Implementation of a Feature-Based Object-Oriented CAD/CAM Package

by

Vance William Unruh

B.S., Kansas State University, 1987

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Mechanical Engineering

Kansas State University
Manhattan, Kansas

1988

Approved by:

Major Professor

Table of Contents

List of Figures

Acknowledgment

I would like to thank

1) Professor J. G. Thompson for acquiring the resources for this project.
2) Professor P. Krishnaswami for helping define my part of the project.
3) Richard League for making me program more elegantly.
4) Professor Brad Kramer for playing racquetball with me.
5) Mom and Dad for excusing my long absences from home.
6) The rest of the world for putting up with me.

This material is based upon work supported under a National Science Foundation Graduate Fellowship.

Chapter 1
Introduction

1.1 Background

In an effort to counteract the decline in the supremacy of American manufacturing, managers and engineers intensified their search for a new technology or management strategy which was going to once again give them the edge economically. Born of this search are several exciting and promising new technologies. A short list would include Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), links between CAD and CAM (CAD/CAM), Flexible Manufacturing Systems (FMS), robotics, Group Technology (GT), Data Base Management Systems (DBMS), and Computer Integrated Manufacturing (CIM). Most of these acronyms have become buzz words in industry so that their meaning is somewhat ambiguous. This ambiguity is unfortunate because when a new CAD/CAM product is introduced, it is unclear if it is a new 2-D drafting system or a real link between the design functions of CAD and the

manufacturing functions of CAM.

Among managers, CIM is used to mean anything from stand alone NC machines to a complete FMS, but neither of these components by themselves really integrate the various aspects of manufacturing. A true CIM system would provide all the functions of manufacturing - from production planning, to scheduling, to process planning, to material handling, to robotic controllers, to NC machining, to assembly - all under a hierarchy of supervisory computers. Current industrial CIM implementations combine at best a couple of these components. However, the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards (NBS) is close to a realization of a true CIM system. A scheduling program on the supervising computer initiates computer controllers in several devices on the factory floor. This allows for the interfacing of NC machines with robots which transfer materials from Automated Guided Vehicles to the NC machines. Numerous other interactions between different CAM components are also possible with the setup at the NBS facility.

Instead of focusing on the integration of manufacturing components, CAM usually refers to the many stand alone computer tools used by manufacturing engineers. Engineers use CAM products to schedule production, to implement Group Technology, to simulate manufacturing policies and plant layouts, to describe the tool paths necessary to machine a part, and to help with a variety of other functions. To qualify as a CAM application, a program need only aid in some aspect of manufacturing.

Computer Aided Design refers primarily to interactive graphics programs which allow for the creation of models of mechanical or electrical parts. At first CAD was little more than computer aided drafting. It was used to speed the production of and ease changes to the 2-D engineering drawings which characterized a part. Gradually, CAD moved from solely 2-D descriptions of parts using lines and points to full 3-D geometric descriptions using a higher level of geometry which included solids. This new data description allowed a more complete model. New data entry methods and Data Base Management Systems were devised so that a designer could enter a part using a variety of techniques such as constructive solid geometry, boundary representations, sweeping of curves, and cell decomposition depending on what CAD package was being used. Unfortunately, the only output of such a system was still the same 2-D engineering drawing showing the dimensions and geometry of the part.

As the power of computers increased and the Finite Element method of analysis became more usable, many CAD packages started to include a link to Finite Element (FE) programs. This link was usually achieved through a neutral data format that both programs could access. Thus, a part was designed on a CAD system and then the geometry of the design was written to a file using the neutral data format. Finally, the FE program was invoked which read the geometry from the neutral file. The designer could then create a FE mesh and perform the needed analysis to evaluate the design. Some CAD packages internalized the entire process by including a FE program as part of the package. This process was one of the first ways to share data between different systems.

As the CAD market grew and companies invested in more than one CAD package, the need arose to transfer the part data between CAD systems. Once again, a neutral data format was the answer. This time, however, the neutral format was standardized and called the Initial Graphics Exchange Specification (IGES). The existence of a standard, however, did not solve all the problems. Each vendor of a CAD package had to supply a translator between their own data types and IGES entities. As can be imagined, some translators worked a lot better than others making the early IGES transfers subject to errors. On top of these errors, an inherent problem of translating between greatly dissimilar systems was encountered. When one system was dealing with a 3-D constructive solid geometry (CSG) model and another was using 2-D lines and points, there was going to be, as with all languages, something lost in the translation. IGES is still evolving to improve the transfer, but current IGES versions provide an adequate bridge between CAD systems for the transfer of geometric part data.

As CAD continued to evolve, it became obvious that there were serious shortcomings to the available forms of output, those being IGES files or engineering drawings. The IGES files were inadequate because they were concerned primarily with graphics. There was no way, other than as notes, to transfer all of the other data such as materials, finishes, and tolerances which make up the entire description of the part. Even though engineering drawings contained the entire part description, they were also undesirable since none of the data could be used with computer tools until it was reentered - a time consuming and error prone process.

This lack of a good method of communication was one of the main

reasons that so called "Islands of Automation" developed in the manufacturing world instead of CIM applications. Instead of CAM applications being built which used the output of previous CAM programs and which output data in a form which later programs could use, vendors produced applications which could not communicate easily. In fact, most programs needed a human interpreter before the output was ready for the next step. Thus CIM applications were not developed because it was impossible for the different processes to communicate the necessary information between programs.

In an effort to solve this problem and to provide compatibility between contractors, the United States Air Force funded the Product Definition Data Interface (PDDI) project with the McDonnell Douglas Aircraft Company in St Louis, MO. The purpose of the project was to determine the requirements of a system which could replace the engineering drawing with a computer data file [1]. This accomplishment would allow the sharing of product data between computer systems without the costs and errors associated with data reentry and human interpretation. But before this exchange was possible, a standard analogous to IGES needed to be developed which addressed the complete product definition data necessary to unambiguously define a part. To satisfy these needs, work was begun on the Product Data Exchange Specification (PDES). Like IGES, PDES will only be a standard with which vendors will comply. When PDES version 1.0 is released, (and for some time thereafter), PDES translators will have the same problems that early IGES translators had, but eventually, PDES should provide a

means for adequate communication of product definition data.

Even if a neutral data format like PDES were currently available, it would be of little use since most CAD systems do not deal with the entire product data. All of the information about materials, finishes, special processes, tolerances etc. is usually entered as notes which are meant to go on an engineering drawing to be interpreted by a human. Before the long term gains of automation and increased communication can be fully realized, a language which can describe more than just the geometry of the part must be developed. It is hoped that feature technology can provide this language.

Features are the next higher conceptual level of part description above geometry. In a standard CAD system, a hole would be described as some circles and a line or possibly as a block with a cylinder removed depending on the system. With features, however, the hole would merely be an instance of a "hole" entity. The feature description would, of course, still have to contain the geometric data necessary to describe the position, orientation, and size of the hole, but it would tie this information together into a single feature - a hole. Additionally, the hole entity would contain information about the material, tolerance, and finish. All of this information would be combined to define a hole. A feature, then, is a high-level entity which contains all of the data necessary for its complete description.

Once the data is in this form, several manufacturing processes take a big step toward automation. With the hole, for example, a computer aided process planning program could make decisions on how to drill the hole based on the materials and tolerances associated with it. Without the

feature description, it is difficult for any thing but a human process planner to know that the geometry stands for a hole which should be drilled.

Research in feature technology is being conducted at several other sites. At Bendix Co. in Kansas City, MO., engineers are developing an expert system which does generative process planning from feature input [2] [3] [4]. The features are extracted interactively from a solid modeler and then fed into the rule-based expert system which performs the process planning. At Purdue University and the University of Arizona, automatic feature extraction is being researched [5] [6]. If successful, this would allow the data from standard CAD packages to be translated automatically into feature data. At the University of Massachusetts, "design with features" is being explored in the domains of extrusion, injection molding, and casting [7]. Here at K-State, the Center for Research in Computer Controlled Automation is exploring ways that feature-based design can be one of the building blocks for a completely integrated design, analysis, manufacturing, and assembly system which conforms to industry standards and automates many of the functions after the initial design.

1.2 Thesis Statement

The purpose of this project is to show that design features work well as the building blocks for a CAD system and that from a design feature representation of a turned part, all of the steps necessary to drive an NC lathe to manufacture that part can in most cases be done

automatically.

1.3 Method of Evaluation

These objectives were shown by the development of a prototype CAD/CAM package in which turned parts are designed by features and the Automatic Programmable Tool (APT) statements necessary to drive an NC lathe to machine the parts are generated automatically.

1.4 Equipment

The work was done and demonstrated on Apollo workstations using the GMR3D and GPR graphics routines with all computer code being written in the C language. Additionally, ElCam™ was used to verify the APT output.

1.5 Summary

In this chapter, the background of CAD/CAM was discussed and feature-based design was proposed as a means of overcoming some of the shortcomings of present implementations. In chapter 2, the requirements of the BigFeat CAD system are established and the implementation used to meet these requirements is discussed in detail. Additionally, the method of applying feature technology to the CAD/CAM problem through object-oriented programming is outlined. Chapter 3 outlines the steps needed for the CAM system to produce APT code to machine the part and outlines the implementation used to follow those steps. Finally chapter 4 presents the conclusions and discusses ideas for future research.

Chapter 2
Description of BigFeat CAD System

In chapter 1, the background of CAD/CAM was discussed giving impetus for the development of a more conceptually natural approach to the problem using feature-based design. In this chapter, features are defined more completely, the advantages of feature-based design are outlined, and the implementation of the CAD system is detailed.

2.1 Requirements of CAD System

Even if a feature-based CAD system provides help in integrating CAD and CAM, it will not be accepted if it is hard to use or if it is too restrictive. Designers should be able to create the same range of parts that are possible with a standard CAD package and they should be able to create them more easily. With these things in mind, a feature-based CAD

system should have the following properties:

1) Ease of use
2) Extensible feature set
3) Adaptability to individual needs
4) Powerful editing capabilities
5) Capability for entire product definition
6) Consistent reliable output

Section 2.4 covers the advantages that designing with features yields especially in the area of ease of use. The extensible feature set and adaptability of the system are covered in section 2.5. The power of the system and the ability to include the entire product data are discussed in section 2.6. Sections 2.7 and 2.8, give the implementation details of the CAD package, and finally, section 2.9 discusses the output.

2.2 Definition of Features

The first problem in developing a feature based CAD/CAM system is to define exactly what a feature is. Reference [7] describes a feature as

"a geometric form or entity :

1. whose presence or dimensions are relevant to one or more CIM functions ... or
2. whose availability to a designer as a primitive facilitates the design process."

The first part of this definition basically states that any bit of information needed to completely describe a part or product is a feature. The problem with such definitions is that the geometric entities used in standard CAD packages also fit this definition although they are by no means features. If the intent of the definition is understood, however,

then it is a fairly good description of a feature. In a pragmatic sense, features are the high level language used to describe parts when humans interact with humans instead of with a CAD package. When speaking of the hole in Figure 1, humans speak of a chamfer, not of the negation of a cone, or of the slanted line used in CAD packages. It is this higher level of abstraction which is the basis and intent of features.

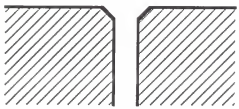


Figure 1: Cross-section of a hole with a chamfer

So why is it so hard to come up with a good definition of a feature? First, the field of feature technology is fairly young and some of the basic tenets have not yet crystallized as is discussed in the next section. Second, many parts do not have names any more abstract than their geometric descriptions. For example, Figure 2 shows a feature whose most abstract name is "free form geometric surface". What then is the proper name for the surface in Figure 2 in a feature based modeler which is supposed to be more abstract than geometry? There is really no choice other than to consider the free form surface to be a feature in some sense.

A feature, then, is just the highest level of abstraction available to describe a section of a part. Whether a high level word exists such as hole, or fillet, or gear, or if the low level attributes like free form surface are the most prominent, the highest one available is the current

feature description. As the language of features is further developed, more and more entities will be conceived as high level features which "aid in CIM functions and facilitate design".

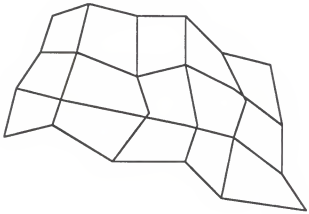


Figure 2: Free form surface

Once the features are chosen, they must be attributed, i.e. the detailed attributes which make up the feature must be established. Although many features have several different valid definitions, one possible set of attributes for a fillet are its radius and the edge which is filleted. There are several other properties which can be used to further define the feature such as tolerance, material, and finish which do not need to explicitly appear in the list of attributes. Some attributes, such as material, can themselves be defined as entities which can reference other features. To specify the material, an instance of steel, or some other material, is created which references the original feature. This gives two ways of associating an attribute with a feature. First the feature can explicitly reference the attribute by including it in the attribute list, or

second, the primary feature can be referenced by the generic attribute as is the case with the material.

2.3 Design vs. Manufacturing Features

Although the above definition helps in knowing whether or not an entity is a feature, it provides no help in choosing a good set of features for a given application or in attributing the features once they are chosen. There is in fact, some disagreement about the basic type of features which should be used. At Bendix, the user-extracted features are primarily features which describe the material which when removed from the blank will leave the part, i.e. they describe the volumes to be removed [3]. At the University of Massachusetts, on the other hand, the features add together to make up the part [7]. Since it appears that the difference comes with whether one is thinking of design or manufacturing, features such as those at Bendix are referred to as "manufacturing features" and the others "design features". This description is not meant to imply that manufacturing functions cannot be automated from a design feature description, it merely reflects the different ways in which design and manufacturing engineers think.

In most cases where feature extraction is being researched, and especially where the goal is process planning or some other manufacturing function, manufacturing features are being used. This choice of features is logical since most manufacturing features correspond directly to a machining sequence. Where design by features is being researched, however, design features are the logical choice since each feature corresponds to a distinct section of the part being designed.

Some who think in terms of manufacturing features feel that designing with features places too much unnecessary burden on the designer, and this is completely correct if manufacturing features are used. The designer would more or less do the entire process planning as he cut away sections to reveal the part he was trying to design. Design features on the other hand are fully as easy to use as conventional CAD systems and in many cases even easier. The disadvantage of design features is that the manufacturing steps do not necessarily flow as easily, although, in most cases, they can be derived.

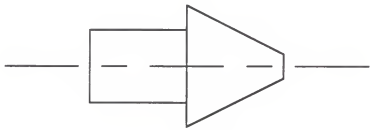


Figure 3: Simple turned part

As an example of the differences, consider designing the simple turned part shown in Figure 3. With design features, the part could be specified as a step and a taper, or as a step, taper, and three faces depending on the definition of the features. (See Figure 4.) With manufacturing features, the designer would start with the blank as shown in Figure 5 and remove the indicated volumes. Process planning from the manufacturing features is now straight forward. Merely remove each feature in the order of access. For the design features, the process is more difficult but still possible. A method needs to recognize the volume

between the part and the blank and output the correct APT motion statements which would remove the material in a workable manner.

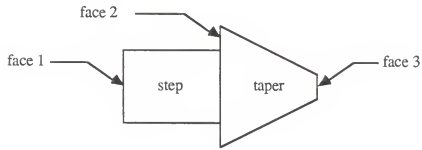


Figure 4: Design feature representation

Note that this extraction can also be done from a geometric description if the insight is given to the process planner that turning is needed to remove the material. Either the feature description or assumptions are needed to provide this extra data for automation. Thus, design features are much easier to use for design while manufacturing features make manufacturing automation much easier.

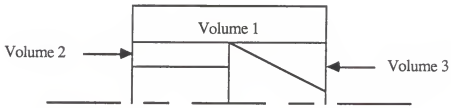


Figure 5: Manufacturing feature representation

One way to solve the problem of design and manufacturing features might be to extract manufacturing features from a design feature data base. Because of the extra information available, this task might be easier

than extracting features from a purely geometric database.

2.4 Advantages of Features

Basing a CAD system on features provides the following advantages:

1) More intuitive, higher-level design
2) Easier automation of manufacturing and analysis and
3) Ability to interface with an expert design system

The more intuitive higher-level design is a product of the higher level language used to describe the part. Designers are no longer forced to design with low-level lines, cones, or points. With features, they are able to describe a part to the CAD system with the same language they would describe it to another designer. Slots, holes, chamfers, and a large variety of other features can be entered directly into the model relieving the designer of the task of mentally translating the features of the design into their geometric counterparts.

The easier automation of manufacturing and analysis comes from the extra information which feature-based design includes in the data base. It is much easier for an automated process planner to identify a hole if it is an explicit entity in the data base than if it has to be extracted from the geometry.

Finally, the designer can be given expert advice during the design process if feature-based design is used. For example, standard size holes and shafts can be suggested or even enforced if desired. With a purely geometric description, the system would not immediately know that a hole was intended and thus could not access the knowledge about standard

size holes. Tolerancing issues can also be handled guiding the engineer away from unnecessarily tight tolerances which do nothing but increase the cost of the part. Questions of manufacturability could also be handled as the designer works. These advantages are further enhanced by the flexibility that comes with an object-oriented implementation.

2.5 Flexibility of a Feature-Based Object-Oriented System

Flexibility is a necessity for a feature-based system since the general field of features is not well defined and is subject to change. Because of this need, the easy addition of new features was given a high priority in this project. New features which are added can either be fully functional new features or macros built up from features which already exist. The addition or deletion of features does not require that large portions of the CAD package be rewritten or even recompiled to accommodate the change in the feature set as is the case for some data base management systems. Providing this kind of flexibility required that a kernel be developed which was independent of the features. This kernel is the heart of the CAD/CAM system and is fully described in [8]. The following is a brief description of how the flexibility of the kernel is achieved.

In a special file named "directory", the features are listed and arranged hierarchically into classes so that inheritance is possible . Each feature is then attributed in a separate ASCII file which is named "featurename.form". For the current listing of the hierarchical structure of classes and features used, see Appendix 2. These files are processed at run time by the kernel to set up the schema for the data base management

system. The dynamic nature of the feature list precluded the use of a standard data base manager.

The ability to change the set of features was deemed more important than using a traditional DBMS for two reasons:

1) It provides the ability to change and grow.

2) It provides the ability to customize.

The advantage of being able to easily change the set of features is especially apparent now when the field is young. As better ways of attributing the features are developed and as new features are devised which make design even easier, they can be built into the system without a lot of changes to the kernel of the CAD system.

The ability to customize is important because of the many features which can be used. A company which makes screws has no need for all the features used to describe composite materials. Thus, by editing the "directory" file, the system can easily be flavored to a given company's needs without carrying a lot of dead weight in unwanted features.

The next step in creating a CAD/CAM system is determining how the features can be accessed and manipulated and what operations can be applied to them. With the kernel of the CAD system being independent of the features, where should the knowledge needed to perform operations on the features be? For example, how does the system display an external step without previous knowledge of what one is? The solution used was an object oriented approach. Each feature comes with a set of subroutines, "methods", to perform the necessary operations. So in order to display a turned step, the kernel would execute the "display" method of the turned step.

It is reasonable to assume that if new features are being created, then it is also likely that new types of methods will be needed. For example, if the department gets a new holograph, it might be nice to display parts on it while still maintaining the capability to display objects on the standard terminal. This would require that every displayable feature have a previously undefined "holograph" method. Thus, the list of features and the list of available methods both needed to be variables. A list of possible methods was therefore included at the top of the "directory" file. The "directory" file gave a means of associating a subroutine name with a certain type of method for any feature. For example, the subroutine "display_a_turned_step" could be designated as the display method for a feature called "turned_step".

The inheritance mentioned earlier greatly reduces the number of methods needed. In many cases, one method can be written for a whole class of features eliminating the need for each feature to have its own method. If a turned step does not have its own special display function, it inherits the display method of its class. Methods for features can also be associated with the current modes of the package - design, analysis, and manufacturing.

Since an object oriented language such as C++ with its capabilities and possible restrictions was not available, a way of invoking the methods of a feature had to be developed. The problem boiled down to "How can a subroutine whose name is not known at compile time be referenced?" The name of the subroutine could be looked up at run time by means of the "directory", but a way of executing it was needed. On a PC, the

loader could be invoked providing an address which could be dereferenced, but Apollo workstations do not give the user access to the loader. Eventually, the problem was solved by binding all the subroutines to the main module and then deciphering the binary file to determine where in memory the subroutines had been put. For a complete description of this process and of the kernel in general, see [8].

In summary, to meet the requirements of flexibility, a kernel was built which was independent of the features and of the methods which acted on the features. The current features and the names of the methods are read from ASCII files at startup making the system completely changeable and adaptable to a given application's needs.

2.6 Power of a Flexible System

An open flexible system provides the following advantages:

1) Extra features can be added to describe the entire product definition data.
2) Extra methods can be added to supply powerful, personalized editing functions.

The ability to add extra features aids in two ways. First, it allows features to be customized to the user's needs and second, it allows generic attributes to be added to features. The example of a material given in section 2.2 applies here. One attribute of the material entity is the features which are made of that material. Other processes such as heat treatment and painting can also be defined so that the entire product definition data can be added to features in this way.

The ability to add methods provides the real power of the system.

If a desired input method is not available, the user can simply add it, making the system perform exactly as desired. For example, if a user needs a circle for part of a free form feature to be tangent to two lines with a given radius, he can write a method which will appear on the menu along with the rest of the design functions which will do exactly that - produce a circle tangent to two lines. That is the power of a flexible CAD system which is independent of the methods. The user can tailor it to his own needs making it very powerful and very easy to use.

2.7 List of Features Used

Since one purpose of this project is to show that manufacturing can be done from a design feature database, design features are used as the building blocks for the CAD system. The following features were chosen to design turned parts.

Name	Description
external	
step	Turned section parallel to centerline; geometrically produces a cylinder.
face	Turned section perpendicular to centerline; primarily used as transition between other features.
taper	Turned section at some other angle to centerline; geometrically produces a cone.
groove	Small linear or non-linear indentation usually cut with a form tool.
freeform	Arbitrarily shaped nonlinear turned section; geometrically, an arbitrary surface of revolution.
corner round	Rounded corner at an intersection of features.
fillet	Filleted corner at an intersection of features.
thread	Threaded section on another feature.

In addition to the features listed, there are also internal features which are analogous to the external features shown above. All features except for corner rounds, fillets, and threads have two edges which link them to their neighbors. Corner rounds and fillets are associated with the edge at the filleted intersection while a thread is associated with the threaded feature.

A more complex set of features such as was used with PDDI [9] was not chosen because of the burden it seemed to place on the designer. Another set of features called facing features was considered, but it was decided that this addition would only complicate the design process and added very little to the knowledge base needed by manufacturing. Facing features were to be the external features which would be accessed by a tool parallel to the centerline instead of perpendicular to it as shown in Figure 6. The drawback to this addition is that it would force the designer to decide if the taper in Figure 6 should be entirely faced or if part of it should be turned. These decisions need not be part of the design process. For this and similar reasons, the feature set was constrained to the simple set shown above.

Any turned part can be easily constructed from these primitives without requiring manufacturing decisions by the designer. This move away from manufacturing decisions does not mean, however, that nothing can be done about manufacturability questions; it only means that the designer is not burdened with detailing the manufacturing plan as he designs. Dixon in [7] provides a good example of addressing manufacturability questions from a design feature data base demonstrating that design features are a feasible tool both for design and

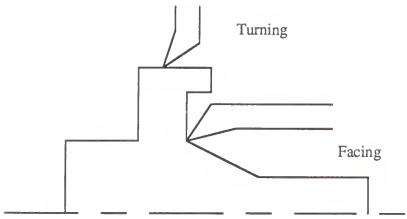


Figure 6: Application of facing features

manufacturing applications.

There are several supporting entities necessary to attribute and describe these features. For the detailed attributes of the complete list of features and entities used, see Appendix 1.

2.8 Implemented Methods of CAD System

The following methods were used in the design part of the system.

EditExist

These methods are used to edit an existing instance of an entity. The EditExist method of the root which all other features can inherit is a general purpose editor which allows the keying in of all data. It pops up a pad in which it writes the data type and name of all the attributes of the entity and a box in which changes can be made as shown in Figure 7.

The cursor can be moved between boxes by moving the mouse or hitting the tab key. Fields where another entity is required allow picking an existing entity from a list, editing that entity, or keying in the instance

number if it is known. The general purpose editor allows all new features to be attributed immediately, even before more convenient specialized input routines are written.

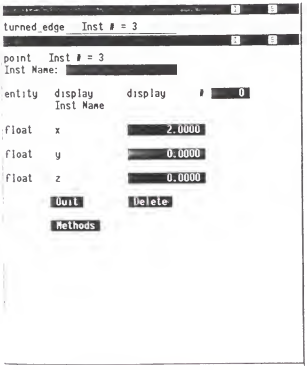


Figure 7: Example of editor

Only one class of entities needed modification to this routine, that being "machines". Each machine has a set of "machine functions" which it can perform like drilling or 2 axis turning. For speed and space considerations, these functions are stored as bits in an array of long integers. The special EditExist function for machines converts this array to the string representations of the machining functions. Thus, when the user is attributing a machine, he can work with the strings instead of with

the bits in the array.

BoundBox

These methods are used to get the physical limits of an entity in the global x, y, and z directions.

Hilight

These methods are used to in some way highlight a feature on the screen. In most cases, a box is drawn using the limits returned by the BoundBox routine. The Hilight methods are used primarily to help pick an entity from a list. For example, if someone is creating a turned step and needs to use a turned edge which already exists, a pad can be brought up which lists the available turned edges. As the cursor moves over the list of edges, the appropriate one is highlighted on the screen making the selection very easy.

SmoothCurve

These methods return the points along the curve. It is assumed that they return sufficient points so that the curve will appear smooth to other methods.

PointSlope

These methods return the endpoints of an entity and the slope of the entity at the two endpoints. The slope is represented as a vector tangent to the curve pointing from the endpoint back in the direction of the curve as shown in Figure 8. The direction of the PointSlope vector is important to several of the manufacturing methods.

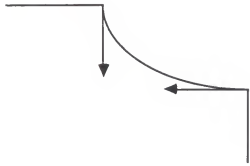


Figure 8: Direction of PointSlope vectors

Display

These methods draw the features on the screen using Apollo's GMR3D graphics package. GMR3D is a 3-D graphics package complete with advanced shading functions and hidden surface removal. The turned parts are displayed as a series of shaded rectangles. With bilinear shading on an 8-plane color graphics monitor, the shading appears smooth.

Most modes of display are changeable from the screen. The view volume, background color, type of shading, etc. are all variables which can be set. Thus the designer can switch from a 3-D shaded view of the part to a wireframe to a 2-D cross section quite easily.

Design

The following methods were included to make the design process easier. Instead of keying in all the information in the general editor, these methods can be used.

EdgeLength allows the complete specification of an internal or external step simply by picking an existing edge and keying in the length of the step along the centerline.

EdgeRadius allows an internal or external face to be specified by

picking an edge and giving the radius of the other edge.

EdgeRadiusLength permits the input of a new taper by picking an edge and specifying the length of the taper and the radius of the other edge.

InputBspline allows interactive graphic creation of B-splines which can be used with free form turned features. Input is mouse driven and can be easily edited. Existing control points can be moved or deleted and new control points can be inserted in any position. This method takes advantage of the local nature of B-splines and only redraws the small section of the curve which changes when a control point changes making a B-spline with 2000 control points as easy to work with as one with 10. The order of the curve can be changed and the control points and control mesh can be turned on and off for viewing purposes.

2.9 Available Output and Summary

The CAD section of BigFeat allows interactive graphic creation of a computer database which contains the complete product definition of the part being made. Hard copy output such as is shown in Figure 19 in the next chapter can be produced, but the primary purpose of the CAD program for this project was to prepare the database for use by the CAM program. For that reason, engineering drawings and other typical forms of CAD output were not supported.

This chapter has described features and feature-based design while detailing the implementation used for this program. Chapter 3 will describe how the database created by the CAD system can be used to make the manufacturing decisions necessary to machine a part.

Chapter 3
Description of BigFeat CAM System

In the previous chapter, the CAD section of BigFeat was considered. The advantages of designing with features were outlined and the specific object-oriented implementation was discussed. In this chapter, the steps necessary to take a design from the CAD system and use the CAM system to automatically produce APT statements for turned parts are detailed. Additionally, the efforts made to perform process planning for general parts are covered.

3.1 Overview of CAM System

The CAM section of BigFeat uses the same feature model as the CAD section to perform the manufacturing functions. Since the part is designed with features, many of the manufacturing steps can be done

automatically. In general, the steps required for a process planner to manufacture a part are as follows:

1) Pick the raw stock
2) Pick the machines
3) Pick the workholding
4) Pick the tool
5) Generate tool paths for that setup
6) Change the tool, workholding, or machine as needed to finish the machining

Unfortunately, these steps do not always follow the neat linear progression given above. There are many interdependencies among the steps which can make some of the decisions quite difficult if all of the factors are considered. Normally, a particular lathe or drill is chosen setting the available workholdings and tools which can be used by the process planner to make the necessary cuts. Sometimes, however, a part has a special need for a certain workholding or tool which is only available on one machine, thus influencing the choice of machine. So, in reality, steps 2 - 6 all have to be considered simultaneously to handle the general case. The general case, however, is simply too general to be handled efficiently at this stage making the system assume certain limiting conditions. The primary assumptions in the CAM section of BigFeat are that the steps above do follow the linear progression shown and that the workholding and tool are both adequate for the cuts proposed by the automated tool path generator. User intervention is allowed to override any of these assumptions. The limitations are not a prerequisite of the

structure of the system, only of the sanity of the author. Modules such as FIXPERT [10] could be interfaced with BigFeat to give workholding suggestions throughout the process, but, instead, the user is given the responsibility of changing the tool and workholding as needed.

Other packages make similar limitations. The previously mentioned FIXPERT chooses fixtures for turned parts only after the entire process plan has been formulated. And DMAP [11], which picks tools, makes no mention of workholding when it creates the tool paths. The problem is simply too big to try to handle the general case with the first try.

Even with these limitations, every effort was made to make the CAM section capable of handling all parts, not just the turned parts which were fully implemented. The CAM system was constructed so that it had the same generality as the CAD section. This generality meant that most decisions in the process planner are made at the feature level by executing the methods of each feature.

3.2 Picking the Raw Stock

Picking the raw stock interactively is relatively easy. Simple geometries such as bar stock are created automatically while more complex geometries are designed using the CAD system in the same way that the designer made the original part. Once the raw stock exists, the user simply picks it from a list to specify it as the desired raw stock.

3.3 Picking the Machine

Picking the machine (or machines for more complicated parts) which is needed to process the part is done automatically. The first

option considered for selecting the machine was to just assume a certain machine as is done with some systems which produce automated tool paths. But this option limited the system too drastically and violated the goal of a general CAD/CAM package. If output were needed for a different machine, almost all of the CAM program would have to be rewritten. Obviously, for a general CAD/CAM package, assuming a certain machine is unacceptable.

The second option was to allow the user to describe the machines in the factory and any other available resources to the BigFeat program. In this scheme, a lathe and a 2-axis milling machine are entities to the CAD/CAM package. At installation time, the factory is described and all of the machines are attributed. When the user is ready to pick a machine for a particular part, he loads the file which describes the factory into memory so that the process planner knows what machines it can pick to process the part. A manufacturing tool with this generality could be used by a manager to decide which factory site could best handle the production of a part by loading different factories and evaluating the output. This option provided the flexibility and generality needed and was therefore implemented. Austin [12] also suggests this approach of variable machines and processes in the Computer Managed Process Planning (CMPP) project.

Now that a method of fixing the available machines was established, the next step was to route the parts to the correct machines for processing, i.e. a hole needed to be routed to a drill and a turned step to a lathe. The first option considered was for each feature to request a

certain type of machine. A hole would request a drill which could meet the size and tolerance specifications of the particular hole. This method had a problem, however, when it came to multi-purpose machines. The features would have no way of knowing that an NC lathe with live tooling was capable of cutting a slot in the end of the part. They would only know that a lathe could do turning. This lack of knowledge would produce unnecessary additional setups.

This problem made it clear that the important thing about a machine is not what kind it is, but what functions it can perform. Thus, the second and preferred option for establishing the correspondence between features and machines was to make the functions of the machine part of the attribute list. When a hole needs "drilling", it can then be matched with a machine which can do "drilling". Multi-purpose machines can list all of their machining functions giving the features the knowledge necessary to avoid the additional setups of the previous scheme. To speed the progress of matching a feature with a machine which can produce it, all of the machining functions are represented as a member of a set. A hole then decides that it needs "drilling", determines that "drilling" is the 5th member of the set, and returns the fact that it needs the 5th machining function to the process planner. The list of possible machining functions is declared in the same "directory" file which declares the entities and the hierarchical class structure.

Features are not limited to one machining function. If a hole needs to be both drilled and reamed, then its method can return both machining functions to the process planner. The capability of alternate machining

functions should also be included for cases when more than one function can supply the needed machining. For example, in some cases, an internal step could either be drilled or bored on a lathe. The current implementation allows for multiple functions from one feature, but it does not support alternate functions.

Once features could be routed to machines based on the machining functions needed, the next problem was to pick the best machines for the job in the correct order using the same high-level reasoning as a human process planner. That undertaking in artificial intelligence, however, is outside of the scope of this thesis. Expert systems such as XCUT [2] [3] could possibly take the input from this stage and make the necessary decisions, but no attempt was made to emulate the high-level reasoning of a process planner in this project. Once the framework was set up so that more elaborate process planning could be done, a process planner was developed to meet the needs of this project, namely to route turned parts to a lathe.

In the implemented process planner, the PickMachine method of each machine evaluates the machine's ability to process the part. First, the dimensions of the part are checked to be within the size limitations of the machine, and then a score is computed to compare it with other machines. Currently, the score is simply the number of needed machining functions that a machine can supply to the part. If a feature needs more than one function but the machine under consideration can only supply the 2nd and 3rd ones, then nothing is added to the machine's score since it is assumed that the functions must be done serially. The machine will have to wait until another machine supplies the first

machining function before it can add the 2nd and 3rd ones to its score.

After all the machines have been evaluated, the one with the high score is chosen as the current machine. All of the features that can be processed by that machine are marked and the cycle is repeated to select all subsequent machines until either all of the features are processed or until it is determined that the part cannot be manufactured with the current resources. This simple "most is best" algorithm allows for multiple machines to be selected in a workable manner, but it by no means makes the best selection for the general case. It does not consider accessibility questions which would have to play a big part in the process planning of prismatic parts but which are relatively unimportant with turned parts. Nonetheless, the framework is in place upon which more elaborate process planners can be built.

3.4 Picking the Workholding and Tool

The selection of the workholding and tool are both straightforward. In each case, the user is asked to select from a list of available workholdings and tools which each machine has as part of its attribute list. Again, this step could be automated with an expert system making the choices, and the entire module could be integrated into the tool path generation step.

3.5 Generate Tool Paths

Taking the output from the process planner, the last step in manufacturing a part is to describe the actual tool paths needed to process

the part. In order to verify the generated tool paths, they will be written using the Automatic Programmable Tool (APT) language. Both the geometry and the tool paths will be verified using ElCam™, a CAM program which can be used for part programming and NC verification.

The steps in generating the toolpath statements are as follows:

1) Generate the geometry statements

2) Trim the geometry and

3) Generate the motion statements

Generating and trimming the geometry is handled at the feature level by a method. Each feature outputs the correct APT statements to produce and trim its own geometry.

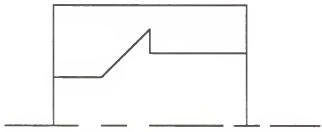


Figure 9: Simple turned part

The motion statements, however, could not easily be handled at the feature level. A more global view was needed to produce adequate tool motion than is generally possible at the feature level. For some features, such as holes, a method can be written which will produce the APT code to orient the drill correctly and make the cut. For other features, like an external turned step, the tool motion which can be produced at the feature

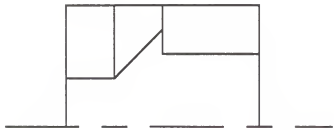


Figure 10: Unnatural manufacturing features

level is unacceptable. Consider the part shown in Figure 9 as an illustration. The method for any given feature could only remove the material above the feature. This strategy would produce the unnatural volumes to be removed shown in Figure 10. A better and more natural result is shown in Figure 11. These volumes are best extracted from a global view of the geometry.

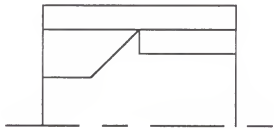


Figure 11: Natural manufacturing features

Although these volumes might not represent the optimal cutting sequence, they do present a workable and reasonable approach to machining the part. The feed, speed, and depth of cut of each tool motion can be varied to meet the tolerance and finish requirements of the completed part. Working from a global view provides the most satisfactory volumes to be removed. Extracting volumes is equivalent to

extracting manufacturing features from the database.

3.6 Apt Geometry

For turned parts, the geometry and tool paths can all be described in 2 dimensions so the APT geometry is written in the xy-plane. It would have been relatively easy to translate the part geometry to APT statements if no attention were given to the intent behind the features. The linear sections (tapers, steps, and faces) could have all been described as lines passing through two points, but to give a truer picture of the intent of the features, the richness of the APT language was used. Steps were described as lines parallel to the centerline and faces as lines perpendicular to the centerline. Then, if the part programmer needs to modify anything, this higher level of description will ease the changes by highlighting the important aspects of the design. For example, with a step, the programmer would not be left worrying about the significance of the two points which the line passes through. The important concepts which the designer used would be obvious, namely that the line is parallel to the centerline with a certain radius.

The same reasoning was used with fillets and corner rounds. They could have been described as circles with a center point and radius, but the intent of the feature is conveyed more accurately by a circle with a given radius which is tangent to two other features. The possible syntaxes for an APT circle of this type are as follows:

id = CIRCLE/mod,line1,mod,line2,RADIUS,r

id = CIRCLE/mod,line,mod1,mod2,circle,RADIUS,r

id = CIRCLE/TANTO,line,mod,tabcyl,mod,pt,RADIUS,r

id = CIRCLE/TANTO,circle,mod1,mod2,tabcyl,mod,pt,RADIUS,r

The id is the name that the APT program associates with the circle, and the modifiers are used to distinguish between cases similar to the four shown in Figure 12.

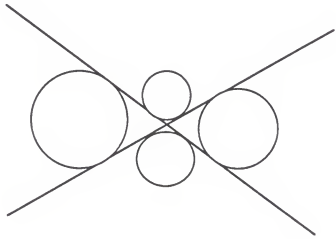


Figure 12: Possible circles tangent to two lines

The AptGeometry method for a fillet or cornerround calls the AptEdgeModifier method of the two features which intersect at its edge to determine the appropriate modifiers. For a linear feature, the possible APT modifiers are XLARGE, YLARGE, XSMALL, and YSMALL. The correct one was chosen according to the following observation. The slope of the other feature always points into the half-plane, created by the current feature's slope, which contains the circle. Figure 13 illustrates this condition. Notice that regardless of whether the circle is part of a corner round or a fillet, the argument still holds.

If the current feature has a horizontal slope, then YLARGE or YSMALL is chosen based on the y-component of the second feature's

slope. Similarly, if the current feature is vertical, XLARGE or XSMALL is chosen according to the x-component of the second feature.

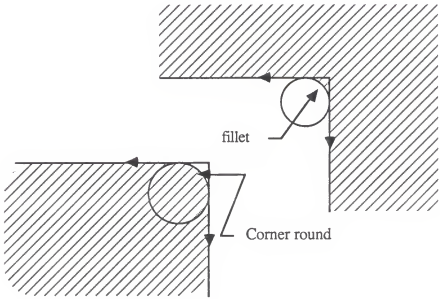


Figure 13: Scheme for choosing between XLARGE, YLARGE etc. Slopes point to half-plane which contains circle.

If the current feature is at some other angle with the x-axis, the cross product of the two slope vectors (other_slope X current_slope) is used to determine the correct modifier. If the slope of the current feature points into the first or second quadrant, then XSMALL and XLARGE correspond to negative and positive z-components respectively of the cross product. If the slope of the current feature points into the third or fourth quadrants, the roles of XLARGE and XSMALL are reversed.

For a circle, two modifiers are needed, one of the set XLARGE, YLARGE, XSMALL, YSMALL and another from the set IN, OUT. The two possibilities for IN and OUT are illustrated in Figure 14. The first

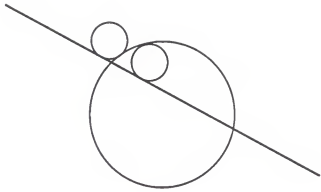


Figure 14: Two possible circles tangent to
a line and a circle

modifier for a circle is determined in the same way as for the linear features. The second modifier is chosen based on whether the slope of the other feature points toward the center of the circular feature. Figure 15 shows the slope actually passing through the center of the circle, but as long as the tangent points into the half-plane which includes the center and is created by the tangent to the circle, then IN is the correct modifier. In similar ways, fillets and corner rounds can be placed between any two intersecting features.

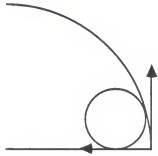


Figure 15: Scheme for choosing between IN and OUT
Slope points to half-plane which contains the circle

Freeform features pass the message to create APT geometry on down to the curve which defines their shape. The curve will then make the appropriate APT entity depending on the type of curve it is. Arcs and B-splines are the only curves which are fully implemented in this way.

3.7 Trimming the APT Geometry

APT normally works with unbounded geometry so that when viewed on a terminal, all lines extend completely across the screen and all arcs are full circles. This convention, however, leads to a very confusing picture quite quickly. In order to make sense of the drawing on the screen, the lines and circles have to be trimmed using some of ElCam's extensions to the APT language. These extensions appear as comments to a standard APT compiler so no loss of generality results from using them.

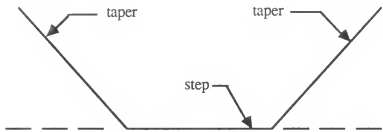


Figure 16: Trimming a step with a taper

Linear features are trimmed by their neighbor on either side, where neighbors are defined as the other users of the feature's two turned edges. Thus, the step in Figure 16 is trimmed by the two tapers. If the edge is filleted, however, there is more than one neighbor on that side

and a choice must be made about which feature should be used. In all cases, the fillet is the correct choice, but instead of hard coding this fact, another method called AptTrimPriority was written. After calling the AptTrimPriority methods of the taper and fillet, the step in Figure 17 will know that the fillet has the higher priority so it will be used to trim on that side.

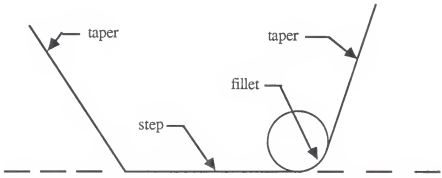


Figure 17: Step trimmed by fillet

Fillets were once again the trickiest features to implement. To know which part of the circle should be erased and which part left, ElCam requires that the two trimming entities be given in the correct order. Figure 18 shows the two possibilities. The correct trimming occurs when the slope of the first feature crossed with the slope of the second feature produces a vector whose z component is negative. Thus, the second fillet would be trimmed by the statement

$$EXTRIM,fillet2,step,taper1.

The incorrect trimming of the first fillet is caused by the wrong order of the two trimming surfaces in the statement

$$EXTRIM,fillet1,step,taper2.

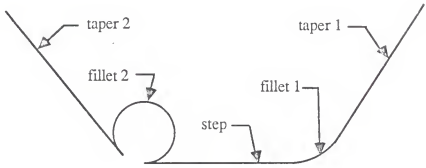


Figure 18: Trimming a fillet

A part similar to the turned part used in the PDDI project [9] was entered into the CAD system for testing purposes. Figure 19 shows the 2-D graphical picture created in ElCam™ when the file in Figure 20 is loaded and executed. Figure 20 is a listing of the output of the AptGeometry routines and Figure 21 is the wireframe representation of the part.

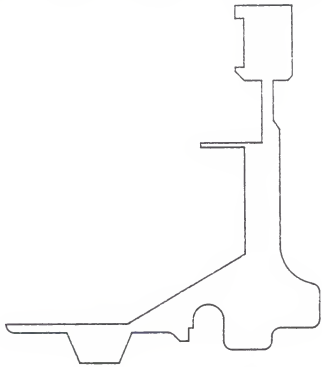


Figure 19: ElCam™ verification of geometry

```
$$ABOUND
PL1=PLANE/XYPLAN,0
LN1 = LINE/0.000000,0.000000,0.000000,-4.500000,0.000000,0.000000
T241 = LINE/PARLEL,LN1,YLARGE,1.562500
T231 = LINE/-0.687500,1.562500,0.000000,-0.687500,1.250000,0.000000
T242 = LINE/PARLEL,LN1,YLARGE,1.250000
T232 = LINE/-1.375000,1.250000,0.000000,-1.375000,1.687500,0.000000
T233 = LINE/-1.812500,1.687500,0.000000,-1.812500,1.562500,0.000000
T243 = LINE/PARLEL,LN1,YLARGE,1.562500
T234 = LINE/-1.875000,1.562500,0.000000,-1.875000,1.375000,0.000000
T244 = LINE/PARLEL,LN1,YLARGE,1.375000
T251 = LINE/-2.000000,1.375000,0.000000,-2.125000,1.500000,0.000000
T245 = LINE/PARLEL,LN1,YLARGE,1.500000
T252 = LINE/-2.687500,1.500000,0.000000,-2.875000,1.062500,0.000000
T246 = LINE/PARLEL,LN1,YLARGE,1.062500
T253 = LINE/-3.437500,1.062500,0.000000,-3.625000,1.500000,0.000000
T247 = LINE/PARLEL,LN1,YLARGE,1.500000
T281 = LINE/PARLEL,LN1,YLARGE,1.625000
T291 = LINE/-2.750000,1.625000,0.000000,-1.062500,2.625000,0.000000
T282 = LINE/PARLEL,LN1,YLARGE,6.125000
ARC2 = CIRCLE/CENTER,(POINT/-1.593750,1.687500,0.000000),$
                (POINT/-1.375000,1.687500,0.000000)
FIL1 = CIRCLE/YSMALL,T245,XSMALL,T251,RADIUS,0.125000
FIL2 = CIRCLE/YLARGE,T281,XSMALL,T291,RADIUS,0.125000
FIL3 = CIRCLE/YSMALL,T241,XLARGE,T231,RADIUS,0.125000
CRD2 = CIRCLE/YLARGE,T242,XSMALL,T231,RADIUS,0.125000
CRD3 = CIRCLE/YLARGE,T242,XLARGE,T232,RADIUS,0.125000
T283 = LINE/PARLEL,LN1,YLARGE,4.125000
T284 = LINE/PARLEL,LN1,YLARGE,4.187500
T285 = LINE/PARLEL,LN1,YLARGE,5.062500
T286 = LINE/PARLEL,LN1,YLARGE,5.250000
T287 = LINE/PARLEL,LN1,YLARGE,5.937300
T288 = LINE/PARLEL,LN1,YLARGE,5.062500
T301 = LINE/-4.500000,1.500000,0.000000,-4.500000,1.625000,0.000000
T302 = LINE/-1.062500,2.625000,0.000000,-1.062500,4.125000,0.000000
T303 = LINE/-1.687500,4.125000,0.000000,-1.687500,4.187500,0.000000
T304 = LINE/-0.812500,4.187500,0.000000,-0.812500,5.062500,0.000000
T305 = LINE/-1.187500,5.187500,0.000000,-1.187500,5.250000,0.000000
T306 = LINE/-1.062500,5.250000,0.000000,-1.062500,5.937300,0.000000
T307 = LINE/-1.187500,5.937500,0.000000,-1.187500,6.125000,0.000000
T308 = LINE/-0.375000,6.125000,0.000000,-0.375000,5.125000,0.000000
T309 = LINE/-0.650000,5.062500,0.000000,-0.650000,4.500000,0.000000
T3010 = LINE/-0.562500,4.375000,0.000000,-0.562500,2.750000,0.000000
T3011 = LINE/0.000000,2.187500,0.000000,0.000000,1.562500,0.000000
T292 = LINE/-1.062500,5.062500,0.000000,-1.187500,5.187500,0.000000
T293 = LINE/-0.375000,5.125000,0.000000,-0.437500,5.062500,0.000000
T294 = LINE/-0.650000,4.500000,0.000000,-0.562500,4.375000,0.000000
ARC1 = CIRCLE/CENTER,(POINT/0.000000,2.750000,0.000000),$
                (POINT/-0.562500,2.750000,0.000000)
CRD1 = CIRCLE/YLARGE,T241,XSMALL,T3011,RADIUS,0.125000
CRD4 = CIRCLE/YLARGE,T247,XLARGE,T301,RADIUS,0.125000
CRD5 = CIRCLE/XSMALL,T3011,YSMALL,OUT,ARC1,RADIUS,0.125000
```

Figure 20: Output of APT Geometry routines

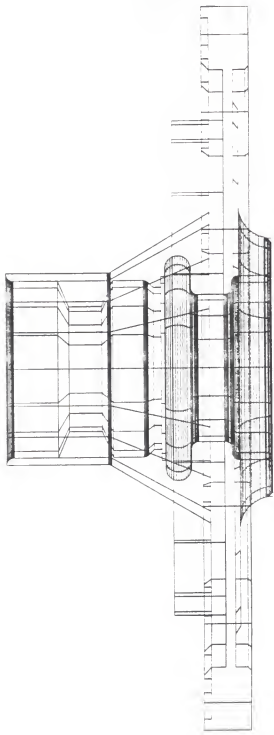


Figure 21: Wire frame representation of test part

3.8 Generating the Motion Statements

As is discussed in section 3.5, the motion statements are produced from a global view of the geometry. Given this fact, the strategy used by humans to extract manufacturing features from turned parts was examined. It was discovered that,whenever possible, the volumes extracted were the largest contiguous regions with a horizontal top and bottom.

To extract these regions, the tool can be imagined starting at the far right edge of the manufacturing feature and moving to the left until some interference is found. The interference can be the other end of the raw stock, the part, the workholding, or an overhang of any of the above. The condition of stopping at an overhang is shown in Figure 22.

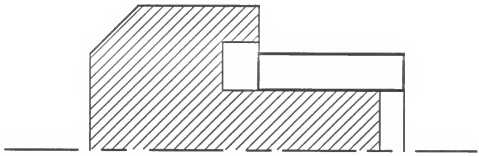


Figure 22: Manufacturing feature ending at overhang

The obvious and straight-forward approach to finding the interference is to calculate the first intersection of the line of motion of the tool path with every feature of the part, raw stock, and workholding. The intersection with the highest x-value less than the starting point is

then the stopping point for the tool. Complex geometries make it difficult or impossible to limit the set of entities checked for intersections making this strategy too slow and computationally expensive for anything other than simple parts.

As an alternative, the following graphically intuitive method was used. First, a grid of bits was developed and the geometry was drawn onto the grid. A 2-dimensional array of unsigned long integers provided the grid of bits. Drawing the geometry to the grid was then exactly the same problem as lighting the correct pixels on a display screen. Once the geometry is drawn on the grid, the task of finding the first intersection reduces to finding the first row with a bit set at or above the current y-position. Finding the intersection can now be done with integer comparisons instead of intersection calculations so the potential speed is greatly improved for complex geometries. The arrangement of the grid is shown in Figure 23.

Bit i,j is an interference point if one of the integers above grid[i][j/bpl] is greater than zero or if grid[i][j/bpl] is greater than the numerical representation of the current y-position (overhang case) where bpl is the number of bits per long integer. Additionally, interference is encountered if the (j mod bpl) bit of grid[i][j/bpl] is set (collision case).

Using this method, manufacturing features can be extracted from the data base. A manufacturing feature was defined as having start surfaces, end surfaces, and a bottom surface, all of which pointed to the various design features comprising the surfaces. In addition, each surface was of a particular type, either part, blank, workholding, or overhang depending on which contributed the surface.

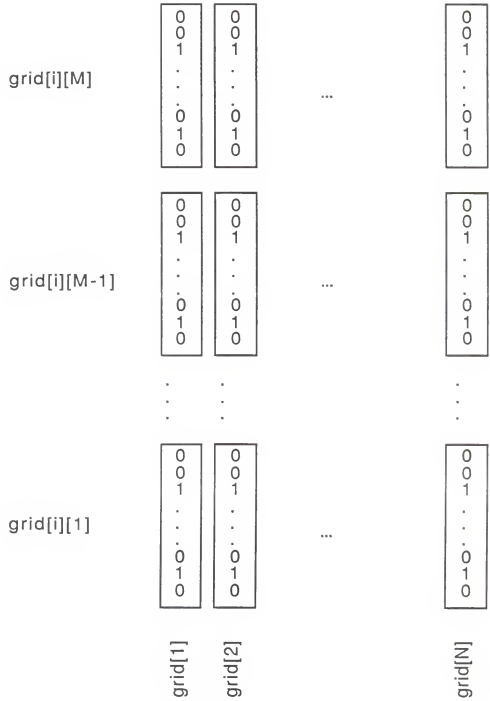


Figure 23: Arrangement of bit grid

The actual process of creating the motion statements consisted of the following steps:

1) Put the part and blank surfaces in order
2) Find a starting position
3) Establish the end surface
4) Establish the bottom surface
5) Output the motion statements
6) Reflect the effects of cutting in the blank
7) Reorder the new blank surfaces
8) Find a new starting position and repeat the process

The fist step, ordering the part surfaces, is accomplished by finding the two ends of the part. An end is identified by an edge of zero radius or by an edge used by both an external surface and an internal surface. The external feature containing the rightmost edge is then the first feature in the list. The rest of the features are then put in order until the left edge is encountered. Similarly, the features which comprise the blank are ordered from right to left.

Finding the start surface is the next task. Possible start surfaces are identified as surfaces on the raw stock with a negative slope in the y-direction on the left end of the feature. Starting surfaces on the part must have a negative slope in the y-direction on the right end of the feature. Several possible start surfaces are shown in Figure 24. The dashed lines represent the blank and the solid lines represent the part. The next start surface is found by first checking all the blank surfaces from left to right and then the part surfaces from right to left so that the start surfaces

would be found in the order shown in Figure 24.

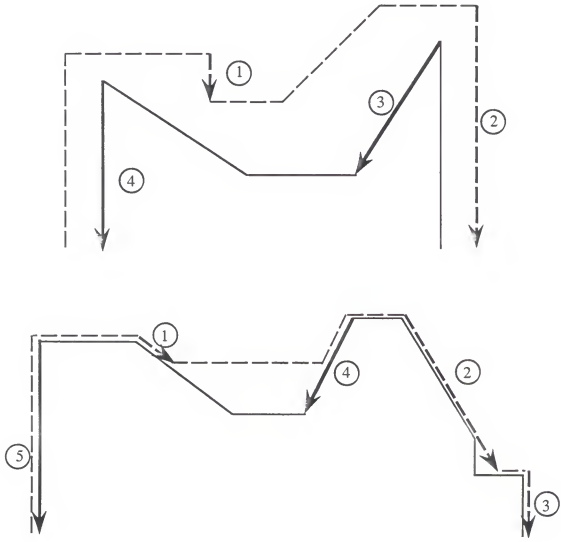
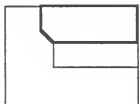


Figure 24: Possible starting surfaces

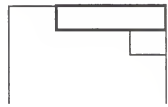
When a possible start surface is found, it is tested to determine if any cutting is needed with that surface as the starting surface. If there is nothing to do, the next start surface is found until either some cutting is needed or until it is determined that no more can be done with the current workholding and orientation. If at that point, there is still additional

material to be removed, the user can change the workholding or orientation and restart the process.

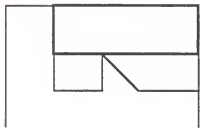
Once a start surface for the current manufacturing feature is found, the next step is to find the end surface. This is done by moving across the grid searching for the first interference as is described above. When found, the data about the entity which created the end surface is entered in the manufacturing feature. If the interference is an overhang which has no specific entity of its own, a phantom face is created to act as the end surface.



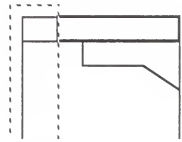
Dual end surface
Bottom at third feature
bottom = blank



Bottom at step
bottom = part



Bottom at part surface
which was not next surface
bottom = overhang



Workholding as end surface
bottom = part

Figure 25: Extracted manufacturing features with bottom type

The final step in extracting the manufacturing feature is to find the bottom surface. The bottom surface can be set either when an actual surface on the part or workholding is found at the bottom or when the start or end surface changes. If the start or end surface changes before the tool descends by one depth of cut, the new start or end surface is added to the original one and the feature is continued. If the second feature is found after one depth of cut, then an artificial bottom surface is set with a type of blank. Finally, if 3 start or end surfaces are found before one depth of cut, the feature is ended and a phantom step is created to act as the bottom surface. Figure 25 shows several different manufacturing features extracted using this strategy. The extracted features are shown with bold lines and the workholding is shown with a dashed line.

Once the manufacturing feature is identified, the APT statements needed to remove it must be written to the APT file. Generally, a motion statement instructs the tool to move along the drive surface until it is in the correct orientation with the check surface. The reduced syntax of an APT motion statement used in this project is

$$\begin{bmatrix}\text{TLLFT}\\ \text{TLRGT}\end{bmatrix}, \text{TLONPS}, \begin{bmatrix}\text{GOBCK}\\ \text{GOFWD}\\ \text{GOLFT}\\ \text{GORGT}\end{bmatrix} / \text{drive\_surface}, \begin{bmatrix}\text{TO}\\ \text{PAST}\\ \text{TANTO}\end{bmatrix}, \text{n,intof,check\_surface}$$

The first and second parameters describe the orientation of the tool with the drive surface and part surface respectively. The third parameter picks which direction along the drive surface should be taken. The last

parameters identify the check surface and the correct ending relationship of the tool with the check surface.

The correct parameters are all chosen depending on the slopes of the check and drive surfaces and on whether the surfaces are contributed by the blank, the part, the workholding, or an overhang. Sufficient loops are made with the tool taking the prescribed depth of cut until within one depth of cut of the bottom. If the bottom surface has a type of anything other than blank, then the final cut is made at the bottom surface. Since a type of blank for the bottom surface indicates that some artificial condition such as switching start surfaces ended the manufacturing feature, the final cut is not taken and the bottom of the manufacturing feature is moved up to the current tool position. This prevents an extra cut from being taken every time the start or end surface switches.

The only step left is to reflect the removal of material in the model of the raw stock and on the grid of bits. This is needed so that the feature extracting routines will not extract the same feature again. The change in the raw stock is made by deleting all of the features on the blank between the surfaces corresponding to the start and end surfaces of the manufacturing feature. The appropriate new features are then created to reflect the tool motion. Figure 26 shows the deleted features for an example together with the features they are replaced with. Non-linear features are replaced with linear approximations so that small pieces of them can be modeled correctly.

The blank surfaces are then reordered and the blank is redrawn to the bit grid. If more machining is possible at the current tool position,

the next feature is extracted and the process is repeated. If no more cuts are possible, the next starting surface is found and features are extracted from it. This process continues until no more machining is possible.

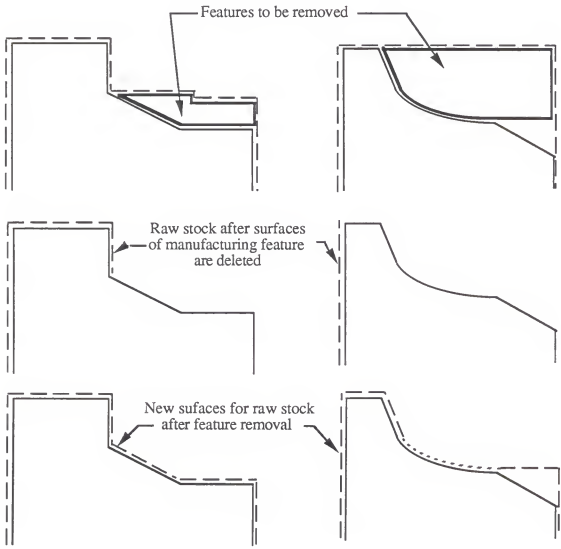


Figure 26: Replacement of Blank Surfaces

3.9 Results and Summary

The useful output of the CAM system is a file with APT statements

which will drive the correct machines to produce the part. This output is only implemented for turned features which can be processed on a lathe. Output for other types of machines such as drills would require a GenerateToolPath method for the particular machine.

The file with APT statements is created directly from the feature-based CAD database making BigFeat a true CAD/CAM program. All of the advantages of a general CAD system are maintained while a direct link to manufacturing is provided without the usual disadvantages of human interpretation and data reentry.

This chapter has described how the CAM system takes the output from the CAD system and transforms it into the useful output file of APT statements mentioned above. The final chapter presents the conclusions of the project and mentions a few areas which need further research.

Chapter 4
Conclusions

This thesis has described the BigFeat CAD/CAM system which was implemented as part of a project being conducted in the Center for Research in Computer Controlled Automation at Kansas State University. Based on this project, it was concluded that design features do work well as the basis for a CAD/CAM environment. The manufacturing steps can be derived from a design feature database for turned parts. The prototype CAD/CAM system demonstrates the ease with which parts can be designed by feature and produces the APT statements necessary for an NC lathe to produce the part.

Future systems will need to add a solid modeler under the feature-based interface to insure validity of the part and to ease some of the operations such as material removal from the raw stock. The area of extracting manufacturing features from design features also needs a more thorough consideration. The geometric grid-based approach presented in

this thesis worked and is quite fast, but a more rigorous method needs to be developed. It would be interesting to determine if a similar scheme would be feasible in 3-dimensions for prismatic parts. Additionally, more research is needed to determine the role that feature-based design can play with the other aspects of engineering such as analysis, quality assurance, etc.

References

[1] Product Definition Data Interface Technical Workshop, McDonnell Aircraft Company, St Louis, MO, December 9-10, 1986.

[2] Hummel, K. E., and Brooks, S. L., "Using Hierarchically Structured Problem-Solving Knowledge in a Rule-Based Process Planning System", *Symposium on Integrated and Intelligent Manufacturing, ASME Winter Annual Meeting,* Anaheim, CA, December 7-12, 1986.

[3] Hummel, K. E., and Brooks, S. L., "Symbolic Representation of Manufacturing Features for an Automated Process Planning System", *Proceedings, Symposium on Knowledge-Based Expert Systems for Manufacturing, ASME Winter Annual Meeting,*Anaheim, CA, December 7-12, 1986.

[4] Brooks, S. L., and Hummel, K. E., "XCUT: A Rule-Based Expert System for the Automated Process Planning of Machined Parts" *Symposium on Integrated and Intelligent Manufacturing, ASME Winter Annual Meeting,* Anaheim, CA, December 7-12, 1986.

[5] Henderson, M. R., "Extraction of Feature Information From Three-Dimensional CAD Data", PhD Thesis, Purdue University, May, 1984.

[6] Henderson, M. R., "Automated Group Technology Part Coding from a Three-Dimensional CAD Database", *Proceedings, Symposium on Knowledge-Based Expert Systems for Manufacturing, ASME Winter Annual Meeting,* Anaheim, CA, December 7-12, 1986.

[7] Dixon, J. R., Libardi, E. C., Luby, S. C., Vaghul, M., and Simmons, M. S., "Expert Systems for Mechanical Design: Examples of Symbolic Representations of Design Geometries", *Applications of Knowledge Based Systems to Engineering Analysis and Design,* Editor: C. L. Dym, ASME, 1985.

[8] Schmidt, Larry E., "A Computer Engineering Environment for Feature-Based Design and Manufacture", M.S. Thesis, Kansas State University, December, 1988.

[9] Product Definition Data Interface System Specification Document, Draft Standard, July, 1984.

[10] Bidanda, Bopaya and Cohen, Paul H., "An Integrated CAD-CAM Approach to the Selection of Workholding Devices for Concentric, Rotational Parts", *Proceedings, Manufacturing Processes, Systems and Machines 14th Conference on Production Research and Technology*, University of Michigan, College of Engineering, Ann Arbor, Michigan, October 6-9, 1987.

[11] Wong, C. L., Bagchi, A., and Ahluwalia, R. A., "DMAP: A Computer Integrated System for Design and Manufacturing of Axisymmetric Parts", *Proceedings, Symposium on Knowledge-Based Expert Systems for Manufacturing, ASME Winter Annual Meeting,* Anaheim, CA, December 7-12, 1986.

[12] Austin, B. L., "Computer Aided Process Planning for Machined Cylindrical Parts", *Symposium on Integrated and Intelligent Manufacturing, ASME Winter Annual Meeting,* Anaheim, CA. December 7-12, 1986.

Appendix 1
List of Entities

---

aluminum.form

```
attributes  {
            float     E          10.E6;
            float     poisson    .33;
            float     density    168.5;
            short     type;
            }
```

arc.form

```
attributes  {
            entity          vector;
            entity point    center;
            entity point    start_point;
            float           angle;
            }
```

assembly.form

```
attributes  {
            entity                          transform;
            array[1:many] class any         constituents;
            }
```

bspline.form

```
attributes  {
            entity                          display;
            short                           order 4;
            array[0:many] short             knot;
            array[0:many] entity point      control_point;
            }
```

centerline.form

```
attributes  {
            entity      display;
            entity      line;
            }
```

circle.form

```
attributes  {
            entity          vector;
            entity point    center;
            float           radius;
            }
```

circular_runout.form

```
attributes  {
            array[1:many] class turned_features
                  toleranced_entity;
            float                       tolerance;
            }
```

concentricity.form

```
attributes  {
            array[1:many] class turned_features
                  toleranced_entity;
            float                       plus_tolerance;
            float                       minus_tolerance;
            class  turned_features      datum;
            }
```

copper.form

```
attributes  {
            float       E           16.E6;
            float       poisson     .3;
            float       density     555.3;
            short       type;
            }
```

```
display.form
    attributes  {
                boolean   shown  TRUE;
                short     color  1;
                }

douglas_fir.form
    attributes  {
                float     density  31.2;
                float     E        1.81E6;
                }

drill.form
    attributes  {
                float                    clearance;
                float                    max_diameter;
                float                    max_rpm;
                array[1:4] long          machine_functions;
                array[1:many] string     workholding;
                array[1:many] entity     lathe_tool;
                boolean                  NC;
                }

e_face.form
    attributes  {
                entity                   display;
                entity                   centerline;
                array[0:1] entity        turned_edge;
                }

e_free_form.form
    attributes  {
                entity                   display;
                entity                   centerline;
                array[0:1] entity        turned_edge;
                class                    curve;
                }
```

e_groove.form

```
attributes    {
              entity                  display;
              entity                  centerline;
              array[0:1] entity       turned_edge;
              class                   curve;
              }
```

e_step.form

```
attributes    {
              entity                  display;
              entity                  centerline;
              array[0:1] entity       turned_edge;
              }
```

e_taper.form

```
attributes    {
              entity                  display;
              entity                  centerline;
              array[0:1] entity       turned_edge;
                  }
```

i_face.form

```
attributes    {
              entity                  display;
              entity                  centerline;
              array[0:1] entity       turned_edge;
              }
```

i_free_form.form

```
attributes    {
              entity                  display;
              entity                  centerline;
              array[0:1] entity       turned_edge;
              class                   curve;
              }
```

i_groove.form

```
    attributes    {
                  entity                  display;
                  entity                  centerline;
                  array[0:1] entity       turned_edge;
                  class                   curve;
                  }
```

i_step.form

```
    attributes    {
                  entity                  display;
                  entity                  centerline;
                  array[0:1] entity       turned_edge;
                  }
```

i_taper.form

```
    attributes    {
                  entity                  display;
                  entity                  centerline;
                  array[0:1] entity       turned_edge;
                  }
```

lathe.form

```
    attributes    {
                  float                   dist_between_centers;
                  float                   swing_over_carriage;
                  float                   horsepower;
                  float                   max_spindle_rpm;
                  float                   max_feed_rate;
                  float                   max_rapid_traverse;
                  array[1:4] long         machine_functions;
                  array[1:many] string    workholding;
                  array[1:many] entity    lathe_tool;
                  boolean                 NC;
                  }
```

lathe_tool.form

```
attributes  {
            float                rake_angle;
            float                back_angle;
            boolean              right_handed;
            boolean              exterior;
            float                cutting_length;
            float                cutting_radius;
            }
```

line.form

```
attributes  {
            entity                        display;
            array[0:1] entity  point      end_points;
            }
```

location.form

```
attributes  {
            array[1:many] class feature   toleranced_entity;
            float                         plus_tolerance;
            float                         minus_tolerance;
            entity  vector                path;
            entity  point                 base;
            }
```

matrix.form

```
attributes  {
            array[1:16] float   element;
            }
```

part.form

```
attributes  {
            entity                        transform;
            array[1:many] class any       constituents;
            }
```

point.form

```
attributes  {
            entity      display;
            float       x;
            float       y;
            float       z;
            }
```

polyline.form

```
attributes  {
            entity                      display;
            array[1:many] entity        point;
            }
```

resources.form

```
attributes  {
            array[1:many] class machines constituents;
            }
```

size.form

```
attributes  {
            array[1:many] class feature     toleranced_entity;
            float                           plus_tolerance;
            float                           minus_tolerance;
            }
```

surface_of_revolution.form

```
attributes  {
            class       curve;
            entity      centerline;
            }
```

```
steel.form
    attributes  {
                float   E           30.E6;
                float   poisson     .3;
                float   density     493.;
                short   type;
                }

transform.form
    attributes  {
                array[1:3] float    rpy_angles;
                array[1:3] float    translation;
                float               scale 1.;
                }

turned_cornerround.form
    attributes  {
                entity      display;
                float       radius;
                entity      turned_edge;
                }

turned_edge.form
    attributes  {
                entity      display;
                entity      vector;
                entity      point;
                float       radius;
                }

turned_fillet.form
    attributes  {
                entity      display;
                float       radius;
                entity      turned_edge;
                }
```

vector.form

```
attributes  {
            float   x;
            float   y;
            float   z;
            }
```

white_oak.form

```
attributes  {
            float   density   42.4;
            float   E         1.78E6;
            }
```

white_pine.form

```
attributes  {
            float   density   23.7;
            float   E         1.46E6;
            }
```

Appendix 2
Directory Structure

```
class user_entity;
        {
        entity      part;
        entity      assembly;
        entity      resources;
        }

class curve;
        {
        entity      line;
        entity      polyline;
        entity      circle;
        entity      bspline;
        entity      arc;
        }

class surface;
        {
        entity      surface_of_revolution;
        }

class quasi_geometry;
        {
        entity      display;
        entity      vector;
        entity      centerline;
        entity      transform;
        entity      matrix;
        }
```

```
class geometry;
        {
        class       quasi_geometry;
        entity      point;
        class       curve;
        class       surface;
        }

class composite_features; {}

class sheet_metal_features; {}

class machined_features; {}

class turned_internal;
        {
        entity      i_step;
        entity      i_face;
        entity      i_taper;
        entity      i_groove;
        entity      i_free_form;
        }

class turned_external;
        {
        entity      e_step;
        entity      e_taper;
        entity      e_face;
        entity      e_groove;
        entity      e_free_form;
        }
```

```
class turned_features;
        {
        class       turned_internal;
        class       turned_external;
        entity      turned_edge;
        entity      turned_fillet;
        entity      turned_cornerround;
        }

class features;
        {
        class       turned_features;
        class       composite_features;
        class       sheet_metal_features;
        class       machined_features;
        }

class tolerances;
        {
        entity      circular_runout;
        entity      concentricity;
        entity      location;
        entity      size;
        }

class metal;
        {
        entity      steel;
        entity      aluminum;
        entity      copper;
        }

class wood;
        {
        entity      white_oak;
        entity      douglas_fir;
        entity      white_pine;
        }
```

```
class plastic; {}

class material;
        {
        class           metal;
        class           wood;
        class           plastic;
        }

class admin_data;
        {
        class           material;
        }

class tools;
        {
        entity          lathe_tool;
        }

class machines;
        {
        entity          lathe;
        entity          drill;
        class           tools;
        }
```

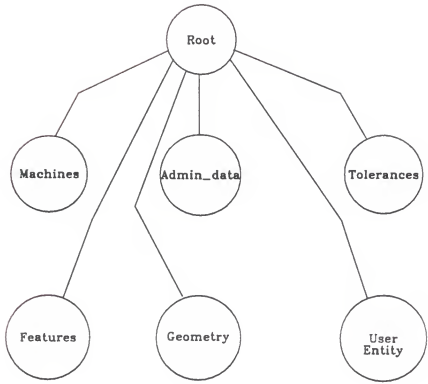
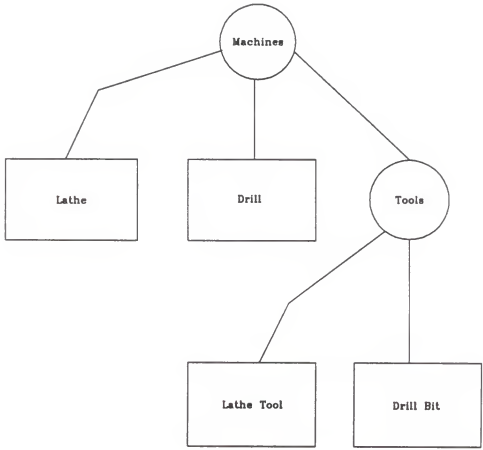
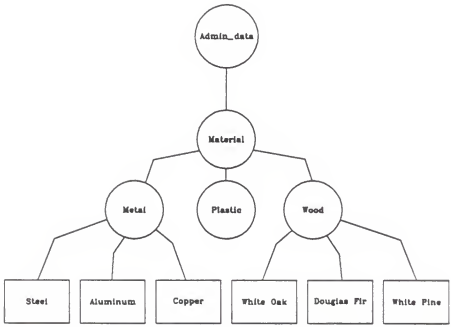
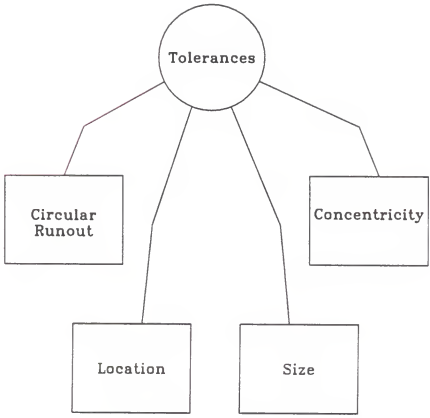
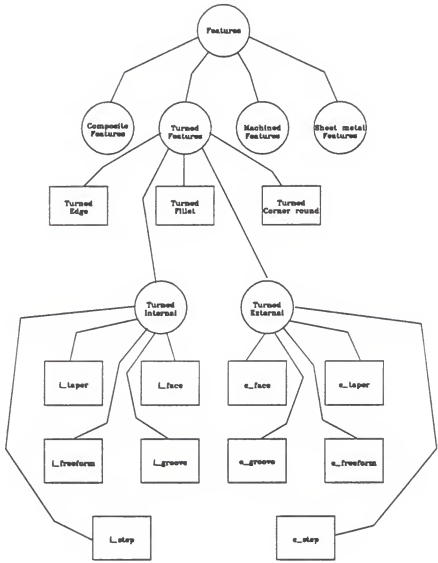


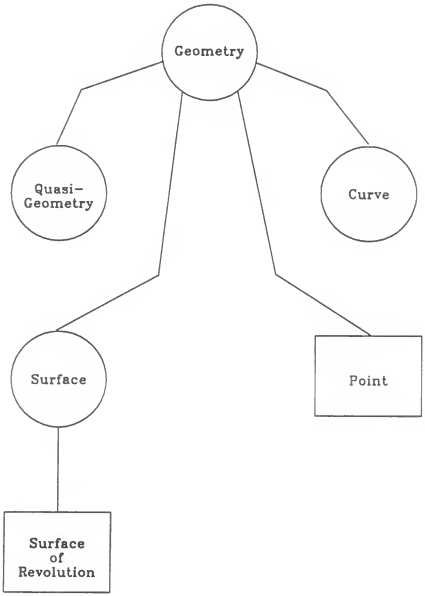
Figure 27: Hierarchical Directory Structure

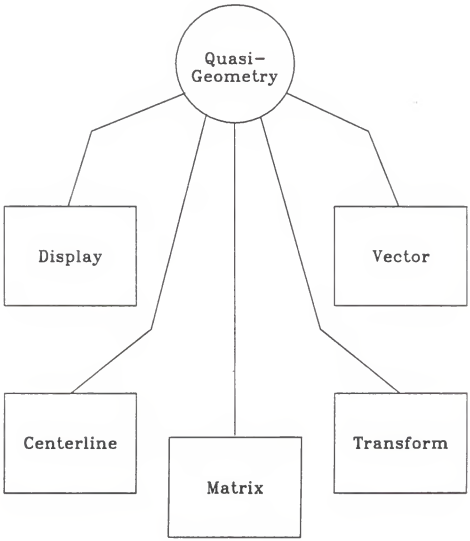

Machines
Lathe
Drill
Tools
Lathe Tool
Drill Bit

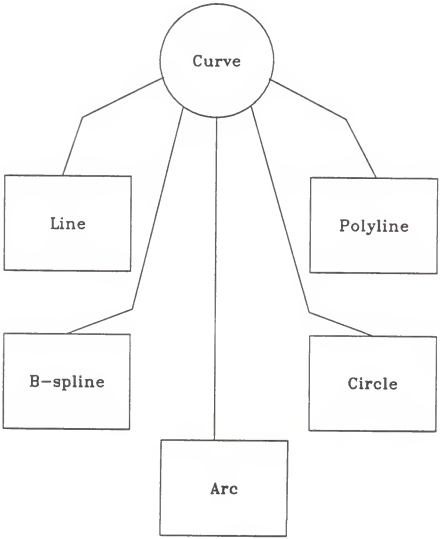

Admin_data
Material
Metal
Plastic
Wood
Steel
Aluminum
Copper
White Oak
Douglas Fir
White Pine

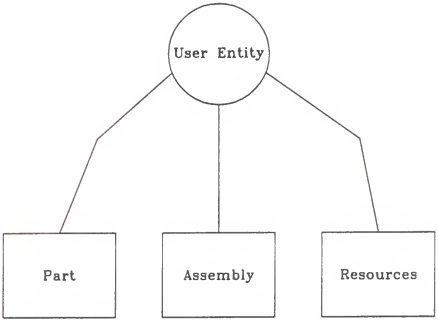

Tolerances
Circular Runout
Concentricity
Location
Size


Features
Composite Features
Turned Features
Machined Features
Sheet metal Features
Turned Edge
Turned Fillet
Turned Corner round
Turned Internal
Turned External
i_taper
i_face
e_face
e_taper
i_freeform
i_groove
e_groove
e_freeform
i_step
e_step


Geometry
Quasi–
Geometry
Curve
Surface
Point
Surface
of
Revolution


Quasi-
Geometry
Display
Vector
Centerline
Matrix
Transform


Curve
Line
Polyline
B–spline
Circle
Arc


User Entity
Part
Assembly
Resources

Implementation of a Feature-Based
Object-Oriented CAD/CAM Package

by

Vance William Unruh

B.S., Kansas State University, 1987

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Mechanical Engineering

Kansas State University
Manhattan, Kansas

1988

Standard CAD and CAM applications normally do not deal with the entire product definition data which is needed for automation. Since this information is not available, the link between design and manufacturing is usually a manual link which requires data reentry and human interpretation of design intent, two processes which are slow and error-prone.

Designing with features is proposed as a means of overcoming these shortcomings. Features are the next higher conceptual level of design above geometry. The designer is freed from having to work with low-level geometric entities such as lines and points. Instead, he can design with high-level features such as slots, holes, chamfers, etc. Designing with features not only makes the computer design process more intuitive and natural, but it also provides the means to include the entire product definition.

The part model is represented with design features instead of with the manufacturing features which are used in some feature-based implementations. Design features are the solids and voids used to build the part, while manufacturing features are the volumes removed from the raw stock to leave the part. Some feature such as holes, qualify as both design and manufacturing features since they are used by both design and manufacturing engineers. Since design features are used, the CAM program must interpret some of the design features to extract the volumes of material which must be removed from the raw stock before it can do the process planning.

An object-oriented implementation using design features is

presented which allows input of the entire produce definition data. The CAM program accesses the database directly and generates the machining instructions necessary to machine the part. Currently, only turned parts which can be produced on a lathe are implemented. Thus, feature-based design is used to integrate CAD and CAM so that parts can be produced automatically from the design database.