15

A Feature Recognition Project

Paul Gibson, Hossam Ismail and Malcolm Sabin
Department of Industrial Studies, University of Liverpool

Abstract

This paper describes the results from a research project into Feature Recognition, carried out at the University of Liverpool, Department of Industrial Studies between April 1994 and July 1996, which was funded by the Engineering and Physical Sciences Research Council of the United Kingdom under grant GR/J/97458.

After a brief exposition of the ongoing need for feature recognition, even within a design-by-features context, the paper identifies previous work most influential in this project, and then presents the project objective of building a programmable feature recogniser in which the input feature descriptions do not need either to pander to the specific data structures of the solid (or other) modeller holding the domain, or to express a recognition algorithm.

The characterisation of features is then discussed and the range of domains in which recognition might need to take place. This leads on to the presentation of a basic, correct but inefficient algorithm for generic recognition, and a substantial section on how the inefficiency can be rectified while retaining correctness.

1 WHY FEATURE RECOGNITION ?

When feature recognition research started, solid models were solid models, representing geometry, but nothing else. Any application which needed to access that geometry intelligently had problems, because the structures giving explicit meaning to the shape were absent.

Now, however, we have feature-based modellers and they are used in design-by-features CAD systems. The argument is therefore that it should be unnecessary to recognise features; they should already be explicit in the CAD-system's database.

This is largely true. Although various kinds of hole are excellent examples for feature recognisers to use in demonstrations, they are totally unrealistic as examples of typical usage, because they will almost certainly have been designed as such.

M. J. Pratt et al. (eds.), *Product Modeling for Computer Integrated Design and Manufacture*

However, the conflict between the design community and the production engineering community over how feature-based design systems should be used stems from the fact that features are in the eye of the beholder. The designer sees a thin plate with stiffeners; the N/C part programmer sees a thick billet with pockets. Yes, the designer can see the pockets, too, but he doesn't want to design in terms of them. It is the thickness of the ribs that matters to him. Sometimes it matters to the N/C programmer, too, because the cutting forces have to be kept low in the region of a fragile rib.

Thus even when a part has been designed in terms of some set of features, other features emerge from the geometry when looked at in the right way, and a really intelligent CAD system will be able to notice them.

2 PREVIOUS LANDMARKS.

The work we have done has not been a reinvention of the wheel. As the next section will show, it is significantly different in objectives from most previous work. However, we have learned many things from the literature, and many of the ideas which we find useful are already there, though possibly viewed differently, in others' papers,

This section is not a full survey of the literature: our current bibliography contains over 350 entries in total, and even to list them would take more pages than are available. Rather it identifies the papers which guided our thinking and those where ideas that we rediscovered had already been published.

The first widely cited work in this area was that of Kyprianou (1980), whose PhD thesis, set the scene for much of the subsequent work. The motivation was probably the extreme frustration felt in the solid modelling community over the fact that mere 2D drafting systems could be used to support 2.5D NC machining quite effectively, by their use of profiles which a cutter could be driven along, while the much more powerful solid modellers, informatically more complete, were less usable in the machining context. Grayer's PhD thesis(1976), in which he recovered profiles from a solid model, may be seen as the precursor of the fuller feature recognition attempted later.

Woo's development of the ASVP algorithm (alternating sum of volumes with partitioning) for generation of CSG descriptions in either conjunctive or disjunctive normal forms (Woo 1982), similarly stemmed from the desire to make sense of neutral geometry which might not have been defined in terms of the cavities which the part-programmer sees (Woo 1977).

Following Kyprianou's thesis, Henderson and Anderson (1984) and Choi, Barash and Anderson (1984) used similar technology of crawling over a Brep model, using adjacency to find specific patterns of faces, edges and vertices, using convexity and concavity to steer the matching.

The following year the group led by de Floriani started to describe, in Ansaldi, de Floriani and Falcidieno (1985), how use of particular graph representations allowed use of powerful graph theoretic methods, and made some recognition much easier. A sequence of papers followed from this group, which were explicitly recognised as sources by other workers, such as Gavankar and Henderson (1990) and Corney and Clark (1991). The face adjacency hypergraph (FAH) became the preferred representation for the network of adjacencies which the boundary of a solid contains.

In the same proceedings as the Corney and Clark paper is one by Ames (1991), which introduced a lovely word for a very important concept, that of **featurettes**. Although in hindsight the idea here (of using several simple features in place of one complex one, and thus reducing the combinatorial complexity) can be seen to be foreshadowed in the work of Kaul (1983 and 1987), we did not find it anywhere else in the feature recognition literature. It is a key one for achieving good performance. Faux (1990) does suggest the splitting of features into parts, but his motivation is primarily not to lose the important tolerancing information, rather than to achieve combinatorial performance in recognition.

The Kaul work comes from a strand of computer science which was known to the group Kyprianou worked in. It views the problem as a grammatical one, and recognition as parsing. Shape grammars had been introduced as a concept by Stiny, who published properly in (Stiny 1991), but unfortunately Stiny himself was interested in them only as ways of generating interesting combinations of features, not in parsing them.

In fact the strong relationship between the graph representation of a feature, the expression of that feature as a set of rules and the grammatical formulation was pointed out by Wu and Liu, in a paper (Wu and Liu 1996) that appeared after we had done most of our work.

The grammatical approach leads to the thought on which we have leant strongly, that it might be possible to separate the recognition process from the description of what is to be recognised. Although many papers use grammatical methods to describe the features which their systems recognise, few invert this process and let the user define the feature which the system is to find. Notable exceptions are Shah and Rogers(1988), van't Erve (1989) and, more recently, d'Haeyer(1992) and Laako and Mantyla (1993). Even there, the feature definitions are not purely concerned with describing what the feature is in declarative terms.

Another rather important idea, which appeared right in the middle of a paper which addressed a lot else (Gadh and Prinz 1992), played the trick of limiting the number of edges that they had to consider by ruling out (and pruning from their graphs) all the edges which for some application reason could not become parts of the features which were being sought. Limiting the number of edges has a quite disproportionate effect on the computing times. This technique was also used by Corney and Clark.

More recently, the idea of achieving good performance by starting from relatively infrequent feature components and searching only locally to them has been given the names 'hints' and 'traces'. This has been fairly widely used, from the Xcap work at UMIST in the early 1980s onwards.

A totally different approach, becoming known as Woodwark's method, was introduced by Perry-Barwick and Bowyer (1995). This does not regard a feature as part of the boundary of an object, but as a volume, and the finding of an instance of a feature becomes looking for maximal intersections in a space of a very large number of dimensions. The view that a feature is a volume is one also found in the strand of work which built on Woo's ideas, developed, among others, by (Sakurai and Gossard,1990), (Dong, Parsaei and Gornet,1993)and (Vandenbrande and Requicha,1993).

Also notably different was the paper by Meeran and Pratt (1993) which describes recognition of features not in solid models, but in 2D drawings.

Although the mainstream of feature recognition has been based on adjacency of bounding elements in Brep solid models, and we started out by following this lead, this is not the only strand.

3 THE OBJECTIVES OF THIS PROJECT.

Because we were starting with the benefit of a lot of hindsight, we had a clear picture of the problems most in need of attention.

We felt that a major obstacle to most of the previous work was that each project had had to tackle three hard problems at the same time.

1. The first was the navigation of a particular modeller's data structures. Although most current (boundary) modellers can trace their roots back to Baumgart's winged edge structures, there are a number of variants used. Recent extensions to handle non-manifold shapes have generated truly baroque plexes. Tracing through such pointer structures is not exactly of the essence of feature recognition.
2. The second was the finding of a good feature recognition algorithm. The problem is combinatorially complex, and the search for reasonable performance is not trivial. As computers get faster and disks get bigger, the models within which features are to be recognised have got bigger and will continue to get bigger as designers put more and more detail into their models.
3. The third was the problem of accurately articulating what the feature is that you want to recognise. This is the hardest of all, for the very simple reason that the boundaries between what is an example of an X and what is not depend on who you are talking to and what aspects of X they are interested in. (Rosenman and Gero 1996) treats this at a general level, and (Wu and Liu 1996) gives some pointed examples of the way that a simple slot definition needs to be expanded noticeably to avoid the inclusion of configurations which are not intuitively slots. There is a great temptation to assume that what is recognized by a plausible algorithm must be the feature you want.

In fact the Face Adjacency Hypergraph stream of research separated out the first of these, by having a pre-process which converted whatever the model was held as into a form appropriate for the recognition algorithm.

Other projects focussed on performance. In particular the aspect graphs ideas used by (Gadh and Prinz,1992) have an important effect on the actual run times even though they do not alter the asymptotic complexity of the recognition process. Also the featurette ideas of (Ames,1991) provide an important tool for reducing the complexity exponent. When we started to articulate the issues in our own approach we came to both these ideas from a rather more generic viewpoint, but it is quite clear that the other authors cited got to essentially the same places first.

Our solution to these three problems was to throw all of them over the garden fence.

We said that because the writing of pre-processors was always possible and not a research topic, we would not spend time on it. Our project is built on a very flexible interpretive language within which whatever data structures are needed for today's purpose can be declared at run-time. The language also has the ability to parse fairly general user-defined syntax, and so we expect to be able to read the legible files of other systems with a minimal amount of editing,

We addressed performance by initially ignoring it, following Brooks' maxim of "Achievement mode first, then Performance", though in fact this paper will consider some ways in which good performance can be achieved systematically and generically.

Finally, we treated the fuzziness of the target as the main focus for the project. The definition of exactly what characterises the feature you are looking for has to be left to the user of our work.

The project's objective was therefore **to build a *programmable* feature recogniser which would be able to recognise, within a database built of instances of user-defined objects, instances of features matching a user-defined description.**

Further, the feature descriptions should be descriptions of the features, not thinly disguised recognition algorithms.

It may sound as if we traded a hard collection of problems for an impossible one, but in fact we were able to make useful progress.

3.1 Who is the User ?

In today's CAD world, there is a long supply chain. We envisage that software of the kind we have developed might well be a component assembled into the integrated CAD system by a system integration company which also uses specialist components provided by other companies for design analysis, process planning and many other subtasks. Each of these software houses has both core function developers and tuners who address the needs of specific customers by layers of software on top of the core function.

The task of getting the feature descriptions right is one which may largely fall to the tuners in these companies, but also possibly to those technical people in end user companies, referred to by Nardi in (Nardi 1993) as 'gardeners', who know the software aspects better than most end-user engineers, and who provide a level of adaptation to local needs.

4 WHAT IS A FEATURE ?

4.1 Definitions from the literature

Defining what the word 'feature' means, (not the same as defining the specific feature you want to recognize) is incredibly hard. Most of the papers in the literature attempt it, and usually fail by trying so hard to avoid being too specific, and thus missing some kind of feature, that they land up saying little that is actually useful.

Frequently the defining concept is that the feature has engineering significance. This is undoubtedly true, but not of much help in guiding a recognition project.

Also, quite often, the assumptions of the modeller underneath show through, so that a feature is 'a set of surfaces' or 'a volume'.

An important question when looking at such definitions is whether the features are perceived as forming a partition of the object (or of the surface of the object, or of the difference between the object and a stock volume enclosing it) or whether features can overlap and lead to parts of the object being unclassified. These are significant differences.

4.2 Our definition

The word '*feature*' is used at different times for three related but distinct concepts. Here are first some circular definitions to give you the feel of these concepts.

1. A **Feature Class** is a set of Feature Instances.
2. A **Feature Description** is the set of criteria whereby a candidate is tested for membership of a Feature Class, articulated in terms of the components of the candidate and their relationships.
3. A **Feature Instance** is an object which matches the criteria of the Feature Description

To be more precise, we say that a Feature Instance is

A structured collection of subfeature instances where those subfeatures already satisfy certain criteria and relationships.

The Feature Description sets out the structure of the collection of subfeatures and identifies the relationships between them and the criteria they have to satisfy. Clearly it must do this. For greatest ease in the use of a feature description language it should do nothing more. Ideally even the sequence of items and of criteria should be arbitrary.

At the start of the project we promised to define a feature description language. In fact we discovered that the EXPRESS entity description gives almost exactly what we need, and so we made a minimal set of changes to the relevant part of the EXPRESS language, to enable our parser to parse it. These changes were mainly concerned with maintaining a small window, so that parsing could create prompt diagnostics. (Others following our path should be recommended to use EXPRESS without modification.)

The definition of a feature identifies

1. a set of subfeatures, each being given a name.
   A subfeature instance can either be a leaf in the tree, (an object in the geometry model), or else a lower level feature with its own feature description. The facility to allow a feature instance to have another instance of the same feature class as a subfeature is an important one for defining features with an indefinite number of subfeatures.
   The structuring of the collection is essentially the naming of the subfeatures.
   These are denoted by attributes in the EXPRESS language, to which we prepend the token WITH to ease parsing.

2. a set of conditions which those named subfeatures must satisfy.
   Each condition is some expression evaluating to either true or false, written as a function of the subfeatures named in the WITH clauses.
   These appear as WHERE clauses.

4.3 Relationship to other Feature Description Languages

We are in the camp which regards features as independent of each other. We are not looking for a disjoint partitioning of the model, because the features which are seen by different users may well overlap. The side face of the pocket is also the side face of a thin wall.

We do not make the assumption (which Wu and Liu find in the methods which they studied) that all the subfeatures are of the same type.

We do not partition our criteria into topological and geometric. Indeed, we do not limit them to those two categories either. If some feature were to require that some subfeature had a particular tolerance or a particular colour, that would be equally acceptable as a criterion.

We do not ask the user to include within the feature description any issues of how to organise efficient recognition. The feature description itself should merely be a statement of what the feature is.

We focus on what is actually in the domain, and so the problem of the further combinatoric explosion due to each feature in a part being manufacturable in many different ways is not addressed per se.

4.4 What do you recognise features in ?

We originally expected to be finding features in solid models held as Boundary representations. We expected to be using primarily the Adjacency relation between solid and face, face and edge, and edge and vertex. (Hanrahan 1984)

During the course of the project our industrial partners provided us with examples, most of which were drawings for which they had DXF files. To our delight we discovered that our technology was as applicable to this data as to solid models.

We have also looked at higher level models where the actual geometry is not even explicit, and discovered that it works for those, too.

5 HOW DO YOU RECOGNISE THEM ?

We started by totally disregarding performance issues. This was quite deliberate.

Given the definition we had adopted for what a feature description is, the simplest algorithm which finds all instances in the database is just to generate all possible combinations of subfeatures which match the specified types and then just test each combination for all the criteria. Those that get through are created as feature instances.

Really dumb; really slow; but really simple, and trivially provably correct.

We call our system interpretive, but in fact it translates from the input character stream into an internal representation as the text is read. This internal form has parts for the schema, which holds the definitions of objects and features, for the grammar which holds the information for steering the parser, and a forest of execution trees for holding the bodies of macros.

We split the recognition task into two parts. In the first, the built-in MAKE FIND facility translates a schema entry into an execution tree, (in internal form) and hangs it on the syntax FIND X, where X is the name of the feature whose schema has been translated. The second part is the execution of this generated code.

Each WITH clause is translated into a FOR EACH loop which cycles through all instances of the requisite type. Once inside all the loops, each WHERE clause is translated into an IF THEN construction. Inside the innermost THEN is the command for creating the feature instance from the current collection of subfeatures (the loop variables of all the loops).

The proof that the code generated does indeed find all instances of the defined feature is so trivial that it is hard to articulate.

We have a facility for printing out the internal form of this execution tree so that we can check the translation process, but we do not have to re-parse any generated text.

6 PERFORMANCE ISSUES REVISITED

Although our starting point ignored performance, that position cannot be maintained for long. The combinatorics are just too serious to ignore.

Suppose that the database contains L lines, and that a feature is defined in terms of eight of them. Then a database of only 1000 lines requires the checking of $L^8 = 1000^8 = 10^{24} = 2^{80}$ combinations, which at only a millisecond each would take longer than most of us want to wait.

There are a number of ways of trimming this figure, some in the feature recognition literature, some in the literature of query optimisation in relational databases, some in both, but under different names.

- strength reduction
  This is a standard compiler optimisation technique. It consists of moving anything out of a loop which is always going to give the same answer, and so any IFs which do not reference the innermost loop variable can be migrated out of that loop, thereby being executed much less often. Further, and even more important, moving an IF can also have the benefit that when the test fails, the execution of complete inner loops is also avoided.
  In our terms this is the migration of WHEREs up the WITH list to the point that the next WITH defines a subfeature referenced inside the WHERE. This is a simple shuffling of records in a chain.
- loop resequencing
  Because the only assignment we are making in this nested loop structure is a single assignment at the very kernel, we can freely alter the sequence of the loops. This permits strength reduction and the other optimisations to have more effect.

One useful heuristic here is to move those loops which are through small domains to the outside. Strictly this demands that the optimising should be carried out immediately before the actual finding each time, but it is sufficient that it should be carried out when a typical database is available.

Another is to move to the outside, loops on whose variable many WHEREs depend. Again just a sorting of records on a chain.

- entity classification

  Suppose that each line is easily identifiable as being as one of four classes, and that each line of the feature must be of one particular class. For simplicity suppose that each classes has an equal number of lines.

  By defining a subfeature for each line class with only one WITH and one WHERE, we can reduce the main recognition task to $L/4^8 = 250^8 = 2^{64}$

  At a millisecond each this is now down to 2^{54} seconds, faster by a factor of 2^{16}. Still not fast enough, but an impressive reduction.

- featuretting

  Now consider splitting the eight-line feature into a feature with two similar sub-features, each of four lines. Suppose that there are $M = 128$ of the four-line subfeatures.

  Finding them takes $L/4^4 = 2^{32}$ and then finding the original feature takes a further $M^2 = 2^{14}$ cycles. The first of these dominates, and we are now down to 2^{22} seconds

  A second level of nesting, so that the finder never has to deal with more than two subfeatures at once, brings the time down to just a few minutes.

  We have been using classification and featuretting manually in our examples. They make it quite practical to run realistic examples in small databases of 50 to 100 data items.

  In fact these two techniques are very reasonable to use in the description of features as it provides some kind of structure round which to do one's thinking. The equivalent of structured programming.

  When a high level feature references more than one instance of a potential featurette or classification it is far easier to use a separate description than to spell it all out in the main description.

- assignment

  If there is a criterion taking the form of an equality, with one of the subfeatures on one side, an essential optimisation is to convert the search and test into an assignment, generating that subfeature from the expression on the other side of the equality.

  Clearly this requires the ability within the system to be able to understand what an equality constraint is, rather than just moving criteria from WHERE clauses to IFs, but this is not hard with a small anount of symbolic processing.

- indexing

  This powerful technique is an extension of the assignment idea. Suppose that a feature has a face and an edge and that one of the WHERE clauses says that the edge must bound the face.

  No hard-wired recogniser would ever consider taking all the edges in the model and testing whether they were bounds or not ! They would use the modeller's pointer plex to generate only those lines which satisfied that condition.

  This reduces the loop count at that level from hundreds or thousands to tens, and also means that there is no need to carry out the test.

The same principle is also applicable in the more generic context, and possibly to even better effect. This conversion of a WHERE clause from being a filter to being a generator can be applied whenever there is a binary relation, and the necessary indexes can be set up by a single pass through the database. (The modeller plex can be regarded as providing the indexes for some access paths to make them faster.) Making the indexes you need is like having the perfect modeller pointer structures present for every enquiry all the time.

A really sophisticated version of this could even build a binary index for an inner loop from a ternary relation within each cycle of an outer loop. This would be even better than having very rich pointer structures in the modeller.

Note that the hint/trace ideas may be viewed as the systematic exploitation of the indexes present in the modeller, taken together with the use of a subfeature (the trace) of which there are relatively few instances in the database, as the outermost loop.

Optimisation within the MAKE FIND can exploit all these opportunities for the cost of a little analysis of the WITH-WHERE references bipartite graph. As suggested above, classification and featuretting are useful ways of getting your mind round a complex feature description. However, the opportunities for applying them are all deducible from the schema. For example, classification can always be applied if there is a WHERE which references only a single WITH.

We have not yet got very far with implementing these optimisations. It is clear however, that each of them can be introduced fairly straightforwardly in a way which does not prejudice the correctness of the dumb approach. It is also clear that their combination will leave us with only the interpretation overhead, relative to a hard-wired recogniser. The option of generating the source code in some compilable language for a hard-wired optimiser from the representation at present only interpreted also remains open.

We regard the interpretation penalty as one well worth paying in the context of experiments with feature descriptions to make them say what you meant.

7 SUMMARY

Features exist in the eye of the beholder, and each application has its own specific criteria for the features with which it deals. Even in a system supporting design by features, there are always emergent features which need to be recognised if applications are to be fully intelligent.

We have built a feature recogniser capable of finding user-defined features within a database of objects of user-defined structure.

At present this is slow, but there are reasons to believe that its speed can be enhanced to within a factor, due only to our interpretive working, of that of a hardwired recogniser working on native modeller datastructures. This factor is the price to be paid for tight interaction in the development and debugging of feature descriptions.

REFERENCES

Ames,A.L. 1991 Production ready feature recognition based automatic group technology part coding. *pp 161–170 in Proc Symp on Solid Modelling Foundations and CAD/CAM applications. (ed Rossignac and Turner)*

Ansaldi,S., de Floriani,L. and Falcidieno,B. 1985 Geometric Modelling of solid objects by using a face adjacency graph representation. *Computer Graphics vol 16 no 3 pp131-139*

Choi,B.K., Barash,M.M. and Anderson,M.R. 1984 Three-dimensional shape pattern recognition using vertex classification and vertex-edge graphs. *CAD vol 16 no 2 pp 81-86*

Corney,J. and Clark,D.E.R. 1991 A feature recognition algorithm for multiply connected depressions and protusions in 2.5D objects. *pp 171-184 in Proc Symp on Solid Modelling Foundations and CAD/CAM applications. (ed Rossignac and Turner)*

Dong,J., Parsaei,H.R. and Gornet,T. 1993 Manufacturing Features extraction and recognition in automated process planning. *Computers and Ind Engineering vol25 nos1-4 pp325-328*

van't Erve,A.H. 1989 PART, a feature based CAPP system *PA89-008 feb 1989 (Twente University)*

Faux,I.D. 1990 Modelling of components and assemblies in terms of shape primitives based on standard dimensioning and tolerancing surface features. *pp 259–275 in Geometric Modeling for Product Engineering (eds Wozny, Turner and Preiss) North-Holland 1990*

Gadh,R. and Prinz,F.B. 1992 Recognition of geometric forms using the differential depth filter. *CAD vol 24 no 11 pp583-598*

Gavankar,P. and Henderson,M.R. 1990 Graph-based extraction of protrusions and depressions from boundary representations. *CAD vol 22 no 7 pp 442-450*

Grayer, A. A computer link between design and manufacture *PhD dissertation, Cambridge University, 1976*

d'Haeyer,J.P.F. 1992 Object Representation for knowledge based recognition *pp19-24 in Proc International conference on manufacturing automation (ed Ko and Tan)*

Hanrahan 1984 An introduction to relational topology *pp 461–469 in Proc CAD84 (ed Wexler) Butterworth 1984*

Henderson,M.R. and Anderson,D.C. 1984 Computer recognition and Extraction of Form Features: a CAD/CAM link. *Computers in Industry vol 5 pp329-339*

Kaul,M. 1983 Parsing of graphs in linear time *pp 206-218 in Lecture Notes in Computer Science 153 Springer Verlag ISBN 3-540-12310-5*

Kaul,M. 1987 Practical applications of precedence graph grammars *pp 326-342 in Lecture Notes in Computer Science 291 Springer Verlag ISBN 3-540-18771-5*

Kyprianou,L.K. 1980 Shape classification in Computer Aided Design *PhD thesis. Cambridge University.*

Laako,T. and Mantyla,M. 1993 pp 333–342 in Proc 2nd ACM Solid Modeling, ACM 1993

Meeran,S. and Pratt,M.J. 1993 Automated Feature recognition from 2D drawings. *CAD vol 25 no 1 pp7-17*

Nardi,B. 1993 A small matter of programming. (chapter 6) *MIT Press 1993 ISBN 0-262-14053-5*

Perry-Barwick,S. and Bowyer,A. 1995 Multidimensional set-theoretic feature recognition. *CAD vol 27 no 10 pp731-740*

Rosenman,M.A. and Gero,J.S. 1996 Modelling multiple views in a collaborative environment. *CAD vol28 no 3 pp193-206*

Sakurai,H. and Gossard,D.C. 1990 Recognizing shape features in solid models. *IEE CGand A Sept 1990*

Shah,J.J. 1991 Assessment of features technology. *CAD vol 23 no 5 pp331-343*

Shah,J.J. and Rogers,M.T. 1988 Expert form feature modelling shell. *CAD vol 20 no 9 pp515-524*

Stiny,G. 1991 The algebras of design. *Research in engineering design vol 2 no 3 pp171-181*

Vandenbrande,J.H. and Requicha, A.A.G. 1993 Spatial Reasoning for the Automatic Recognition of Machinable Features in Solid Modelling. *IEEE PAMI vol 15 no 12 pp1269–1285*

Woo,T.C. 1977 Computer aided recognition of volumetric designs. *pp 121–136 in Advances in Computer-Aided manufacture (ed McPherson) North Holland 1977*

Woo,T.C. 1982 Feature Extraction by Volume Decomposition TR 82-4 Dept of Industrial and Operations Engineering, University of Michigan 1982

Wu,M.C. and Liu,C. 1996 Analysis on machined feature recognition techniques based on Brep. *CAD vol 28 no 8 pp603–616*