considered. There are extreme cases where even a pairwise consideration is insufficient, though these are sufficiently unlikely in real components as to be comfortably ignored. (In the algorithm in Figure 54, x , $x1$ and $x2$, are partners in the relationship that are only instantiated upon answering the *numfacts* query. This query will return all items in the database that match the body of the query with x as a wildcard. The algorithm is thus slightly simplified for clarity.)


```

for i = 1 to n
  if (numfacts(i " has access problem with " x)=1
  then assert(i " machined_after" x)
  endif
  if (numfacts(i " has access problem with " x)=2
    //assume partners are now features x1, x2
  then
    if ( $A_i \cap N_{x1} = A_i \cap N_{x2}$ )
    then assert(i "machined_after (or " x1 x2 ")")
    endif
    if ( $A_i \cap N_{x1} \subset A_i \cap N_{x2}$ )
    then assert(i "machined_after " x2)
    endif
    if ( $A_i \cap N_{x2} \subset A_i \cap N_{x1}$ )
    then assert(i "machined_after " x1)
    endif
  endif
endif
next i

```

Figure 54 Improved Anteriority Algorithm

4.5.4 Through Feature Detection

Up to now, one access direction has been considered for each feature. Holes are frequently through-holes, and the option of two access directions can remove a large number of constraints on the subsequent process plan. To this end, for each feature, an access check is performed with a *blind* access body. The blind access body is that body which would intersect with a component in the event of a blind hole. Often this will reveal alternate access directions for other features, though it will not infer access directions in the 'ends' of slots or the 'sides' of pockets. This information is not currently used in conjunction with the other access checks.

```

for i=1 to n
  if  $B_i \cap C = \emptyset$ 
  then assert(i " is a through feature")
  endif
endif
next i

```

Figure 55 Through Feature Detection

This algorithm can be extended in much the way that the previous access algorithms were. Note that the blind access body extends downwards from the top of the feature. This is because all currently implemented features are guaranteed to have their

maximum radius at the top of the feature and may narrow lower down. To ensure that features that narrow below the top of the feature it is important to check blind access through the feature itself.

4.5.5 Proximity Detection

An area of particular interest to the author, detecting proximity problems or thin walls appears trivial. Upon closer inspection, it becomes remarkably complex.

Firstly, it is necessary to define what is meant by a thin wall.

The chief reason for detecting a thin wall in this domain is the possibility of that wall rupturing due to machining forces during manufacture. It is sensible to assume that the design passed to the geometric reasoning module is functionally valid, that is the design will meet its 'in use' requirements.

First, a naive method for the detection of thin walls is discussed, both the problems and the disadvantages. A more complete method is subsequently discussed.

Let the thickness below which a wall is declared thin be δ (not to be confused with the δ used in access problem detection). If it is possible to dilate all features by the distance $\delta/2$ then intersections of pairs of these dilated feature bodies will indicate thin walls. The problem of producing dilated bodies has already been covered in Chapter 3 and through the dilated tool profiles described in section 4.4 earlier in this chapter.

As with many of the other tests, the number of intersections to try is $\frac{n^2 - n}{2}$ where n is the number of features in the component. A routine has been written that cheaply tests whether the intersection of the bounding boxes of two objects exists. This radically reduces the number of full Booleans that have to be computed. The result of this test determines whether a full Boolean test is needed³.

³ Though one would imagine ACIS already performs this test before performing a true Boolean, the test seems to result in a significant speed up, possibly as fewer *entity:copy* functions need to be called (both by FODDS2 and ACIS).

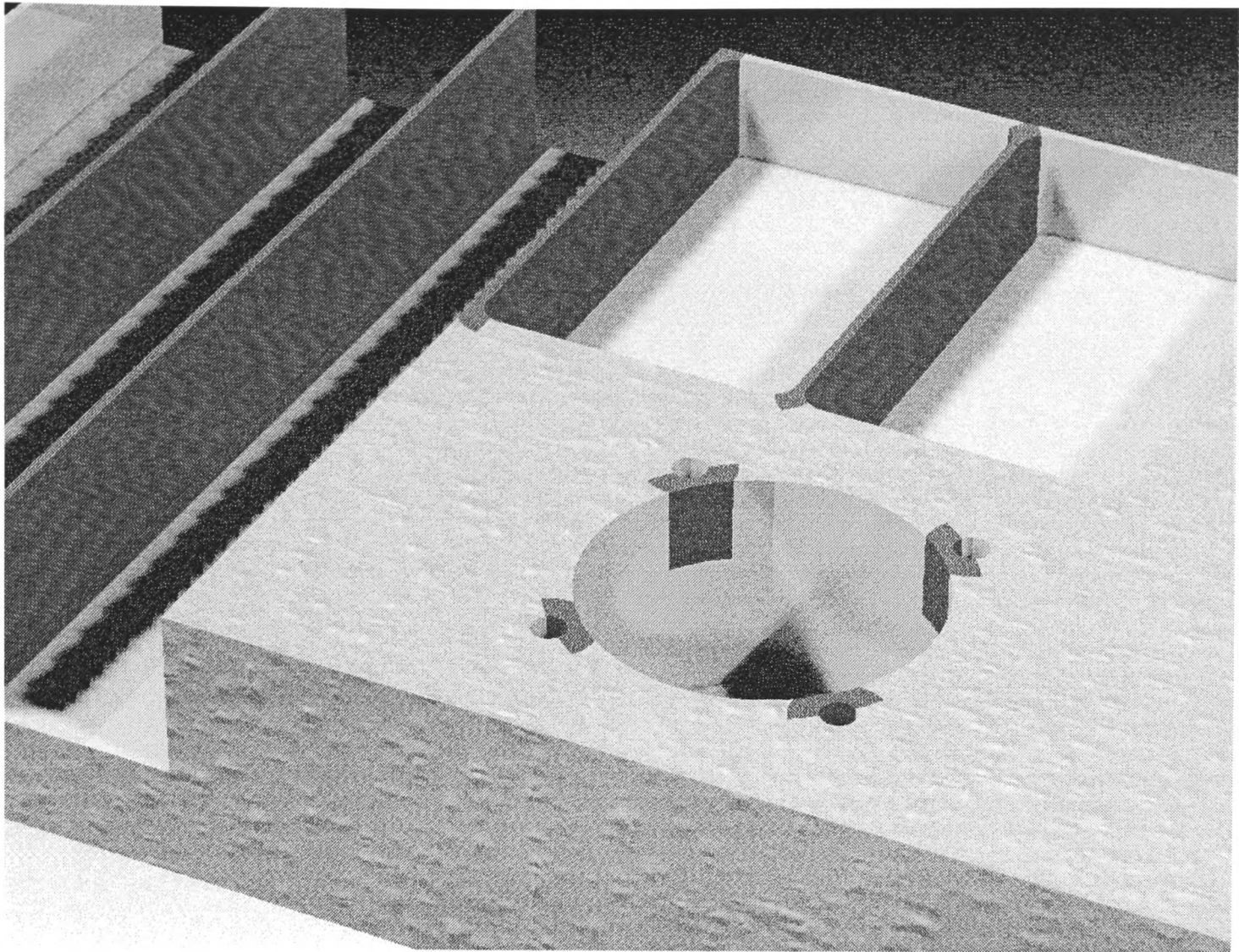


Figure 56 Thin Wall Detection

This technique has been implemented and is illustrated in Figure 56. Thin walls are highlighted and can be seen between the large hole and its surrounding holes and between the various slots and pockets.

The one major drawback that can be seen of this simple technique is that thin walls are not detected between the features and the outside world. This can be overcome for a range of cases if dilated feature bodies with a dilation of δ (twice that used previously) and check that no portion of the dilated feature lies outside the blank. First, though a check must be made that the original feature had no portion outside the blank. Any feature that already has some part of its volume lying outside the blank will necessarily have some portion of its dilated volume outside the blank, so a check for thin walls on these features will fail. More details of these problems are examined in the next chapter with the Thin Walls test piece.

Dilated bodies using the dilated tool profile have been dilated with a hemisphere in the $-z$ halfspace. This was to prevent dilated features whose top surface lies flush

with a face of the blank from having their dilated body protrude out of the blank except where they meet a thin wall criteria. This algorithm is shown in Figure 57.

Vandenbrande [Vand91] suggested a similar technique that he left unimplemented and the problems unresolved.

```
// the algorithm is in two parts
// feature-feature thin walls
// feature-universe thin walls
for i=2 to n
  for j=1 to i-1
    if (and
      ( $N_i \cap N_j \neq \emptyset$ )
      ( $D_i(\delta/2) \cap D_j(\delta/2) \neq \emptyset$ ) )
      then assert (i " has a thin wall with "j)
    endif
  next j
next i
for i = 1 to n
  if (and ( $B - N_i \neq \emptyset$ )
    ( $B - D_i(\delta/2) \neq \emptyset$ ) )
    assert (i " has a thin wall with the blank")
  endif
next i
```

Figure 57 Thin Walls Algorithm

This algorithm still has a number of drawbacks. If the insistence on volumetric solutions is dropped then a number of other approaches are open. Gupta's approach in section 2.5 [Gupt97] shows promise. Alternatively approaches using the medial axis of the component might be possible, though these two approaches may converge.

4.5.6 Intersection Detection

The detection of intersections between features is vital as almost all problems of interest when process planning derive from intersections between features. A component with a thousand distinct features is easy to process plan when compared with a component with two intersecting holes. Reasoning about the intersection between features can lead us to redefine features in order to simplify process planning, these are called alternate representations. The system evaluates all feature intersections and records them in the database for the information of downstream

packages using the algorithm in Figure 58. With the exception of those intersections used for anteriority evaluation, the current system does not use this information.

Applications for this information include the possibility of simultaneous multi-feature machining and feature redefinition.

```
for i=2 to n
  for j=1 to i-1
    if  $N_i \cap N_j \neq \emptyset$ 
      then assert (i intersects j)
    endif
  next j
next i
```

Figure 58 Intersection Detection Algorithm

4.5.7 Hole Interference Detection

Hole interference detection is used to detect whether features, or indeed the blank, will cause problems for the machining of holes.

This class of problems is normally not a significant issue with slots or pockets, so it is necessary first to isolate what is special about holes that causes this to be a problem.

This algorithm only applies to features machined solely with an approach/retract movement.

The problem arises when the approaching hole end contacts an incomplete or angled surface. To successfully machine a hole, the intersection of the entry face of the component and the cutter must be a complete circle of the same radius as the hole. This must be true if the hole exits and re-enters material during machining. In the limit this is true for exit faces, though not usually so critical. The problem of constraining the exit face is neglected here.

Firstly this condition must be checked for the hole and the blank. If it does not some feature must be machined before the hole in order to leave the entry surface in the required condition.

Then all other features that intersect with the hole must be checked to ensure that the intersection body of the hole and the feature meet the requirements. An algorithm that handles hole interference detection is shown in Figure 59 below.

For simplicity the following functions are assumed:

hole_access_interaction(body, vec, radius) that given some solid body, will return true if that body has a planar face whose surface normal is parallel to *vec* and that will contain a circle of radius *r*. (note that a surface normal antiparallel to *vec* is not acceptable).

hole?(i) returns *true* if negative feature *i* is a hole, *false* otherwise

A Hole Access Interaction problem is referred to as an *HAI* in the interests of brevity. So an *HAI* between a hole and a feature means that the hole cannot be machined after the feature if that feature has been removed first (and possibly not even then).

```
for i = 1 to n
  if hole?(i)
  then
    if hole_access_interaction(B  $\cap$  Ni)
    then assert("hole " i "has a HAI with the blank")
    endif
    for j = 1 to n
      if (Ni  $\cap$  Nj)
      then
        if hole_access_interaction(Ni  $\cap$  Nj)
        then assert(i "has an HAI with" j)
        else assert(i "has no HAI with" j)
        endif
      endif
    next j
  endif
next i
```

Figure 59 Hole Access Interaction Algorithm

The problem with this algorithm is that it is not clear how to spot how one feature might resolve a HAI problem a hole has with another feature or the blank. Consider how a feature is often used to provide a flat spot on a blank to use as a starting surface for a feature. For this reason, when a feature that intersects with the hole does not have a problem with the hole, this fact is asserted. It is this second class of feature that can be used to provide flat spots to start holes. This then leaves the

problem of analysing whether the problem with a feature lies with the top surface or the bottom surface. HAIs are then left as hints for the designer.

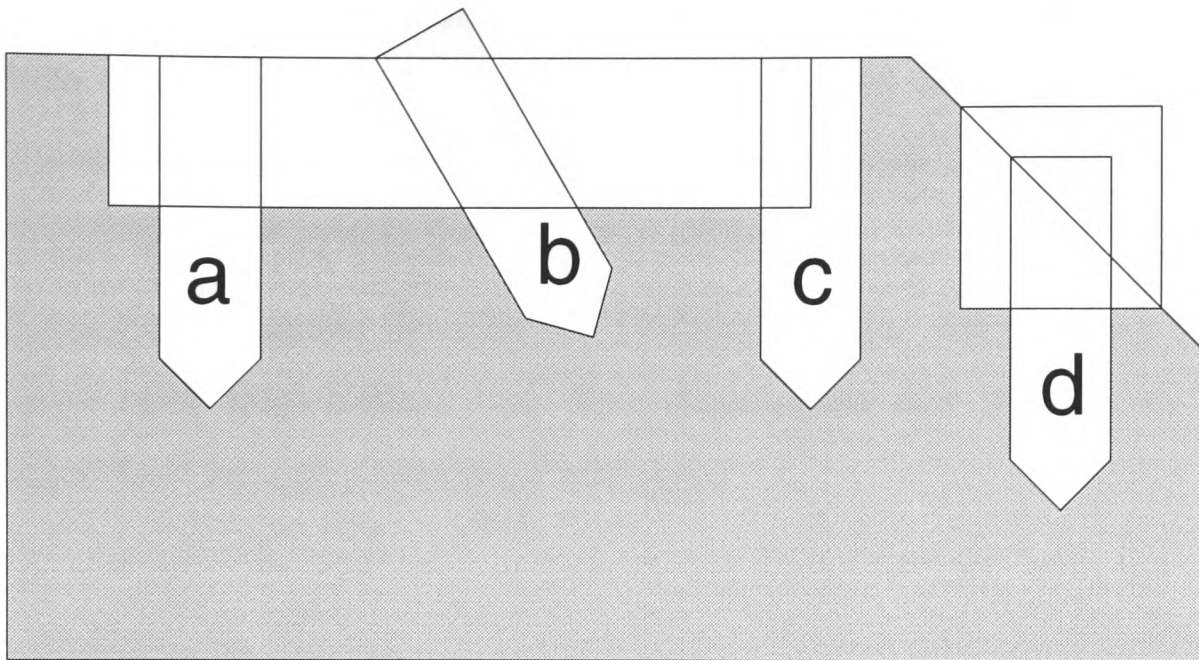


Figure 60 Problems with Hole Access Interference

From Figure 60 it can be seen that hole a) has no problem either with the blank or the pocket feature. Hole b) has a problem with both the blank and the feature. Hole c) has a problem with the pocket, but not with the blank, so can be machined first. Hole d) has a problem with the blank but not with the cut out, so can be machined after the cutout, but only because the cutout intersects with the problem area on the blank.

In general, if any feature exists that creates an HAI, then a feature not causing a HAI that intersects with the entire volume of the entry face of the HAI must also be machined before the hole is machined.

4.5.8 Alternate Access Direction

Potential Alternate Access Directions (AADs) lie normal to both the Principal Access Direction (PAD) and any straight edges forming the cutter path. Evaluating AADs requires feature redefinition and is beyond the scope of this thesis. An alternative approach might be to relax the geometry of features in the first place as in [Mant89].

4.6 Summary

This chapter has introduced a description for volumetric material removal features that allows all subsequent volume oriented geometric manufacturability algorithms to

operate in a general way on all feature types. Algorithms have been presented for a number of important validation tests. Most importantly, novel algorithms have been presented to generate anteriority constraints between features and for detection of hole access interference problems. The algorithms do not check for any alternate access directions with the exception of the opposite direction for holes. This is a drawback of the system that should be addressed.

Subsequent chapters describe the FODDS2 system within which these algorithms have been implemented and then demonstrate the validity of the anteriority algorithms through a number of test cases.

5 Feature Oriented Detail Design System

This chapter introduces the Feature Oriented Detail Design System 2. It starts with a review of the user requirements and translates those into system requirements.

The system design decisions are reviewed in light of Shah's requirements for a feature based design system.

It is important to realise that unlike most CAD systems, designed for ease of use of the designer, FODDS2 has been developed for two distinct classes of user. Firstly, there is the human designer, but secondarily, and arguably of more importance are the downstream applications, in particular the CAPP system.

5.1 User Requirements

There are two classes of user associated with the FODDS2 Feature Based Design System. Firstly there is the designer at the 'front-end' of the system. The majority of CAD systems are geared towards the user as designer.

Secondly, there are the downstream applications, in particular CAPP and CAM systems. These also have a set of requirements, at least of equal importance with the front-end user.

FODDS2 attempts to satisfy both classes of user.

5.1.1 Requirements of the 'Front-End' User

In order to satisfy two sets of users, some compromises may have to be made. The system should still allow design to be a straightforward task.

The user of a Feature Based System requires a system that allows rapid design of components and good editability, so mistakes and refinements of design can be easily made. The user also requires that resulting designs are valid (according to some set of rules) and of acceptable quality for downstream operations. In particular this means to the designer that realistic tolerancing information can be added to a design and that the designer is warned about some potential problems in downstream processing. The user requirements can be summarised below.

Must allow relatively simple component creation

Must allow straightforward editing of the component.

Should flag designer of a variety of potential CAPP and CAM problems.

5.1.2 Requirements for Downstream Applications

For process planning, and in particular for process planning without an integral solid modeller, the following information is important for the process planning system.

Dimensions, feature type and in particular principal access direction of all features requiring manufacture.

Feature validation. All features must play a part in the final component, and all features must be (at some time) accessible to be machined on the component.

Anteriority constraints between features. That is information regarding implied ordering relationships that *should* hold in order for a successful process plan to be generated.

Tolerance constraints between features. In order that, for instance, the process planner can endeavour to have those features with tight tolerance constraints planned in the same setups.

This set of requirements is sufficient for some process planning tasks. Without solid models of workpiece and features it is still inadequate for certain other tasks in particular fixturing [Chia97a].

As the feature based design system is also just the front-end to a number of additional downstream computer aided activities, such as process planning, but additionally Finite Element Analysis (FEA) and fixture design as well as other more exotic analysis techniques such as Computational Fluid Dynamics (CFD) packages, it is necessary to cater in some way for these downstream activities. In particular this will require that the exporting of models in suitable formats be sufficiently straightforward. With respect to CFD and FEA analysis, FODDS2 can export ACIS '.sat' files, a *de facto* standard. Using external software incorporating ACIS such as CADfix, Bentley Microstation, or SolidEdge, ACIS models can be converted to any

of a number of standards. Additionally, ANSYS, perhaps the leading finite-element system, has an add-on to accept ACIS '*.sat*' files. FODDS2 feature models are also saved in Access database format ('*.mdb*' files).

5.2 Shah's Feature Based Design System

Characteristics

[Shah95] identifies the following important characteristics of a Feature Based Design System

- Representation of Feature Definitions
- Level of Support of User-defined features
- Type of the linkage with a Geometric Modelling System
- Application Context
- Support for Feature Validation

It is important that each of these characteristics are addressed at the design stage in order that an appropriate solution can be incorporated into the system, or occasionally, that the reasons are known and clearly thought out when a problem arises.

5.2.1 Representation of Feature Definitions

There are two particular strategies with regard to the implementation of features that must be examined.

- Procedural versus Declarative definition
- Hard coded versus interpreted feature definitions

It becomes obvious from these alternatives that Shah has already decided on the solution he requires, however that is not to say his subsequent choice of language is necessarily wrong.

Taking a practical point of view and comparing C++ with Scheme leads to the following conclusions.

Without writing an interpreter, C++ encourages the writing of procedural definitions of features, Scheme naturally encourages a declarative approach. C++ however has the advantage that object-oriented techniques are easy to implement (though difficult to implement well and in a user-extensible way), in Scheme an object-oriented viewpoint on any data structures can be arranged, and indeed, object-oriented methods can be layered on top of Scheme, (see [Abel96], Chapter 12 and [Laak96]).

The argument for hard-coded versus interpreted feature definitions follows similar lines. Interpreted feature definitions are generally preferred because the user (or more likely the consultant installing a system in a particular company) can add feature definitions more easily. The downside to making addition of features easy is that, especially when the design system is the front-end to a process planning system, there is far more to adding a feature than merely adding a function that builds the geometry of a feature. In particular, methods to perform access checks and proximity checks need to be added. With skill, the original system designer can make suitable functions available to calculate these bodies from an arbitrary new feature body, however it is foreseeable that the user may want to add a new class of feature, appropriate to some new manufacturing technique perhaps for which the conventional checks are not suitable. It is then necessary to either add substantial code or allow the user to be warned that checks on this new feature class are disabled.

However having said all this, it is also true that an interpreted feature definition also makes it easier for the original developer, provided the evaluation time of the interpreted code does not become critical. In the case of an ACIS Toolkit based product, the evaluation time for a feature is dominated by the solid modelling code rather than the time taken to interpret the function and so the build time for any feature is almost independent of whether the code is written in C++ or Scheme, ACIS is implemented in C++ either way. The time saving in prototyping the code however greatly reduces the lead-time of the system programmer.

5.2.2 Level of Support of User-defined Features

The level of support for user-defined features has already been discussed in the previous section, and the conclusion reached is that there is a greater overhead in

providing user-defined features in a system intended to perform geometric reasoning for process planning than in a straight forward feature based system. However, a trained user could add features to such a system provided the code was interpreted.

A particular advantage of the FODDS2 system in defining new features is that all features are defined in terms of a *tool-profile* and a *cutter-path*. It is much easier to define a machinable feature in this way than by writing volume generation routines, and it is also easier to automatically generate the various volumes required by the geometric reasoner in terms of the cutter-path and alternate tool-profiles.

Tool-profile and cutter-path here do not refer to any real tools necessarily, but to an abstraction of a 3D feature into two 2D profiles. By choosing this model, it is easier to generate real tool-sets and cutter-paths. Holznagel [Holz98] demonstrates this functionality in the NC generator (NCgen) from FODDS2 models.

5.2.3 Type of Linkage with a Geometric Modelling System

Shah advocates a strong link with a geometric modelling system in order that features can be validated in a number of contexts, including their geometry. The FODDS2 system has a strong unidirectional link from GUI and database to the modeller. The link back from the modeller to the front-end is weaker. This split ensures that the two areas of code concentrate on their domain (either the GUI or the geometry) and this encourages clear separation of functionality ensuring robustness and clarity.

5.2.4 Application Context

The application context of this work is principally in the machining of 2½D components (as defined in Chapter 2) and concentrates on a class of features for which a tool can be found that spins around its vertical axis and travels either linearly in the z direction or along linear or circular paths in the x-y plane, thus utilising a subset of motions defined by Sungurtekin [Sung86]. Other manufacturing processes such as those described in [Gupt97] are outwith the application context.

5.2.5 Support for Feature Validation

The system explicitly performs feature validation in the form of feature presence and void detection and access problem detection. It also performs some feature validation routines not commonly found in other feature-based systems, such as proximity detection.

5.3 The Design Features in FODDS2

Though the mechanism by which features are built and reasoning is performed has already been discussed in detail in Chapter 4, the view that the designer has is slightly different, and are detailed here. The designer can also build models using various operators that are discussed briefly.

The features are discussed in the following order.

- The positive workpiece construction features
- The primitive material removal (negative) features.
- The feature operators

5.3.1 The Positive Workpiece Construction Features

For workpiece construction two features are supplied, the cuboid and the cylinder, matching the most common form of billets and bar. Surface finish information can be added to these. More complex workpiece shapes can be built up through the union of a number of these positive features.

A third form of workpiece feature proposed is the casting, where an arbitrary ACIS ‘.sat’ file can be loaded into the system as the blank workpiece. This allows total flexibility of initial workpiece geometry.

The following design features from the FODDS2 Design Feature Library. Some of the features are not solids but operators acting on other solid features. Other features are peculiar to the Mandelli part family. Most of these specific features could also be implemented as compound features.

All solid material removal features implemented in the current system are ideally suited to 2½D milling and can be expressed as the solid generated when a tool (specified by its profile) travels along a cutter-path. This simplifies the specification of the features and also simplifies the generation of access bodies and the like for subsequent geometric reasoning. The sweep needed to generate the feature volume can be thought of as a restricted Minkowski sum.

5.3.2 The Primitive Material Removal (Negative) Features.

The following features are fully implemented in the current system.

- Holes
- Slots
- Pockets
- Rings
- Screwholes (as a subclass of Hole)

The following features have been considered for inclusion in the system.

T-slots, Complex pocket, Bearing seat, Pitch circle diameter, Matrix

All the volumetric material removal features can be described as the Minkowski sum of a tool volume with a cutter path. The tool volume can always be constructed from a rotary sweep of a tool profile through 360° as described in Chapter 4. The mapping from design feature attributes such as length, width and depth is shown in Table 2.

For pocket features, the cutter path becomes a cutting surface and the actual cutter path is a route along the surface such that tolerances on the surface are met.

For hole features, the cutter path is a single point.

For slots and rings, the cutter path is some curve or wire.

For all features, the cutter path can be summarised as a collection of planes or edges (either straight line or circular arcs) in the x-y plane. The tool profile can be represented equally as a collection of straight line or circular arc edges in the y-z plane. All features are transformed (rotated and translated) into their final position in

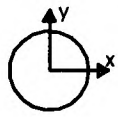
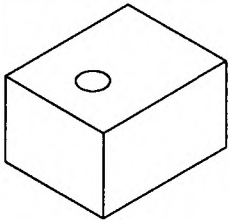
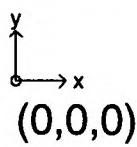
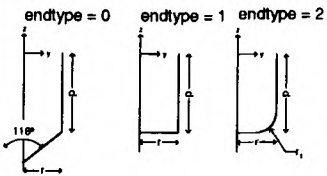
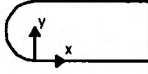
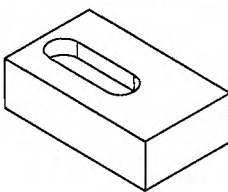
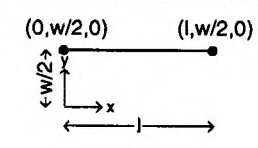
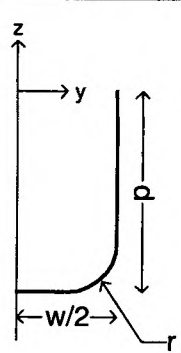
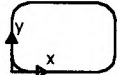
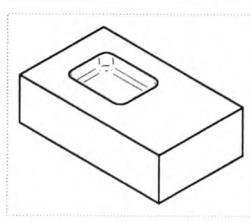
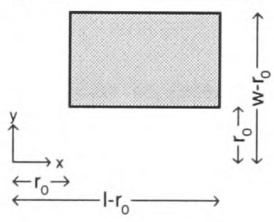
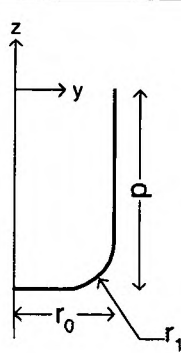
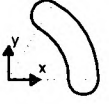
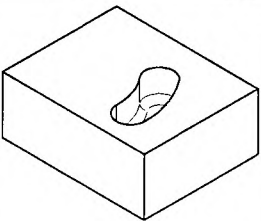
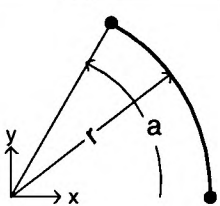
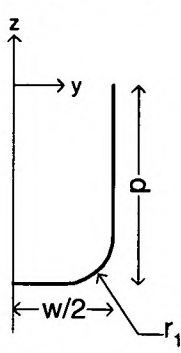
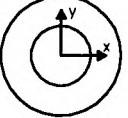
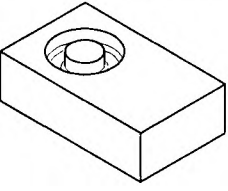
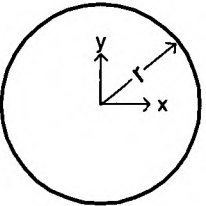
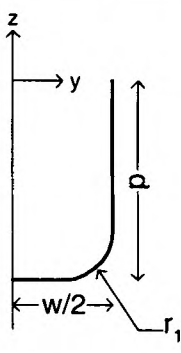
the design after their generation at the origin. This can be likened to changing setup prior to machining a set of features.

The following table summarises the parameters needed for simple features, and the subsequent tool profile and cutter paths needed.

It is not always clear where the origin of a feature is. This is important for knowing how the transform will affect it. Chang’s Quick Turnaround Cell (QTC) [Ande90] uses multiple handles to which any transform can be attached. The multiple transforms approach to features in someway obviates the need for multiple handles, as a simple translation can be used to move the effective origin of the feature. The first version of FODDS however had handles [Mill93].

The cutter paths for the features are not always centred on the origin, instead they are placed to give a convenient handle or origin on the design feature.

Table 2 The Primitive Material Removal Features

Name	Origin	Isometric	Cutter path	Tool profile
Hole diameter(2r) depth(d) threaded? endtype [drillmill] drill-angle(a) bottom_radius(r1)				
Slot length(l) width(w) depth(d) bottom_radius(r)				
Pocket length(l) width(w) depth(d) corner_radius(r0) bottom_radius(r1)				
Curved Slot Central_radius(r) width(w) depth(d) finish_angle(a) bottom_radius(r1)				
Ring central_radius(r) width(w) depth(d) bottom_radius(r1)				

Holes

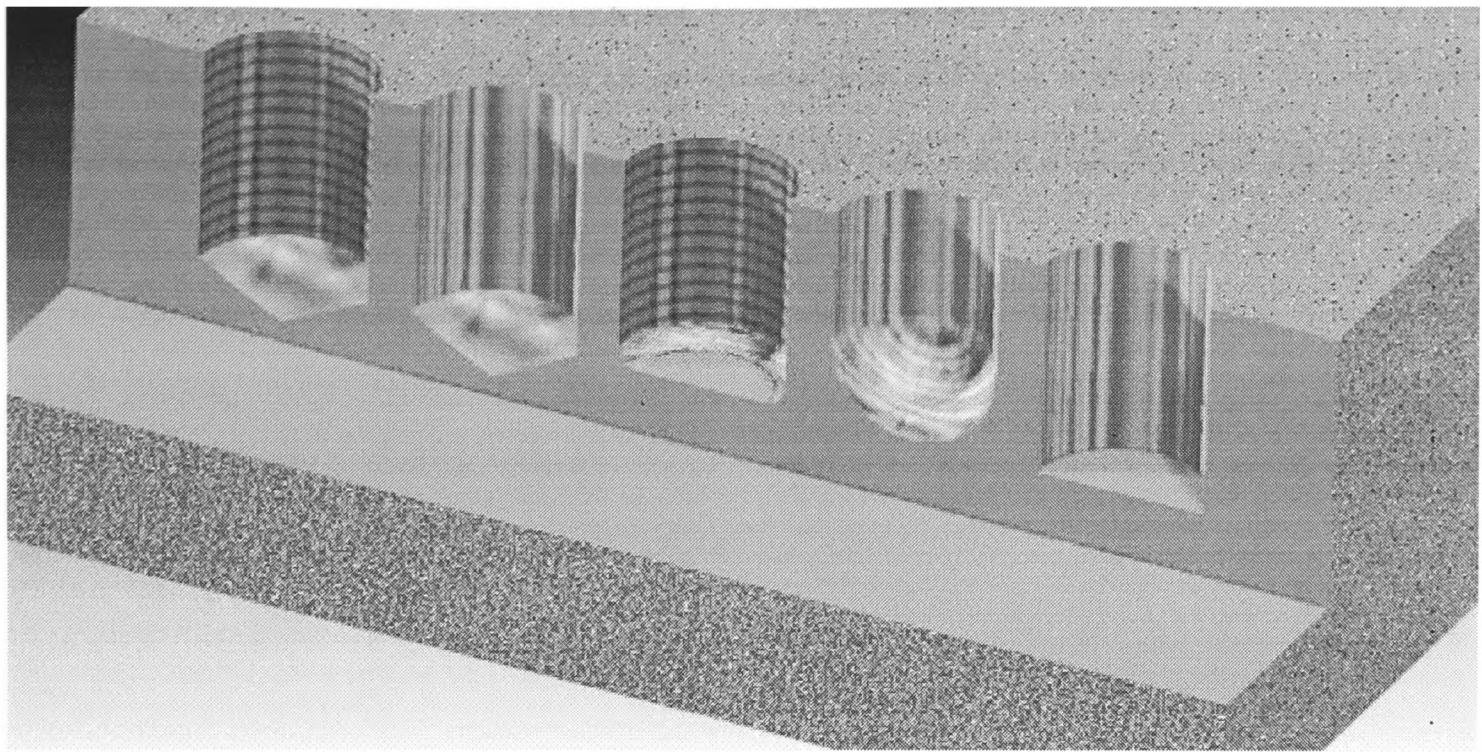


Figure 61 Example of Hole Types

Figure 61 shows examples of the different possible hole types. From left to right the figure shows a threaded hole with a 118° (drilled) end, a drilled hole unthreaded, a threaded hole with a radiused end, a ball-ended hole and a flat-ended hole. A flag selects whether holes are threaded or unthreaded and standard M-types are assumed. Hole type is 'drill' or 'mill' allowing selection of drill-angle or bottom radius. A bottom radius of zero gives a flat ended hole, and a bottom radius equal to the hole radius gives a ball-ended hole.

T-slots

A T-slot can be designed into the system by specifying two slots one on top of each other. A T-slot is a contender for a primitive feature in a future incarnation of the system, but has some problems when performing geometric reasoning. In particular, though the access body for machining a T-slot is merely the extrusion of the top surface of the slot, the access body for advance and retraction of the T-cutter is the extrusion of the largest part of the T-cutter. In order to ensure safe advance and retraction locations, either a search must be undertaken to find a suitable location, or the T-slot volume must be redefined to incorporate a volume for advance and retract operations.

The first of these two solutions is outwith the scope of this project whereas the second solution, though used in earlier versions of FODDS precludes generation of the T-slot in terms of a sweep of a tool-profile along a cutter-path (though an extension of the cutter-path to include the advance and retract phase might be possible.

It is felt that it is better at this time to enforce generation of T-slots as the combination of two slots. This means the geometry can be used by incorporation of compound features. An extension of the reasoning to include features with partial accessibility can be regarded as an extension of this work.

A feature similar to the T-slot is the dovetail slot. Cutter profiles have been developed for both the T-slot and the dovetail slot though problems regarding access for a dovetail slot are the similar to those for the T-slot above.

Screwholes

In early versions of FODDS, complex screwholes with chamfers, countersinks and complex specifications of thread type and depth were allowed. Though useful to the designer, they were somewhat specialised and did not fit the generalised model of features used throughout the remainder of the system. Furthermore, any of these complex screwholes can be represented as a compound feature composed of a number of more simple holes.

The only exception to this is that the concept of a threaded hole needed adding to the simple hole. It is now possible to add a thread to the simple hole. This addition is reflected in the display, either in the simple display as a darker colour or in the rendered display as an attempt to render the thread.

Given the use throughout the project of the metric system, there are only a handful of possible normal thread specifications for any particular dimension of hole. Specialised threads are beyond the bounds of current CAPP systems anyway. Thus, the type of thread is recorded and passed onto downstream applications, and the thread is rendered, but no further work uses the thread attribute of the simple hole.

Pitch Circle Diameter

The Pitch Circle Diameter and the Matrix are useful design operators, providing a useful abstraction and subsequent saving on design time. It should be noted that passing on the information that a feature is a member of one of these groupings can offer solutions (such as multiple tooling) that could otherwise be missed. However, they also add some minor problems, for instance, in naming the sub-features of a group feature. Though the system allows the user to generate these features, none of the test examples use such a feature.

5.3.3 Tolerancing and Surface Finish

Though much work has been undertaken on tolerancing and surface finish [Voel97] [Henz95], and though it is extremely important for process planning, only a simple tolerancing model has been included in FODDS2. The manufacturing feature viewpoint of design means that full tolerancing models after ANSI or BS standards are unnecessary. Instead each material removal feature has a position, orientation and surface finish tolerance associated with it. It is up to the process planner to ensure that all faces of the feature will match this tolerance. A fuller tolerance model would allow the specification of different tolerances for different faces. This model however acknowledges the importance of tolerancing.

The default surface finishes for workpiece and features have been selected so that drilling and milling can easily achieve the required surface finishes. The surface finish of any feature can be tightened if required by the designer.

A limiting factor of this model is that the bottom surface and side surface of any feature are allocated the same surface finish, which is untrue of most machining processes. The extension to allow different surfaces of a feature to have different surface finishes is merely a matter of adding some extra parameters.

An additional extension to add intra-feature relationships suitable for the different feature types has been discussed, as has the ability to add inter-feature geometric tolerance relationships. Neither of these extensions are difficult, however at present the geometric reasoning routines cannot make use of this information. The work of

Naish [Nais97] requires this information however, so these extensions may be added in the near future.

A simple object designed in FODDS2 and its accompanying feature tree is shown in Figure 62. In the feature tree, the ‘-’ signs only represent where the tree may be collapsed in the GUI. The operators in the diagram are ‘Assembly’, ‘Component’ and ‘Transform’, and the features are ‘Block’ (the positive feature) and ‘Slot’ and ‘Hole’.

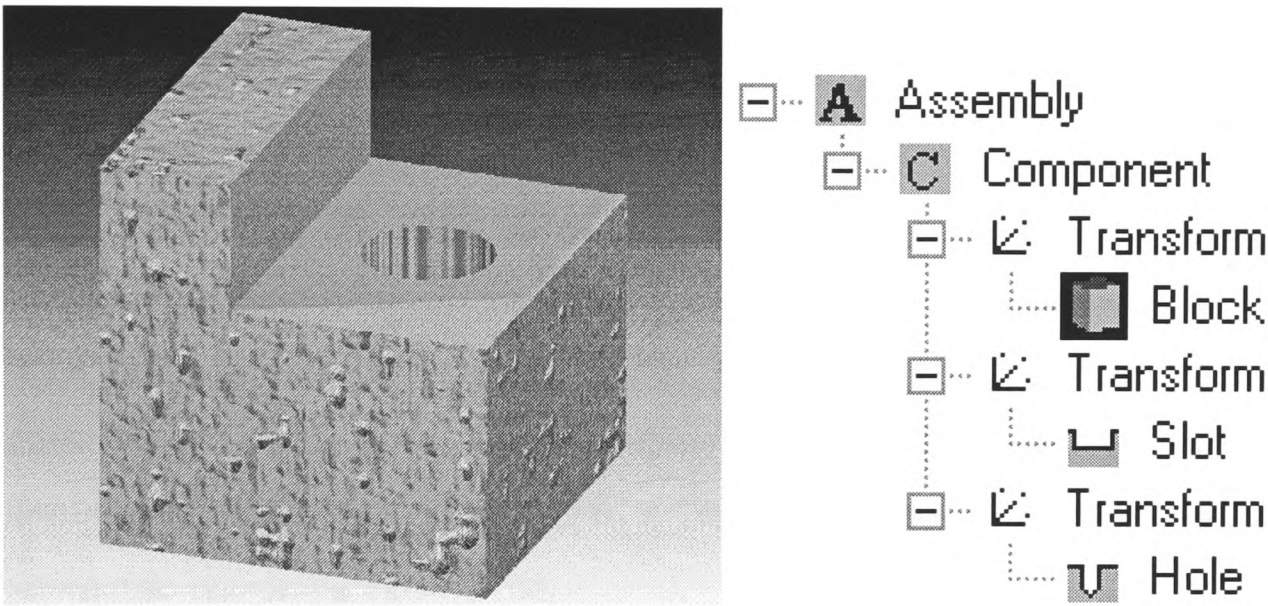


Figure 62 The Simple 'Ell' Component

6 Experiments

There are two varieties of experiments that have been performed.

The first variety is a series of focussed examples designed to verify the performance of one or more design or geometric reasoning algorithms in a controlled situation. These experiments were initially conducted as thought experiments and were used throughout the period of study in order to direct the research towards the ultimately successful goals.

The second variety of test is with more complex components, either test components from Edinburgh or from other research groups, or real components. These components have been drawn from a variety of sources. Certain components have been used as test components within the group for a number of years. Other components are from the CIE97 Feature Recognition competition [Litt97b]. These components have been modelled to a greater or lesser degree and had some of the geometric reasoning carried out.

6.1 Focussed Experiments

6.1.1 Feature Variety Test

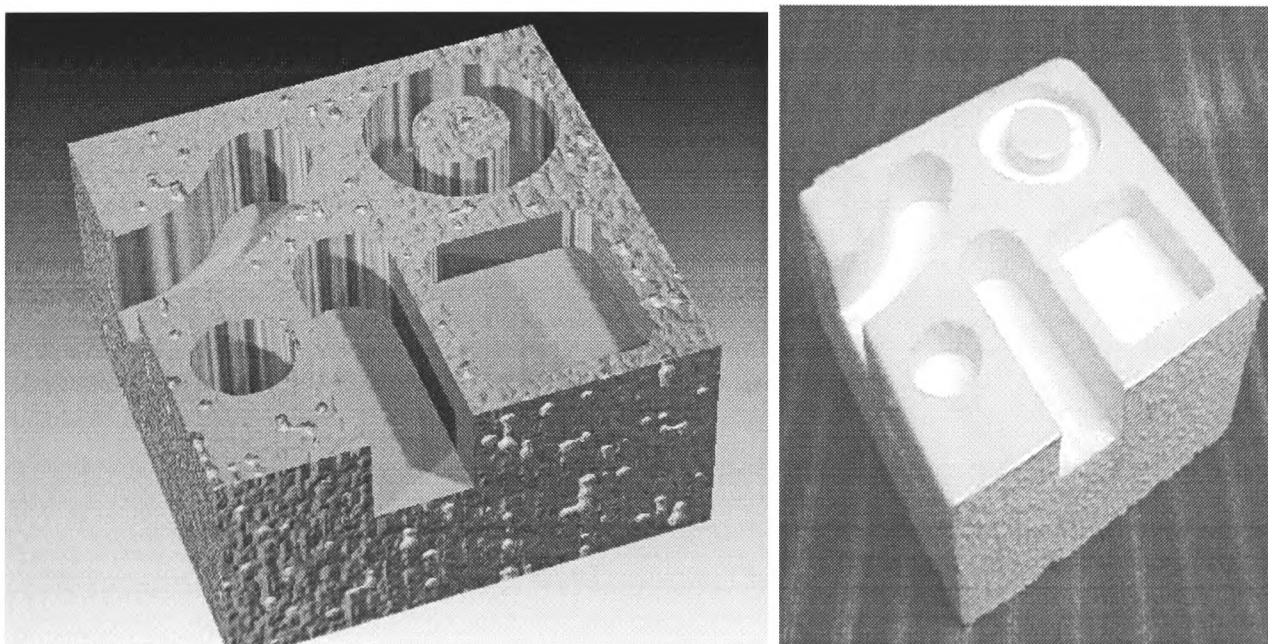


Figure 63 The Feature Library Demonstration Part

The Feature Variety Component is shown in Figure 63, both a rendered image and a real NC machined component.

This component tests five different feature types in a simple block. There are (deliberately) no feature intersections, access problems, through holes or thin walls, and the object is strictly 2½D. This test merely demonstrates that the system is capable of creating feature based designs.

The NC code for this part has been automatically generated by a prototype NC generation system (NCgen) [Holz98] from the feature based description output directly from FODDS2. NCgen assumed a single end mill whose radius is smaller than the radii in all features, and computed complete toolpaths for each feature on this basis. The NC generator will provide valid toolpaths provided features are supplied in a machinable order (i.e. satisfying anteriority constraints), and that no setup changes are required (the component is strictly single sided). The NC generator is a validation tool and not a full scale CAM package, so it also neglects speed and feed information and the cutter paths are not always optimal.

6.1.2 Access test for Crossed Slots with Hole

There are three aspects to this test.

The first is to ensure that the system can handle the ordering problem when there is an access problem with two features. The second shows that the test produces valid but different results when the intersections are modified slightly.

The third test illustrates an extreme case where the code produces a valid but suboptimal result.

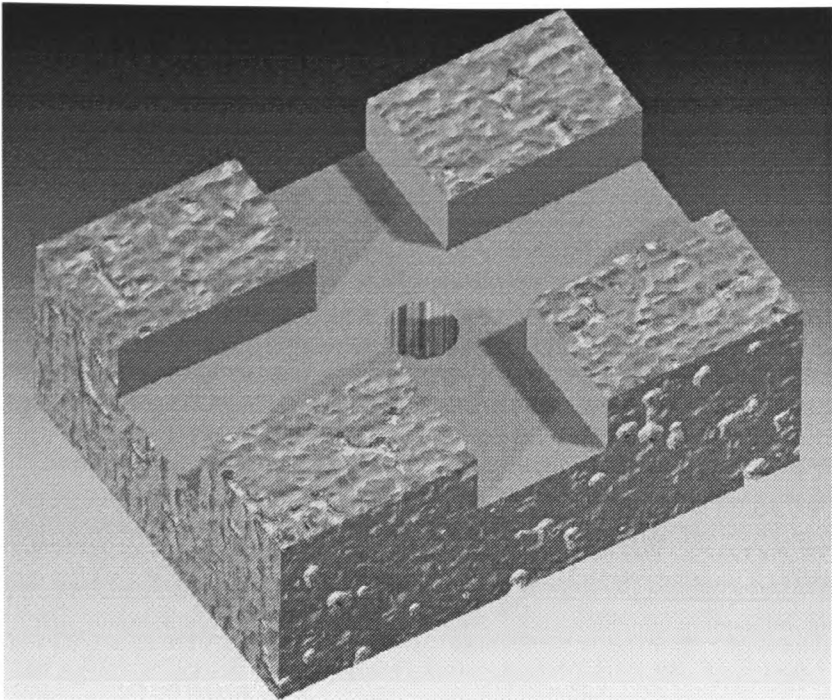


Figure 64 Crossed Slots with a Hole

The component in Figure 64 consists of a pair of crossed slots with a blind hole at the crossing. The hole has been designed with its top at the base of the slot. Because no feature modifications are made in the system this implies that the hole cannot be machined first. It is clear by inspection that the hole can be made after either of the slots has been machined (or indeed after both).

The analysis automatically produces the following access list

```
(  
  ((Hole Hole Node 10 access) intersects (blank))  
  ((Hole Hole Node 10 access) intersects (Slot Slot Node 6 feature))  
  ((Hole Hole Node 10 access) intersects (Slot Slot Node 8 feature))  
)
```

The conservative implication of the automatically generated list is that both slots must be machined before the hole. In fact it is obvious that either slot can be machined before the hole and so ideally a way of representing the logical operator ‘*or*’ on the graph of the anteriority constraints. More importantly the algorithm must be improved so that this ‘*or*’ constraint can be identified. Though in the case of the crossed slots above it is unlikely the hole would be machined after one slot but before another, other examples show different behaviour.

The algorithms give the correct anteriority constraints.

There are no other problems with this object.

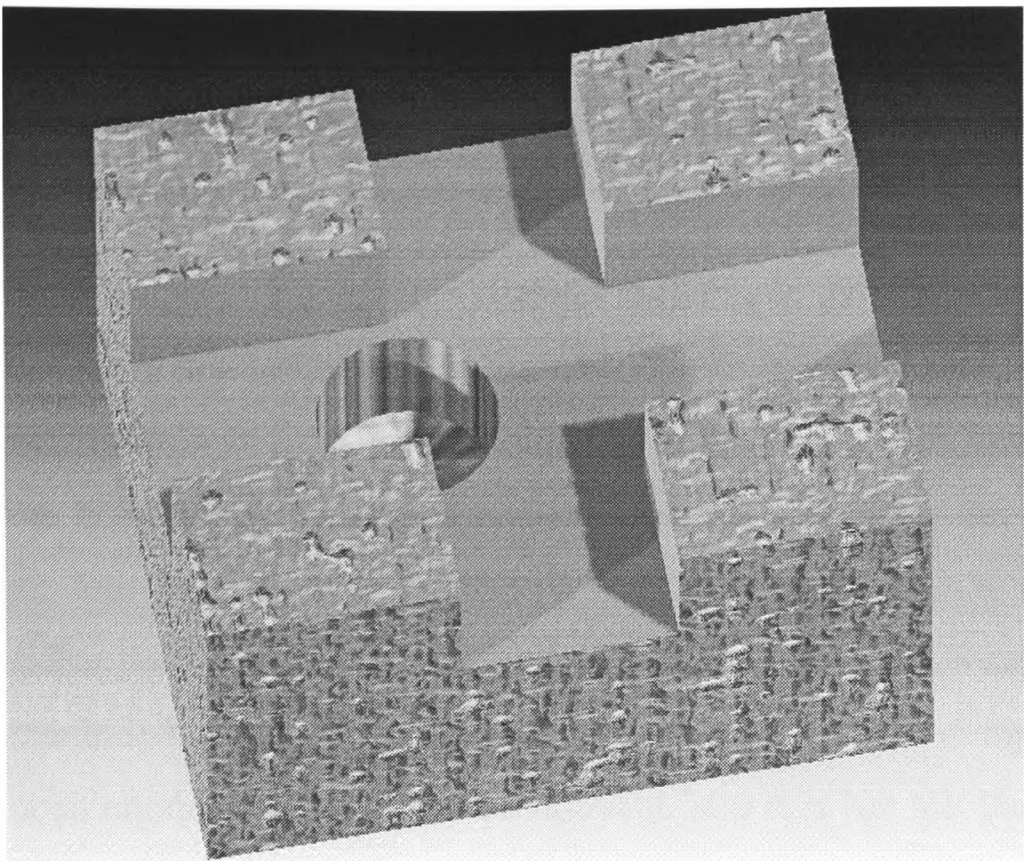


Figure 65 Offset Hole in Crossed Slots

This component (in Figure 65) is to ensure that the previous case of Figure 64 is not repeated. The hole now lies in the base of one of the slots, but only partially in the base of the other. Here, the algorithm correctly spots that the access body of the hole intersects with both slots, so two access problems are posted. Then further reasoning shows that the volume of intersection with the shorter slot is completely subsumed in the volume of intersection with the longer slot, and so only one “machined_after” constraint is posted. that with the longer slot. and the correct result is realised.

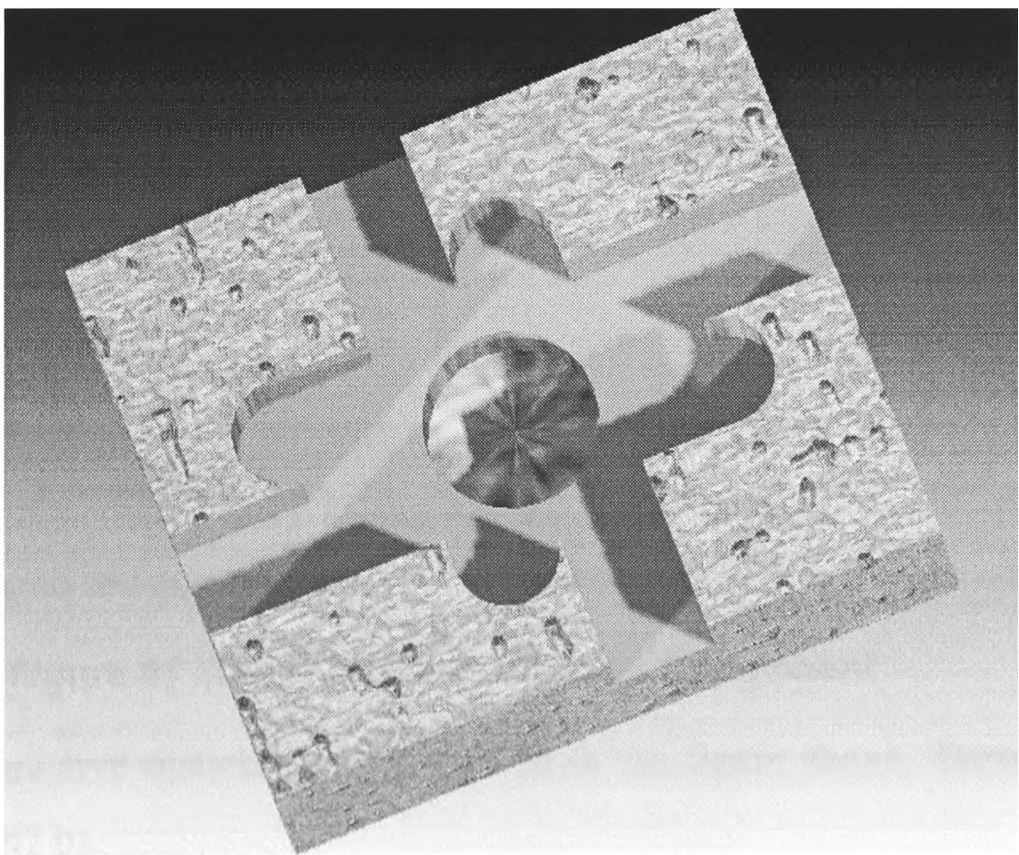


Figure 66 Four Crossed Slots with Hole

The previous example uses pairwise analysis of access-feature intersections to discover that only one anteriority constraint is necessary.

Th example in Figure 66 illustrates that simple pairwise analysis of these intersection bodies is not sufficient. Preliminary constraints are posted for each of the four slots, but there is no pairwise combination of slots for which the intersection volume with the hole can be subsumed one in the other, so finally, the requirement emerges that all four slots must be machined before the hole. This is infact untrue as provided either the North-South pair are machined, or, the East-West pair then the hole can be machined. A situation such as this where the pairwise comparison fails has never yet been encountered in a real component, and it is not felt that generalising the current algorithm for this case is necessary.

6.1.3 Thin Wall Demonstration

The test component in Figure 67 test shows a variety of thin wall situations on a single component and illustrates where correct interpretations are made and where further problems arise. All images come from the FODDS model of the part. In Figure 67 c) the dilated volumes can be seen, and it is the intersection of these volumes with other dilated feature volumes or with the volume outside the blank that are identified.

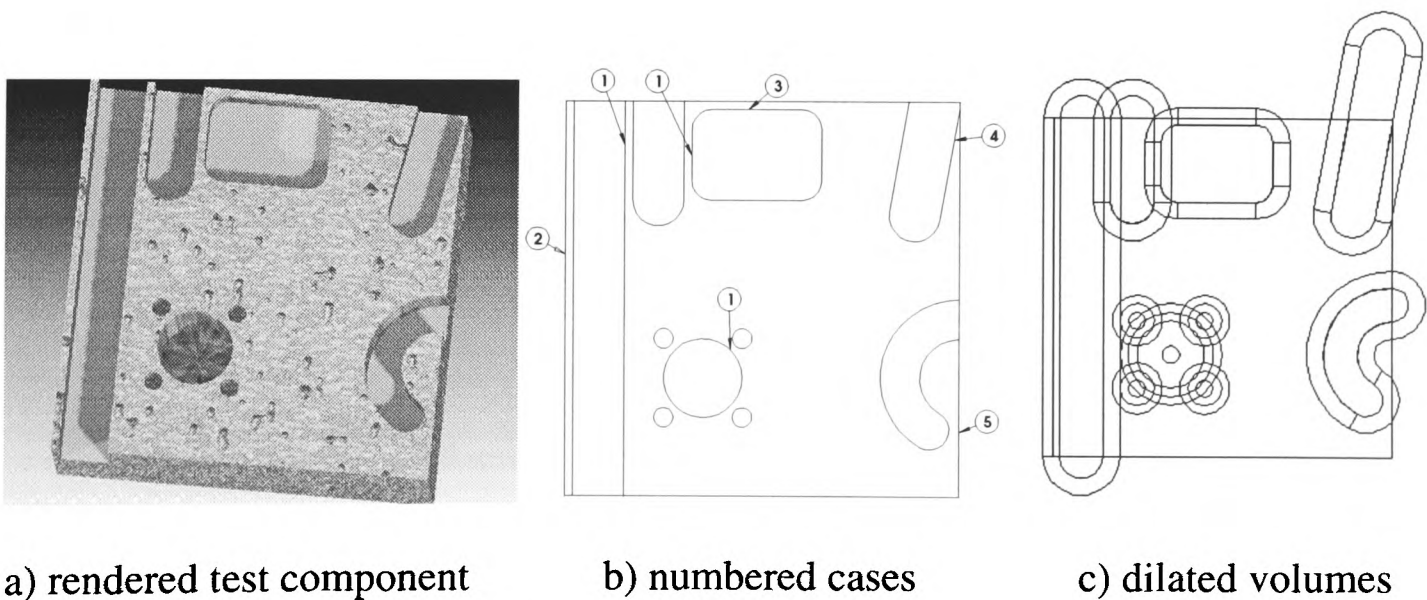


Figure 67 Thin Wall Demonstration Component

There are five distinct cases identified in the figure above. These are numbered in Figure 67 b).

Feature-Feature Intersections. These correctly identify thin walls between features such as the slots and the pocket and the holes.

Feature-Blank Intersections. Case 2 shows a feature-blank intersection where the slot already pushes into space, so the thin wall is not detected with the existing algorithm.

Feature-Blank Intersection. Case 3 shows a successful diagnosis of a thin wall between a feature and the blank.

1. Case 4 shows a questionable case that can arise either in a feature-feature or feature-blank interaction. Where a solid wedge is at the junction of two features, and the angle of the wedge is acute then it is possible that this should be considered as a thin wall. Because the two features (or in this case the feature and the outside world) intersect, the algorithm cannot test for this.
2. Case 5 shows a similar case, but here the question of whether we should consider this to be a thin wall is more clear cut. Again, we cannot look for the thin wall because of an existing intersection between feature and the outside world. Whereas case 4 is perhaps excusable, Case 5 can always arise where features with concavities are allowed.

It becomes clear from the analysis of this thin wall component that the model of dilated bodies and solid intersections is not powerful enough to detect all desired thin wall cases. It seems likely that looking for thin walls in terms of face-face proximities [Gupt95] is also fraught with similar difficulties, particularly in the case of acute wedges again. An alternative approach might be to use a Minkowski erosion algorithm on the finished component and look for the difference between the medial axis [Arms94] of the component before and after erosion. Significant changes in the medial axis would show the disappearance of the thin wall under erosion.

6.2 Test Components

6.2.1 The HAPPI component

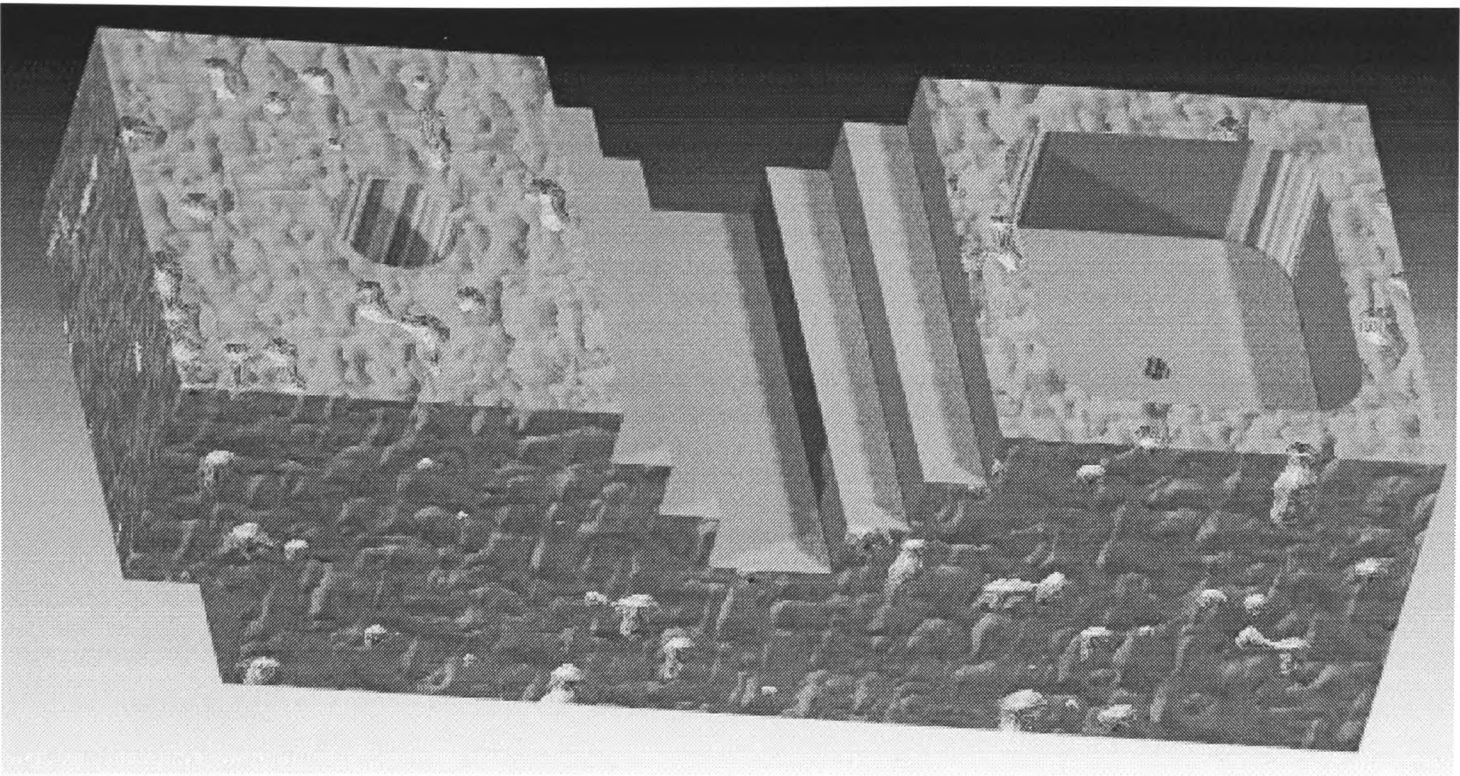


Figure 68 The HAPPI Test Component

Table 3 The HAPPI Test Component Access List

(
((Slot MidSlot Node 10 access) intersects (blank))
((Slot BtmSlot Node 12 access) intersects (blank))
((Hole LtlHole Node 16 access) intersects (blank))
((Slot MidSlot Node 10 access) intersects (Slot TopSlot Node 8 feature))
((Slot BtmSlot Node 12 access) intersects (Slot TopSlot Node 8 feature))
((Slot BtmSlot Node 12 access) intersects (Slot MidSlot Node 10 feature))
((Hole LtlHole Node 16 access) intersects (Pocket Pocket Node 14 feature))
((Hole BigHole Node 6 blind-access) through (component))
)

The list above is automatically generated from the component in Figure 68 by the Access Checking Geometric Reasoning Algorithms. The first three relationships identify access problems for the middle and bottom slot and the hole in the pocket. The next three relationships identify which features must be removed before the three problem features can be machined. The last relationship identifies the BigHole as being a through hole, that is it has access through the component on its ‘blind’ side.

The HAPPI test component is again shown in Figure 69 complete with access bodies shown in ‘transparent gold’, and the negative features in chrome to illustrate the nature of the geometric algorithms working in practice.

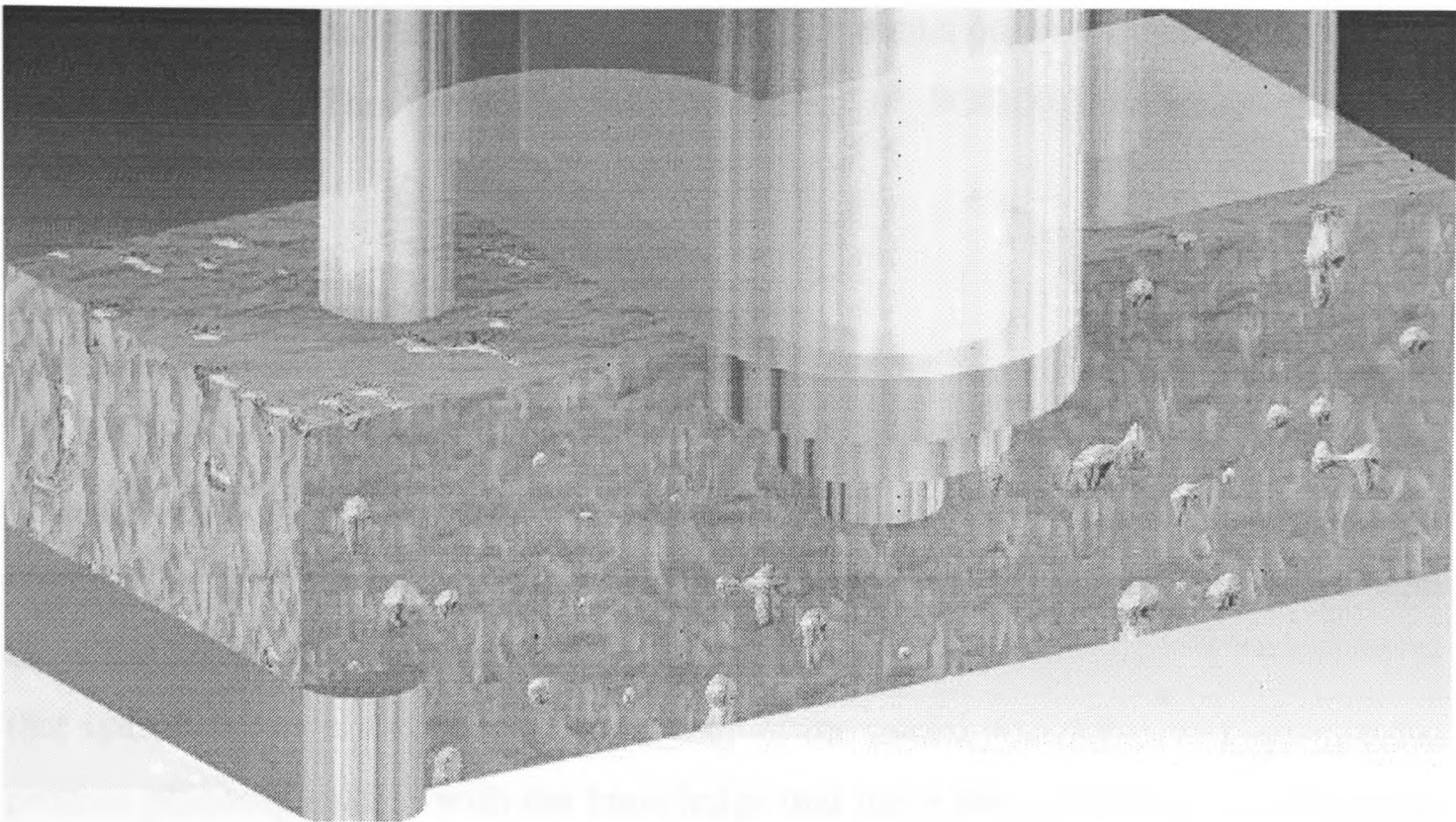


Figure 69 The HAPPI Test Component with Features and Access Bodies

6.2.2 The Edinburgh Composite Component (TECC)

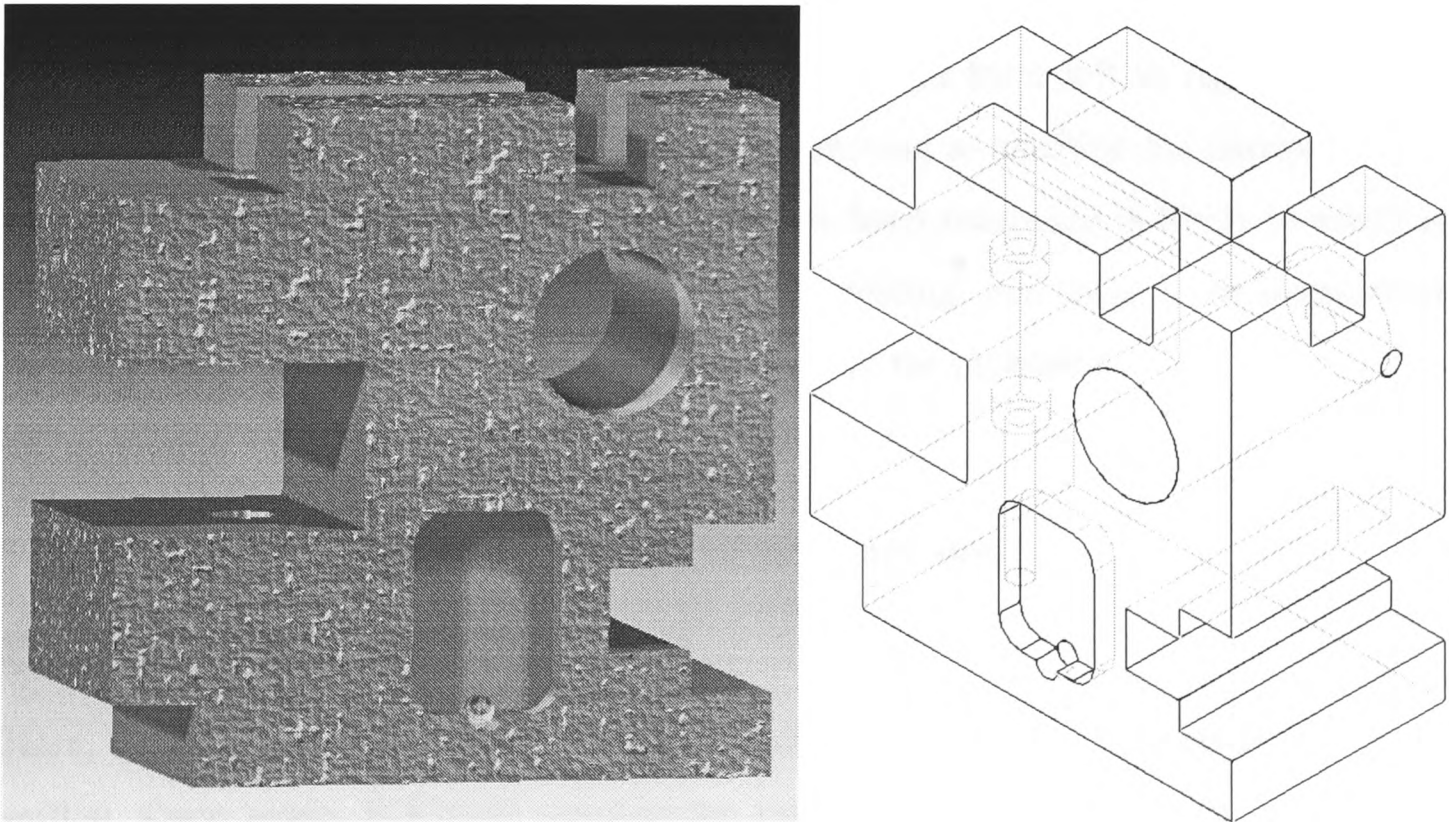


Figure 70 The TECC Component

The TECC Component shown in Figure 70 consists of 13 features and was modelled using 4 approach directions using compound transforms for all but the default setup.

This Component has a set of process planning problems on it as described by Mill et al. in [Husb91]. The problems and the ability of FODDS2 to tackle them are discussed.

Crossed slots

On the top surface of the TECC, a crossed slot can be seen. The issue in the TECC paper was one of alternate representations. Were the slots two crossed slots or four meeting slots or some other combination? Can alternate representations of the features be used for process planning? The answer in the case of FODDS2 is that there are no alternate representations, the only representation known in FODDS2 is that specified by the designer on generation of the model. FODDS2 does provide the process planning system with the knowledge that these slots intersect, so it becomes the responsibility of the downstream applications to use an alternate representation if required.

Crossed holes

The large diameter hole in the side of the TECC component is crossed by a smaller diameter hole running entirely through the component from left to right. The issue here is similar with one important addition, attempting to machine the narrow hole in one machining operation after the wide hole has been machined is likely to result in tool breakage. Again FODDS2 supplies the knowledge that there is an intersection between these features, but doesn't attempt to solve the problem.

Nested Slots

FODDS2 correctly orders the machining of the nested slots.

Pocket with hole in side

Here, there would be a problem trying to machine the hole after the pocket has been milled. Once again, FODDS2 supplies the information that there is an intersection, but does not solve the problem.

Countersinks with access problems

FODDS2 correctly spots that machining the countersinks cannot be performed with conventional 2½D milling and tools.

Steps

There are two steps on this component. FODDS2 does not treat a step any differently from a slot or pocket. This means that the only access direction checked is the access direction that was specified when the feature was entered into the design.

Through holes

There are three through holes on the component. FODDS2 correctly spots all of these.

Ultimately FODDS2 finds that all features with the exception of the countersunk holes are machinable, though a number of intersection bodies are posted as hints of possible problems for the process planner to examine when tool selection is undertaken.

6.2.3 The Heriot-Watt 2 ‘MacTaggart Scott’ Component

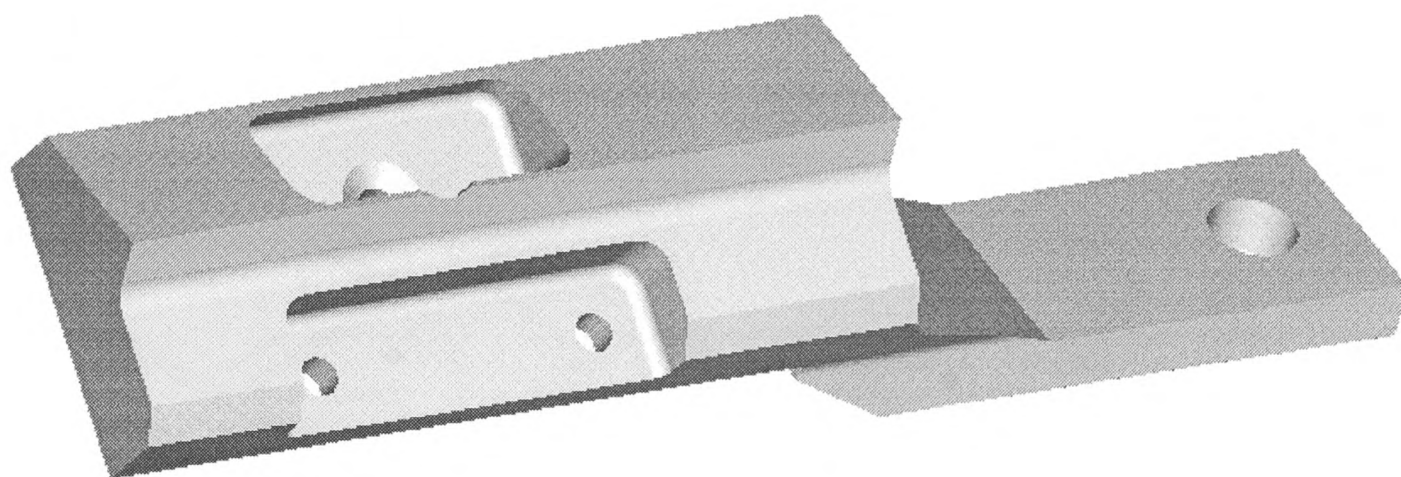


Figure 71 The Heriot-Watt 2 Component

The Heriot-Watt 2 component, originally from MacTaggart-Scott of Loanhead, Edinburgh has been completely modelled in FODDS2 and can be modelled easily with the standard feature library. This feature model consists of 13 holes, 4 slots and 13 pockets (30 negative features in all) and is the largest component to be successfully analysed by the system. An unusual problem is that because of its function the component is clearly split into two ends with a tricky angle between the two. In FODDS2 it was easy to model this by introducing an extra transform before the set of features at the far end. In FODDS2, the component was modelled from a block, MacTaggart’s themselves have planned the component (using the PEPS CAM

system from CAMTEK) using a round bar. The bar may be somewhat more efficient in material use.

152 relationships were discovered by the reasoning algorithms, predominantly access problems, but with 8 through holes and 3 through pockets.

6.2.4 The Heriot-Watt ‘Teddy Bear’

Unfortunately, this is one of the few ‘real’ components to defeat the design side of the current incarnation of FODDS2. Examination of the object (seen in Figure 72) reveals that every pocket has opposite sides that are not parallel. FODDS2 currently only contains a rectangular pocket, not sufficient for this feature.

Additionally, the Heriot-Watt version of this body contains some bottom edge blends that are not machinable with conventional machining.

The ‘teddy bear’ is predominantly single sided however, so with the addition of the compound pocket, running the algorithms on the component and subsequently planning the component are not imagined to be a problem.

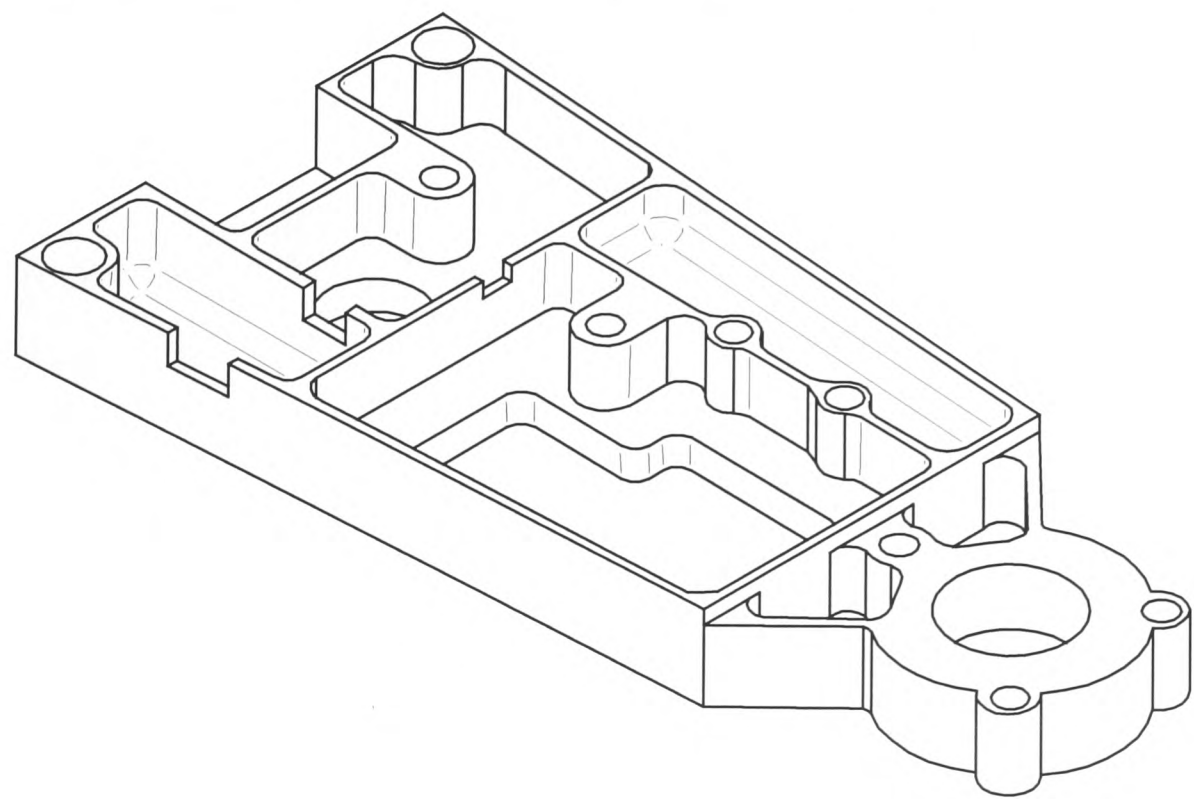


Figure 72The Heriot-Watt ‘Teddy Bear’

The majority of the pockets in the real features examined have been rectangular, and for this reason non-rectangular pockets have been neglected. A more complex pocket can still be defined in the ‘tool profile’, ‘cutter path’ model of features, but with a

more complex cutter path. This ensures that all the algorithms will still work for more complex pockets.

The plan for the next generation of pockets is that the user specifies a list of 2D points representing the vertex of a polygon, then a separate radius can be specified for each of convex corner radii, concave corner radii and the bottom edge blend. A wire polygon can then be turned into the necessary surface through the use of 2D Minkowski blending and this then passed to the feature construction functions to produce the 3D feature. An example of the solid volume that such a compound pocket might have is given in Figure 73. This figure shows the inverted volume to be removed from an imaginary pocket. The different radii chosen for convex, concave and 'bottom' radius blends can clearly be seen. The overall profile of this pocket has been specified by six vertices of a polygon at which each adjacent pair of planar faces intersect. Defining a polygon this way encourages manufacturability. The radii chosen provide constraints on the tooling that the process planner can choose.

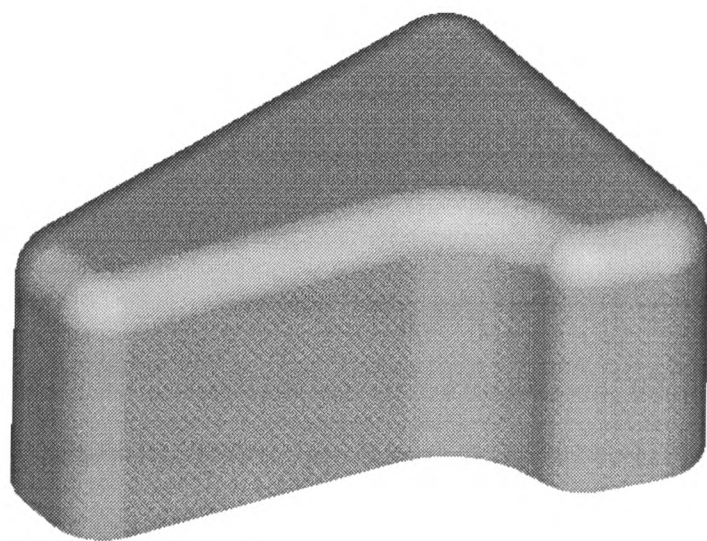


Figure 73 Example of the 3D Geometry of a Complex Pocket

6.2.5 The Han1 Component

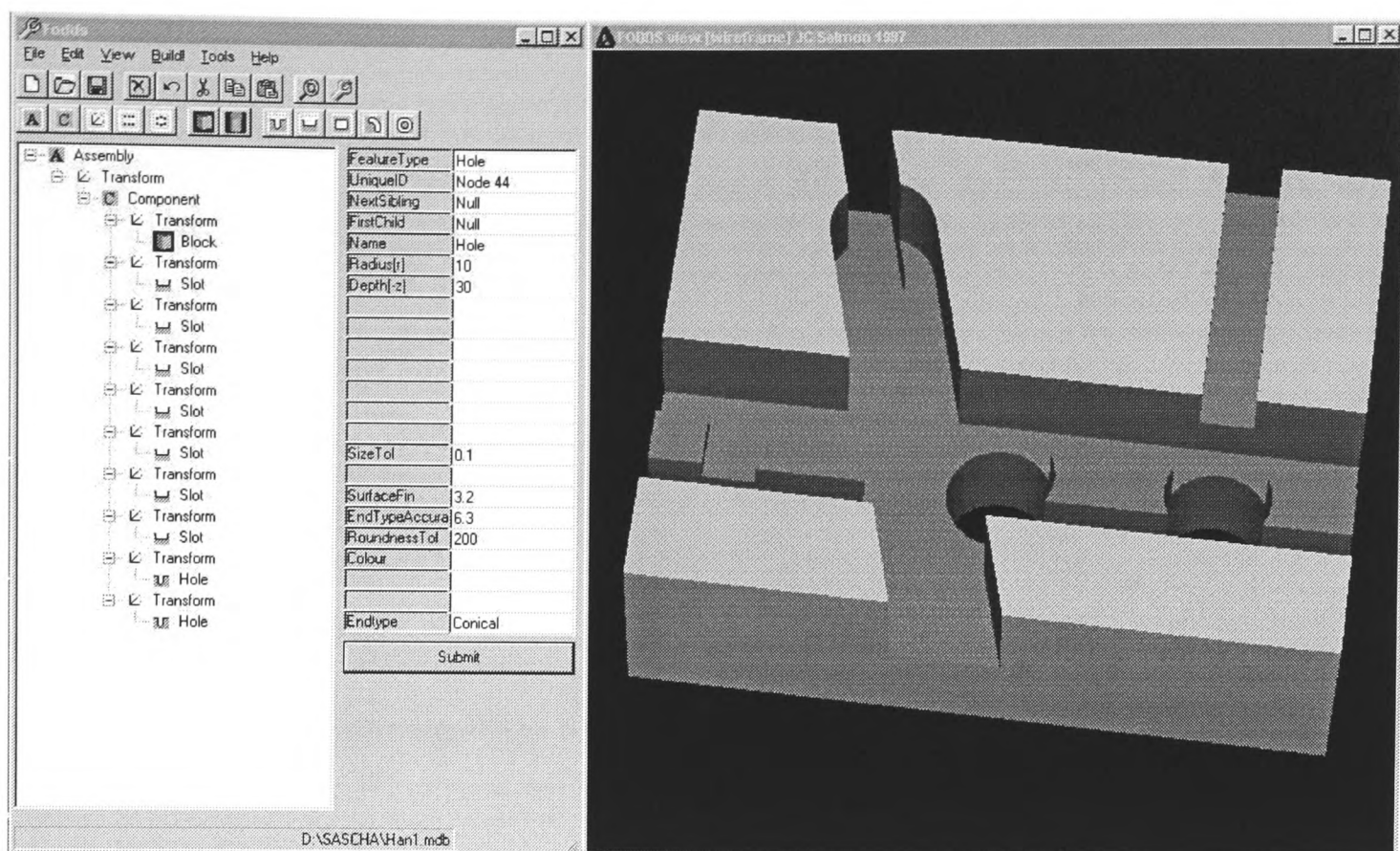


Figure 74 The Han1 Test Component in the FODDS2 GUI

The Han1 test component has been designed as a feature recognition test component. It is single sided but contains through holes and two slots that are not orthonormal features. The Han1 component presented no significant problems for modelling in FODDS2. Indeed, the analysis of the component was also straightforward. The component consists of a block, 7 slots and 2 holes. All are designed into the top surface of the component with the exception of the small slot on the left hand side of the image and the two holes which have been designed as low as possible.

The small slot and two holes have access problems against various slots. The two holes are correctly identified as through holes. The results of the anteriority checks as produced by FODDS2 on the Han1 component are shown in Table 4.

Table 4 Results of Anteriority Checks on Han1 Component

(
(("Slot" "Slot" "Node 36" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 42" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 44" "access") "intersects" ("blank"))
(("Slot" "Slot" "Node 36" "access") "intersects" ("Slot" "Slot" "Node 28" "feature"))
(("Hole" "Hole" "Node 42" "access") "intersects" ("Slot" "Slot" "Node 28" "feature"))
(("Hole" "Hole" "Node 44" "access") "intersects" ("Slot" "Slot" "Node 28" "feature"))
(("Slot" "Slot" "Node 36" "access") "intersects" ("Slot" "Slot" "Node 30" "feature"))
(("Hole" "Hole" "Node 42" "access") "intersects" ("Slot" "Slot" "Node 30" "feature"))
(("Hole" "Hole" "Node 44" "access") "intersects" ("Slot" "Slot" "Node 30" "feature"))
(("Slot" "Slot" "Node 36" "access") "intersects" ("Slot" "Slot" "Node 32" "feature"))
(("Hole" "Hole" "Node 42" "access") "intersects" ("Slot" "Slot" "Node 32" "feature"))
(("Hole" "Hole" "Node 44" "access") "intersects" ("Slot" "Slot" "Node 32" "feature"))
(("Hole" "Hole" "Node 44" "access") "intersects" ("Slot" "Slot" "Node 38" "feature"))
(("Hole" "Hole" "Node 42" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 44" "blind-access") "through" ("component"))
)

There are no thin walls, and the component can be machined (and designed) as a single sided component.

6.2.6 The Gadh2 Component

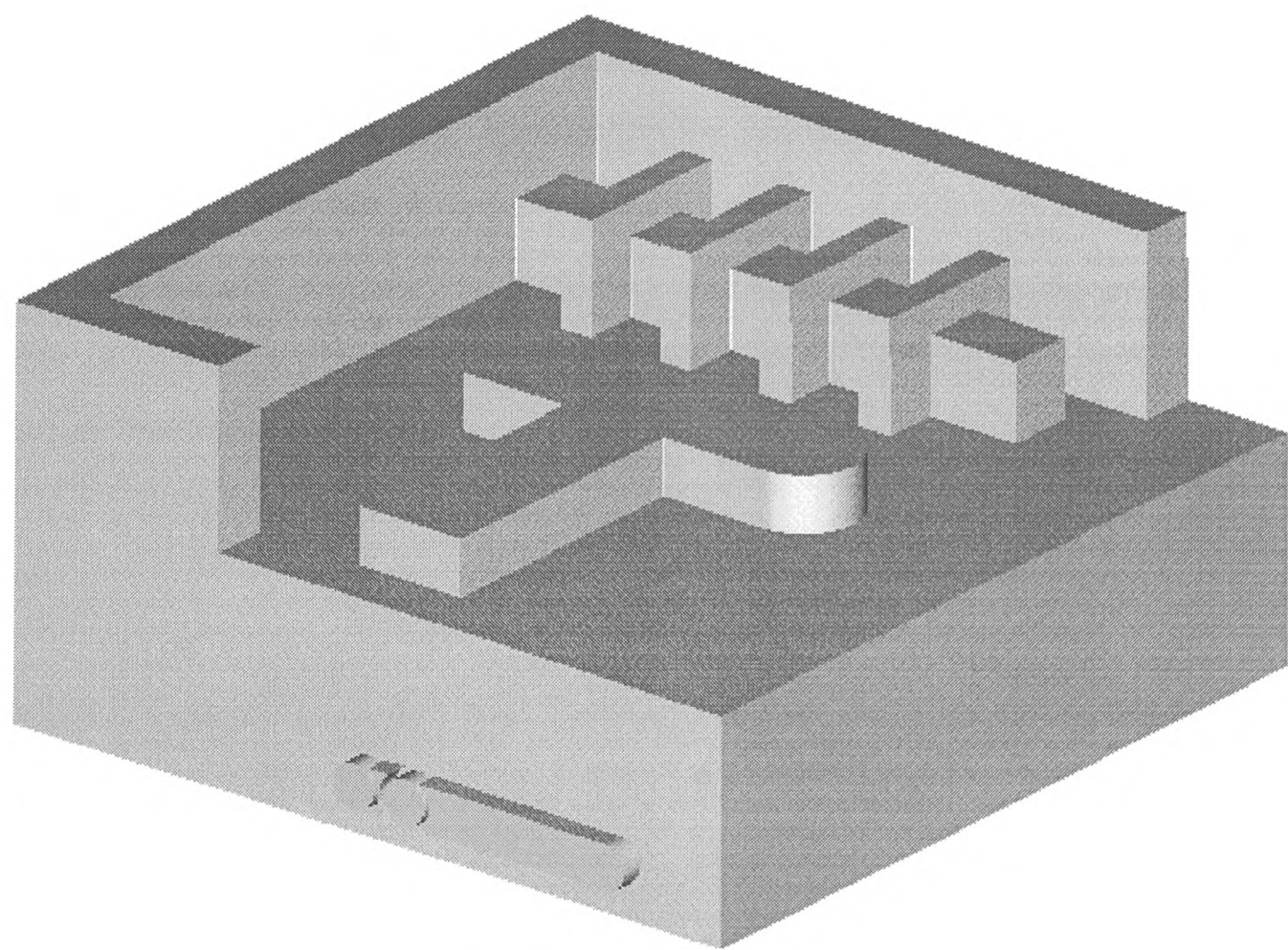


Figure 75 The Gadh2 Test Component

The FODDS2 was able to model the Gadh2 component except for one exceptional area. The area on the outer left face showing a lozenge shaped protrusion with two ring features. The original design called for 5 hemispherical protrusions here. These

protrusions would defeat most feature recognition systems and most manufacturing systems. There is no convenient feature, or indeed method of manufacture of convex spherical surfaces. The design shown in Figure 75 has started to approximate the hemispherical bumps using ring features. These are the only complex features on the component, and require significant external machining just to reveal the area where the bumps should be. Despite an inaccurate model, already 6 negative features have been used to create the remaining island.

With the exception of the protrusions on the side the remainder of the component is strictly 2½D. The design shown consists of 4 slots, 24 pockets, 3 curved slots and 2 rings. Unfortunately the system suffered from combinatorial explosion on analysis and the system crashed in the geometric algorithms having consumed over 200Mb of swap space.

In order to design the rounded end of the T-shaped protrusion in the middle of the component a curved slot was used in conjunction with surrounding pockets. The intersections between these features that should have been recognised could give a subsequent process planner opportunity to optimise the NC cutter paths to avoid remachining empty space.

The failure has been useful in indicating that the code needs some optimisation to be able to tackle very large components, however the success on smaller components shows the theory to be sound.

6.2.7 The Regli component

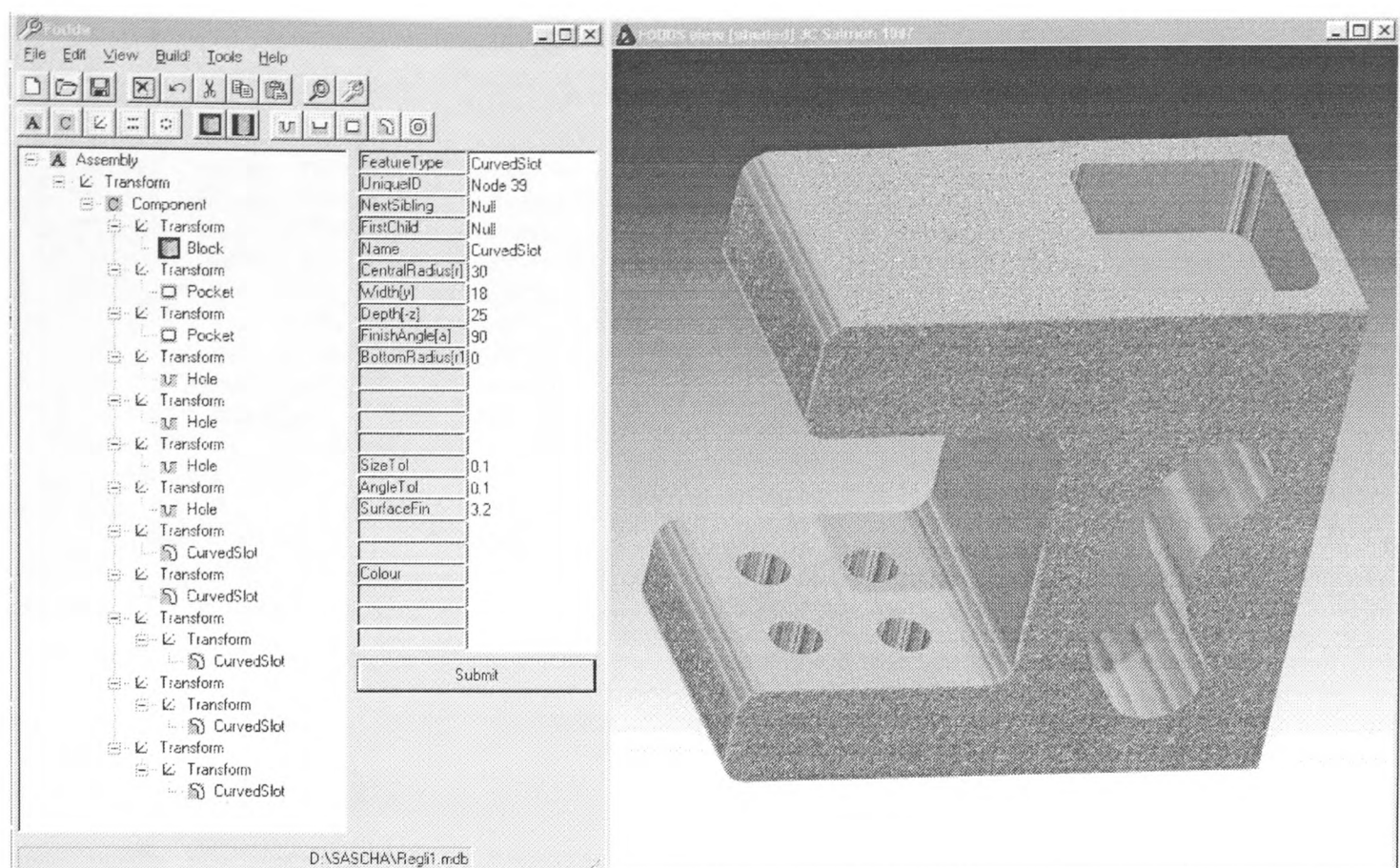


Figure 76 The Regli Component in FODDS2

This component has 2 pockets, 4 holes and the rounded external corners have been constructed with 5 curved slots.

The side pocket on the Regli component (see Figure 76) has a spline profile in the original. This has been roughly approximated by a circular slot feature for modelling in FODDS2. The problem is in specifying the spline profile in a way flexible enough to be useful and easy enough to be practical. It is not proposed to add a spline bounded pocket in the immediate future.

A problem with spline pockets is determining the smallest concave radius, to determine cutter sizes, and in particular having determined a cutter size, ensuring that there is no isthmus too narrow for the chosen cutter to pass through.

The blended corners on the left of the component have been modelled with curved slots, suggesting access directions from the front or the back of the component. This method of making these features is unlikely to be the choice of most process planners. The aspect ratio of the ‘curved slot’ would probably reveal the problem.

The relationships discovered are given in Table 5 below.

Table 5 Relationships of the Regli Component

(
(("Hole" "Hole" "Node 16" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 17" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 18" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 19" "access") "intersects" ("blank"))
(("Hole" "Hole" "Node 16" "access") "intersects" ("component"))
(("Hole" "Hole" "Node 17" "access") "intersects" ("component"))
(("Hole" "Hole" "Node 18" "access") "intersects" ("component"))
(("Hole" "Hole" "Node 19" "access") "intersects" ("component"))
(("Hole" "Hole" "Node 16" "access") "intersects" ("Pocket" "Pocket" "Node 11" "feature"))
(("Hole" "Hole" "Node 17" "access") "intersects" ("Pocket" "Pocket" "Node 11" "feature"))
(("Hole" "Hole" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 11" "feature"))
(("Hole" "Hole" "Node 19" "access") "intersects" ("Pocket" "Pocket" "Node 11" "feature"))
(("Pocket" "Pocket" "Node 11" "blind-access") "through" ("component"))
(("CurvedSlot" "CurvedSlot" "Node 24" "blind-access") "throu
gh" ("component"))
(("CurvedSlot" "CurvedSlot" "Node 29" "blind-access") "through" ("component"))
(("CurvedSlot" "CurvedSlot" "Node 32" "blind-access") "through" ("component"))
(("CurvedSlot" "CurvedSlot" "Node 35" "blind-access") "through" ("component"))
)

The four holes at the bottom of the component are blind. This means they are also inaccessible because of the overhang. This is successfully detected and is shown in the first twelve relationships above, where the four holes have access problems with the blank, the component and the pocket in which they have been placed.

The curved slots that have been used to make the rounded external corners are identified as being accessible from either end, they are ‘through’ slots. In fact, they have many more access directions as they are on the edge of the component. These alternate access directions are not searched for.

6.3 Summary

The algorithms for thin walls, anteriority constraints and through hole detection have been demonstrated. A number of real components have been tested. Design failed entirely in one case where a more complex pocket shape was required and to a lesser degree in two other cases where peculiar geometry had been specified, probably in order to catch out feature recognition systems.

The analysis failed in one of the successfully designed case, but only due to the size of the problem. The algorithms can be optimised to consume less memory.

In all other cases the algorithms have been successful, and the number and type of relationships discovered shows the importance of these detection algorithms for process planning.

7 Conclusions

This thesis has shown that for successful process planning it is necessary to generate anteriority constraints between features.

Anteriority inferencing algorithms have been developed and in the absence of other constraints the anteriority constraints result in valid process plans for manufacture of components is successful.

This is confirmed through the test cases in Chapter 6.

The FODDS2 Feature Based Design System is capable of designing a significant range of mechanical components and that the inferencing algorithms do indeed detect anteriority errors.

NC code can be generated from the plans so produced and this has been tested

The modelling method used, that of negative features as a Minkowski sum of tool and toolpath is a powerful method and new features can be added with a modicum of effort. The algorithms demonstrated will all continue to work with new features so designed.

FODDS2 is a complete CAD/CAM system focussing on Geometric Reasoning for Process Planning. That is, there is substantial work on a Feature Based Design front-end, allowing design of real mechanical components in a near professional environment. There is a Geometric Reasoning Subsystem capable of inferring additional knowledge about a design, not given explicitly by the designer but necessary for the process planner. Lastly, there is a CAM system [Holz98] capable of taking this knowledge and generating NC code suitable for the mill in the department.

The process planning system itself will be strengthened by ongoing work by Mill and Naish [Nais98].

The algorithm for proximity detection leaves much to be desired. An alternative approach using Minkowski sums and medial axes shows more promise.

7.1 Summary of Conclusions

A functioning Feature Oriented Detail Design System has been developed.

The system has been proven with a set of test components from a variety of sources.

The system satisfies the criteria laid down by Shah [Shah95] for a Feature Based System.

All features have been implemented using Minkowski sums in a manner sufficiently general to allow easy addition of features. Treating the Swept Volumes as Minkowski sums and the use of Minkowski sums in proximity detection is a novel approach. The use of Swept Volumes is in line with the work of Kramer [Kram92] and Sungurtekin [Sung86].

The mathematical specification of the geometric reasoning is for the most part new and is an extension of the component in terms of features specification of Requicha and extends or modifies the work of Vandenbrande, Sungurtekin and Kramer. The specification ensures an elegant and robust method of describing the geometric reasoning with predictable results, and allows the feature set to be easily extended with no reduction in generality.

A drawback of this method is that alternate approach directions other than the principal access direction, and in the case of holes, the opposite direction are not considered. This is a significant drawback for setup planning, but would require some dynamic feature redefinition to solve.

The set of anteriority constraints produced by the Geometric Reasoning is suitable for input into a CAPP system such as HAPPI or its descendant.

7.2 Further Research

There are a large number of areas of further research, some of which are already being tackled.

1. Addition of more powerful features
2. Additional feature validation.

3. Feature relaxation techniques.
4. Alternate feature view. Work from Little addresses some of this. An interface to a feature recognition system would be of interest.
5. Agent based approaches to speed up the overall system.
6. Improved tolerancing mechanism
7. Parametric and constraint modelling
8. Geometric Algorithms

7.2.1 Addition of More Powerful Features

A number of form features suitable for inclusion in the system have been identified from the list in Table 2 Chapter 3, and others have been identified from the test objects. Particularly these include pockets with complex profiles including rounded polygons and splines and would allow those test pieces that were not completely realisable to be completed.

7.2.2 Additional Feature Validation

Though some feature validation methods have been included in FODDS2 they are not an exhaustive set. To develop the feature based design part of the system, additional feature validation methods would aid the user in producing valid designs. The feature validation methods in Bidarra's Spiff system [Bida97] could be incorporated.

7.2.3 Feature Relaxation Techniques

A particular drawback of the current system is its insistence on one representation for each feature and a restricted number of approach directions. Work in the area of feature relaxation, to allow alternate representations of features and thus allow alternate access directions, or to allow modification of the height or depth of the machined volume in order to effectively modify the anteriority constraints without changing the component geometry, could be effective.

7.2.4 Alternate Feature Views

Though some work by Little [Litt97a] has addressed the feasibility of allowing design by positive features and transforming those features into the negative manufacturing features necessary for FODDS2, it is clear more work could be done in this area.

7.2.5 Agent Based Approaches to Speedup the Overall System.

Achieving simultaneous design and planning, in order to decrease lead time still further, and provide immediate feedback of planning and machining problems to the designer through the design cycle is already the subject of work by Salmon and Jacquel [Jacq97][Jacq98], through the adoption of a fine-grained agent-based approach. Other work in this area includes [Bala96].

7.2.6 Improved Tolerancing Mechanism

A manufacturing oriented tolerancing mechanism has been implemented without the full rigour of a complete tolerancing mechanism. Further work in this area should be undertaken based on the ongoing work of Voelcker [Voel97].

7.2.7 Parametric and Constraint Modelling

The design system can be described as a parametric design system, but adding a constraint-based mechanism would greatly increase its power and flexibility whilst leaving the geometric reasoning and downstream applications largely unaffected.

7.2.8 Geometric Algorithms

The work on computational geometry tools such as Minkowski sums, medial axes and other algorithms such as convex hulls and Delaunay triangulation, to allow modification of features and component geometry in a way not supported by current generation's of commercial solid modellers is an area of great interest to the author and is the subject of a draft grant proposal.

References

- [Abel96] Abelson, H, Sussman, GJ, Sussman, J, *Structure and Interpretation of Computer Programs*, MIT Press, 1996.
- [Agos97] d'Agostino, N, *Feature Based Modelling of Parts with Sculptured Surfaces*, [http://www.ifw.uni-hannover.de/bereich5/forschen/ 56_1e.htm](http://www.ifw.uni-hannover.de/bereich5/forschen/56_1e.htm), 1997.
- [Ande90] Anderson, DC, Chang, TC, *Geometric Reasoning in Feature-Based Design and Process Planning*, Computers & Graphics 14(2), pp225-235, 1990.
- [Arik92] Arikan, MAS, Totuk, OH, *Design by Using Machining Operations*, Annals of the CIRP 41(1), 1992, pp185-188.
- [Arms94] Armstrong, Cecil G, *Modelling Requirements for Finite-Element Analysis*, Computer-Aided Design, Volume 26 Number 7, July 1994 pp573-578, Butterworth-Heinemann.
- [Arms97] Armstrong, C, Bowyer, A, Cameron, S, Corney, J, Jared, G, Martin, R, Middleditch, A, Sabin, M, Salmon, J, Woodward, J, *Djinn: A Geometric Interface for Solid Modelling – Part I, Functionality Rationale*, Project Report EPSRC Grant GR/K 00841 (to be published as a book in 1999).
- [Bala96] Balasubramanian, S and Norrie, DH, A, *Multiagent Architecture for Concurrent Design, Process Planning, Routing, and Scheduling*, Concurrent Engineering: Research and Applications, 4(1) pp7-16, March 1996.
- [Berg97] de Berg, M, van Kreveld, M, Overmars, M, Schwarzkopf, O, *Computational Geometry: Algorithms and Applications*, Springer 1997.

-
- [Bida97] Bidarra, R, Dohmen, M, Bronsvort, WF, *Automatic Detection of Interactions in Feature Models*, proceedings of the 1997 ASME Design Engineering Technical Conferences September 14-17, 1997, Sacramento California.
- [Boog94] Boogert, RM, *Tool Management in Computer Aided Process Planning*, PhD thesis, University of Twente, Enschede, Netherlands, 17th June 1994.
- [Bow95] Bowyer, A, Cameron, S, Jared, G, Martin, R, Middleditch, A, Sabin, M, Woodwark, J, *Introducing Djinn: A Geometric Interface for Solid Modelling*, Information Geometers Ltd, Winchester, ISBN 1-874728-08-9, 1995.
- [Bron93] Bronsvort, Willem F and Jansen, Frederik W, *Feature Modelling and Conversion - Key Concepts to Concurrent Engineering*, Computers in Industry 21 (1993) 61-86, Elsevier, 1993.
- [Brun95] Brunetti, G, De Martino, T, Falcidieno, B, Hassinger, S, *A Relational Model for Interactive Manipulation of Form Features Based on Algebraic Geometry*, proceedings of Solid Modelling '95, Salt Lake City, Utah, pp95-103, 1995.
- [Burk97] Burkhart, R., *Schedules of Activity in the Swarm Simulation System*, position paper for OOPSLA'97 workshop on Object Oriented Behavioural Semantics, Oct 5-9, 1997, Atlanta, Georgia.
- [Butt86] Butterfield, W.R. et al., *Part Features for Process Planning*, CAM-I report R-85-PPP-03, 1986.
- [Case93] Case, K, Gao, J, *Feature Technology: An Overview*, International Journal of Computer Integrated Manufacturing, 7(2) pp77-99, 1993.
- [Case94] Case, K, Gao, JX, Gindy, NNZ, *The Implementation of a Feature-Based Component Representation for CAD/CAM Integration*, proceedings of the Institute of Mechanical Engineers, Journal of Engineering Manufacture, Vol.208 B1, pp.71-80, 1994.
-

-
- [Case97] Case, K, Wan Harun, W.A., *A Representation of Assembly and Process Planning Knowledge for Feature-based Design*, proceedings of Advances in Manufacturing Technology XI, NCMR13, Glasgow Caledonian University, pp73-78, September 1997.
- [Cham93] Chamberlain, MA, Joneja, A and Chang, T-C, *Protrusion-Features Handling in Design and Manufacturing Planning*, Computer Aided Design, volume 25 number 1, pp19-28, Butterworth-Heinemann, January 1993.
- [Chan81] Chang, TC, Wysk, RA, *An Integrated CAD/Automated Process Planning System*, AIIE Transactions, pp223-233, September 1981.
- [Chan85] Chang, TC and Wysk, RA, *An Introduction to Automated Process Planning Systems*, Prentice-Hall, 1985. (229 pages)
- [Chax94] Chaxel, F, Bajic, E, and Richard, J, *From STEP Product Modelling to Product Manufacturing: an Approach using Identification Tags*, European Workshop on Integrated Manufacturing Systems Engineering, Grenoble (France), December 12-14, 1994.
- [Chia97a] Chia, SC, Mill, FG, Salmon, JC, *Geometrical Approach to Fixture Planning in a Features Defined Environment*, MPGIR36, Edinburgh University, 1997.
- [Chia97b] Chia, SC, *Fixture Planning in a Feature Based Environment*, PhD thesis, Edinburgh University, 1997.
- [Corn90] Corney, J, Clarke, DER, *A Feature Recognition Algorithm for Multiply Connected Depressions and Protrusions in 2½D objects*, proceedings of International Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM/SIGGRAPH, ACM Press, pp171-183, June 1991.
- [Corn97] Corney, Jonathan, *3D modeling using the ACIS kernel and toolkit*, Wiley 1997.
-

-
- [Cutk88] Cutkosky, MR, Tenenbaum, JM, Muller, D, *Features in Process-based Design*, proceedings of Computers in Engineering 1988, San Francisco, CA, USA, pp557-562, 31 July - 04 August 1988.
- [Cutk91] Cutkosky, MR, Tenenbaum, JM and Brown, DR, *Working with Multiple Representations in a Concurrent Design System*, Galley proof, ASME Journal of Mechanical Design, 1991.
- [DeFa93] De Fazio, TL et al., *A Prototype of Feature-Based Design for Assembly*, Journal of Mechanical Design, Vol.115, pp723-734, December 1993.
- [DeJo94] DeJong, Mitchell T and Fuchs, Alex, *PART White Paper*, ICEM Technologies, Minnesota, USA, 1994.
- [DeKr95] De Kraker KJ, Dohmen M, and Bronsvort WF (1995) *Mutiple-way Feature Conversion to Support Concurrent Engineering*, proceedings of Solid Modeling '95, Third Symposium on Solid Modeling and Applications, 17-19 May 1995, Salt Lake City, USA, Hoffman C, and Rossignac J (eds), ACM Press, New York, pp 105-114, 1995.
- [Denz93] Denzel, H and Vosniakos, G-C, *A Feature-Based Design System and its Potential to Unify CAD and CAM*, proceedings of IFIP workshop in Interfaces in Industrial Systems for Production and Engineering, IGD, Germany, March 15-17, 1993.
- [Desc81] Descotte_Y, Latombe_JC, *GARI: A Problem Solver that Plans how to Machine Mechanical Parts*, proceedings of the 7th International Joint Conference on Artificial Intelligence, Vol.2, pp766-772, Vancouver, BC, Can, 24-28 Aug 1981.
- [Dilg93] Dilger, Werner and Kassel, Stephen, *A Distributed AI System for Job Shop Control*, Proceedings of Industrial and Engineering Applications of AI and Expert Systems 6, Edinburgh, June 1-4, 1993.
- [Dowl94] Dowlatshahi, Shad, *A Comparison of Approaches to Concurrent Engineering*, International Journal of Advanced Manufacturing Technology (9) pp106-113, Springer-Verlag, 1994.
-

-
- [Erve88] Erve, AH van't, *Generative Computer Aided Process Planning for Part Manufacturing*, PhD thesis, University of Twente, Enschede, 1988.
- [Giac90] Giacometti, F., Chang, T.-C., *A Model for Parts, Assemblies and Tolerances*, Proceedings, Workshop on Design for Manufacturing, Enschede, 1990.
- [Gind89] Gindy, N.N.Z., *A Hierarchy of Form Features*, International Journal of Production Research, Vol 27 No.12, pp2089-2103, 1989.
- [Gind92] Gindy, NNZ, *A Product Data Model for Computer-Aided Process Planning Systems*, International Conference on Manufacturing Automation, University of Hong Kong, pp428-433, 1992.
- [Glas84] Glassner, A.S., *Space Subdivision for Fast Ray Tracing*, IEEE Computer Graphics and Applications, pp15-22, October 1994,
- [Güle93] Gülesin, M and Jones, RM, *Stock Selection and Blank Part Modelling in an Expert Process Planning System*, proceedings of Industrial and Engineering Applications of AI and Expert Systems 6, Edinburgh, June 1-4, 1993.
- [Gupt95] Gupta, S.K., Nau, D.S., *A Systematic Approach for Analyzing the Manufacturability of Machined Parts*, Computer Aided Design, 27(5) pp343-352, 1995.
- [Gupt97] Gupta, SK, Das, D, Regli, WC, and Nau, D, *Automated Manufacturability Analysis: A Survey*, Research in Engineering Design, 9(3), 1997.
- [Henz95] Henzold, G, *Handbook of Geometrical Tolerancing - Design, Manufacturing and Inspection*, Wiley, ISBN 0-471-94816-0, 1995
- [Holz98] Holznagel, U., *NC Code Generation from FODDS Plans*, ERASMUS Project Report, Edinburgh University, February 1998.
- [Houn97] Hounsell, M.S., *Morphological and Volumetrical Feature-based Designer's Intents*, proceedings of Advances in Manufacturing Technology XI, NCMR13, Glasgow Caledonian, pp64-68, Sept 1997.
-

-
- [Hout89] van Houten, FJAM, van't Erve, AH, *PART; a Parallel Approach to Computer Aided Process Planning*, 4th International Conference of Computer Aided Production Engineering, pp281-288, Edinburgh 1989.
- [Hout91] van Houten, FJAM, *PART: A Computer Aided Process Planning System*, PhD thesis, University of Twente, Enschede, Netherlands, 2 May 1991.
- [Husb88] Husbands, P., *The Optimisation of Process Plans*, Manufacturing Planning Group, Internal Report No.3, Edinburgh, March 1988.
- [Husb89] Husbands, P., Mill, F., Warrington, S.W., *Part Representation in Process Planning for Complex Components*, Geometric Reasoning, ed. Woodwark, Open University Press, pp203-210, ISBN 0-19-853738-7, 1989.
- [Husb90] Husbands, P., Mill, F., Warrington, S.W., *Generating Optimal Process Plans from First Principles*, in Expert Systems for Management and Engineering, ed. Balagurusamy, E. and Howe, pp130-152, J.A.M., Ellis Horwood, ISBN 0-13-296690-5, 1990.
- [Husb91] Husbands, P., Mill, F.G. Pedley, A.G., Warrington, S.W., *The Edinburgh Composite Component*, proceedings of Manufacturing Technology International Conference (Holland) (MSTF'91), Enschede, 1991.
- [Jack80] Jackins, CL, Tanimoto, SL, *Octrees and their Use in Representing Three Dimensional Objects*, Computer Graphics and Image Processing 14, pp249-270, 1980.
- [Jacq97] Jacquel, D., Salmon, J.C., Mill, F.G., *Features as Autonomous Agents: An Alternative Paradigm for Concurrent Engineering*, International Conference for Manufacturing Automation 1997, Hong Kong, pp1087-1092, April 1997.
- [Jacq98] Jacquel, D., Salmon, J.C., *Solving the 3D Localization Problem for Feature Agents*, INCOM'98 - 9th Symposium of the International Federation of Automatic Control on INformation COntrol in Manufacturing Systems III, pp141-146, Nancy & Metz (FRANCE) , 24-26 June 1998.
-

- [Jare89] Jared, G., *Recognizing and Using Geometric Features*, Geometric Reasoning, ed. Woodwark, pp169-188, ISBN 0-19-853738-7, Open University Press, 1989.
- [Jone93] Jones, R, Mitchell, S and Newman, S, *Feature-based systems for the Design and Manufacture of Sculptured Products*, International Journal of Production Research, vol.31, no.6, pp1441-1452, 1993.
- [Jonk92] Jonkers, Frank, *A Software Architecture for CAPP systems*, PhD thesis, University of Twente, Enschede, Netherlands, 16th April 1992.
- [Josh87] Joshi, S, Chang, TC, *CAD Interface for Automated Process Planning*, Proc. 19th CIRP Int. Seminar on Manufacturing Systems, University Park, Pennsylvania, June 1987.
- [Kaul92] Kaul, Anil, *Minkowski Sums: A Simulation Tool for CAD/CAM*, Computers in Engineering Vol.1, ASME, pp447-456, 1992
- [Kram91] Kramer, TR, *The Off-line Programming System (OLPS) A Prototype STEP-Based NC-Program Generator*, proceedings of Product Data Exchange for the 1990s, New Orleans, Louisiana, NCGA, Vol. 2, February 1991.
- [Kram92] Kramer, TR, *Issues Concerning Material Removal Shape Element Volumes (MRSEVs)*, NIST Internal Report 4804, Gaithersburg, MD, March 1992.
- [Krip94] Kripac, Jiri, *Topological ID System - A Mechanism for Persistently Naming Topological Entities in History-Based Parametric Solid Models*, PhD Dissertation, Czech Technical University in Prague, 1994.
- [Laak93] Laako, T, Mäntylä, M, *Feature Modelling, by Incremental Feature Recognition*, Computer Aided Design Journal, Vol.25 No.8, pp479-492, August 1993.
- [Laak96] Laakko, T., Mäntylä, M., *User-defined Features in EXTDesign++*, ASME Computers in Engineering Conference, 1996.

-
- [Lend94] Lenderink, Albert, *The Integration of Process and Production Planning in Small Batch Part Manufacturing*, PhD thesis, University of Twente, 28 April 1994.
- [Litt97a] Little, G., Mill, F.G., Salmon, J.C., *Transforming Positive Features within a Feature Based Design System*, MPGIR 35, Edinburgh, 1997.
- [Litt97b] Little, G, Tuttle, R, Clark, DER, Corney, J, *The Heriot-Watt Featurefinder: A Graph-Based Approach to Recognition*, Proc. ASME Design Engineering Technical Conferences, Sacramento, California, September 14-17, 1997.
- [Mänt88] Mäntylä, M, *An Introduction to Solid Modelling*, Computer Science Press, 1988.
- [Mänt89] Mäntylä, M, Opas, J., Puhakka, J., *Generative Process Planning of Prismatic Parts by Feature Relaxation*, proceedings of the 15th ASME Design Automation Conference, Montreal, pp49-60, 1989.
- [Maro95a] Maropoulos, PG, *Review of Research in Tooling Technology, Process Modelling and Process Planning Part I: Tooling and Process Modelling*, Computer Integrated Manufacturing Systems, Vol.8 No.1, pp5-12, Elsevier, 1995.
- [Maro95b] Maropoulos, PG, *Review of Research in Tooling Technology, Process Modelling and Process Planning Part II: Process Planning*, Computer Integrated Manufacturing Systems, Vol.8 No.1, pp13-20, Elsevier, 1995.
- [Mars95] Marshall, S and Griffiths, JG, *A New Cutter Path Construction Technique for Milling Machines*, International Journal of Production Research, No.6, pp1723-1736, Taylor & Francis Ltd. , 1995.
- [Mart96] Martin, Edward C., *Getting started with ACIS using Scheme*, Fort Lauderdale: Schemers Inc., 1996.
- [Meag82] Meagher, D, *Geometric Modelling using Octree Encoding*, Computer Graphics and Image Processing 19 pp129-147, 1982.
-

-
- [Midd88] Middleditch, *The Representation and Manipulation of Convex Polygons*, Theoretical Foundations of Computer Graphics and CAD (NATO ASI Series F, Volume 40) pp211-252, ed. Earnshaw, R., Springer Verlag, 1988.
- [Midd97] Middleditch, A, Reade, C., *A Kernel for Geometric Features*, proceedings of the Fourth Symposium on Solid Modelling and Applications, Atlanta, Georgia, pp131-140, May, 1997.
- [Mill93] Mill, F.G., Pedley A.G., Salmon J.C., *Representation Problems in Feature Based Approaches to Design and Process Planning* International Journal of Computer Integrated Manufacturing, vol. 6, Nos 1 & 2, pp27-33, Jan/April 1993.
- [Mill94] Mill, FG, Naish, JC and Salmon, JC, *Design for Machining with a Simultaneous Engineering Workstation*, Computer Aided Design, Vol. 26 No. 7, pp521-527, July 1994.
- [Mill96] Mill, F, Naish, JC, Rieken, HR, Salmon, JC, Warrington, SW, *Industry Based Survey of Feature Oriented Engineering*, (submitted to the Journal of the IMechE)
- [Nais97] Naish, JC, Mill, FG and Salmon, JC., *Implementation of Process Capability Models to Support Computer Aided Process Planning*, Advances in Manufacturing Technology XI, Proceedings of the XIIIth National Conference on Manufacturing Research, pp375-379, Glasgow Caledonian University, Glasgow, 9-11 September, 1997.
- [Nais98] Naish, JC, *Process Capability Modelling for Manufacturing Process Selection in an Integrated Simultaneous Engineering Workstation*, Ph.D Thesis, University of Strathclyde, 1998.
- [Opas94] Opas, J, Kanerva, F, Mantyla, M, *Automatic Process Plan Generation in an Operative Process Planning System*, International Journal of Production Research, 32(6), pp1347-1363, 1994.
-

-
- [Ovtc92] Ovtcharova, J, Pahl, G, Rix, J, *A Proposal for Feature Classification in Feature-Based Design*, Computing and Graphics 16(2), pp187-195, 1992.
- [Prat84] Pratt, M.J., *Solid Modelling and the Interface Between Design and Manufacture*, IEEE Computer Graphics and Applications, 4(7) pp52-59, July 1984.
- [Regl95] Regli, W.C., Gupta, S.K., Naud, D.S., *Extracting Alternative Machining Features: An Algorithmic Approach*, Research in Engineering Design, 7, pp173-192, 1995.
- [Requ89] Requicha, AAG and Vandenbrande, JH, *Form Features for Mechanical Design and Manufacturing*, pp47-52, Computers in Engineering 1989.
- [Salm94] Salmon, J.C., *Component Description Language Definition*, MPGIR 13, Edinburgh University, March 1994.
- [Salo93a] Salomons, OW, van Houten, FJAM, Kals, HJJ, Review of Research in Feature Based Design, Journal of Manufacturing Systems, 12(2), pp113-132, 1993.
- [Salo95] Salomons, Otto, *Computer Support in the Design of Mechanical Products*, PhD thesis, University of Twente, Enschede, Netherlands, 20th January 1995.
- [Shah91a] Shah, J.J., *An Assessment of Features Technology*, Computer Aided Design 23(5) pp58-66, 1991.
- [Shah91b] Shah, Jami, Sreevalsan, Palat and Abraham, Matthew, *Survey of CAD/Feature-Based Process Planning and NC Programming Techniques*, Computer Aided Engineering Journal, pp25-33, Feb 1991.
- [Shah91c] Shah, JJ, Matthew, A, *An Experimental Investigation of the STEP Form Features Information Model*, Computer Aided Design 23(4) pp282-292, 1991.

-
- [Shah95] Shah, Jami J, Mäntylä, *Parametric and Feature-based CAD/CAM: concepts, techniques and applications*, Wiley & Sons, NY, ISBN 0-471-00214-3, 1995,
- [Shah97] Shah, JJ, Dedhia, H, Pherwani, V, Solkhan, S *Dynamic Interfacing of Applications to Geometric Modeling Services via Modeler Neutral Protocol*, Computer Aided Design, 29(12): pp811-824, Dec 1997.
- [Shap95] Shapiro, V, Vossler, DL, *What is a Parametric Family of Solids?*, Proceedings of 3rd ACM/IEEE Symposium on Solid Modeling and Applications, Salt Lake City, Utah, May 17-19, 1995.
- [Subr95] Subrahmanyam, S, DeVries, W, Pratt, MJ, *Feature Attributes and Their Role in Product Modelling*, Solid Modeling '95, Salt Lake City, Utah, pp115-124, May 17-19, 1995.
- [Sung86] Sungurtekin, Ugur A and Voelcker, Herbert B, *Graphical Simulation and Automatic Verification of NC Machining Programs*, IEEE, 1986.
- [Tam91] Tam, TKH, Armstrong, CG, *2D Finite Element Mesh Generation by Medial Axis Subdivision*, Advances in Engineering Software, 13(5/6) pp313-324, 1991.
- [Tutt97] Tuttle_R, Little_G, Clark_DER, Corney_J, *Feature Recognition for NC Part Programming*, International Conference on Manufacturing Automation '97, pp797-802, Hong Kong, April 1997.
- [Vand91] Vandenbrande, J.H., *Automatic Recognition of Machinable Features in Solid Models*, PhD Thesis, University of Rochester, 1991.
- [Vand93] Vandenbrande, JH, Requicha, AAG, *Spatial Reasoning for the Automatic Recognition of Machinable Features in Solid Models*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(12), pp1269-1285, Dec 1993.
- [Vane90] Van Emmerik, *Interactive Design of Parametrized 3D Models by Direct Manipulation*, PhD Thesis, Delft University Press, Netherlands, 1990.
-

-
- [Voel97] Voelcker, H, *The Current State of Affairs in Dimensional Tolerancing*, ICMA'97, pp33-40, Hong Kong April 1997.
- [Wils89] Wilson, PR, Pratt, M.J., *A Taxonomy of Features for Solid Modelling*, ACM SIGGRAPH, , Boston Massachusetts 1989.
- [Wils90] Wilson, PR, *Feature Modelling Overview*, Dallas, TX, 1990.
- [Wing91] Wingard, L, *Introducing Form Features in Product Models, A Step towards CAD/CAM with Engineering Terminology*, Licentiate Thesis, Dept. of Manufacturing Systems, Stockholm, 1991.
- [Yama84] Yamaguchi, K, Kunii, TL, Fujimura, K, *Octree-Related Data Structures and Algorithms*, IEEE Computer Graphics and Applications, pp53-59, Jan 1984.
- [Yang97] Yang, ZX, Joneja, A, Zhou, J, *Backward Growing-based Geometric Reasoning for Manufacturing Feature Recognition*, International Conference on Manufacturing Automation '97, Hong Kong, pp791-796, 1997.

Appendix A. FODDS2 Implementation

Details

This Appendix contains brief details of the computer systems under which the original FODDS and the FODDS2 systems were built.

In addition the GUI of FODDS2 is explained.

The first version of FODDS was written in C++ on a Sun (Unix) workstation with a GUI written in X/Motif and using ACIS as the kernel modeller. FODDS2 retains ACIS as the kernel modeller, but has moved platforms to PCs under Windows95/NT4.0 and has been entirely rewritten using Visual Basic for the GUI, Access for the database kernel and Scheme⁴ and the ACIS 3D Toolkit for solid modelling and geometric reasoning. These can be used in a largely coherent development environment with Scheme/ACIS code tested under the 3D Toolkit before being incorporated in the application. The system runs happily on a P133 with 48Mb of RAM running Windows95 (minimum spec. 32Mb), but realistically needs around 100Mb of free hard disk for the necessary applications and a minimum of 100Mb of swap space to deal with reasonable sized components.

The system uses Dynamic Data Exchange (DDE) to form the link between both Visual Basic and Access (through a set of VB functions) and between Visual Basic and ACIS. In particular this means that the commands used to drive ACIS are identical to those typed at the command line of the ACIS 3D Toolkit (renamed for ACIS version 3.0 the Scheme ACIS Interface Driver Extension). This allows code to be developed using Scheme independently of the GUI. The enforced separation of

⁴ Scheme is a close relative of Lisp, and is often argued to be both a more compact and elegant language.

GUI and core code improved both sets of code, as there was no mismatch of function between the two. This is exemplified by the fact that the move from ACIS 2.1 to ACIS 3.0 was accomplished in half a day and required no code rewriting or recompiling. This is despite huge changes in the way ACIS is distributed (now through DLLs rather than static libraries). Instead the new binaries were used ‘straight out of the box’. The software packages used in FODDS2 are listed below.

Visual Basic Pro v5.0, Microsoft.

Access v7.0, Microsoft.

ACIS v4.2, Spatial Technology Inc.

Scheme, Schemers Inc.

Scheme ACIS Interface Driver Extension, v4.2, Spatial Technology Inc.

System Architecture

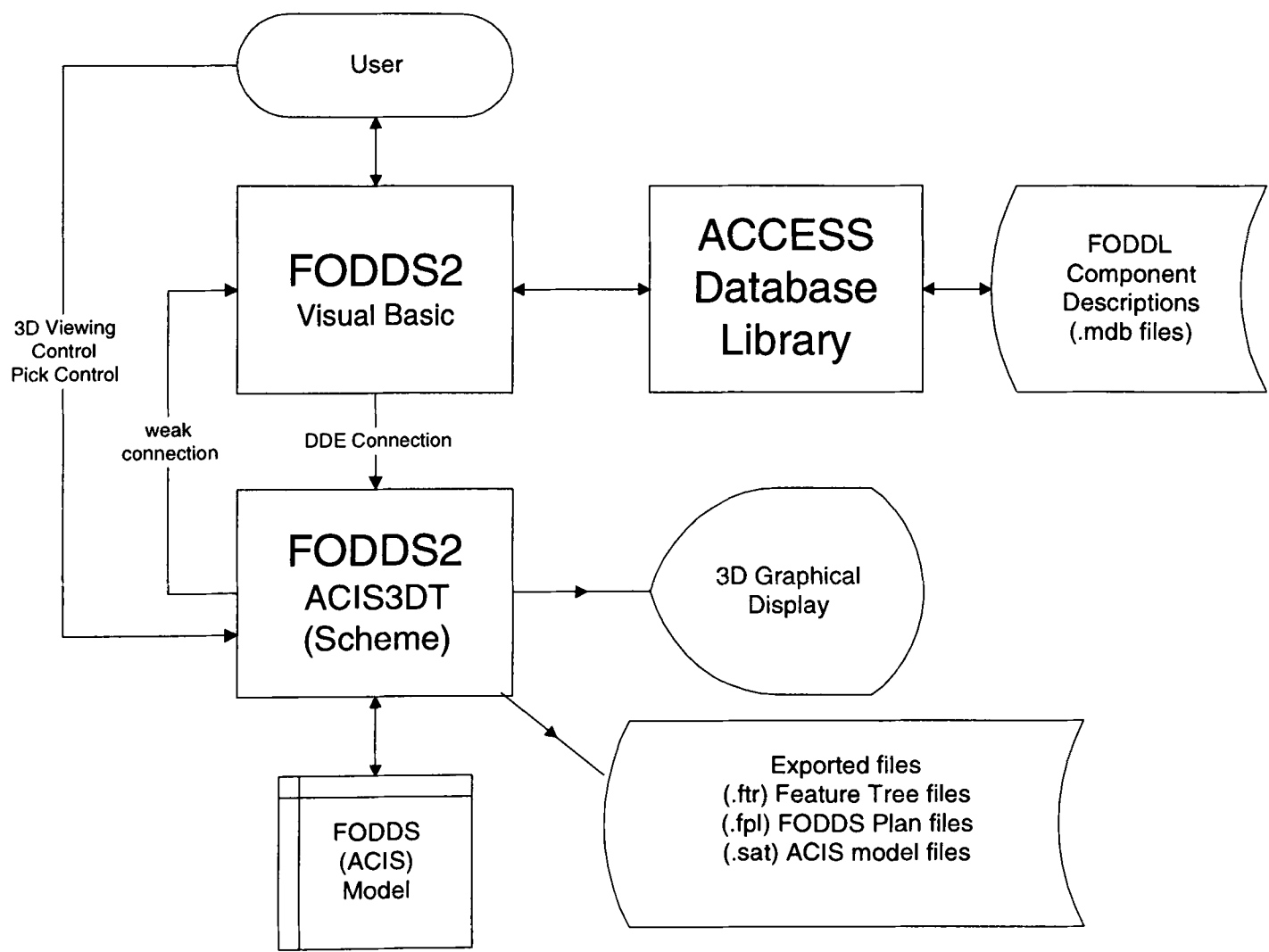


Figure A.1 System Architecture

Figure A.1 shows the system architecture in terms of the modules used. The User interacts chiefly with the FODDS2 User Interface. This in turn modifies a database containing the component definition and informs the 3D Toolkit about the current state of the database. The database contains no solid models of the component, but contains sufficient information in a feature based table to rebuild the component at any time.

The 3D Toolkit handles modelling the component and displaying the component on the screen. The toolkit is also primarily responsible for geometric reasoning about the component. The Toolkit can also write component descriptions to file (i.e. including feature information), ACIS bodies of any of the generated bodies including most importantly the final component, and writes files describing the results of the geometric reasoning that has taken place.

Components can be displayed and rotated in real-time using a wireframe or shaded OpenGL mode, and can be rendered.

The FODDS2 Graphical User Interface

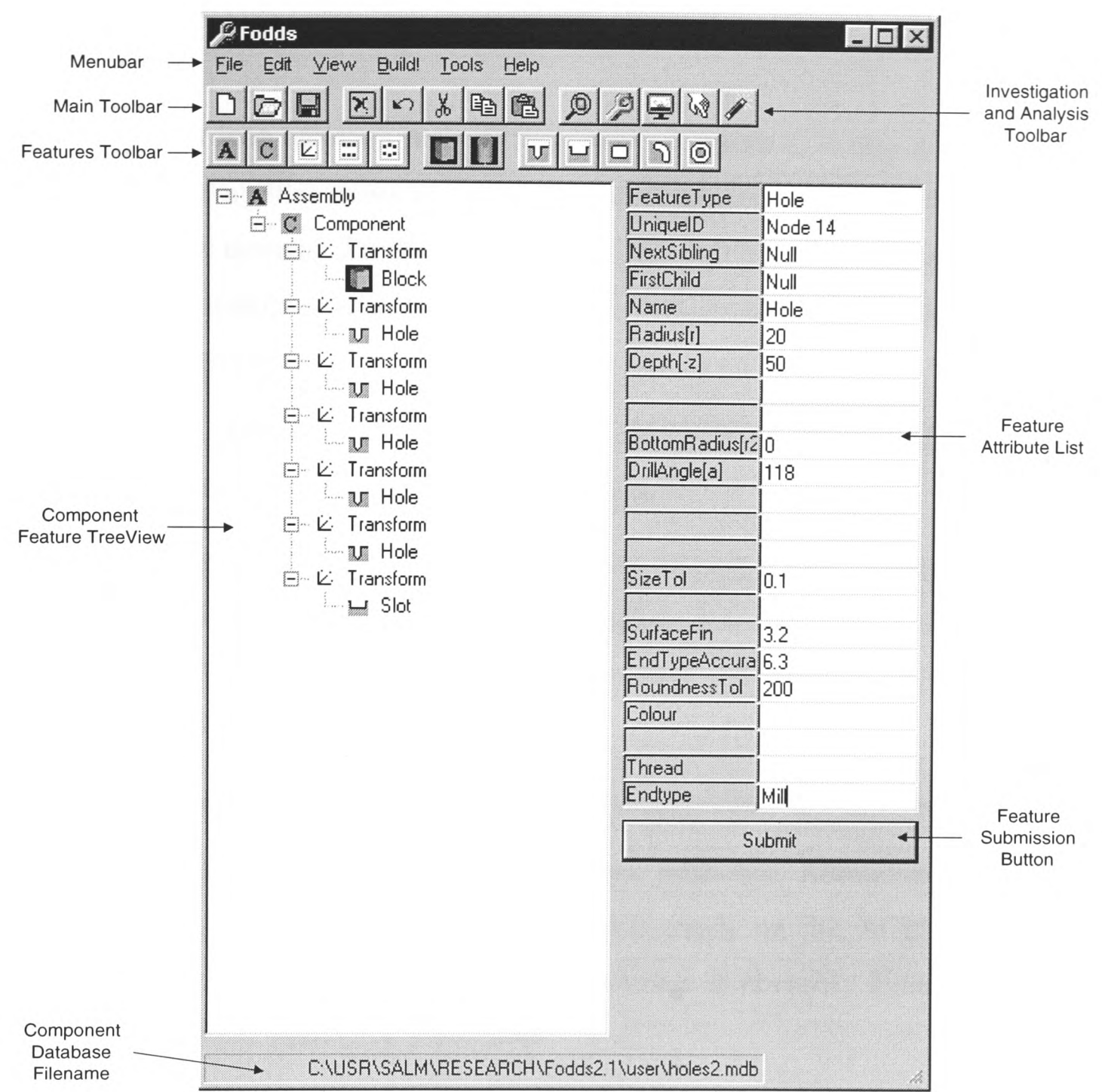



Figure A.2 The FODDS2 Graphical User Interface


On the interface are a number of buttons in the main toolbar, with conventional icons and usage, namely:


New File, File Open, File Save working as expected on Component database files.


Delete, Undo, Cut, Copy and Paste, acting on nodes in the feature tree.


The investigation and analysis toolbar however contains novel buttons.

 The magnifying glass zooms to cause the current component to centre and fill the current viewports. Additional panning and zooming is possible using the mouse and keyboard.

 The spanner icon causes a build of the current component to take place. Because no usage of history files exists in the prototype, a build of a complex component can take up to a few minutes. To allow rapid design, users can enter a number of changes (in the form of feature modification, addition or deletion) and select a suitable time to rebuild the component.

 The render button invokes photorealistic rendering of the current view. Most images in this thesis have been generated this way.

 The pick button once selected allows the user to pick a face of the current view and information in a dialogue is supplied regarding that face. This makes mistakes easier to identify, helping to close the representation gap between the Feature Tree and the Component subsequently designed.

 Lastly the magic wand icon causes all analysis algorithms to be performed.

All these commands are also available from the menu bar. Additionally, the same effect can be generated by typing suitable commands on the ACIS 3D Toolkit command line interface, though the toolkit interface is normally hidden. This was used to demonstrate individual algorithms.

FODDS2 Data Structure

The primary FODDS2 data structure is a tree and is stored primarily as a table of nodes in an ACCESS database.

Each node represents, in general, either a primitive feature or an operator on a set of features.

Each node contains three data items specifying its position in the tree.

- A Unique Node ID (UNID) (must be present and unique to the tree)
- The UNID of the Next Sibling of that node in the tree (this is set to a NULL value if there is no Next Sibling)

- The UNID of the First Child of that node (this is set to a NULL value if there is no First Child)

This is sufficient information to model a tree, and is compatible with the TreeView Widget in Visual Basic (4.0 and later), which thus allows easy on-screen representation of the feature hierarchy of any component.

In addition, each node contains some domain specific information, in particular identifying the type of node.

Lastly, each node contains a fixed number of attributes. These are used to carry data depending on the node type.

The specification for each type of node is held in one table in the database known as the CODLSpec Table. This is so called as it embodies the historical Component Description Language [Salm94] [Mill93]. Tables A.1 and A.2 show the specification of the Feature Operators and Primitives respectively.

In Table A.1 the specifications for operators unite, intersect and subtract can be seen, the set operators. These can be seen only in the table, but not in the GUI. At the present time, the Scheme code supports these operators, but they would allow the user to break the Feature Based Design model if they were made available. A development of the system would allow design of the initial workpiece with these operators, or the addition of a feature recognition module would allow use of these CSG operators.

Table A.1 The FODDS2 Feature Operators

Key	Assembly	Component	Transform	PCD	Matrix	Intersect	Unite	Subtract
UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD
FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType
SuperType	Operator	Operator	Operator	Operator	Operator	Operator	Operator	Operator
NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling
FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild
Name	Name	Name	Name	Name	Name	Name	Name	Name
NodeAttr01	OODLver		Xtranslation	Radius[r]	Xpitch			
NodeAttr02			Ytranslation	AngularPitch[a]	Ypitch			
NodeAttr03			Ztranslation	Nolterns	NolternsX			
NodeAttr04			VecX		NolternsY			
NodeAttr05			VecY		Filled?			
NodeAttr06			VecZ					
NodeAttr07			Angle(deg)					
NodeAttr08								
NodeAttr09								
NodeAttr10			PositionTol	SizeTol	SizeTol			
NodeAttr11			AngleTol	AngleTol				
NodeAttr12								
NodeAttr13								
NodeAttr14								
NodeAttr15	Author	Author						
NodeAttr16	Title	Title						
NodeAttr17	Date	Date						
NodeAttr18	OtherInfo	OtherInfo						

Table A.2 The FODDS2 Primitives

Key	CurvedSlot	Pocket	Slot	Hole	Ring	Block	Cylinder
UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD	UniquelD
FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType	FeatureType
SuperType	NegFeature	NegFeature	NegFeature	NegFeature	NegFeature	PosFeature	PosFeature
NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling	NextSibling
FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild	FirstChild
Name	Name	Name	Name	Name	Name	Name	Name
NodeAttr01	CentralRadius[r]	Length[x]	Length[x]	Radius[r]	InnerRadius[r0]	Length[x]	Radius[r]
NodeAttr02	Width[y]	Width[y]	Width[y]	Depth[-z]	OuterRadius[r1]	Width[y]	Height[z]
NodeAttr03	Depth[-z]	Depth[-z]	Depth[-z]	ThreadType	Depth[-z]	Height[z]	
NodeAttr04	FinishAngle[a]	CornerRadius[r]		ThreadDepth			
NodeAttr05	BottomRadius[r1]	BottomRadius[r1]	BottomRadius[r1]		BottomRadius[r2]		
NodeAttr06							
NodeAttr07							
NodeAttr08							
NodeAttr09							
NodeAttr10	SizeTol	SizeTol	SizeTol	SizeTol	SizeTol	SizeTol	SizeTol
NodeAttr11	AngleTol				AngleTol		
NodeAttr12	SurfaceFin	SurfaceFin	SurfaceFin	SurfaceFin	SurfaceFin	SurfaceFin	SurfaceFin
NodeAttr13							
NodeAttr14							
NodeAttr15	Colour	Colour	Colour	Colour	Colour	Colour	Colour
NodeAttr16							
NodeAttr17							
NodeAttr18				Endtype			

In addition to those features listed in the table above, a number of features have been acquired from other sources, particularly Mandelli. Though the Mandelli features have not been included in the current system, they show how the system might be expanded to take into account application and company specific features. It was felt important to ensure that the generic features were fully understood before going on to include more specific features with more complex geometry. In particular many of the Mandelli features are examples of compound features and can be integrated using the compound feature mechanism rather than generating new feature primitives.

Display of the component can take one of three forms depending on user preference:

- a wireframe view - supporting fast real-time rotation
- an OpenGL view - supporting slightly slower real-time rotation but with colour-coded surfaces

- a rendered view - using a number of light sources and material and transparency effects this produces realistic component views for both clarity and product demonstration⁵. Generation of a large rendered view can take a few minutes.

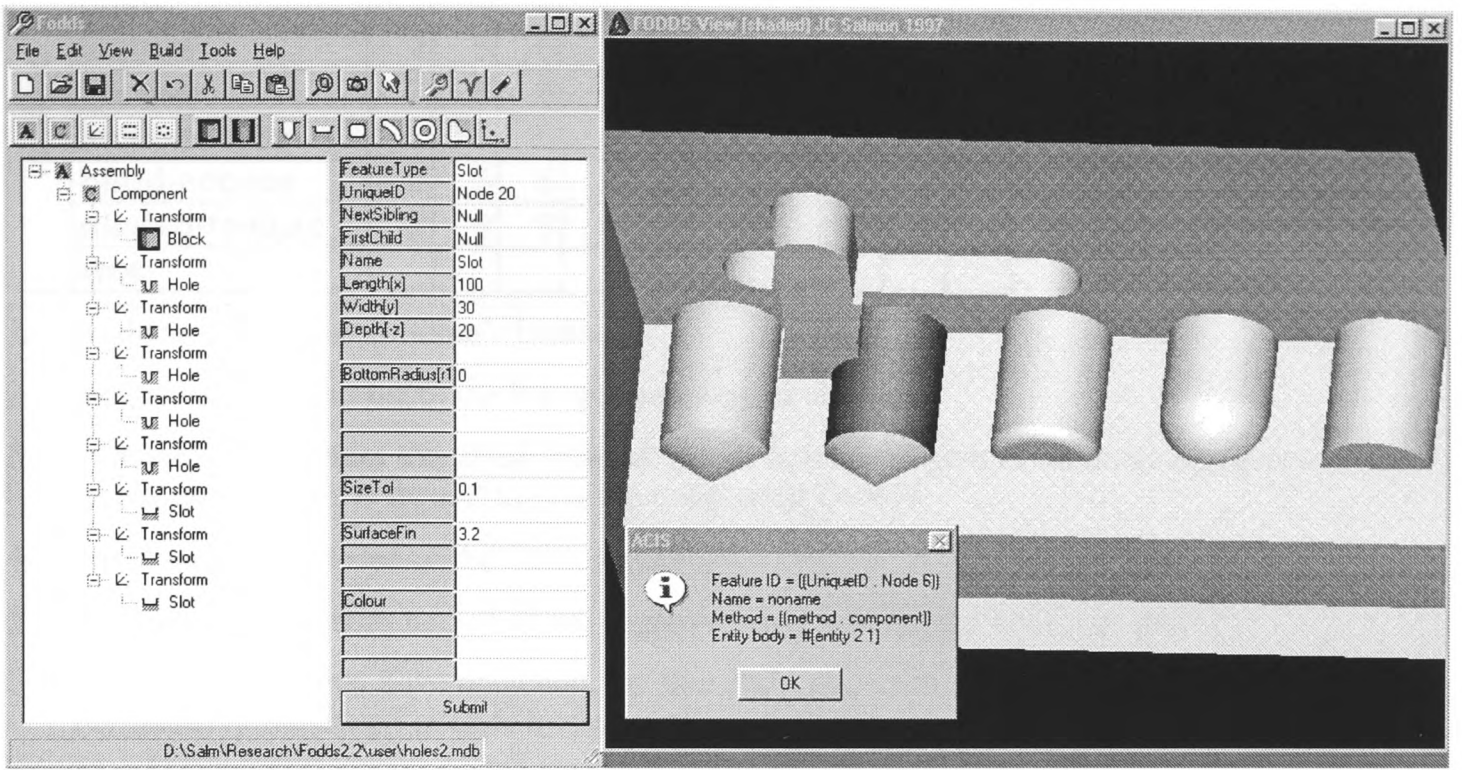


Figure A.3 The Pick Operation

The user can also pick any feature on the screen and its identity and other information is relayed to the user, allowing easy editing of obviously misplaced features (see Figure A.3).

Implementation of the Method/Featuretype call

For each of the methods that produce volumes for the geometric algorithms it is possible to call the method with a feature type as mentioned in the table below. The behaviour of the method/featuretype combination is described in the accompanying notes. The note is given by the number in the method/featuretype matrix of Table A.3.

Table A.3 The Featuretype/Method Matrix

⁵ Most of the rendered images in this thesis have been generated directly from the software.

		feature type									
		assembly	component	transform	block	cylinder	hole	slot	pocket	curved-slot	ring
method	component	1	3	6	7	7	7	7	7	7	7
	blank	1	4	6	7	7	8	8	8	8	8
	feature	2	5	6	8	8	7	7	7	7	7
	access	2	5	6	8	8	9	9	9	9	9
	local-access	2	5	6	8	8	9	9	9	9	9
	blind-access	2	5	6	8	8	9	9	9	9	9
	dilated-feature	2	5	6	8	8	9	9	9	9	9
	wcs										

1. produces a list of its children's bodies

2. produces a flattened list of the lists generated by its children

3. to make a component this must unite all *blank* features to give *P*, unite all negative features to give *N* and subtract *N* from *P* to give the component $C=P-N$.

4. this is merely the union of the blank features *P*.

5. this is an *entlist* (entity list) of all the negative features or feature body (not united)

6. a transform can only have one child, but that child may return either a single entity or an *entlist*. Fortunately, transforms operate transparently on these two types.

7. returns bodies as expected

8. returns an empty list

9. returns a single body appropriate to feature type and method.

Table A.4 Features and Nodes in FODDS2

Name	Icon	# subnodes	Comment
Reserved		0	Reserved
Hole		0	Negative Design Feature
Slot		0	Negative Design Feature
Rectangular Pocket		0	Negative Design Feature
Curved Slot		0	Negative Design Feature
Ring		0	Negative Design Feature
Complex pocket		n	Negative Design Feature
2D point		0	2D Point Definition
Block		0	Geometric Primitive
Cylinder		0	Geometric Primitive
Sphere		0	Geometric Primitive
Cone		0	Geometric Primitive
Torus		0	Geometric Primitive (not imp.)
Sculptured Surface		n	Unimplemented
Reserved		n	Reserved
Component op		1	This is the root of any component
Compound Feature op		1	For a user-defined features
Subtract		n	CSG Boolean
Unite		n	CSG Boolean
Intersect		n	CSG Boolean
Coordinate transform		n	Geometric Op
Matrix		1	Manufacturing Design Op
pitch circle diameter		1	Manufacturing Design Op

An important question is “what impact does this tree structure have on subsequent downstream tasks such as process planning?”. The answer is that the structure here is of little impact. The tree structure can be envisaged as merely a design data structure in which case the tree structure can be reinterpreted as a flat structure. This can be done by writing the design structure into another tree structure with the same underlying properties, but where the tree is flattened by evaluating some of the nodes between the top level and the leaf nodes, particularly multiple transforms. This is done at present when producing the ‘.fpl’ process plan structure.

The impact of this tree structure on the subsequent process planning is an important consideration, but is found to be slight. The process planning system will predominantly plan single features and then perform some (global) optimisation in order to produce the optimal process plan. Whereas this optimal process plan may contain some of the structure of the design, there should be no constraint that this is the case. Indeed process planning systems that generate process plans for

manufacturing features in the order in which they are designed are likely to be seriously sub optimal, and frequently no valid process plan can be generated in this manner. However, because much previous work has been carried out on a flat feature structure [Mill93] there exists a function that takes one feature tree and produces a flat feature tree that evaluates to the same component.

Tree Building, Editing and Interrogation

Just allowing the existence of a tree structure is not sufficient. Tools are required to manipulate this tree structure. Fortunately, by adopting the use of Access to hold the tree nodes, and by making use of the built in functionality of the Tree Widget to handle trees, much of the tree handling functionality can be handled by Visual Basic and Access. All that must be ensured is that the *FirstChild* and *NextSibling* attributes in the TreeView widget are kept synchronised with the nodes in the database, and that any time any node is modified in the database, the entire contents of the node are passed through the DDE to the mirror of the data structure held in Scheme.

The Scheme Mirror

Using Visual Basic and Access is fine, but a mechanism is needed to hold the same datastructure in Scheme, though the tools to edit this datastructure need not be so sophisticated as entire nodes of the tree can be modified at once.

The data structure found in Scheme therefore is a list of nodes (in Scheme as in Lisp, almost everything is a list).

The definition for one particular node, that of the hole in the 'ell' example discussed previously, is given in Figure A.4 below.

The node contains a string with the Unique Node Identifier, and then a list of the attributes associated with that node. Unlike the format held in the Access database (with a fixed number of nodes in a fixed position) this version of the tree can handle any number of attributes, and consists of key-value pairs. All values are held as strings whether they are strings or numbers so the possibility of adding parametric functions instead of single values is available for the future.

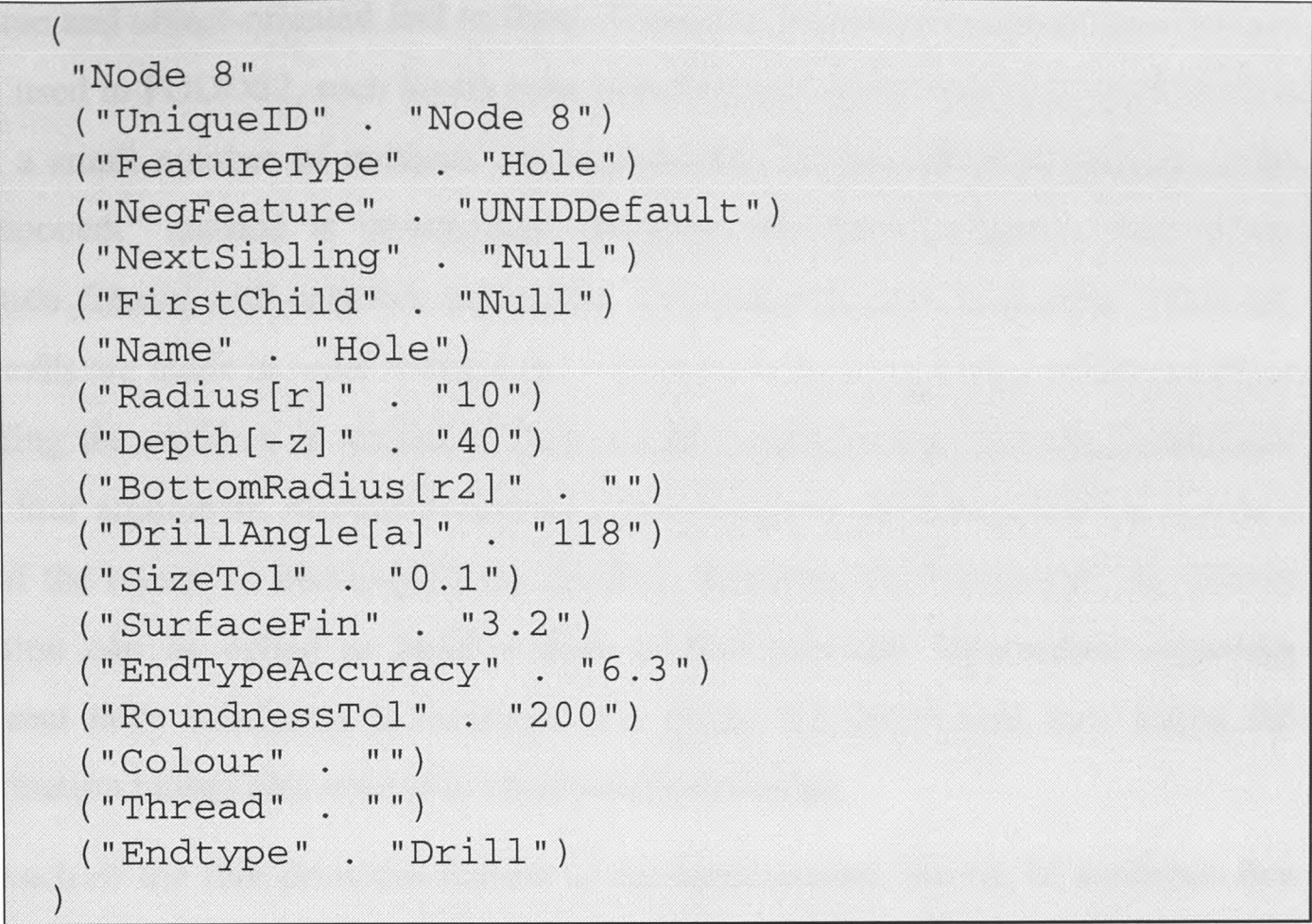


Figure A.4 Representation of a FODDS2 Node in Scheme

A deliberate consequence of this structure is that no explicit information is held in a node as to whom its parent node is. This is chiefly to avoid redundant information, as when a tree is evaluated we rarely need to climb up in the tree. When the possibility of functions attached to attributes is added the ability to browse around the tree in a more flexible manner is required, but this is still possible without the knowledge of parents via a search from the root node downwards for any particular node. The time taken to browse a tree this way is still likely to be small compared with the time taken to execute the solid modelling functions, so the gain of no redundancy and with it consistency, makes this worthwhile.

If run-time speed of FODDS2 is to be improved the lazy-evaluation schemes and a history (or subtree preservation) mechanism need to be investigated rather than re-evaluating the entire tree every time a modification is made.

Building Features

In order to build the tree, the root object has its build method called. In order to do this, it must first build its children and so on down the tree. The tree can be built in a number of different ways depending on the method called. This gives the nodes in

the tree and object-oriented feel to them. Though a full object-oriented layer has not been used in FODDS2, such layers exist [Abel96] and could have been used. As it is only a small number of methods are required for the tree. Here the action of the “component” method is investigated. Building any node requires a call to the function *f2build* with only two arguments, the *method* and the *nodename*. From this first calls are made in order to build the *children* of a node, and a list of the results of building the *children* is obtained. Then a call is made to the function *f2nodebuild* with four arguments, *method nodename featuretype* and *children*. *Children* is now a list of the results of building all the children. Knowing the *featuretype*, the correct function can be called to build a node of that subclass. Information regarding different node subclasses is localised in a single file *f2feats.scm*, thus aiding the information hiding idea central to object-oriented design.

For each of the five primitive feature so far implemented, the list of attributes that have to be recovered from the node and the specification of the geometry of the feature so built is shown previously in **Error! Reference source not found.**

Operator or Branch Node Generation Functions

All these operator nodes take lists of subnodes. This must be a list of nodes, though it can be an empty list. Where the description says subnode, the function still takes a list of nodes, but for these functions to be evaluated properly this must be a singleton list. Though it is legal to give these functions empty lists, all require non-empty lists to function properly, though all are guaranteed not to fail on empty lists. Similarly those expecting lists of two or more items will return sensible results for singleton lists.

Assembly	Merely combines the output of all the Components into one list
Component	If the method is component then this operator subtracts the feature list from the blank list, otherwise it returns the list of the objects below it.
Subtract	This performs a regularised Boolean subtract from the first subnode of all subsequent subnodes. This implies that subnodes are ordered in the tree.

Unite	This performs a regularised Boolean unite of its subnodes
Intersect	This performs a regularised Boolean intersect of its subnodes.
Transform	This generates a coordinate transform node composed of a translation by some vector consisting of 3 cartesian components, and a rotation about an arbitrary axis defined by 3 cartesian components by an arbitrary number of degrees. Effectively, the rotation is performed first about the global origin and the translation is then performed relative to the global axes. (or the origin and axes of the transform higher up in the tree)
Matrix	Creates a matrix of the child feature given an x and y spacing, number of items along the x and y axis, and a flag to indicate whether the matrix is filled or empty.
Pitch Circle	Creates a repeated feature around the boundary of a circle. The radius is specified along with the number of copies and the angular spacing between copies.

The Requirements of a Feature Oriented Design System

The following is a list of desirable features of a Feature Oriented Design System and a mechanism by which such a system can be built. This list has been derived from a study performed by the Manufacturing Planning Group *entitled Feature Oriented Engineering - An Assessment* [Mill96]

- User Defined Features – No set of predefined features will be complete for any domain.
- Compound Features – Provides a simple mechanism for many user defined features
- Parametric Design/Constraints
- Opportunities for Lazy Evaluation

It can be shown that the tree structure provides a simple structure through which all the above requirements can be satisfied.

We can split the tree structure into two main components

- Leaf nodes consisting only of primitive features or geometric building blocks
- Non-terminal nodes consisting of operators

The operators include the traditional regularised Booleans of unite, intersect and subtract, so it could be argued that we have generated a CSG tree via the back door. This may be true, in which case when editing the tree some restrictions must be imposed such as not being able to add subtractive features.







Other operators are also included. These include operators for transforming objects (restricted to translation and orientation transforms)

Other operators allow generation of multiple features, such as a matrix of holes or a pitch-circle of holes.

Editing facilities allow cut, copy and pasting of subtrees throughout the overall component tree. This introduces questions of identification. Each node in the tree is allocated a unique integer to allow pointers to nodes to function properly. As nodes are added and deleted, and particularly as subtrees are imported, maintaining these unique nodes becomes problematic.

A solution involves a two part naming scheme. Each node has two identifiers, either of which can be empty. One identifier identifies the type as the user sees it and the other a unique name for that particular instance of the type. Problems that arise are similar to that of handling a disk directory structure, often requiring unique file names in each directory, but also requiring that the operating system has some unique way of referring to the location of a file or directory.

Even when this problem is solved, there remains the problem of ensuring consistency of relationships, i.e. geometric tolerances between features and anteriority constraints between features. The current system does not attempt to retain these relationships after editing of the feature tree. For anteriority constraints these can always be rebuilt when required. Geometric tolerancing that has been specified by the user is harder to rebuild and some consistency control is needed.

Every Tree starts with an  Assembly icon, so any tree can be an assembly of any number of components (though this is limited to a simple component in the current implementation. At the next level in the tree is a list of  components (again, just one currently). The Ell component consists of three features, a  block,  slot and  hole, each attached to the tree via a  transform. This allows each feature to have its position and orientation modified without having to modify the node containing the individual feature parameters.

In fact a transform operator can be placed at almost any position in a feature tree, and multiple transforms can be chained and affect one or more features. This allows the common functionality associated with a transform to be separated from the features, and so avoid unnecessary code replication. In addition, allowing multiple transforms means that a complex transform can be decomposed into simpler transforms. This is particularly useful if a set of features have an unusual approach direction for instance, where a transform can set up a new working coordinate system, and subsequent transforms can be defined relative to this. This approach became particularly useful when building the Heriot-Watt 2 component (see Chapter 4).

Appendix B. Relationship Lists

This appendix contains some of the sets of relationships discovered for the test components in Chapter 6, particularly when those lists do not warrant inclusion in the main thesis.

Han1 Relationships

```
(
  ("Pocket" "Pocket" "Node 12" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 18" "access") "intersects" ("blank"))
  ("Slot" "Slot" "Node 26" "access") "intersects" ("blank"))
  ("Slot" "Slot" "Node 28" "access") "intersects" ("blank"))
  ("Slot" "Slot" "Node 30" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 41" "access") "intersects" ("blank"))
  ("Slot" "Slot" "Node 43" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 53" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 55" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 57" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 59" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 62" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 64" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 66" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 68" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 70" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 72" "access") "intersects" ("blank"))
  ("Hole" "Hole" "Node 74" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 77" "access") "intersects" ("blank"))
  ("Pocket" "Pocket" "Node 12" "access") "intersects" ("component"))
  ("Slot" "Slot" "Node 43" "access") "intersects" ("component"))
  ("Pocket" "Pocket" "Node 45" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Hole" "Hole1" "Node 47" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Hole" "Hole" "Node 49" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Pocket" "Pocket" "Node 57" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Pocket" "Pocket" "Node 64" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Pocket" "Pocket" "Node 77" "access") "intersects" ("Pocket" "Pocket" "Node 12"
"feature"))
  ("Hole" "Hole" "Node 16" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Pocket" "Pocket" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Slot" "Slot" "Node 28" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Slot" "Slot" "Node 30" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Hole" "Hole" "Node 59" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Pocket" "Pocket" "Node 62" "access") "intersects" ("Pocket" "Pocket" "Node 14"
"feature"))
  ("Slot" "Slot" "Node 30" "access") "intersects" ("Hole" "Hole" "Node 16"
"feature"))
)
```



```
((("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 16"
"feature"))
(("Pocket" "Pocket" "Node 62" "access") "intersects" ("Hole" "Hole" "Node 16"
"feature"))
(("Hole" "Hole" "Node 20" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Slot" "Slot" "Node 26" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Slot" "Slot" "Node 28" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Hole" "Hole" "Node 53" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Hole" "Hole" "Node 55" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Pocket" "Pocket" "Node 18"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 20"
"feature"))
(("Pocket" "Pocket" "Node 51" "access") "intersects" ("Hole" "Hole" "Node 20"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Hole" "Hole" "Node 20"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Hole" "Hole" "Node 16" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Slot" "Slot" "Node 30" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Pocket" "Pocket" "Node 62" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Hole" "Hole" "Node 70" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Hole" "Hole" "Node 72" "access") "intersects" ("Pocket" "Pocket" "Node 22"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Hole" "Hole" "Node 20" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Hole" "Hole1" "Node 47" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Pocket" "Pocket" "Node 51" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Pocket" "Pocket" "Node 57" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Pocket" "Pocket" "Node 64" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Slot" "Slot" "Node 26"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Slot" "Slot" "Node 28"
"feature"))
(("Hole" "Hole" "Node 16" "access") "intersects" ("Slot" "Slot" "Node 30"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Slot" "Slot" "Node 30"
"feature"))
(("Pocket" "Pocket" "Node 62" "access") "intersects" ("Slot" "Slot" "Node 30"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Pocket" "Pocket" "Node 22" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Pocket" "Pocket" "Node 34" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Hole" "Hole" "Node 66" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Hole" "Hole" "Node 68" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Hole" "Hole" "Node 70" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Hole" "Hole" "Node 72" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
```



```
((("Hole" "Hole" "Node 74" "access") "intersects" ("Pocket" "Pocket" "Node 32"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Hole" "Hole" "Node 16" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Slot" "Slot" "Node 30" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Pocket" "Pocket" "Node 32" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Hole" "Hole" "Node 41" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Pocket" "Pocket" "Node 62" "access") "intersects" ("Pocket" "Pocket" "Node 34"
"feature"))
(("Pocket" "Pocket" "Node 39" "access") "intersects" ("Slot" "Slot" "Node 37"
"feature"))
(("Hole" "Hole" "Node 41" "access") "intersects" ("Slot" "Slot" "Node 37"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Slot" "Slot" "Node 37"
"feature"))
(("Hole" "Hole" "Node 41" "access") "intersects" ("Pocket" "Pocket" "Node 39"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 41"
"feature"))
(("Pocket" "Pocket" "Node 39" "access") "intersects" ("Hole" "Hole" "Node 41"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 41"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Slot" "Slot" "Node 43"
"feature"))
(("Hole" "Hole" "Node 41" "access") "intersects" ("Slot" "Slot" "Node 43"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 45"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole1" "Node 47"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 49"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Slot" "Slot" "Node 26" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Slot" "Slot" "Node 28" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Hole" "Hole" "Node 53" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Hole" "Hole" "Node 55" "access") "intersects" ("Pocket" "Pocket" "Node 51"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 53"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 55"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Pocket" "Pocket" "Node 14" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Slot" "Slot" "Node 26" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Slot" "Slot" "Node 30" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Slot" "Slot" "Node 37" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Pocket" "Pocket" "Node 39" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Pocket" "Pocket" "Node 57"
"feature"))
```



```
((("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 59"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Hole" "Hole" "Node 59"
"feature"))
(("Slot" "Slot" "Node 28" "access") "intersects" ("Hole" "Hole" "Node 59"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Pocket" "Pocket" "Node 22" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Pocket" "Pocket" "Node 32" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Pocket" "Pocket" "Node 34" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Hole" "Hole" "Node 41" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 62"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Slot" "Slot" "Node 26" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Slot" "Slot" "Node 37" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Pocket" "Pocket" "Node 39" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Pocket" "Pocket" "Node 77" "access") "intersects" ("Pocket" "Pocket" "Node 64"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 66"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 66"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 68"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 68"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 70"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 70"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 72"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 72"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Hole" "Hole" "Node 74"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Hole" "Hole" "Node 74"
"feature"))
(("Pocket" "Pocket" "Node 12" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Pocket" "Pocket" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Slot" "Slot" "Node 26" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Slot" "Slot" "Node 30" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Slot" "Slot" "Node 37" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Pocket" "Pocket" "Node 39" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Slot" "Slot" "Node 43" "access") "intersects" ("Pocket" "Pocket" "Node 77"
"feature"))
(("Pocket" "Pocket" "Node 12" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 16" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 20" "blind-access") "through" ("component"))
(("Pocket" "Pocket" "Node 34" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 41" "blind-access") "through" ("component"))
(("Slot" "Slot" "Node 43" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 49" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 53" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 55" "blind-access") "through" ("component"))
(("Hole" "Hole" "Node 59" "blind-access") "through" ("component"))
```

```
(( "Pocket" "Pocket" "Node 62" "blind-access") "through" ("component"))
)
```

Gadh2 Relationships

```
(
  (( "Hole" "Hole" "Node 16" "access") "intersects" ("blank"))
  (( "Hole" "Hole" "Node 17" "access") "intersects" ("blank"))
  (( "Hole" "Hole" "Node 18" "access") "intersects" ("blank"))
  (( "Hole" "Hole" "Node 19" "access") "intersects" ("blank"))
  (( "Hole" "Hole" "Node 16" "access") "intersects" ("component"))
  (( "Hole" "Hole" "Node 17" "access") "intersects" ("component"))
  (( "Hole" "Hole" "Node 18" "access") "intersects" ("component"))
  (( "Hole" "Hole" "Node 19" "access") "intersects" ("component"))
  (( "Hole" "Hole" "Node 16" "access") "intersects" ("Pocket" "Pocket" "Node 11 "
"feature"))
  (( "Hole" "Hole" "Node 17" "access") "intersects" ("Pocket" "Pocket" "Node 11 "
"feature"))
  (( "Hole" "Hole" "Node 18" "access") "intersects" ("Pocket" "Pocket" "Node 11 "
"feature"))
  (( "Hole" "Hole" "Node 19" "access") "intersects" ("Pocket" "Pocket" "Node 11 "
"feature"))
  (( "Pocket" "Pocket" "Node 11" "blind-access") "through" ("component"))
  (( "CurvedSlot" "CurvedSlot" "Node 24" "blind-access") "throu
gh" ("component"))
  (( "CurvedSlot" "CurvedSlot" "Node 29" "blind-access") "through" ("component"))
  (( "CurvedSlot" "CurvedSlot" "Node 32" "blind-access") "through" ("component"))
  (( "CurvedSlot" "CurvedSlot" "Node 35" "blind-access") "through" ("component"))
)
```

HAPPI Relationships

```
(
  ((Slot MidSlot Node 10 access) intersects (blank))
  ((Slot BtmSlot Node 12 access) intersects (blank))
  ((Hole LtlHole Node 16 access) intersects (blank))
  ((Slot MidSlot Node 10 access) intersects (Slot TopSlot Node 8 feature))
  ((Slot BtmSlot Node 12 access) intersects (Slot TopSlot Node 8 feature))
  ((Slot BtmSlot Node 12 access) intersects (Slot MidSlot Node 10 feature))
  ((Hole LtlHole Node 16 access) intersects (Pocket Pocket Node 14 feature))
  ((Hole BigHole Node 6 blind-access) through (component))
)
```