An Application of Distributed Solid Modeling: Feature Recognition

by W.C. Regli, S.K. Gupta, D.S. Nau

T.R. 94-82

An Application of Distributed Solid Modeling: Feature Recognition [*]

William C. Regli[†]
National Institute of Standards and Technology
Manufacturing Systems Integration Division
Building 220, Room A-127
Gaithersburg, MD 20899
regli@cme.nist.gov

Satyandra K. Gupta
Mechanical Engineering Department
Institute for Systems Research
University of Maryland
College Park, MD 20742 USA
skgupta@src.umd.edu

Dana S. Nau
Computer Science Department
Institute for Advanced Computer Studies
Institute for Systems Research
University of Maryland
College Park, MD 20742 USA
nau@cs.umd.edu

Available as **CS-TR-3375, UMIACS-TR-94-126, ISR-TR 94-82**.

Abstract

The availability of low-cost computational power is a driving force behind the growing sophistication of CAD software. Tools designed to reduce time-consuming build-test-redesign iterations are essential for increasing engineering quality and productivity. However, automation of the design process poses many difficult computational problems. As more downstream engineering activities are being considered during the design phase, guaranteeing reasonable response times within design systems becomes problematic. Design is an interactive process and speed is a critical factor in systems that enable designers to explore and experiment with alternative ideas during the design phase. Achieving interactivity requires an increasingly sophisticated allocation of computational resources in order to perform realistic design analyses and generate feedback in real time.

This paper presents our initial efforts to develop techniques to apply distributed algorithms to the problem of recognizing machining features from solid models. Existing work on recognition of features has focused exclusively on serial computer architectures. Our objective is to show that distributed algorithms can be employed on realistic parts with large numbers of features and many geometric and topological entities to obtain significant improvements in computation time using existing hardware and software tools. Migrating solid modeling applications toward a distributed computing framework enables interconnection of many of the autonomous and geographically diverse software tools used in the modern manufacturing enterprise.

This has been implemented on a network of SUN workstations using the ACIS solid modeler and the NIH C++ class library; inter-processor communication is handled with TCP/IP-based network communication tools.

Keywords: Multiprocessor Solid Modeling, Feature Recognition, Feature-Based Modeling, Distributed Computing.

[*]This work was supported in part by NSF Grants IRI9306580, DDM-9201779, EEC 94-02384 and a forgivable loan from General Electric Corporation awarded to the first author. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

[†]Also affiliated with: Computer Science Department and Institute for Systems Research, University of Maryland, College Park.

1 Introduction

The availability of low-cost computational power is a driving force behind the growing sophistication of CAD software. Tools designed to reduce time-consuming build-test-redesign iterations are becoming essential for increasing engineering quality and productivity. Examples include tools for finite element analysis, mechanism analysis, simulation, and rapid prototyping. Such tools have become crucial components in research in concurrent engineering and engineering design.

However, automation of the design process and construction of such tools pose many difficult computational problems. In order to realize the advantages of concurrent engineering, more downstream engineering activities are being considered during the design phase. As design is an interactive process, speed is a critical factor in systems that enable designers to explore and experiment with alternative ideas during the design stage. Achieving interactivity requires an increasingly sophisticated allocation of computational resources in order to perform realistic design analyses and generate feedback in real time.

It is becoming increasingly evident that one necessary component of an automated design analysis tool is a subsystem for recognizing manufacturing features directly from a CAD or solid model. This problem has been the focus of extensive research over the last decade. Feature recognition has been used in a variety of applications, including to generate process plans, to translate between design and manufacturing features, and to produce redesign suggestions. What has also become evident is that feature recognition, for realistic classes of parts with multiple and interacting feature interpretations, is computationally expensive. Hence, generating the features from a part may become a computational bottleneck within a design system. Further, existing feature recognition research has dealt exclusively with serial computer architectures.

In this paper we present our initial efforts toward developing a methodology for recognizing a class of machining features using a distributed, multi-processor computational architecture. Feature recognition has been approached using a variety of techniques, some of which are easier to parallelize than others. In previous work [25], we described trace-based, serial algorithms for finding feature instances from solid model data. This current work indicates that trace-based feature recognition methodologies are particularly well suited for parallelization. The basic steps in this approach are:

1. **Identify a task decomposition.** Task decomposition is performed at four levels: (1) the features to be recognized; (2) the types of trace information used to construct the feature instances; (3) the geometry and topology of the part; and (4) simplification of the part geometry to reduce the costs to solid modeling operations.

2. **Distribute the tasks.** Divide the problem using the task decomposition, isolating independent portions of the recognition problem and identifying a suitable computational resource for solving it.

3. **Synthesis of results.** Combine the results obtained by each separate processor into a global solution. This solution set can then be passed on the application at hand—in the context of our previous work, this is a subsystem for performing manufacturability analysis for machined parts [12, 11].

The benefits of applying this approach include:

- It increases the complexity of parts that are now computationally feasible. In the feature recognition area, serial approaches have experienced great difficulty when scaled to address complex, real-world parts which have thousands of geometric and topological entities and several hundred interacting feature instances. This approach is best suited for parts where there might be thousands of feature instances, but the individual features themselves are simple in structure. A distributed approach can put a greater number of such components within reach of existing tools.

- It makes use of an existing commercial solid modeling system directly. In this way, one is not required to wait for parallelized versions of common algorithms or the implementation of true multi-threaded, multi-processor solid modeling systems.

- It provides interactive analysis and feedback to the designer. Recognition of the many alternative features can be done for realistic parts in real-time. This can facilitate more comprehensive of analyses of manufacturability of the part at hand.
- It exploits the growing ubiquity and power of networked computing facilities to provide a flexible means of utilizing networked computational resources. Effective utilization of large collections of inexpensive processors enables applications to perform computationally intensive CAD/CAM activities efficiently and interactively.

The remainder of the paper is organized as follows: Section 2 gives an overview of related work on multi-processor solid modeling and recognition of features. Section 3 outlines the problem domain. Section 4 presents our method for dividing the recognition problem to be solved distributedly on multiple machines. Section 5 briefly discusses the implementation and presents an example of the performance improvements. Lastly, Section 6 contains concluding remarks and discussion.

2 Related Work

The bibliography of work on multi-processor algorithms for solid modeling applications is limited but growing. Currently, most have focused on parallel operations on CSG trees and other CSG representations of polygonal or polyhedral entities. Ellis *et al* [7] have developed the RayCasting Engine: a hardware-implemented facility for sampling solids represented in CSG for a variety of purposes, including rendering and mass-property calculations. They outline how this special-case hardware makes possible brute-force solutions to difficult computational problems, such as spatial sweeping and offsetting.

Narayanaswami and Franklin [20] presented a parallel multi-processor method for calculating the mass properties of polygonal CSG objects and outlined some extensions for applying the techniques to 3-D polyhedra. Banerjee *et al* [2] have developed parallelized algorithms for evaluating CSG trees that operate with a fixed number of processors with shared memory.

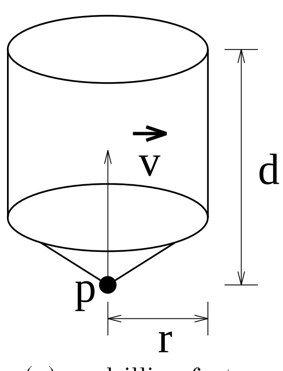
In the domain of boundary representation modeling, Karinthi *et al* [16] have produced a parallel algorithm for performing boolean set operations on polygons and polygons with holes. In Almasi *et al* [1], these techniques are extended to more general loops of edges.

Existing work on recognition of features has dealt with exclusively serial computer architectures. These feature technologies are based heavily on the geometric and topological manipulation capabilities of solid modeling systems and deal predominantly with form or machining features.

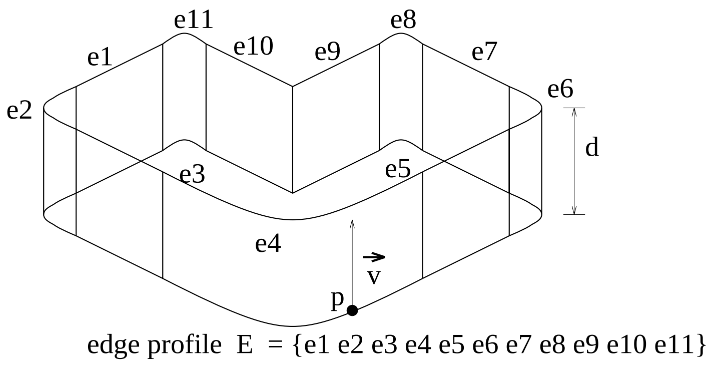
The work of Henderson has continually brought new computational techniques to address the feature recognition problem. The work described in [14], was the first to apply expert systems to the feature recognition problem. Gavankar and Henderson [9] presented techniques to identify protrusions and depressions in the boundary model of a part. More recently, Prabhakar and Henderson [22] described the use of neural networks to recognize and classify features. A strength to this approach is that they exploit the trainability of a neural net to incorporate new feature types. Further, neural nets have been demonstrated to be effective in classifying patters in domains where there is “noise.” For the feature recognition problem, noise comes in the form of incomplete or missing feature data that has been lost due to feature interactions.

Graph-based algorithms have proven useful for extracting some classes of features. These methods fall into two categories: those based on graph search [6, 4] and those based on pattern matching [15, 21, 26]. A common difficulty for both categories of graph-based approach is that the graph-based representations for solid models of parts are difficult to extend to the complex geometry and topology found in real industrial parts. Secondly, methods based on pattern matching and finding subgraph isomorphisms (a problem known to be NP-hard) are prone to combinatorial difficulties.

Chuang and Henderson [3] explored graph-based pattern matching techniques to classify feature patterns based on geometric and topological information from the part. Sakurai [29] provided for limited user-defined feature types with a graph-based feature recognition system. Efforts at Carnegie Mellon University [21, 26] have employed graph grammars for finding features in models of injection molded parts. Recently, Corney



(a): a drilling feature



(b): an end-milling feature

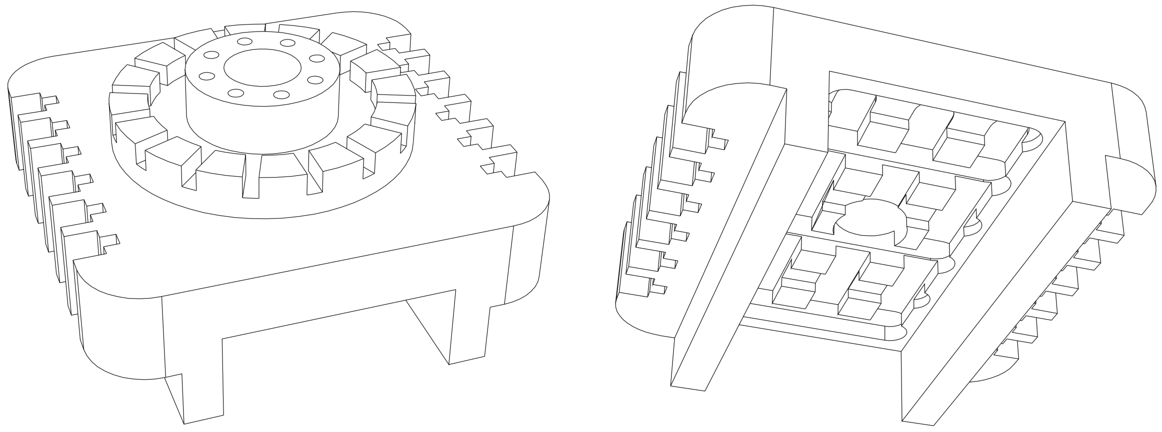
Figure 1: Examples of machining features

and Clark [4] have employed graph-based algorithms to find general feature classes from $2\frac{1}{2}$-dimensional parts.

Geometric algorithms for finding convex hulls have been employed to decompose polyhedral parts and identify form features. In an early effort in this direction, Woo [33] proposed a method for finding general depression and protrusion features on a part through decomposing the convex hull of the solid model. Recent work by Kim *et al* [17, 32] addresses some of the limitations of Woo's approach, improving upon its utility for generating descriptions for polyhedral parts and extending it to handle parts with some kinds of non-planar surfaces. More recently, other approaches based on cellular decomposition have emerged, most notably [27, 28]. Such methods are computationally intensive, often producing a large number of cells with a large (often exponential) number of ways for them to be combined into features. They have several nice properties, however, including that they can be employed to produce alternative feature decompositions and, in the case of machining, that the cells can be used to generate many of the tool paths of interest in planning applications.

Gadh and Prinz [8] were the first to describe techniques for combating the combinatorial costs of handling realistic industrial parts (i.e. those with thousands of topological entities). In such cases, traditional knowledge-based, decomposition, and pattern-matching techniques are computationally impractical because the fundamental algorithms (i.e. forward chaining in a frame-based reasoning system or subgraph pattern matching) are inherently exponential. Gadh and Prinz's method is to abstract an approximation of the geometric and topological information in a solid model and find shape features in the approximation. Their approach employs a differential depth filter to reduce the number of topological entities. A second pass maps the topological entities onto structures called "loops." In this work, features are defined using the higher-level loops as opposed to being defined as patterns in the boundary representation's geometry and topology. This approach significantly reduces the number of entities that need to be searched in order to build feature instances. While this kind of approach holds much promise for addressing combinatorial problems, yet to be addressed are how to extend the techniques to better handle interacting features and non-linear solid models.

Many aspects of the feature recognition problem are still open and active areas of research. Among these are: recognizing and representing interacting features [31], incremental recognition of features [18, 13], modeling alternative feature interpretations and completeness [19, 24], reasoning about the manufacturability of features [12], and incorporation of user-customizable feature classes.



(a): part (after machining) (b): part (underside view)

Figure 2: An example part.

3 Problem Specification

3.1 Machining Features

A *machining feature* is the portion of the workpiece affected by a machining operation. An instance f of a machining feature will be created by some machining operation using a single cutting tool in one tool setup. To perform the machining operation, one sweeps the tool along some trajectory that is characterized by some set of parameters. Only a portion of this swept volume corresponds to the volume of material that is to be removed by the machining feature. This volume is called *removal volume* of feature f.

Machining features will be referred in terms of the operations used to create them. For example, we say that the hole h in Figure 1(a) is an instance of a *drilling feature.* The pocket p in Figure 1(b) is an an instance of an *end-milling feature* and is characterized by the edge profile bounding the area swept by the milling tool. These two particular features defined above correspond to roughing operations.

3.2 Machining Feature Recognition

The initial workpiece, S, is represented as a solid model of raw stock material to be acted upon by a set of machining operations. The machined part is a solid object, represented by a solid model of the part P, to be produced as a result of a finite set of machining operations. The *delta volume* is the regularized difference of the initial workpiece and the part: $\Delta = S -^* P$.

In general, there may be several alternative interpretations of the part as collections of machining features, each corresponding to a different way of manufacturing it. A *feature-based model* is a collection of features that model a single, unique interpretation of the part. The feature recognition problem is be defined as follows: given a collection of machining features, $\mathcal{M} = \{m_1, m_2, \ldots m_l\}$, a part P, and a piece of stock S, return the set $\mathcal{F}$ of feature instances from $\mathcal{M}$ found from P and S. In existing literature, the members of $\mathcal{M}$ are often defined as parameterized geometric templates. The feature set $\mathcal{F}$ is a finite set of features comprised of those features belonging to the alternative feature-based models for the part [24].

3.3 Trace-based Recognition of Features

A trace represents the partial information left in the model by an instance of a feature. This partial information, in a manner similar to approaches to computer vision problems, is used to reconstruct feature instances.

For example, in the case of the drilling feature in Figure 1(a), a trace could be the conical ending surface of the hole. Similarly, in the case of an end-milling feature such as the one in Figure 1(b), as trace might be a portion of its bottom surface.

Trace-based techniques have their origin in the work of Marefat and Kashyap [19]. They expanded on the work of [15] and augmented it with hypothesis testing techniques. In this method, information from the solid model is used to generate hypotheses about the existence of features. These hypotheses are tested to see if they give rise to valid feature instances.

Vandenbrande and Requicha [31] were the first to formalize trace-based (or hint-based) techniques for constructing features from information in a solid model. In the work of Vandenbrande, the traces are used to fill "feature frames" in a frame-based reasoning system. After filling frames with the trace information present in the part, the system classifies the partial frames and attempts to complete the frame information for those that appear promising using a variety of geometric reasoning and computational geometry techniques.

Regli *et al* [25, 23] presented an approach for guaranteeing completeness of a recognition algorithm, i.e., that one could define a class of features and verify that one's approach was capable of producing all features in that class. They presented feature recognition as an algorithmic problem in which part geometry and topology are traversed for traces from which feature instances are constructed. They attempt to formally describe the behavior of their algorithm and calculate a general measure of its complexity. Their approach has been employed for automated design analysis [12] and automated redesign [5].

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

- Feature traces can be produced from a richer variety of design information from the part, such as tolerances, surface finish requirements, and function information associated with surfaces. Traditional feature recognition methodologies often consider only the part's geometry and topology.

- Feature classes can be customized by users. Recognition routines for new features can be built by introducing traces for the new features and methods for building instances of the new features from these traces.

- Trace-based techniques can be employed to recognize features from a variety of manufacturing domains. Existing feature recognition literature focuses mostly on machined parts, due in part to the fact that the functionality of solid modeling systems is well suited for manipulating volumes that describe material to be machined and decomposing these volumes into features.

- Trace-based techniques lend themselves well to parallelization. In the existing work on serial computer architectures, techniques based on cellular decomposition and graph search do not present evident places at which the recognition problem can be divided into independent subproblems. Trace-based techniques provide several obvious levels at which the problem can be broken up. What might be less evident is that, in parallelizing the problem, one can make additional geometric and topological simplifications to independent problem pieces to reduce their computational difficulty.

The basic structure of a trace-based feature recognition system includes:

1. For each feature m in $\mathcal{M}$, there is a finite set of *traces* $t_1, t_2, \ldots t_l$. Intuitively, a trace corresponds to a minimum amount of information contributed to the part by an instance of a feature of class m adequate for determining the parameters of an equivalent feature of the same class.

2. Each trace t_i has associated with it a procedure $\mathcal{P}_{t_i}()$ that can build, from trace data and the model of the part, instances of those features of class m capable of producing the trace t_i.

An outline for a generic algorithm for trace-based recognition of features can be presented as follows:

1. Input a class of features, $\mathcal{M}$, a solid model for the part, P, and for the initial stock material, S.

2. From P and S, identify the set of all potential traces present: $\mathcal{T}$.

3. For each trace t in $\mathcal{T}$ do
   (a) if t matches a t_i, call the procedure $\mathcal{P}_{t_i}()$ and construct (if possible) feature instances, $f_1, f_2, \dots f_k$. Add these to the set of all feature instances, $\mathcal{F}$.

Examples.

For an illustration, the task of recognizing drilling and end-milling features can be accomplished using the following traces:

1. *drilling features:*
   (a) Trace 1: a convex conical surface in the delta volume as a conical ending surface describing the cutting tip of a drilling tool. This trace is used to build an instance of a drilling feature when only a portion of its ending tip surface remains on the boundary of the delta volume.
   (b) Trace 2: a convex cylindrical surface in the delta volume as a side surface created by a drilling operation. This trace is used to build instances of drilling features when a portion of their side surface remains on the boundary of the delta volume.
2. *end-milling features*
   (a) Trace 1: a planar surface in the delta volume as a surface created by the cutting tip of an end-mill. This trace is used to build instances of end-milling features when only a portion of their bottom surface is present on the boundary of the delta volume.

   Traces 2-3 are used to build instances of end-milling features when only a portion of their side surfaces are present on the boundary of the delta volume. In these cases, the end-milling features may extend completely through the stock material. Examples of such features include through pockets.

   (b) Trace 2: a conical (elliptical/cylindrical) surface in the delta volume as a surface created by the side cutting surface of an end-mill.
   (c) Trace 3: a pair of non-parallel planar surfaces in the delta volume, as faces created by the side cutting surface of an end-mill.

A presentation of the details of the various procedures $\mathcal{P}_{t_i}()$ for these traces is beyond the scope of this paper. Such algorithms have been developed in previous work, notably: Vandenbrande [31] for drilling feature traces 1 and 2 and end-milling feature trace 1; and Regli [25, 23] for all of the above traces.

4 Approach

4.1 Motivations

The properties that make trace-based approaches highly suitable for parallelization are evident in the procedure from the previous section: the feature classes and their traces each introduce natural partition lines along which the problem can be divided into independent subproblems to be solved by different processors.

For features in $\mathcal{M}$, the act of recognizing a feature of class m_1 is independent of the recognition of a feature of class m_2. As presented in Section 3.2, the final feature set $\mathcal{F}$ is to contain all those feature instances from $\mathcal{M}$ that are members of some feature-based model of the part. This set contains all instances of each of the feature classes in $\mathcal{M}$; hence the feature instances of type m_1 can be calculated separately from those of type m_2. For example, in the context of this paper, a particular drilling feature f being a member of some feature-based model does not alter the existence of any end-milling features. The feature set $\mathcal{F}$ will contain all of the possible features, as derived from the trace information left in the part.

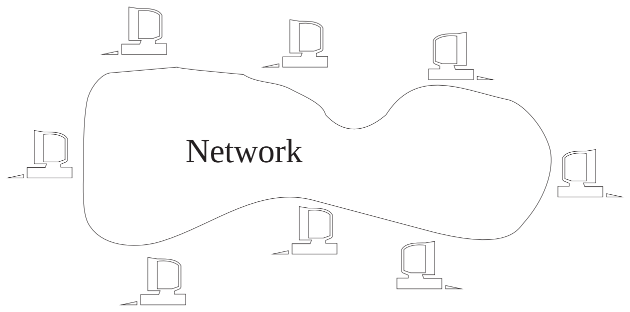


Figure 3: Internetworked computational resources.

Further, the set of traces $\mathcal{T}$ used to build the feature instances from the part and stock information introduces an additional level at which the problem can be partitioned. For each feature class m in $\mathcal{M}$, there is a collection of traces $t_i, t_{i+1}, \ldots t_j$ and associated methods for building instances of features of type m from these traces. One can decompose the problem of finding all features of type m by trace and handle each trace t_i on a different processor. One observation is that this may introduce some redundancy, i.e., it may be possible to find the same feature instance f in different ways using different traces. This redundancy can be handled in two ways. One method is to delete duplicate features while building the final feature set $\mathcal{F}$. A second approach, and the one which we will employ, is to handle the traces capable of producing equivalent feature instances together on the same processor and remove duplicates as they are found. The benefits of doing the latter are that the redundancies are addressed at the level at which they occur, thus simplifying the task of building the final feature set $\mathcal{F}$.

Parallelizing the problem enables other, less obvious, benefits. In particular, a large portion of the costs in a feature recognition system are due to the complexity of operations requiring geometric computations and geometric reasoning. In isolating independent problem pieces, one can make geometric and topological simplifications that identify the information in the original part needed to build and verify the feature instances on each independent processor. In this way, the independent problem pieces may only require a fraction of the information present in the original part. This approach is most suitable for parts for which the feature instances themselves are relatively simple, but there may be thousands of them. It is not as well suited to problems where the feature instances are complex, i.e., a pocket with 5280 distinct curved surfaces comprising its profile.

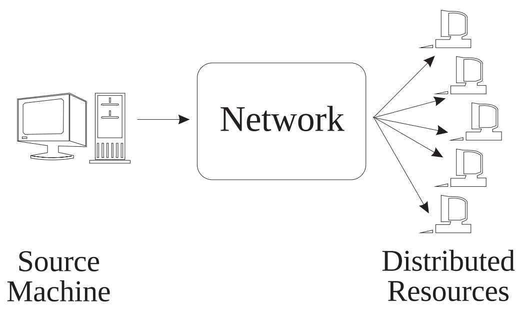


Figure 4: Source host decomposing the tasks to clients.

4.2 Overall Methodology

In the distributed computing paradigm, collections of autonomous computational resources are interconnected on a network, as illustrated in Figure 3 [30]. While these resources do not share main memory, they may share access to common devices such as peripherals, file systems, output devices, etc. Software systems can use the network and shared peripherals to exchange information between the autonomous resources. A fundamental issue when building distributed software systems is how to enable the independent computers to cooperate with each other in solving enterprise-wide problems.

Our approach to feature recognition will be to have a central computing resource set up the problem and serve subtasks to client machines distributed on the network, as shown in Figure 4. Each of the individual client processors will be given an independent portion of the particular global feature recognition problem. The activities of these distributed resources will be coordinated through global mechanisms at the server machine.

There are three technical issues to be addressed:

Identifying a Task Decomposition

There are four levels at which the recognition problem is decomposed; specific details are given in Section 4.3:

1. *Features to be recognized:* different feature classes (in this case drilling and end-milling) are considered by separate computing resources.

2. *Types of feature traces:* different traces for each of the feature classes are considered by separate computing resources.

3. *Part information:* given a specific feature and a trace for recognizing it, partition the geometric and topological information of the part to further subdivide the recognition task. For example, a given feature instance might be created from any one of several traces it leaves in the part. The objective of this phase is to collect together all of the trace information capable of producing equivalent or identical feature instances. While we will only consider geometric and topological information in this paper, this decomposition can be extended to include other data (i.e. tolerances, surface properties, etc.).

4. *Part simplification:* based on the feature type and the trace being used to recognize it, alter the geometric and topological information in the solid model of the part to reduce its complexity. In this way, one can reduce the cost of operations on it during feature recognition. For example, reduce the number of geometric and topological entities while still retaining the information required to construct feature instances from the particular trace. This involves simplifying or eliminating complex part geometry not effecting the feature trace being considered and distributing disjoint portions of the delta volume for consideration on separate computing resources.

Distributing Tasks

Given a task decomposition, the next phase is to distribute the individual tasks to the available computing resources. This is done by invoking the feature recognition procedure on each separate task, each task on its own processor. The basic outline is as follows:

1. Input the part P and the stock material S.

2. Generate the set of traces $\mathcal{T}$ from P and S.

3. Generate the task decomposition by feature type and trace.

4. Identify available computational resources with which to further decompose and simplify the problem and recognize features.

Recombination of Results

Each separate computing resource, upon termination of its portion of the global recognition task, transmits its results back to the server machine. The features returned are then integrated into a solution to the overall problem. This may include modeling the interactions among the features found, eliminating any redundant features subsumed by those features found on other processors, or building compound features or feature groups.

In the domain of machined parts and features we use to illustrate the distributed approach, distributing tasks and recombining results are straightforward problems. Distribution of tasks is executed based directly on the decomposition identified. This would become more complex if we assumed that there were an upper bound on the number of computing resources available.

Recombining results in this domain is a matter of building the final feature set as the union of those sets returned by each distributed resource. However, that fact that these are relatively easy within the context of this paper may not generalize to other manufacturing domains.

4.3 Identifying a Task Decomposition

The task decomposition stage groups feature information and isolates traces to be handled by separate computing resources. There are four levels of task decomposition. For illustration purposes, we shall assume there is no limit on our computational resources. When there is a bound on the number of processors available, the task decomposition or the distribution of the task may vary to more efficiently partition the problem. In our implementation (discussed in Section5), we distribute the tasks evenly over the available processors.

The decomposition by feature type and decomposition by trace, as noted before, are straightforward. In developing techniques for part decomposition and simplification, one is faced with a trade off between the sophistication of techniques and their computational costs. Using very sophisticated techniques to maximize the ability of each individual processor to produce useful feature instances in a minimal amount of time might increase the computational overhead to a degree that mitigates the benefits of parallelization. In choosing the following conditions, we have picked decompositions and simplifications that are computationally cheap. While it is certainly possible to present more complex decomposition criteria, an important consideration is that the conditions themselves cannot be more complex than the original recognition problem. If the decomposition conditions were themselves costly, the overhead considerations might eliminate any of the speedup benefits we hope to achieve using a multi-processor approach.

The remainder of this section discusses the decomposition of part geometry and topology and techniques for model simplification.

4.3.1 Decomposition of Part Information

We present a four-part decomposition for the geometric and topological information in the part. The conditions are based on properties of the traces for constructing feature instances. There may be other conditions that provide an equivalent means of arriving at a task decomposition with the desired properties. Decomposition of the geometry and topology based on feature class and trace proceeds as follows:

1. drilling traces 1 and 2:
   Group together convex cylindrical and conical faces with equivalent axes.

2. end-milling trace 1:
   Group together all coplanar faces. In the example illustrated in Figure 5(b), six disjoint planar part faces are grouped to be handled together on the same processor. This grouping collects the split-faces [32] of the part.

3. end-milling trace 2:
   Group convex cylindrical surfaces with equivalent axis directions.

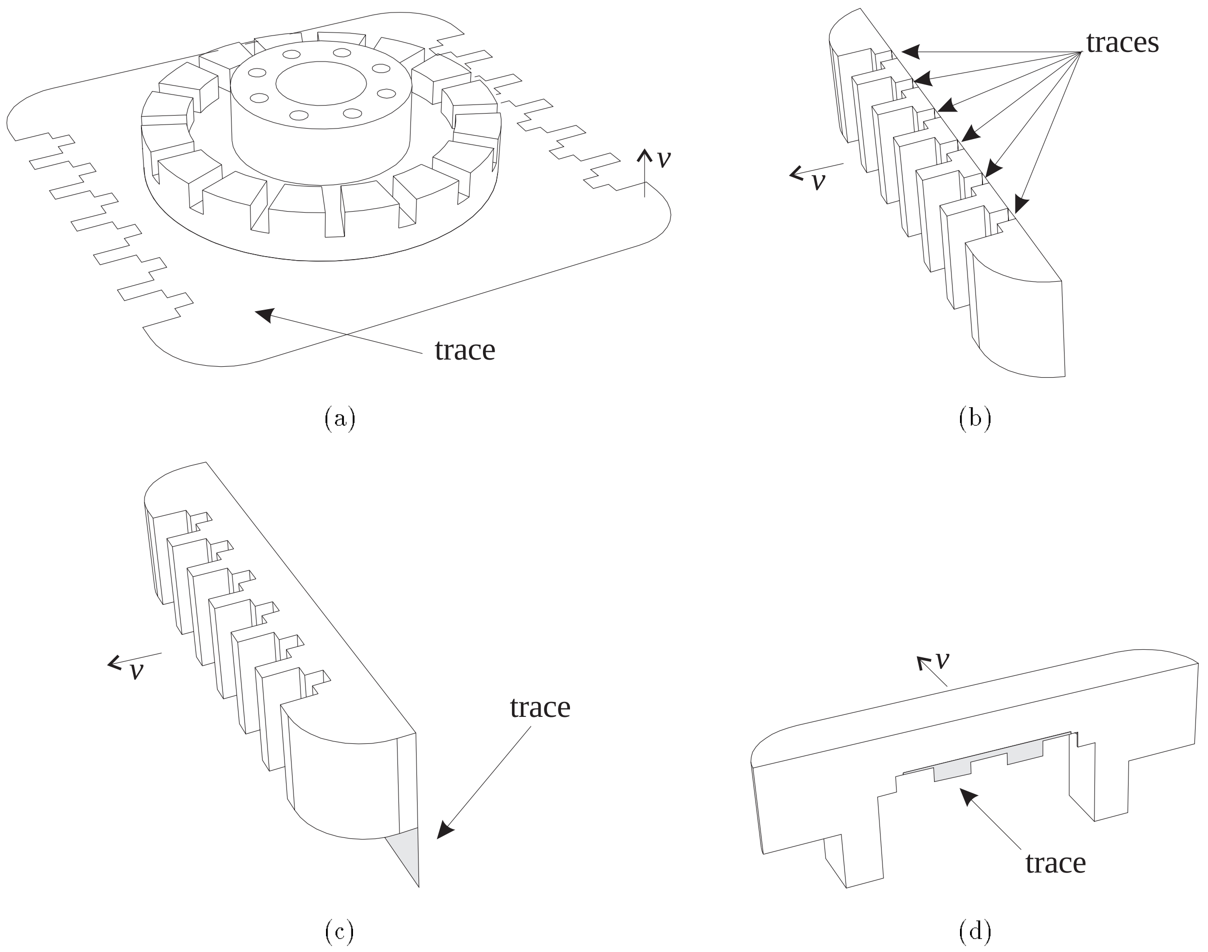


Figure 5: For the example in Figure 2, four end-milling recognition subtasks and their simplified part information.

4. end-milling trace 3:
   Group planar surfaces with normals perpendicular to a common vector, i.e., for each grouping there is a vector v such that, for all surfaces s_i and s_j in the grouping, $\text{normal}(s_i) \cdot v = \text{normal}(s_j) \cdot v = 0$. Note that some surfaces may be present in more than one group.

The above decomposition groups together those traces from the part which could produce equivalent feature instances. In this way, redundancies can be eliminated at the sub-process level and facilitate later recombination of results.

4.3.2 Model Simplification

The goal of model simplification is to reduce the amount of data that needs to be considered by each processor to the minimum amount required to construct feature instances from the traces it has been given.

We simplify the models of the part, stock, and delta volumes in the following ways:

1. The disjoint portions of the delta volume are handled on separate processors. In the example in Figure 2, there are 10 disjoint portions of the delta volume.

2. Based on the trace information and feature types, simplify the geometry and topology of the part. In each case, the model for the part P is simplified to P' as follows:

   (a) drilling trace 1:
   Given a cylindrical surface c in the delta volume of radius r, P' contains all the portions of P that lie within r of the axis of c.

   (b) drilling trace 2:
   Given a conical surface c in the delta volume of radius r and located at point d, P' contains all the portions of P that lie within r of the axis of c and in the half-space above d.

   (c) end-milling trace 1:
   Given a planar surface p in the delta volume with a root point d and normal vector v, P' contains all the portions of P that lie in the half-space defined by d and v.

   (d) end-milling traces 2 and 3:
   No simplifications are made for these traces. Finding these types of end-milling feature instances might require consideration of information from the entire part, and sophisticated simplifications in this case would be costly.

Figure 5 shows four illustrations of part simplification for end-milling trace 1. In the figure, the planar faces are being considered as traces indicating potential bottom surfaces of an end-milled features; vector v denotes the orientation of the potential feature. In each case, the trace information is used to eliminate the portion of the part lying below the trace—information that does not get considered when building a feature instance in direction v. Note that, in making this rudimentary simplification, the number of geometric and topological entities to be considered is greatly reduced.

4.4 Estimated Computational Improvements

In theory, we can expect an ideal speedup of a factor of K, where K is the number of processors available. In reality, the task decomposition to set up parallelization incurs some added cost, as does the recombination of results at the end. However, these additions are negligible when compared with the costs incurred when performing the recognition process on the subproblems.

Within a trace-based methodology, as outlined in Section 3.3, the overall complexity of recognition depends on two factors: the difficulty in generating the set of potential traces $\mathcal{T}$, and the complexity of the methods for generating feature instances from traces.

The size of $\mathcal{T}$ can be computed from the model of the part and the types of traces. The complexity of the feature construction routines is more difficult to assess and is where the majority of the computational costs occur. Much of this cost is due to geometric queries and reasoning operations used to find the parameters of feature instances. While there is no authoritative reference on the general complexity of the solid modeling operations such as booleans, sweeps, and the like, the consensus is that these operations are costly. In particular, the complexity of boolean operations appears to lie between $O(n^2)$ and $O(n^4)$ or $O(n^5)$ time, depending on many implementation-specific details.

The fact that these basic solid modeling routines are at least quadratic in the size of the model means that small reductions in the number of entities in the model, such as those shown in Figure 5, translate into large reductions in computational effort.

In the next section, we have attempted to provide examples to estimate both the speedup factor and the reduction in the number of geometric and topological entities achieved by this approach.

5 Implementation and Example

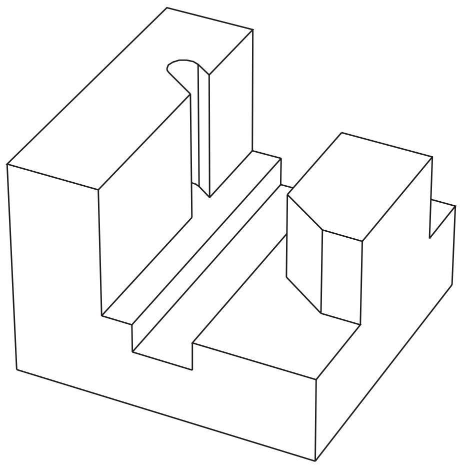
A proof-of-concept implementation of this distributed feature recognition methodology has been done in C++ using version 3.0.1 of the AT&T C++ compiler from SUN Microsystems running on networked SUN

SPARCStations. We employ version 1.5.1 of Spatial Technologies' ACIS© solid modeling system, and version 3.14 of the NIH C++ Class Library developed at the National Institutes of Health. Also being employed are Ithaca Software's HOOPS© Graphics System and the Tcl/Tk embeddable command language and user interface toolkit developed at the University of California at Berkeley.

The system currently runs on a cluster of SUN workstations at the University of Maryland in College Park. Communications between separate processes are performed over the Internet using TCP/IP protocol-based network communication tools and shared disk storage. The data for the examples below has been collected using six processors, one SPARC Station model 10, one model 2, and 4 models IPX. In this version of the implementation, when the number of tasks is greater than 6, the task decompositions are distributed evenly over the 6 available processors.

In general, the methodology we have outlined in the previous sections can be applied to any trace-based feature recognition domain. However, for purposes of this proof-of-concept implementation we have limited the class of features $\mathcal{M}$ to drilling and end-milling features.

The timing results presented below are the elapsed clock time and not meant to be an absolute measure of the intrinsic difficulty of the feature recognition problem (as these algorithms and implementations can likely be improved). They are intended to provide an indication of the time-lag experienced by the user of the system. More significant than the elapsed time statistics are the speedup factors between the serial and parallelized versions.


(a): example from [31]

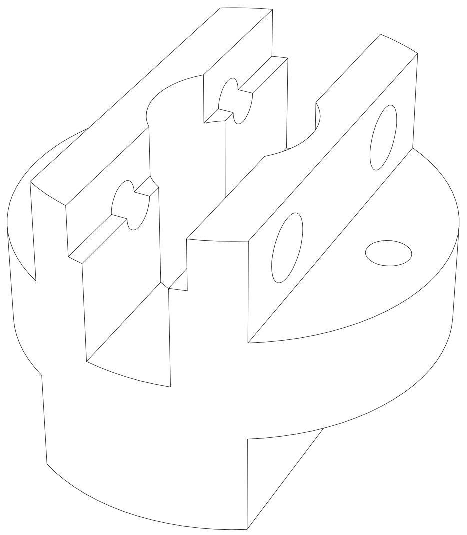

(b): example from [24]

Figure 6: Two example parts addressed in previous literature.

Empirical Results 1.

This example, taken from [31], contains 21 part faces. In serial, 1 drilling and 8 end-milling features were identified in approximately 45-50 seconds. In parallel on 6 processors, it took approximately 5 seconds to set up the decomposition and 8-12 seconds to recognize the features. Using the design information reduction and simplification techniques, there was a 22% reduction in the number of geometric and topological entities that had to be considered.

Empirical Results 2.

The example part in Figure 6(b) is a socket taken from [24]. This part, when machined from a cylindrical piece of stock material, has 37 faces in the delta volume. For this part, there are 12 drilling and 20 end-milling features appearing in its feature-based models that can be produced with the traces given above. In serial running on the SPARC 10, our system identified these 32 feature instances in approximately 65-70 seconds. When run distributedly, using 6 processors, the system took 10 seconds to set up the decomposition and approximately 12-16 seconds to identify the features. In this case, reduction and simplification resulted in a 35% reduction in the number of geometric and topological entities that had to be considered.

Empirical Results 3.

The example part in Figure 2 is a shuttle intended to move along a guideway. The solid model of this part contains 281 faces. Many of the feature instances have been added for weight reduction purposes. In serial, it took over one hour to find the more than 100 feature instances. When running distributedly, it took 2 minutes to set up the task decomposition and approximately 32 minutes to find the features. In this case, reduction and simplification resulted in a 43% reduction in the number of geometric and topological entities that had to be considered.

6 Conclusions

Concurrent engineering and product design are pushing more downstream manufacturing issues into the design phase. Building effective and interactive design and analysis tools to address these needs is making efficient and sophisticated allocation of computational resources increasingly important.

Use of a multi-processor architecture can provide a large increase in computational power by exploiting the available computing resources. The benefits of migrating to a multi-processor architecture include an increase in the complexity of feasible mechanical designs and the ability to produce real-time feature data for realistic parts.

Contributions

In this paper we have presented our initial work toward an approach for performing trace-based feature recognition using a distributed multi-processor architecture. We present a commonly addressed collection of features and illustrate how to identify a task decomposition of the recognition problem. The task decomposition is used then to divide the work among several distributed computing resources whose individual results are integrated into a unified solution for the part at hand. This kind of approach shows promise for domains of complex parts containing, possibly, thousands of features instances, but for which the structures of the feature instances themselves are relatively simple.

Future Work

Application of distributed algorithms to solid modeling and problems in engineering design and analysis holds immediate promise for enhancing existing CAD tools. We hope that this work motivates more study into how to effectively migrate current solid modeling applications toward a distributed computing framework. In the future, as distributed computing technologies become more accessible, algorithms that coordinate efforts between autonomous and geographically diverse computing resources will be commonplace in the modern manufacturing enterprise.

Required to make this transition will be changes in the underlying architecture of solid modeling systems, their data structures, and algorithms to exploit multi-processor computing. In addition, as engineering software applications built on top of modelers continue to grow in complexity, the need to obtain performance improvements will increasingly involve distributed algorithms.

As a new feature recognition technology, we anticipate that a distributed methodology will increase the complexity of mechanical parts within the reach of traditional feature recognition systems, reduce computational bottlenecks, and enable more sophisticated types of analyses. This will, in turn, aid in building an environment that will allow designers to create high-quality products that can be manufactured more economically—thus reducing the need for redesign, lowering product cost, and shortening lead times.

Acknowledgements

This work was supported in part by NSF Grants IRI9306580, DDM-9201779, EEC 94-02384 and a forgivable loan from General Electric Corporation awarded to the first author. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This paper and the example parts can be obtained through the World Wide Web at URL `http://www.cs.umd.edu/~regli/SM95/paper.html`.

References

[1] George Almasi, Raghu Karinthi, and Kankanahalli Srinivas. A parallel algorihtm for computing set operations on loops. Technical Report TR 93-10, Department of Statistics and Computer Science, West Virginia University, August 1993.

[2] Raja P. K. Banerjee, Vineet Goel, and Amar Mukherjee. Efficient parallel evaluation of csg tree using fixed number of processors. In Jaroslaw Rossignac, Joshua Turner, and George Allen, editors, *Second Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 313–322, New York, NY 10036, USA, May 1993. ACM SIGGRAPH, ACM Press. Montreal, Canada.

[3] S. H. Chuang and M. R. Henderson. Three-dimensional shape pattern recognition using vertex classification and the vertex-edge graph. *Computer Aided Design*, 22(6):377–387, June 1990.

[4] J. Corney and D. E. R. Clark. Method for finding holes and pockets that connect multiple faces in $2\frac{1}{2}$d objects. *Computer Aided Design*, 23(10):658–668, December 1991.

[5] Diginta Das, Satyandra K. Gupta, and Dana S. Nau. Reducing setup cost by automated generation of redesign suggestions. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 159–170. ASME, September 1994.

[6] Leila De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8), August 1989.

[7] J. L. Ellis, G. Kedem, T. C. Lyerly, D. G. Thielman, R. J. Marisa, P. J. Menon, and H. B. Voelcker. The RayCasting Engine and ray representations. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[8] R. Gadh and F. B. Prinz. Recognition of geometric forms using the differential depth filter. *Computer Aided Design*, 24(11):583–598, November 1992.

[9] P. Gavankar and M. R. Henderson. Graph-based extraction of protrusions and depressions from boundary representations. *Computer Aided Design*, 22(7):442–450, September 1990.

[10] S. K. Gupta, D. S. Nau, W. C. Regli, and G. Zhang. A methodology for systematic generation and evaluation of alternative operation plans. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing*. Elsevier/North Holland, 1993.

[11] Satyandra K. Gupta, Thomas R. Kramer, Dana S. Nau, William C. Regli, and Guangming Zhang. Building MRSEV models for CAM applications. *Advances in Engineering Software*, 1994. To appear.

[12] S.K. Gupta and D.S. Nau. A systematic approach for analyzing the manufacturability of machined parts. *Computer Aided Design*, 1994. To appear.

[13] JungHyun Han and Aristides A. G. Requicha. Incremental recognition of machining features. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 143–150. ASME, September 1994.

[14] Mark R. Henderson. *Extraction of Feature Information from Three-Dimensional CAD Data*. PhD thesis, Purdue University, West Lafayette, IN, USA, 1984.

[15] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20(2):58–66, March 1988.

[16] Raghu Karinthi, Kankanahalli Srinivas, and George Almasi. A parallel algorithm for computing polygon set operations. Technical Report TR 93-4, Department of Statistics and Computer Science, West Virginia University, April 1993.

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

[18] Timo Laakko and Martti Mäntylä. Feature modelling by incremental feature recognition. *Computer Aided Design*, 25(8):479–492, August 1993.

[19] M. Marefat and R. L. Kashyap. Geometric reasoning for recognition of three-dimensional object features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):949–965, October 1990.

[20] Chandrasekhar Narayanaswami and William R. Franklin. Determination of mass properties of polygonal csg objects in parallel. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[21] J. Miguel Pinilla, Susan Finger, and Friedrich B. Prinz. Shape feature description using an augmented topology graph grammar. In *Proceedings NSF Engineering Design Research Conference*, pages 285–300. National Science Foundation, June 1989.

[22] S. Prabhakar and M. R. Henderson. Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer Aided Design*, 24(7):381–393, July 1992.

[23] William C. Regli, Satyandra K. Gupta, and Dana S. Nau. Extracting alternative machining features: An algorithmic approach. Technical Report ISR-TR94-55, The University of Maryland, July 1994.

[24] William C. Regli, Satyandra K. Gupta, and Dana S. Nau. Feature recognition for manufacturability analysis. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 93–104. ASME, September 1994.

[25] William C. Regli and Dana S. Nau. Building a general approach to feature recognition of material removal shape element volumes (MRSEVs). In Jaroslaw Rossignac and Joshua Turner, editors, *Second Symposium on Solid Modeling Foundations and CAD/CAM Applications*, New York, NY 10036, USA, May 19-21, Montreal, Canada 1993. ACM SIGGRAPH, ACM Press.

[26] Scott A. Safier and Susan Finger. Parsing features in solid geometric models. In *European Conference on Artificial Intelligence*, 1990.

[27] Hiroshi Sakurai. Decomposing a delta volume into maximal convex volumes and sequencing them for machining. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 135–142. ASME, September 1994.

[28] Hiroshi Sakurai and Chia-Wei Chin. Definition and recognition of volume features for process planning. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing*, chapter 4, pages 65–80. Elsevier/North Holland, 1994.

[29] Hiroshi Sakurai and David C. Gossard. Recognizing shape features in solid models. *IEEE Computer Graphics & Applications*, September 1990.

[30] Amjad Umar. *Distributed Computing: A Practical Synthesis*. Prentice-Hall, Englewood Cliffs, NJ 07632, 1993.

[31] J. H. Vandenbrande and A. A. G. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1269, December 1993.

[32] Douglas L. Waco and Yong Se Kim. Geometric reasoning for machining features using convex decomposition. *Computer Aided Design*, 26(6):477–489, June 1994.

[33] Tony C. Woo. Feature extraction by volume decomposition. In *Conference on CAD/CAM Technology in Mechanical Engineering*, pages 76–94, March 1982.