Feature Recognition for Interactive Applications: Exploiting Distributed Resources

William C. Regli*
Department of Computer Science and
Institute for Systems Research
University of Maryland
College Park, MD 20742 USA

Satyandra K. Gupta
Rapid Manufacturing Laboratory
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Dana S. Nau
Department of Computer Science,
Institute for Advanced Computer Studies and
Institute for Systems Research
University of Maryland
College Park, MD 20742 USA

Abstract

The availability of low-cost computational power is enabling development of increasingly sophisticated CAD software. Automation of design and manufacturing activities poses many difficult computational problems. Design is an interactive process and speed is a critical factor in systems that enable designers to explore and experiment with alternative ideas. As more downstream manufacturing activities are considered during the design phase, computational costs become problematic. Achieving interactivity requires a 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 use distributed algorithms to recognize machining features from solid models of parts with large numbers of features and many geometric and topological entities. Our goal is to outline how significant improvements in computation time can be obtained using existing hardware and software tools. An implementation of our approach is discussed.

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

1 Introduction

The availability of low-cost computational power is enabling the development of increasingly sophisticated CAD software. 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 for research in collaborative engineering and engineering design.

Automation of the design process and construction of such tools, however, pose many difficult computational problems. To realize the advantages of collaborative engineering, more downstream engineering

*Also with: National Institute of Standards and Technology, Manufacturing Systems Integration Division, Building 220, Room A-127, Gaithersburg, MD 20899.

activities are 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 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 is used for a variety of applications, including the generation of process plans [29], translation between design and manufacturing features, and production of redesign suggestions [5, 11]. 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 algorithms. Feature recognition has been approached using a variety of techniques, some of which are easier to parallelize than others. In previous work [24], 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. **Task initialization.** Initialization 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 traces; and (4) simplification of the part geometry to reduce the costs to solid modeling operations.

2. **Task distribution.** 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 to the application at hand—in the context of our previous work, this application is a system for automated manufacturability analysis and redesign for machined parts [12, 13, 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 in which 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 computational tools.

- It makes use of existing commercial solid modeling tools directly, and does not require parallelized versions of common algorithms or the implementation of multi-threaded, multi-processor solid modeling systems.

- It enables more interactive analysis and feedback. Recognition of the many alternative features can be done for complex parts in real time. This facilitates faster and more comprehensive analyses of manufacturability for 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 works 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 [21] present 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.* [18] 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.

Strip and Karasick [27] present techniques for performing solid modeling operations on a massively parallel SIMD (single instruction multiple data) computer. They provide a data structure for representation of solid models and a variety of parallel algorithms for implementing solid modeling operations. In addition, they present performance comparisons with serial implementations.

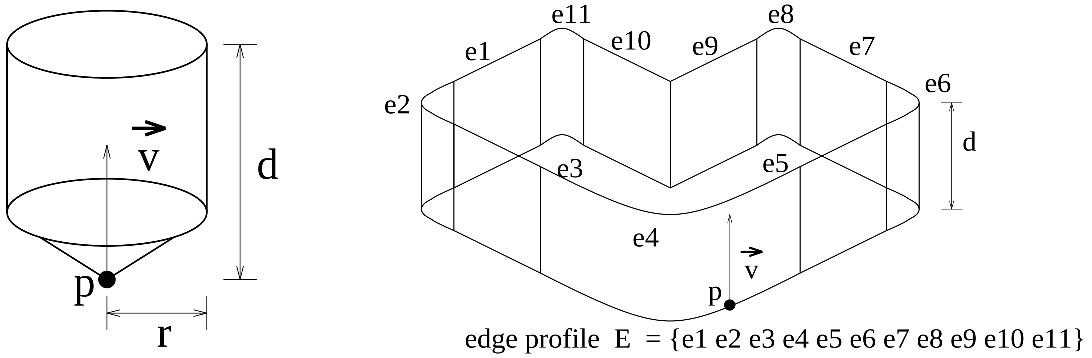
Existing work on recognition of features has dealt with exclusively serial computer architectures. These feature technologies are based heavily on the geometric and topological manipulation capabilities of solid modeling systems and deal predominantly with form or machining features. Much has been written on this topic in the literature and we will not attempt to cover all of this work here. We present below several of the more recent and relevant works.

The work of Henderson has continually brought new computational techniques to address the feature recognition problem. The work described in [15] was the first to apply expert systems to the feature recognition problem. Gavankar and Henderson [10] present techniques to identify protrusions and depressions in the boundary model of a part. More recently, Prabhakar and Henderson [23] described the use of neural networks to recognize and classify features. A strength of this approach is that it exploits the trainability of a neural net to incorporate new feature types. Further, neural nets have been demonstrated to be effective in classifying patterns in domains where there is "noise." This noise is in the form of incomplete or missing feature data lost due to feature intersections.

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 [17, 22, 26]. A common difficulty for both categories of graph-based approaches 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] explore graph-based pattern matching techniques to classify feature patterns based on geometric and topological information from the part. Efforts at Carnegie Mellon University [22, 26] have employed graph grammars for finding features in models of injection molded parts. Recently, Corney and Clark [4] have employed graph-based algorithms to find general feature classes from $2\frac{1}{2}$-dimensional parts.

Gadh and Prinz [9] were the first to describe techniques for combating the combinatorial costs of han-



(a): hole h, a drilling feature (b): pocket m, an end-milling feature

Figure 1: Examples of machining features

dling complex and realistic industrial parts (i.e., those with thousands of topological entities). They point out that, 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 their 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 to build feature instances. While this kind of approach holds much promise for addressing combinatorial problems,it does not address how to extend the techniques to better handle interacting features and more non-linear (non-faceted) solid models.

Fields and Anderson [8] present an approach to feature recognition that overcomes some of the representation and efficiency problems common in previous work. Unlike pattern-based or decomposition-based recognition methodologies, they categorize sets of faces on the surface of the part into classes of general machining features: protrusions, depressions, and passages. The shapes within each class, while sharing many operational similarities, may vary in geometry and topology. For each of their feature classes, they present a linear-time algorithm.

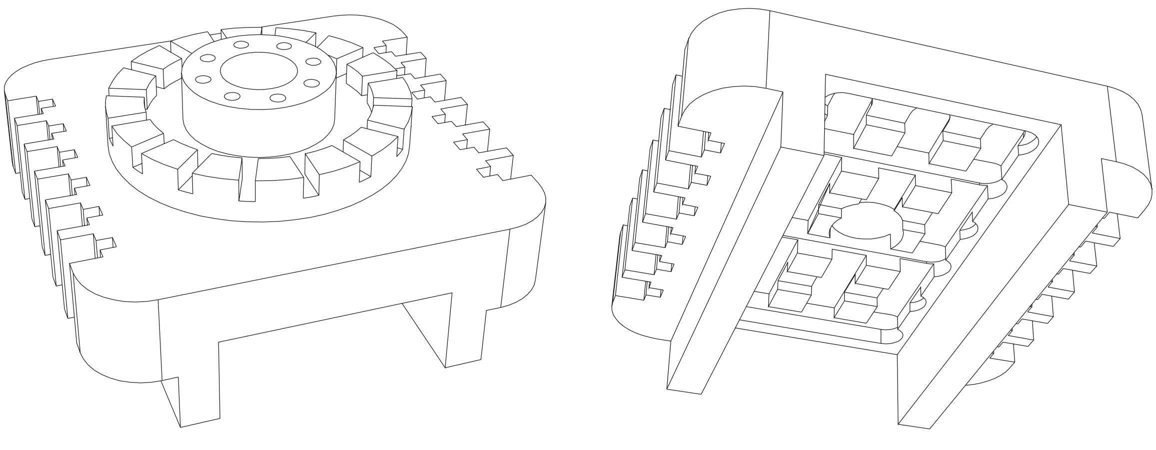
Many aspects of the feature recognition problem are still open and active areas of research. Among these are: recognizing and representing interacting features [29], incremental recognition of features [19, 14], modeling alternative feature interpretations and completeness [20, 24], and reasoning about the manufacturability of features [12].

3 Approach to Feature Recognition

In this section we outline a basic feature recognition technique on which we will build our multiprocessor algorithms in Section 4. We have chosen to adopt a trace-based methodology for this purpose; the reasons for this choice shall become evident. Fundamentally, a trace-based approach to feature recognition attempts to reconstruct feature instances from the information that they contribute to the final CAD model of the product.

The work of Marefat and Kashyap [20] presented an early trace-based technique. They expanded on the work of Joshi and Chang [17], augmenting it with hypothesis testing techniques. In Marefat and Kashyap's 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 [29] were the first to formalize trace-based (or hint-based) techniques for



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

Figure 2: An example part.

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, 24] present an approach for guaranteeing completeness of a recognition algorithm, i.e., it describes how one can define a class of features and verify that a particular approach was capable of producing all features in that class. They presented feature recognition as an algorithmic problem in which traces are found by traversing the geometry and topology of the part and then used to construct feature instances. They formally describe the behavior of their algorithm and calculate a general measure of its complexity. This approach has been employed for automated design analysis [12] and automated redesign [5, 11].

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

- Feature traces can be derived from a variety of design information such as tolerances, surface finish requirements, and functional 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 adapted to recognize features from a variety of manufacturing domains and processes. Existing feature recognition literature focuses primarily 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 also lend themselves well to parallelization, providing several levels at which the problem can be divided. What might be less evident is that, in parallelizing the problem, one can make additional geometric and topological simplifications to independent problem subtasks to reduce their computational difficulty.

The remainder of this section will specify a common example domain of machined parts and some basic trace-based recognition techniques. Using this domain we present a multi-processor recognition methodology in Section 4.

3.1 Machining Features

A *machining feature*, M, is a parameterized volumetric template that represents the solid volume removed from a workpiece by a machining operation. An instance of a machining feature, f, is created by a specific machining operation with a single cutting tool in one tool setup. To perform a machining operation, one sweeps the tool along some trajectory. 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 are referred to 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.

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 [16] 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 interpretation corresponding to a different way of manufacturing it. A *feature-based model* is a collection of features that models a single, unique interpretation of the part. The feature recognition problem can be defined as follows: given a collection of machining feature types, $\mathcal{M} = \{M_1, M_2, \ldots M_j\}$, a part P, and a piece of stock S, find the set $\mathcal{F}$ of feature instances in $\mathcal{M}$ recognized from P and S. The feature set $\mathcal{F}$ is a finite set of features composed of the union of those features in the alternative feature-based models for the part [24].

3.3 Trace-based Recognition of Features

A *trace* represents the partial information in the solid model of the part produced by an instance of a feature. Intuitively, a trace t_M corresponds to the information contributed to the part by an instance of a feature of type M. The trace provides sufficient information for calculating the parameters of a feature instance f of type M. For example, one trace for the drilling feature in Figure 1(a) is the conical ending surface of the hole h. Similarly, a trace for the end-milling feature m in Figure 1(b) is its bottom surface.

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

1. Each feature type M in $\mathcal{M}$ has associated with it set of traces $t_{M1}, t_{M2}, \ldots t_{Mk}$.

2. There is a procedure $\mathcal{P}()$ such that $\mathcal{P}(t_{Mi})$ constructs, from the information in the solid model of the part and stock material, instances of features of type M capable of producing the trace t_{Mi}.

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

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

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

3. For each potential trace t in $\mathcal{T}$ do:
   If t matches a t_{Mi}, call the procedure $\mathcal{P}(t_{Mi})$ and construct (if possible) feature instances, $f_1, f_2, \ldots f_n$ of type M. Add these to the set of all feature instances, $\mathcal{F}$.

Example traces. 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 surfaces are present on the boundary of the delta volume.

   The following two traces 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 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 $\mathcal{P}(t_{Mi})$ for constructing feature instances from these traces is not central to the focus of this paper. Such algorithms have been developed in previous work, notably: Vandenbrande [29] for drilling feature traces 1 and 2 and end-milling feature trace 1; and Regli *et al.* [25, 24] for all of the above traces.

4 Approach to Parallelization

In the distributed computing paradigm, collections of autonomous computational resources are interconnected on a network, as illustrated in Figure 3 [28]. 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 among the autonomous resources.

In this section, we will apply distributed algorithms to the example problem domain from Section 3.

4.1 Motivations

The feature types and their traces each introduce natural partition lines along which the problem can be divided into independent subproblems to be solved by different processors.

As presented in Section 3.2, the final feature set $\mathcal{F}$ contains all those feature instances from $\mathcal{M}$ that are members of feature-based models of the part. $\mathcal{F}$ contains all instances of the feature types in $\mathcal{M}$ present in the given part. Note that for the features in $\mathcal{M}$, the act of recognizing a feature of type M_1 is independent of the recognition of a feature of type M_2—hence the feature instances of type M_1 can be calculated separately from those of type M_2. For instance, in the example domain presented in Section 3, the fact that a particular drilling feature f is a member of some feature-based model does not alter the existence of any end-milling features.

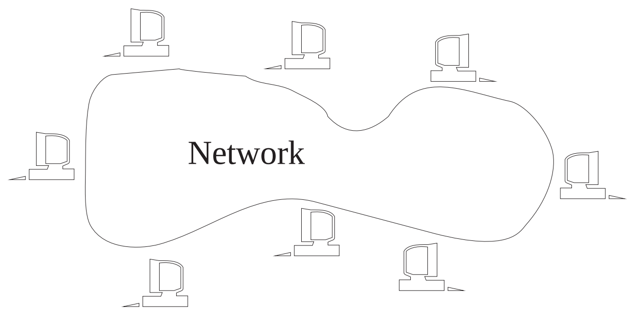


Figure 3: Internetworked computational resources.

Secondly, the set of traces $\mathcal{T}$ (from the generic algorithm in Section 3.3) introduces an additional level for partitioning the problem. Recall that for each feature type M in $\mathcal{M}$, there is a collection of traces $t_{M1}, t_{M2}, \ldots t_{Mk}$ for building instances of features of type M. One can decompose the problem of finding all features of type M using the traces by handling each trace t_{Mi} 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. There are two possible approaches to handling this redundancy. One method is to delete duplicate features while building the final feature set $\mathcal{F}$. A second approach, and the one that 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. This introduces another level of parallelization by dividing the set of traces found into independent subsets. In this way redundancies are addressed at the level at which they occur, thus simplifying the task of building the final feature set $\mathcal{F}$.

Parallelizing feature recognition produces other, less obvious benefits. In particular, a large portion of the costs in a feature recognition system are due to the complexity of geometric computations and geometric reasoning. When isolating independent problem subtasks, one can make geometric and topological simplifications that identify the information in the original part needed to build and verify the feature instances. In this way, many of the subproblems may require only a fraction of the information present in the solid models of the original part and stock.

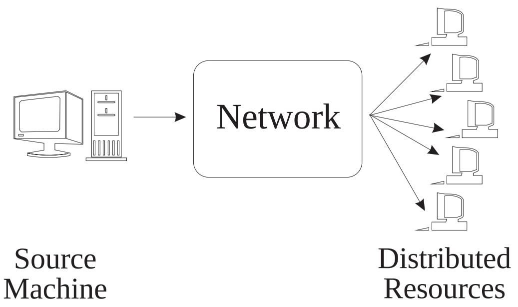


Figure 4: Source machine distributing the tasks to clients.

4.2 Distributed Methodology

For the example domain of Section 3, our approach is to have a central computing resource act as a server to set up the problem and transmit subtasks to client machines distributed on the network, as illustrated in

Figure 4. Each of the individual client processors is given an independent portion of the particular global feature recognition problem.

A distributed algorithm. Recalling the serial trace-based algorithm of Section 3.3, we present an outline for a multi-processor trace-based feature recognition system. There are two main components to this system: a server algorithm and a client algorithm. The server algorithm is presented as follows:

Server algorithm.

1. Input a collection of feature types, $\mathcal{M}$, a solid model for the part P, and a solid model for the initial stock material S. Initialize $\mathcal{F} = \emptyset$.
2. For each feature type M in $\mathcal{M}$ do
   (a) Fork a new process on a free resource
   (b) For each trace type t_{Mi} for feature type M do
      i. Find the set $T_{t_{Mi}}$ of instances of traces of type t_{Mi}.
      ii. Decompose the set $T_{t_{Mi}}$ into independent subtasks, $\tau_1, \tau_2, \ldots \tau_j$.
      iii. For each τ_i do
         A. Decompose the part P using the τ_i. Result P'.
         B. Fork a new process on a free resource to call the client recognition algorithm on P'.
      iv. Let $F_{t_{Mi}}$ be the set of features returned by the client.
3. $\mathcal{F} = \mathcal{F} \bigcup_{\forall t_{Mi}} F_{t_{Mi}}$.
4. Remove duplicate features from $\mathcal{F}$.
5. Return $\mathcal{F}$.

The client algorithm, to be invoked by the server on multiple computational resources, is presented below:

Client algorithm.

1. Input a feature type, M, a trace type, t_{Mi}, a set of instances of T of trace t_{Mi}, and solid models for the part P', and the stock material S.
2. Simplify the solid model of the part P'. Result P''.
3. Call $\mathcal{P}(t_{Mi})$ to build feature set $F_{t_{Mi}}$.
4. Return $F_{t_{Mi}}$.

To implement this client-server algorithm, three technical areas must be addressed:

Task Initialization There are four levels at which the recognition problem is initialized:

- *Types of features to be recognized:* different feature types (in this example drilling and end-milling) are considered by separate computing resources, as discussed in Section 4.1.
- *Types of feature traces:* different traces for each of the feature types are considered by separate computing resources, as discussed in Section 4.1.
- *Trace decomposition:* given a specific feature type and a trace for recognizing it, decompose the set of instances of this trace to independent subsets to subdivide the recognition task. This is discussed in Section 4.3.1.
- *Part simplification:* given a specific feature type and a trace for recognizing it, alter the geometric and topological information in the solid model of the part to reduce its complexity. This is discussed in Section 4.3.2.

Specific details are given in Section 4.3.

Task Distribution. Once tasks are initialized, the next phase is to distribute the individual tasks to the available computing resources. This is done by invoking a client feature recognition procedure for each separate task; each task to be performed on its own processor.

In the example domain in Section 3, distributing tasks is straightforward. This becomes more complex when bounds are placed on the number of available computing resources.

Synthesis of Results Each separate client procedure, upon termination of its portion of the recognition task, transmits its results back to the server machine. The features returned are then integrated into an overall solution. In this domain, recombining results requires building the final feature set as the union of those features returned by each client machine.

However, the fact that this example domain lends itself well to building an overall solution from the separate subtasks may not generalize to other manufacturing domains. For example, this phase might include additional computations such as modeling feature interactions, eliminating redundant features, or identifying compound features or feature groups.

4.3 An Example of Task Initialization

The task initialization 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 Section 5), 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 are 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 part simplification.

4.3.1 Trace Decomposition

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 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.).

We present a four step 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 types and traces proceeds as follows:

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

   Rationale: This collects all possible drilling traces that might be machined in the same orientation. Drilling features with multiple traces (e.g., several separate cylindrical faces) can be isolated and identified.

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 all faces sharing the same underlying surface.

   Rationale: This collects all possible end-milling traces that might be machined in the same orientation. End-milling features with multiple traces (e.g., a bottom surface divided into multiple subfaces) can be isolated and identified.

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

   Rationale: This groups all potential corner radii for end-milling features with the same machining orientation.

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.

   Rationale: This groups traces for end-milled features based on machining orientation; hence through features that can be machined in the same orientation are place in the same group.

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

4.3.2 Part Simplification

The objective of this step is to reduce the amount of data that must be considered by each processor to a minimum amount sufficient to construct feature instances from the traces it has been given. The goal is to reduce the cost of operations during feature recognition. For example, one can reduce the number of geometric and topological entities while still retaining the information required to construct feature instances from the particular trace. In this way, the complex part geometry not effecting the feature trace being considered can be eliminated.

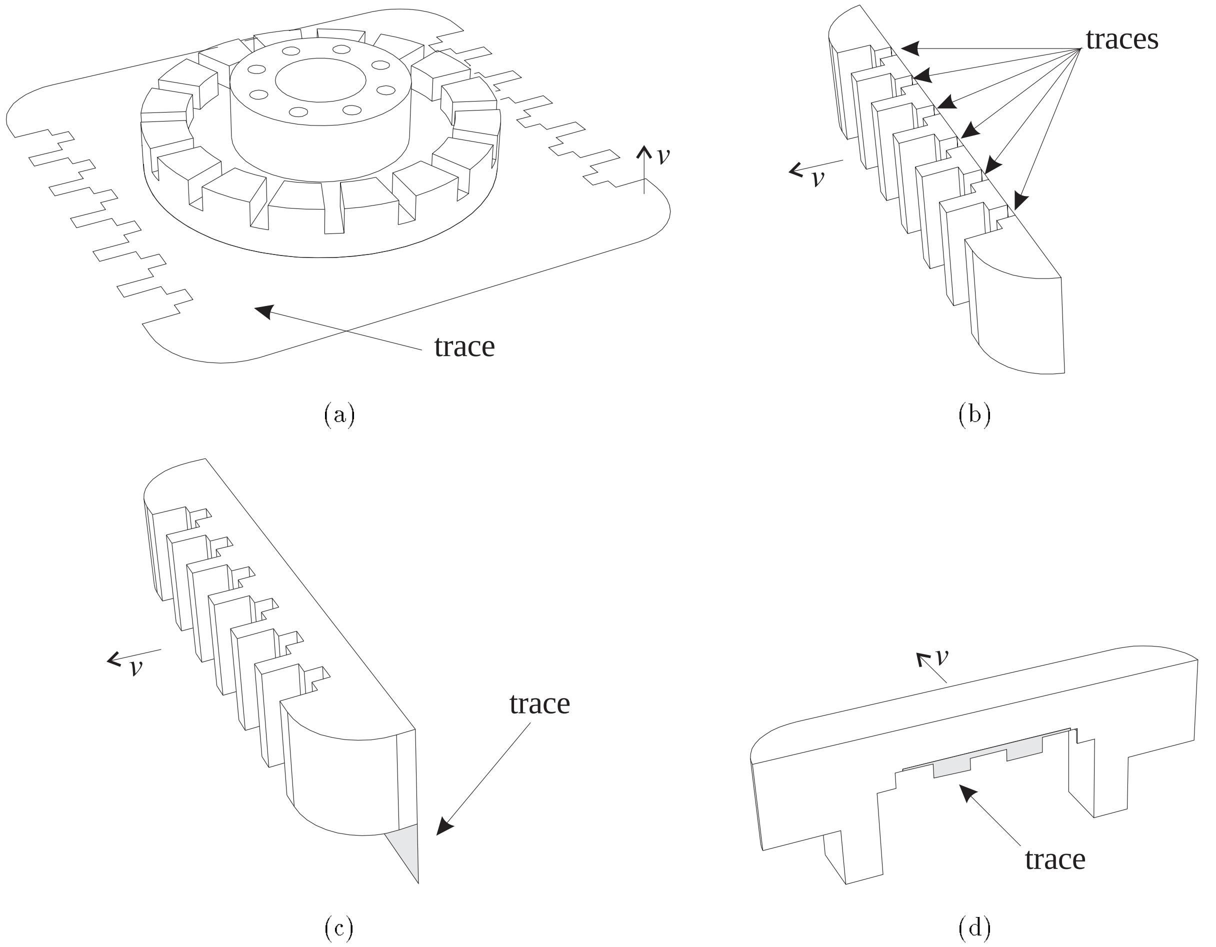


Figure 5: Four end-milling recognition subtasks and their simplified part information for the example in Figure 2.

We simplify the solid models of the part and stock based on the trace information and feature types. In each case, the geometry and topology of the model for the part P is simplified to P' as follows:

1. 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.

   Rationale: This simplification retains enough information to check for tool gouging and interference. To check for interference between the workpiece and the machine tool, this radius may be enlarged depending on the size of the tool assemblies available in the particular set of manufacturing resources.

2. Drilling trace 2:
   Given a conical surface c in the delta volume with a maximum 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.

   Rationale: This simplification is identical in intent to that for drilling trace 1.

3. 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.

   Rationale: This simplification retains all geometric and topological information that lies above the bottom surface of the milling feature and discards all information below it.

4. End-milling traces 2 and 3:
   No simplifications are made for these traces.

   Rationale: 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 several 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 Potential for Computational Improvement

We can expect the speedup to be no more than 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. These additions are negligible, however, when compared with the costs incurred to perform 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 $\mathcal{T}$ of potential traces, and the complexity of the methods for generating feature instances from traces.

A rough upper bound on the size of $\mathcal{T}$ can be computed from the model of the part and the types of traces by counting the number of geometric and topological entities. 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 solid modeling operations such as booleans, sweeps, and the like, the consensus is that these operations account for the majority of the computational cost during feature recognition [29]. The complexity of boolean operations appears to lie between $O(n^2)$ and $O(n^4)$ or $O(n^5)$ time, depending on the particular configuration of geometric entities and many implementation-specific details.

The fact that these basic solid modeling routines are at least quadratic in the size of the model implies that small reductions in the number of entities in the model translate into large reductions in computational cost.

In the next section, we provide rough estimates of both the speedup factor and the reduction in the number of geometric and topological entities achieved by this approach.

5 Implementation and Examples

A proof-of-concept implementation of this distributed feature recognition methodology, dubbed *F-Rex*, has been done in C++ using version 3.0.1 of the AT&T C++ compiler from SUN Microsystems running on networked SUN SPARCStations. F-Rex employs 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.

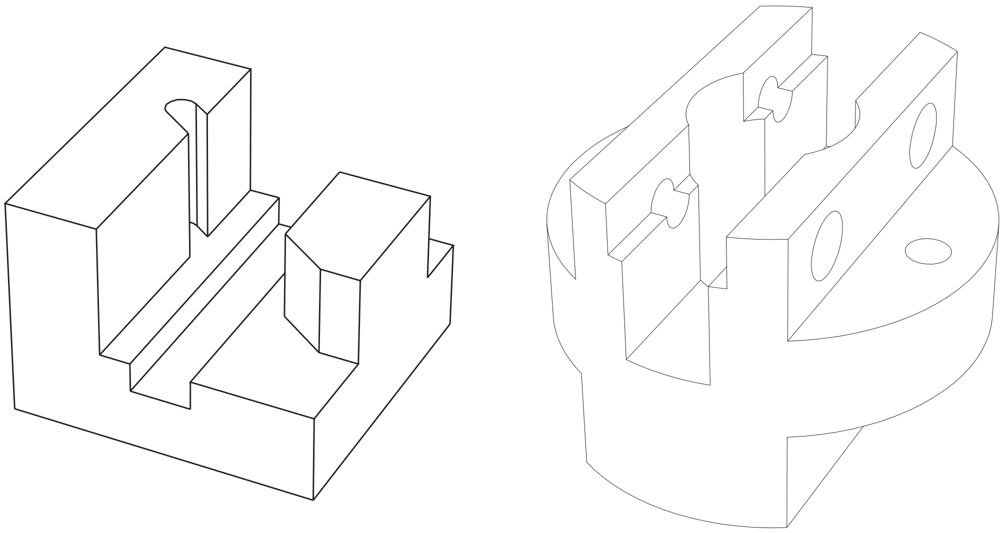
Additional tools include Ithaca Software's HOOPS© Graphics System and the Tcl/Tk embeddable command language and user interface toolkit from the University of California at Berkeley.

F-Rex is the feature recognition subsystem for IMACS, an interactive manufacturability analysis tool under development at the University of Maryland's Institute for Systems Research. One of the fundamental goals of IMACS is to provide interactive feedback and redesign suggestions to the user. Multi-processor algorithms have provided IMACS with a means of handling computational bottlenecks.

F-Rex runs on a cluster of SUN workstations; processes communicate over the Internet using UNIX-based and TCP/IP-protocol-based network software utilities and shared disk storage. The geometric computations required for task initialization are implemented with direct C++ calls to the ACIS kernel; distributed processes are invoked using UNIX remote shell commands; and the resulting feature set is generated by examining the features produced by each processor and eliminating redundancies.

The data for the examples below has been collected using six processors, one SPARCStation model 10, one model 2, and 4 IPX models. In this version of the implementation, when the number of tasks is greater than 6, the tasks are distributed evenly over the available processors.

These timing results represent the elapsed clock and CPU times and are not absolute measures of the intrinsic difficulty of the feature recognition problem—this example domain is not directly comparable to those of other feature recognition efforts. Further, there are hidden costs in the implementation not directly related to the recognition of feature templates (such as feature accessibility analysis) and these algorithms and their implementation can certainly be optimized. The results are intended to provide a rough indication of the time-lag experienced by the user of the system. More significant than any precise calculation of elapsed time is the speedup factor between the serial and parallelized algorithms. Measurements of elapsed CPU time are summarized in the table in Figure 8.



(a): example from [29]   (b): example from [24]

Figure 6: Two example parts addressed in previous literature.

Example 1. The example part in Figure 6(a), taken from [29], contains 21 part faces. Vandenbrande and Requicha [29] report identifying 7 features (3 slots, 3 open pockets, and a step) in two and a half minutes on a SUN 4/360. The OOFF system [29] handles a wide variety of machining features and process planning constraints; hence it is not directly comparable to the approach outlined in this paper. It does, however, provide a general indication of the computational costs required to recognize features in relatively straightforward parts.

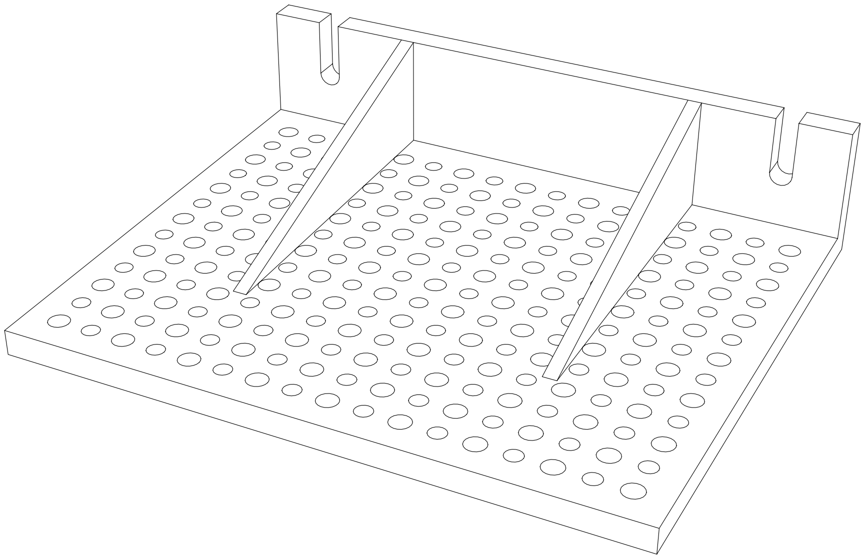


Figure 7: A fixture from ICEM's PART System.

Running in serial on the SPARC 10, our system finds 1 drilling and 8 end-milling features in approximately 1 minute. In parallel on 6 processors, it takes approximately 3 seconds to set up the decomposition and 12 seconds to recognize the features. Using the simplification techniques, the number of geometric and topological entities that had to be considered was reduced by 22%.

Example 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. There are 12 drilling and 20 end-milling features in its feature-based models that can be produced with the traces given above. In serial running on the SPARC 10, F-Rex identifies these 32 feature instances in approximately 65-70 seconds. When run distributedly, using 6 processors, F-Rex takes 10 seconds to set up the decomposition and approximately 12-16 seconds to identify the features. In this case, simplification resulted in a 35% reduction in the number of geometric and topological entities that had to be considered.

Example 3. The example part in Figure 7 is a fixture used in Control Data Corporation's[1] ICEM PART Process Planning System. The solid model for this part contains 245 faces. When running in serial, F-Rex takes over one hour to find the feature instances. In parallel, F-Rex takes 1.3 minutes to set up the problem and approximately 12 minutes to recognize the features. In this case, simplification resulted in a 23% reduction in the number of geometric and topological entities that had to be considered.

[1] This part was used with the permission of CDC and it is available for anonymous ftp from the NIST Process Planning Testbed at `ftp.cme.nist.gov`.

Example	Serial	Distributed	
		Set Up	Recognition
1	53.59s	0.96s	9.79s
2	126.90s	1.23s	6.39s
3	> 1920.0s	74.98s	699.82s
4	> 1920.0s	19.41s	701.47s

Figure 8: Table of elapsed CPU times for each example.

Example 4. The example part in Figure 2 is a shuttle intended to move along a guideway, with many of the feature instances added to reduce weight. The solid model of this part contains 281 faces. In serial, F-Rex takes over one hour to find the more than 100 feature instances. When running distributedly, F-Rex took 2 minutes to set up the task decomposition and approximately 32 minutes to find the features. In this case, simplification resulted in a 43% reduction in the number of geometric and topological entities that had to be considered.

6 Conclusions

Collaborative engineering and product design are pushing more downstream manufacturing issues into the design phase. The need to build effective and interactive design and analysis tools to address these issues 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 complex parts.

Discussion of Results

Our preliminary results confirm that performance gains can certainly be made through effective parallelization of algorithms. However, it is difficult to assess what a typical speedup factor will be. We suspect that the considerable variation in our experiments between parallel and serial speedup can be traced to the complexity of the parts themselves. The example part in Figure 6(a) has only one curved surface, while that in Figure 2 contains many dozens. A more general analysis of speedup factors would require testing the software against a set of benchmark parts of varying degrees of complexity.

We believe that parallelized trace-based feature recognition is highly suitable for parts in which the feature instances themselves are relatively simple, but numerous. It is not as well suited to problems where the feature instances themselves have very complex geometric configurations.

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 research 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.

Making this transition will require changes to 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, obtaining performance improvements increasingly will involve distributed algorithms.

We anticipate that this new distributed feature recognition technology will increase the complexity of mechanical parts within the reach of traditional feature recognition systems and will reduce the computational bottlenecks they pose. This will enable more sophisticated design analyses and, 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 the National Institute of Standards and Technology and National Science Foundation Grants DDM-9201779, IRI-9306580, and NSFD EEC 94-02384 to the University of Maryland. Additional support from the General Electric Corporation Forgivable Loan program was awarded to William Regli. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporting government and commercial organizations.

Certain commercial equipment, instruments, or materials are identified in this document. Such identification does not imply recommendation or endorsement by the University of Maryland, College Park, or the supporting government and commercial entities; nor does it imply that the products identified are necessarily the best available for the purpose.

The authors would like to extend their thanks to NIST readers Ted Hopp and Tom Kramer for their helpful comments.

Solid models for the example parts in this paper can be obtained through the World Wide Web at URL `http://www.cs.umd.edu/~regli/distrib/paper.html`. For more information on parts available through the NIST Process Planning Testbed, please see URL `http://elib.cme.nist.gov`.

References

[1] George Almasi, Raghu Karinthi, and Kankanahalli Srinivas. A parallel algorithm 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 trees 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] M. C. Fields and D. C. Anderson. Fast feature extraction for machining applications. *Computer Aided Design*, 26(11), November 1994.

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

[10] 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.

[11] Diganta Das Satyandra K. Gupta and Dana S. Nau. Generating redesign suggestions to reduce setup cost: A step towards automated redesign. *Computer Aided Design*, 1995. Also available as University of Maryland ISR TR 95:39,CS-TR-3439, UMIACS-TR-95-36.

[12] S. K. Gupta and D. S. Nau. A systematic approach for analyzing the manufacturability of machined parts. *Computer Aided Design*, 1995. To appear.

[13] 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*, 20(2/3):121–139, 1994.

[14] 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.

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

[16] Christoph M. Hoffman. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers Incorporated, CA, 1989.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] William C. Regli, Satyandra K. Gupta, and Dana S. Nau. Extracting alternative machining features: An algorithmic approach. *Research in Engineering Design*, 1995. To appear.

[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] D. Strip and M. Karasick. Solid modeling on a massively parallel processor. *International Journal of Supercomputing Applications*, 6(2):175–192, Summer 1992.

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

[29] 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):1269–1285, December 1993.