NBSIR 88-3717

THE DESIGN PROTOCOL, PART DESIGN EDITOR, AND GEOMETRY LIBRARY OF THE VERTICAL WORKSTATION

January 28, 1988

By:
Thomas R. Kramer
Jau-Shi Jun

THE DESIGN PROTOCOL, PART DESIGN EDITOR, AND GEOMETRY LIBRARY
OF THE VERTICAL WORKSTATION
OF THE AUTOMATED MANUFACTURING RESEARCH FACILITY
AT THE NATIONAL BUREAU OF STANDARDS

Dr. Thomas R. Kramer
Guest Worker, National Bureau of Standards, &
Research Associate, Catholic University

Dr. Jau-Shi Jun
Computer Scientist, National Bureau of Standards

January 28, 1988

NBSIR 88-3717

Funding for the research performed by Dr. Kramer and reported in this paper was provided to Catholic University under Grant No. 60NANB5D0522 and Grant No. 70NANB7H0716 from the National Bureau of Standards.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Page

THE DESIGN PROTOCOL, PART DESIGN EDITOR, AND GEOMETRY LIBRARY OF THE VERTICAL WORKSTATION OF THE AUTOMATED MANUFACTURING RESEARCH FACILITY AT THE NATIONAL BUREAU OF STANDARDS

I. INTRODUCTION

1. CONTENTS

This paper describes the part design protocol (method of describing the geometry of a part), the part design editor, and the geometry library used in the Vertical Workstation (VWS) of the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards. Chapter II describes the design protocol, Chapter III discusses the design editor, and Chapter IV discusses the geometry library. The descriptions pertain to the versions in use during September 1987.

2. AUDIENCE

The paper is intended to be useful to people interested in concepts and technical details of the VWS, particularly AMRF personnel who are running the VWS or maintaining or improving the software for the VWS. The paper is intended to be useful also to other researchers in automated manufacturing. A knowledge of the computer language LISP is useful but not essential to reading this paper. Detailed documentation of the LISP functions that are involved with the systems described here is being prepared separately.

3. BRIEF VWS DESCRIPTION

The VWS is a computer-integrated automated machining workstation. It includes a control system, a computer-aided design system, an automatic process planning system, and an automatic nc-code generator. The principal machinery is a milling center (Monarch VMC-75 with a GE2000 controller) and a robot (Unimate 4070 with a Val II controller) to tend the milling center. There is quite a bit of ancillary hardware. The system is controlled from a microcomputer (Sun 3/160 with 6Mb memory, BW monitor). Running in stand-alone mode, it is possible to design and machine a simple metal part within an hour. The VWS may also be run as an integrated part of the AMRF. The workstation is described in more detail in [KR&J].

The software for the VWS is written in the Franz LISP dialect of the computer language LISP. In this paper this software is called the VWS2 system. Six principal modules comprise the VWS2 system: the Production Management Operating System (the control system), the State Table Editor, the Equipment Program Generator, the Part Design Editor, the Process Planner, and the Data Execution module.

To produce a part from scratch, the user sits at the Sun workstation and creates a design using the Part Design Editor. The Process Planner is then called to write a plan for how to machine a part of that design. Next NC-code is generated automatically from the design and the plan by the Data Execution module. Finally the user tells the control system to make the part. The control system coordinates the activities of the workstation equipment so that the part blank is loaded onto the milling machine, the NC-code is sent to the milling machine and executed (making the part), and the finished part is unloaded.

4. RELATED READING

This paper is one of about a dozen papers being prepared as part of the AMRF documentation to describe all aspects of the VWS. The others are [JUN], [KRA2], [KRA3], [KRA4], [KRA5], [K&S2], [KR&W], [LOVE], and [RUDD]. Other papers, prepared for professional meetings, also describe the VWS [KRA1], [KR&J], [K&S1], and [NA&J].

II. **DESIGN PROTOCOL**

1. DESIGN PROTOCOLS GENERALLY

1.1. Definition

A part design protocol is a method of representing the geometry (and possibly other information) of a family of parts. Non-geometric information might include the material from which a part is intended to be made.

The rules for making a mechanical drawing, for example, may be considered to be a design protocol. A document conforming to a design protocol and representing a particular part will simply be called a design (or a "design document"). A mechanical drawing of a gear, for example, is a design. A design document may be either a paper document, a printout on a computer terminal screen, or a computer file.

A good discussion of design protocols is given by Requicha in [REQU].

1.2. Uses

The challenge to a design protocol for an integrated computer-automated machining system is to be usable for all phases of part design and machining. This includes being:

1. comprehensible to a human user and yielding readable designs,
2. suitable for a computer aided design system,
3. usable by an automated drawing system,
4. usable by an automated process planning system,
5. usable for the automatic generation of nc-code,
6. usable for automated handling (by robot, conveyor, cart, etc.)
7. usable by a computer vision system,
8. usable for automated cleaning and deburring,
9. usable for automated inspection, and
10. usable by an automated analysis system (e.g. finite element analysis)

If the area of interest is extended beyond machining to manufacturing, add:

11. usable for automated assembly, and
12. usable for automatic document control.

The authors are not aware of any design protocol that is available for all these uses simultaneously. It may be that employing several different design protocols for a given part will be superior to trying to use a single protocol for all purposes.

Some of the uses above are closely linked in currently available commercial systems, particularly computer aided design, automated drawing, and automated analysis. They may be separated, however. The design editor described here may be run without drawing, for example.

Producing a document comprehensible to a human may be circumvented by having a sufficiently friendly computer interface between the design document and the user. With such a system, a paper copy or a byte-by-byte printout on a computer terminal screen of a design document may be useless or irritating to a human.

1.3. <u>Other Considerations of Design Protocols</u>

1.3.1. Wide Range

A design protocol should be capable of representing the design of a wide range of machined parts. In practice, a "wide range" is whatever is sufficient to cover a part mix that is interesting to some machine shop or research group.

1.3.2. Ambiguity

A design protocol should allow the expression of all geometric information needed for its intended uses. On critical items the protocol should be unambiguous. Some ambiguity, however, may be allowed. For example, the point on the circumference of a threaded hole at which the threading emerges is almost never specified in common design practice.

1.3.3. Physical Realizability

A design protocol may permit the construction of a design that is not physically realizable -- an infinitely long cylinder, for example. It is desirable but not essential that a protocol (or the computer interface to a protocol) minimize the possibility of designing non-realizable objects.

1.3.4. Uniqueness

One use of a design protocol is to determine if two objects (physical or design only) are the same. If there are many ways to specify the design of an object, it becomes hard to determine whether two objects are the same. Some design protocols make it possible to describe a part in only one way, some permit a small number of different designs for a single object, and some a large number.

Where a protocol is non-unique, it may be made less so by adding rules for orienting objects and rules for how to choose a single method of constructing an object when there are several possibilities. This may remove some of the difficulties of non-uniqueness, but it will add to the difficulty of using the protocol.

Any determination of uniqueness will require a convention for when numbers are equal -- at the very least an agreement on how many significant figures will be recognized in the representation of a number.

1.4. Software

Two kinds of software are essential for dealing with a design: the raw data of a design itself, and routines for using the data. It is important not to confuse raw data with visual displays produced by processing the data.

1.5. Exact vs. Approximate

Some design protocols represent parts exactly while others are only approximate. The usual need for approximation is in the case of curved surfaces or curved solids. A curved surface may be approximated by a set of planar polygons, and a curved solid may be approximated by a set of solid objects (imagine a cylindrical salt shaker approximated by thousands of tiny cubic salt crystals packed together, for example).

1.6. Types

Three common types of design protocol are: primitive instancing, constructive solid geometry, and boundary representation.

In primitive instancing a design is specified by giving the name of a primitive object and (usually) values for some parameters. For example, a gear may be a primitive object in a group technology design protocol. Parameters required to specify a certain gear might be the diameter, the number of teeth, the thickness, and the diameter of a hole through the middle. The protocol could include a convention for the shape of a gear tooth so that the protocol would be unambiguous. We will make no further mention of primitive instancing.

In constructive solid geometry (commonly abbreviated CSG) a design is specified by starting with a set of primitive solids (cubes, spheres, cylinders, etc.) and allowing operations (union, subtraction, intersection, etc.) which form complex solids from the primitives. The design protocol used in the VWS is a type of constructive solid geometry.

In boundary representation a design is specified by giving a description of the surface of an object. The object is what lies within the (necessarily closed) surface. The standard AMRF design protocol is a boundary representation.

Hybrids of these types may be and have been constructed.

Translating the design of a given part from one protocol to another is apt to be a difficult undertaking. Automatic conversion between a boundary representation and CSG is particularly difficult.

1.7. Constructive Solid Geometry (CSG)

1.7.1. Approximate CSG

There are two common subtypes of CSG. In the first, a solid object is approximated by the union of a lot of little solids. In this subtype the only primitive is usually a cube, and the only operation is usually union. This subtype may be further divided into two varieties: one in which the cubes (sometimes called voxels -- short for volume element) are all the same size, and the other in which they are different sizes. In principle, voxels do not have to be cubic, but might be any shape that can pack to fill space.

1.7.2. Octrees

One common protocol using cubes of different sizes creates "octrees". This system breaks space down into cubes of a fixed starting size. The part is placed in this space, and all the cubes which fit entirely inside the part are assigned to the set of inside cubes. Cubes which intersect the surface of the part are subdivided into eight smaller cubes or octants, each of which has a side half as long as a side of the parent cube. Again all the cubes which fit entirely in the part are assigned to the set of inside cubes, and those which intersect the surface are subdivided. This process goes on until either the part is exactly represented or a minimum cube size is reached. Since it is necessary to keep track of the parent cube of each child cube, the data structure for the part is a tree. The prefix "oct" for eight, plus "tree" yields the name "octree".

1.7.3. Exact CSG

In exact CSG a solid object is represented by using a set of parameterized primitive volumes and combining them with a set of operations. The VWS design protocol is of this type.

The primitives in exact CSG are usually common solids from elementary solid geometry: rectangular parallelepipeds (i.e. blocks), spheres, cones, cylinders, toruses, and sometimes prisms. Objects which may be described are usually connected solids which are combinations of the elementary solids. The allowable operations for combining primitives may include union, intersection, and subtraction. Figure 1 shows an L-shaped solid constructed three different ways from block-shaped primitives.

In CSG, either the specification of a primitive must include its position in space, or the description of a combination operation must include spatial information. In general, there is no rule that the primitives must be either disjoint or touching. For example, suppose block B is contained entirely in block A. Then the union of A and B is equal to A. If block B is disjoint from block A, the difference between A and B is equal to A.

Notice that, in principle, it is possible to have a CSG system with subtraction as the only operation. Any physical object can be imagined as contained in a block, and each part of the outer surface of the object can be imagined as part of the surface of one of a number of volumes that have been cut away from the block to produce the object. As long as all of the volumes being cut away are in a set of primitive volumes, the original object may be described in this hypothetical CSG system as the block minus the cut-away volumes.

Figure 1. Constructive Solid Geometry Example

1.7.4. Aspects of CSG

1.7.4.1. Range

In principle, the design of any object may be expressed in an approximate CSG protocol. However, the limit of resolution of the approximation is also a limit on the usefulness of the protocol. For example, if the smallest voxel size is one millimeter on a side, every object whose largest dimension is one millimeter is represented by one voxel -- not very useful if you are designing computer chips.

For the purposes of machining, it would be nice to have the limit of resolution be near the closest tolerance required. If the tolerance requirement is 0.01 millimeter, then voxel size should be about 0.01 millimeter on a side. For a part ten centimeters on a side, using equal sized voxels, this would require a trillion voxels. This is beyond the on-board storage capacity of today's computers, and even a millionth of that number tests the limits of a typical engineering workstation.

This is not to say approximate CSG is not useful for machining; it has been applied to real problems. But using it requires either accepting lower tolerances or using heavy duty computer hardware and software.

The range of designs expressible in an exact CSG is largely dependent on the suite of primitives in the CSG. If all surfaces of a part match the surface of some primitive, the part should be expressible. The range of parts which can be expressed in terms of the CSG primitives listed above is enormous and includes the bulk of what American industry would like to produce by machining. However, if the surface of a part is "sculpted" like the outside of a car or the Winged Victory of Samothrace, it cannot be expressed.

1.7.4.2. Ambiguity

Constructive solid geometry may be ambiguous or unambiguous, depending on the rules of combination.

1.7.4.3. Realizability

Constructive solid geometry inherently makes realizable objects. All the primitives are realizable objects, and the combination methods are all realizable, so the result is always realizable. It is not difficult to generate disconnected objects, however (for example by taking a chunk out of the middle of a long narrow block, splitting it in two), and this must be forestalled.

1.7.4.4. Uniqueness

Exact CSG is inherently non-unique, as shown by the three methods of constructing the simple object in Figure 1.

Approximate CSG is also inherently non-unique.

1.7.4.5. Computer Graphics

Computer graphics for automated manufacturing normally involves either showing the surface of a part or a wire frame for the part.

Exact CSG does not lend itself readily to drawing because when solids intersect, the surfaces which result may be complex, requiring difficult calculations. Moreover, the equations of lines of intersection are often of higher order than the equations of the surfaces which are intersecting and may not have solutions in closed form.

As a rule, drawing the primitives themselves, or combinations of primitives which have simple lines of intersection is not prohibitively difficult or time consuming, although many routines may be required.

1.7.4.6. Automated Machining

Because machining is a process of material removal with a (usually) rotating tool, the objects which can be created by machining lend themselves to expression in exact CSG. If a flat-ended tool moves through a workpiece in a straight line perpendicular to the axis of tool rotation, the solid removed is a block plus two cylinders, for example. If a ball-nosed tool is moved in a circle whose radius is larger than the tool radius in a plane perpendicular to the axis of tool rotation, the removed volume is a torus plus the difference of two concentric cylinders.

The solids generated when a tool moves simultaneously along three axes are much more complex than those generated when the value on one axis is held constant and the motion is along one or both of the other two axes. The primitives in the design protocol used in the Vertical Workstation may all be generated as the swept volume of a common tool with the value of at least one axis held constant.

Figure 2. Boundary Representation Example

1.8. Boundary Representation

In a boundary representation, an object is whatever is contained inside the surface that is the boundary of the object. The surface is described as being composed of a set of surface patches which are joined together at their edges but do not otherwise intersect. Figure 2 shows a boundary representation of the same L-shaped block as is shown in Figure 1.

1.8.1. Approximate Boundary Representation

In an approximate boundary representation, the exterior surface of the object being represented is approximated by a collection of elementary surface pieces. While the elementary pieces could, in principle, be of any surface type, the only elementary surface piece currently in common use is the planar polygon. Planar polygons are very well suited to both computer graphics and geometric calculations.

Since a planar polygon may be defined simply by identifying the points at its vertices, one common method of giving a design definition in a protocol of this sort is to give two lists. The first is a list of points (each of which is a list of three numbers -- the x, y, and z coordinates of the point). The points on the first list are implicitly or explicitly numbered. The second list is a list of polygons. Each polygon is represented by a list of point numbers. The idea is that the polygon is described by drawing a line from the first point to the second, and so on. Since it is possible to describe a polygon in this way by starting at any vertex (this corresponds to circular permutations of the list of points), there may be a convention for choosing the starting point (e.g. the lowest numbered point).

In many such protocols the direction (clockwise or counterclockwise) in which the vertices are traversed serves to show which side of the polygon the inside of the object lies on. This is redundant information, of course, since it could be deduced from an examination of the whole finished object, but it is extremely convenient to retain that information locally.

Care must be taken in creating designs with such protocols that all of the vertices of a polygon really do lie on a single plane.

In existing protocols of this sort there may be additional limitations, such as an upper limit on the number of vertices a polygon may have, or a requirement that only convex polygons be used, or a prohibition on having adjacent coplanar polygons.

1.8.2. Exact Boundary Representation

In an exact boundary representation, the patches of surface used to build the boundary of an object are cut from a collection of primitive surface types: plane, cylinder, sphere, etc. The allowable cut lines on the surface patches are also taken from a set of primitives: straight line, circular arc, ellipsoidal arc, helical arc, etc.

A surface patch is defined by giving descriptions of three sorts of items: 1. the surface from which a patch is cut, 2. the lines which form the edges of the cut, and 3. the points which are the ends of the lines.

As in approximate boundary representation, the order of the edges of a patch may be used to indicate where the inside of the object lies.

In an exact boundary representation information about the numerical parameters of the surfaces, lines, and points may be called "geometry" and separated from information about which lines form edge loops and which edge loops are used to cut which surfaces. Information of the second sort may be called "topology".

1.8.3. The AMRF Standard

The AMRF Standard part design protocol is described in a paper by Ted Hopp [HOPP]. In its geometry and topology sections the AMRF standard part model is an exact boundary representation. The standard also has header, functionality, and features sections, however. The inclusion of the features section allows the addition of information about features, but there is no requirement that the features section be used.

The primitive geometric surfaces included in the AMRF standard are: plane, cylinder, cone, sphere, and torus. The curves which may bound surfaces are limited to straight lines and circular arcs.

1.8.4. Aspects of Boundary Representation

1.8.4.1. Range

The situation for the range of representable parts in boundary representation is similar to the situation in CSG. Approximate methods may represent anything, but often with poor resolution. The range of representables in exact representations depends upon the range of primitives but is limited by the suite of primitives.

1.8.4.2. Ambiguity

The degree of ambiguity of a boundary representation depends upon the individual protocol.

1.8.4.3. Realizability

The object described by a boundary representation is whatever happens to be inside the boundary. If the designer fails to make the boundary closed, then a non-realizable object will be described. The reversal of directionality of an edge loop may indicate to some boundary representations that the object is on the wrong side of the surface surrounded by the loop, also creating a non-realizable object. Peculiar intersections of surfaces may cause other non-realizable conditions. Thus boundary representations are prone to allow description of non-realizable objects.

1.8.4.4. Uniqueness

Exact boundary representations are nearly unique within differences of translation and rotation. Conventions may be required for the direction of axes, the location of points on closed curved edges, and determining which is the first point in an edge loop or the order of surfaces. The uniqueness of approximate boundary representations is more difficult.

1.8.4.5. Computer Graphics

Because the drawing of a part consists of drawing the surfaces or the edges of the part, and the data base contains surface and edge information, boundary representation lends itself to computer drawing of parts.

1.8.4.6. Automated Machining

Because machining operations usually create several surfaces at once and the boundaries of the surfaces which are created depend upon the current shape of the workpiece as well as on the tool path, boundary representation does not lend itself to automated machining.

2. VWS DESIGN PROTOCOL

2.1. Introduction

The VWS2 system employs a feature-based design protocol which satisfies uses one through five from the list in section 1.2 above. Using the protocol for the other purposes listed has not been attempted, but there is no apparent obstacle to its use for these purposes.

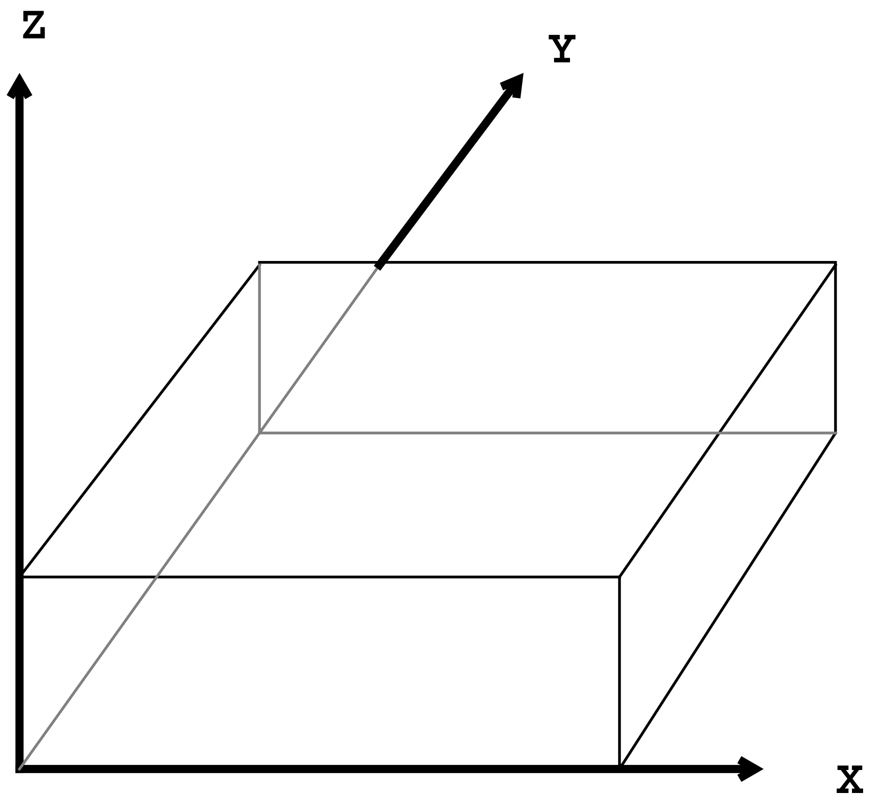
The design of a part is expressed as a list of features on a piece of stock. The piece of stock is always a rectangular block. The stock is fixed with respect to a three-dimensional cartesian coordinate system, as shown in Figure 3.

The primary features in the system in September, 1987 are: chamfer_out, groove, hole, pocket, straight_groove, text, contour_groove, contour_pocket, and side_contour. There are also subfeatures which may be made on the primary features: chamfer_out, chamfer_in, countersink, and thread. A feature is specified in the system by giving its name and the values of several parameters which specify its location, shape, and size. Parameter information is kept by storing the name of each parameter followed by its value. Parameter values representing physical dimensions are given in inches.

The design file has two parts: a header and a list of features. The header includes the id number of the design, a description not more than 30 characters long, the dimensions of the block, and (optionally) the material from which a workpiece of that design is intended to be made. The list of features is numbered sequentially starting with 1, so that each feature has a number. The feature number is important as an identifier of the feature, but the order of the features is irrelevant.

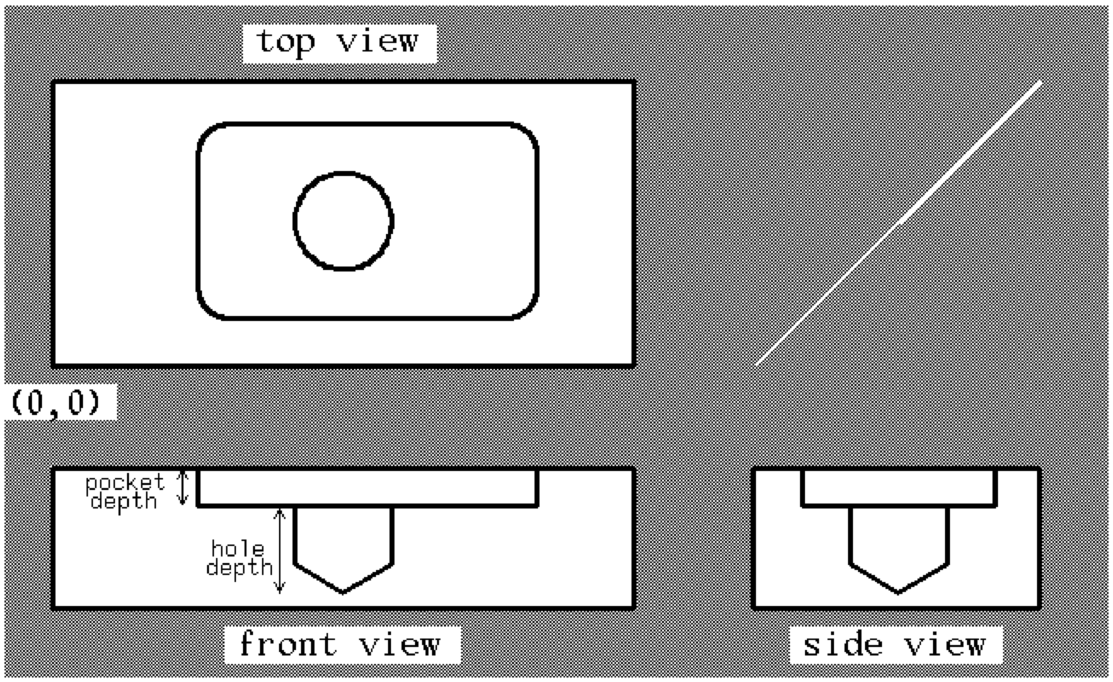
2.2. Reference Features

Figure 3. Coordinate System and Block Location



The design protocol includes the use of "reference features". If feature A is to be made at the bottom of feature B, then one of the parameters of feature A is "reference_feature", and the value of that parameter is the feature number of feature B. As a rule, whenever B is the reference feature for A, the outline of feature A must fit within the outline of feature B, and the bottom of feature B must be flat. An example of a hole which has a pocket as its reference feature is shown in Figure 4. Cases in which the rule is broken are described below.

Figure 4. Hole with Pocket for Reference Feature



2.3. <u>Objectives of the Protocol</u>

The VWS2 design protocol was established to meet the following objectives:

1. Be able to express the design of a large variety of one-sided, two-and-a-half dimensional parts.
2. Be able automatically to produce a process plan (whose structure is the standard AMRF process plan structure) from the design of a part.
3. Be able automatically to generate NC-code to carry out the process plan and machine the part.
4. Be able automatically to make a mechanical drawing of the part.
5. Be able to support automatic verification of design and process plan.
6. Produce design documents easy for a human to read without computer help.

2.4. <u>VWS2 Design Protocol as a Type of CSG</u>

The VWS2 design protocol is an exact CSG representation in which each feature and subfeature is a removed volume. For example, the description of a pocket, in terms of geometric primitives might be as follows: remove (i) a rectangular parallepiped centered on the center of the pocket whose height is the depth of the pocket, whose length is the length of the pocket and whose width is the width of the pocket minus twice the corner radius of the pocket, (ii) a rectangular parallepiped centered on the center of the pocket whose height is the depth of the pocket, whose width is the width of the pocket and whose length is the length of the pocket minus twice the corner radius of the pocket, and (iii) four quarter cylinders (one at each corner of the pocket) whose height is the depth of the pocket and whose radius is the corner radius of the pocket.

Each of the features can be broken down into a set of garden-variety geometric primitives in this fashion, but nowhere in the VWS2 system has this been done. It would not be difficult (although it would take some time) to write a subsystem that would decompose a feature-based design into a set of primitive removed volumes. It has been feasible thus far in the development of the system to treat each feature type as a describing a unified object and develop computational techniques appropriate to each feature type.

Although all the features and subfeatures are purely geometric, they were selected to be included in the system on the basis of being features commonly found on machined parts that could be produced in one, or at most a very few, machining operations.

2.5. Advantages and Disadvantages

The design system currently assumes that all features are being made from one side of the block. A part with features in several sides can be described with the system by specifying a design for each side. It has proved feasible to make very complex parts using this method. During AMRF test runs in September and December, 1986, parts of several different designs, all of which were real designs for parts used by the US Navy, were machined using this technique. On the other hand, having a separate design for each side of the part is awkward, and it is expected that future versions of the system will allow features in several sides in a single design.

An extremely wide range of parts can be described with this design system, and the parts may be complex. An example of a modestly complex design, which we will call the "XYZ Part" since the letters XYZ are machined into it, is given in different formats in Table 1 and Table 2. A mechanical drawing of the XYZ Part, which was generated automatically by the drawing subsystem, is shown in Figure 5. This part contains at least one feature of each type except a chamfer of the whole block.

On the other hand, there are also many parts which cannot be described because none of the feature types fits some portion of the design. For example, a rectangular hole with a sloping flat bottom cannot be described. It is feasible to add features to the design protocol, as described below. Adding a feature and integrating it into all parts of the system is a large project.

There is currently no provision in this design protocol for including tolerance information. It appears feasible to revise the protocol to include tolerance information without much change to the rest of the protocol.

The great advantage of this feature-based design system is that a process plan can be produced automatically to machine any design that can be described in the design protocol and passes design verification. And, if the appropriate tools exist, a part of the design can be machined.

Table 1. Lisp-Readable XYZ Part Design Document

Table 2. Human-Readable XYZ Part Design Document

Figure 5. Drawing of XYZ Part

2.6. Relationship To Other Design Protocols

There is currently no automatic method of extracting a design according to this protocol from the AMRF standard boundary representation of a design or from design documents produced by commercial computer aided design systems. It should be feasible to perform this extraction automatically from a marked AMRF standard design, where the marking has been done by a human. It is believed that a fully automatic system to do this would be extremely difficult to construct.

There is also no method, currently, of automatically producing a design according to some other protocol from the protocol given here. A limited automatic conversion system should be feasible. In particular, an AMRF standard design should be producible from a VWS2 design for those parts in which features do not intersect and which are expressible in the AMRF standard.

Since the AMRF standard design protocol allows only straight lines and circular arcs as edges, there are many parts covering a wide range which can be expressed in the VWS2 protocol which cannot be expressed in the AMRF standard. On the other hand, there are also many parts covering a wide range that can be expressed in the AMRF standard but not the VWS2 system.

The AMRF standard design protocol has the advantage of having the representation of a part be unique, assuming appropriate conventions for locating the part with respect to coordinate axes. The AMRF standard may lend itself better to producing drawings, since it contains much detailed information about points and lines. The AMRF standard and other non-feature-based protocols have the drawbacks of not lending themselves to automatic process planning or generation of nc-code.

The VWS2 design protocol described here has the advantage of being usable as the basis for a fully integrated, automatic CAD-CAM system. In particular, it lends itself to automatic process planning and nc-code generation. The VWS2 design protocol is much easier for a human to read since all the parameter names are meaningful and are printed, and the features are natural. The VWS2 design protocol requires fewer data to represent a given part, since a single feature encompasses points, lines, and faces, and a minimal amount of information is needed to specify a feature.

2.7. Enhanced Design

2.7.1. Introduction

Each feature has an enhanced version, which is prepared automatically by the VWS2 system at the appropriate time. In the enhanced version there are several additional parameter-value pairs and the value of user-given parameters may change.

Except for "z_surf", which is discussed in this section, all details of design enhancement are given under the descriptions of features. The enhanced design of the XYZ Part is shown in Table 3.

The enhanced design is invisible to the user and is prepared automatically by the enhancement subsystem. The enhanced design is not written to a file. The enhanced design is shared among subsystems to a certain extent, however.

In the design editor one feature is enhanced at a time because the user is dealing with one feature at a time. In other parts of the VWS2 system, an entire design is enhanced.

Table 3. Enhanced Design of XYZ Part

2.7.2. Why Have an Enhanced Design?

Having an enhanced version of the design is useful for three reasons:

1. The design protocol is structured to allow the user to specify features in a commonsense manner. For example, if a thru-hole is to be made, the value of the "depth" parameter for the hole is simply "thru". In other parts of the VWS2 system, particularly when nc-code is to be generated, it may be necessary to have a numerical value for the depth of the hole. The enhanced version of a thru-hole has a numerical value.

2. The VWS2 design editor allows the user to specify a feature with different sets of parameters in some cases. For example, the outline of a pocket may be specified either by giving the coordinates of diagonally opposite corners (in which case the feature_type is "pocket_corners") or by giving the coordinates of the center plus the length and width (in which case the feature_type is "pocket_center"). In this case the rest of the system might want to know the coordinates of the corners or the center even if the user has not specified them. In the enhanced version of a pocket, the feature-type is changed to "pocket" in both cases, and all the missing parameters are calculated and inserted.

3. In order to prepare nc-code, it is helpful to have parameters pertaining to the path of tool making a feature, rather than the outline of the resulting feature. In many cases such parameters are calculated and placed in the enhanced design.

2.7.3. z_surf

One enhancement parameter which every feature has is called "z_surf" (short for z_surface). The z_surf is the negative of the distance below the top surface of the block where the top of the feature is located. This parameter is calculated by examining the reference feature situation. If a feature has no reference feature, its z_surf is zero. Otherwise, the z_surf is the negative of the sum of the depths of the nest of reference features above the given feature.

2.8. Feature Verification

The parameters for each feature type must satisfy a number of conditions. For example, the depth of the thread on a hole cannot be greater than the depth of the hole. There is a list of these conditions (called verification rules) for each feature type. The lists are included below. There is a verification subsystem in the VWS2 which checks that the rules are followed. The verification subsystem is described in [K&S1] and [K&S2].

A feature which passes feature verification and reference feature fit checking is guaranteed to be physically realizable. If it fails, like the hole mentioned above, it may not be.

2.9. Data Structure

The design data structure is a tree of attribute-value pairs. The value of some attributes is a subtree. For example, the design itself consists of two pairs: the attribute "header" followed by the header subtree and the attribute "features" followed by the features subtree. The features subtree is composed of pairs in which the attributes are integers and the values are feature descriptions. This pairwise structure goes down to the lowest level of the design, as may be seen in Table 1.

Although the data structure is independent of the choice of computer language chosen to represent it, since the VWS2 system is written in LISP, the system's internal representation is a LISP property list. The VWS2 design editor prepares both a LISP-readable file giving the design, and a file more easily read by humans. Table 2 gives the human-readable version of the XYZ part design document.

2.10. Block Description and Coordinate System

The block-shaped piece of stock material that is always the starting shape is taken to be fixed to a right_handed three_dimensional cartesian coordinate system, as shown in Figure 3. The origin of the coordinate system is at the front lower left-hand corner of the block. The x-axis is coincident with the front lower edge of the block, and is positive towards the right. The y-axis is coincident with the left lower edge of the block and is positive going away. The z-axis is coincident with the front left edge of the block and is positive going upwards. The dimensions of the block are called length (in the x-direction), width (in the y-direction), and height (in the z-direction).

2.11. Feature Depth

The coordinate system is used extensively for specifying x and y values, but not for specifying z-values. Instead of specifying z-values, a parameter called "depth" is used for all the features. The depth of a feature is positive extending downwards into the block, and is measured from the top of the feature. This method of specifying depth was selected as the easiest, most natural to deal with. The depths of the two features in Figure 4 are marked as an example. Remember that the top of a feature is at the top of the block if the feature has no reference feature and at the bottom of the reference feature if there is one. Whenever "depth" is a parameter of a feature, its value either will be a positive number, or it will be the word "thru", which means the feature extends completely through the part.

2.12. Workpiece Description

2.12.1. Introduction

A workpiece description is a set of data that describes an individual workpiece. Every workpiece must refer to a design, but there may be many workpieces made from the same design.

The workpiece description document makes it feasible to represent parts in process, either within the workstation or coming from other workstations. Also by using the workpiece description, it is feasible to use a process plan which was intended to make a part from scratch to write nc-code to finish machining a partially made part. The VWS2 data execution module does this automatically, as described in [KR&W].
A workpiece description may be verified in the same way a design may be verified.

At the current stage of development, the workpiece description loses usefulness if the workpiece is to be turned over, because, as in a design, it is expected that all features are made from one side.

Thus, a single physical object may have several workpiece descriptions if there are features in several sides.

The description of a workpiece is nearly identical to a design. The differences are as follows.

2.12.2. Differences in the Header Section

1. A workpiece description must name the material the workpiece is made of (this is optional in a design). The workpiece material may differ from the design material.

2. A workpiece description must have a "workpiece_id".

3. The block_size may include the "slab" property. This has five sub-properties: front_over, back_over, right_over, left_over, and depth. As shown in Figure 6, the idea is that the block may include overhangs on the four sides which project beyond the given length and width. The overhangs must all be the same thickness, as given by the depth parameter for the slab. If overhangs are present, it looks as if a slab of material is lying atop the block. The slab notion was developed to make it easy to work on the reverse side of a block which was milled flat on four sides to a fixed depth during a previous milling.

The VWS2 system will deal automatically with a workpiece which has this slab property. When the slab on a workpiece is removed, the workpiece must have the same length and width as specified in the corresponding design, and the remaining height must be at least as large as the design height. The description of a workpiece with a slab may have no features.

4. The "description" property, which is required in a design, is optional in a workpiece.

2.12.3. Differences in the Features Section

1. Each feature in a workpiece description must have the same feature number as the corresponding feature in the design. However, the workpiece description does not have to have all the features in the design. For example, it would be feasible to have a workpiece made from the XYZ Part design with only features 1 and 6.

2. A given feature in the workpiece description may not have all the subfeatures of the corresponding design feature. Suppose, for example, that feature 6 on the XYZ part design (which is a hole) had been drilled and countersunk, but not tapped. Then feature 6 would be present in the workpiece description, but it would not have the parameters "thread_depth", "threads_per_inch", and "thread_diameter". The values of all the other parameters for feature 6 would be as given in the design.

Figure 6. Workpiece With Slab

2.13. Adding a Feature to the Protocol

2.13.1. Introduction

This section explains how a feature type may be added to the VWS2 design protocol. To add a feature type, changes must be made in several subsystems: verification, drawing, data execution (particularly nc-code generation), design editor, data base, geometry, and process planning.

The list given below is based on several assumptions:
1. The feature is machinable in a single (new) operation.
2. The feature has no allowable subfeatures.
3. The feature is not flat_bottomed, and, therefore, cannot be a reference feature.
4. A new type of tool, not currently known to the system, will be required for the new operation that makes the feature.

If new tools and a new operation are not needed or if the feature cannot have any reference feature, less work will be necessary. If more than one operation will be required, if the feature has subfeatures, or if the feature can be a reference feature, more work will be necessary.

In the list below, it may be necessary to write several auxiliary functions for some of the principal functions.

Almost every feature has something unique about it that requires unique additions to the system. A few such additions, not listed here, are to be expected.

2.13.2. Verification

1. Write a new function that produces a set of corners of the type used for contour features which has the same outline as the feature. This is so the feature can have a reference feature (we are still assuming it cannot itself be a reference feature for some other feature).

2. Generate verifier:

a. Write verification rules.
b. Make three new sets of entries in the "verify_data" file, so that the automatic verification function generator will be able to interpret the verification rules for the feature.
c. Call "define_verifier" to generate a new verification function.

3. Write a new function that checks the new machining operation.

2.13.3. Drawing

1. Write a function to draw the feature. This normally requires auxiliary functions, such as an outline drawer, a profile drawer, and a masker.
2. Revise "redefine_get_slot", if necessary, to produce a virtual description of the new tool type, so that the design editor will be able to draw the feature.

2.13.4. Data Execution

1. Write a function to generate nc-code. This normally requires auxiliary functions.
2. Revise "add_feature" so that the model of the workpiece can be handled properly.

2.13.5. Design Editor

1. Add appropriate data about the feature to the "FEATURES_LIST".
2. Devise a suitable question and answer routine to guide the user through the process of creating a feature of the new type.
3. Revise the feature changing facility to handle the new feature type.
4. Revise the "array" and "group" facilities to handle the new feature type.

2.13.6. Data Base

1. Add the feature to the "features" file, and revise other parts of that file.
2. Add the new machining operation to the "machine_ops" file.
3. Add appropriate tools to the "tool_catalog" file.
4. Revise the model of the tools in the milling machine to include some tools of the new type.

2.13.7. Geometry

1. Write a function to enhance the new feature. This is an iterative process. As the nc-code generation, verification, process planning, and drawing functions needed to deal with the new feature are developed, it becomes clear what parameters are nice to have available for two or more of these subsystems, and such parameters are candidates for inclusion in the enhancement subsystem.

2. Write specialized geometry functions to deal with geometric aspects of the new feature for which no capability already exists.

2.13.8. Process Planning

1. Write a function to choose the machining operation to make the feature.
2. Revise "order_ops" to determine the place of the new machining operation in a sequence of machining operations.
3. Revise "select_tool_diameter" to include the new tool.
4. Revise "select_tool_type" to include the new tool.
5. Revise the functions that pick spindle speeds, feed rates, and pass depths to include the new tool.

2.14. An Example of a Design

As mentioned earlier, Table 1 gives the design of a part which we have called the "XYZ Part". Table 3 shows the enhanced version of the design, and a picture of that part is shown in Figure 5.

2.15. Feature Descriptions

2.15.1. Introduction

Immediately below there is a discussion of each feature type in the system. Each discussion includes:

1. Comments about the feature type,
2. A description of the appearance of the feature type,
3. A list of required parameters,
4. A list of optional subfeatures,
5. A list of optional parameters,
6. A table of enhancement parameters,
7. The verification rules which a feature of that type must satisfy,
8. A design document consisting of features of the type under discussion, and
9. A drawing of the part made from the design.

Examples of enhanced features are in Table 3 and are not included below.

A summary of feature parameters and subfeatures follows immediately in Table 4.

The appearances described below are based on the assumption that the feature will pass its verification test. If a feature does not pass its verification test, additional appearances are possible, both on the part itself and on the drawing of the part made by the system. But in this case it is possible that the part will not look like the drawing of it. A few of the verification rules, in fact, are included only because the drawing subsystem is limited in what it can draw correctly, and it was desired to be sure that all verifiable features would be drawn correctly.

The appearances given here are based on the assumption that a feature does not intersect any other features. Obviously, the appearance of the actual part will be different if there are feature intersections. The drawing system does not detect feature intersections, so the appearance of features on the drawing will be as described below, even if they intersect.

The groove and pocket features may have the size and location of their outlines specified by giving the x and y coordinates of the upper left and lower right corners of an imaginary box around the feature. Similarly, the location of text is specified by giving the x and y coordinates of the lower left hand corner of an imaginary parallelogram around the outline of the text. Figure 7 shows how this location is done. In the case of grooves and pockets, the feature fits completely inside the box. In the case of text, the outline of the text extends beyond the imaginary parallelogram because of the line_width of the text. The parallelogram around text is located with respect to the center line of the text.

A few conventions have been followed in selecting parameter names. Any parameter name that starts or ends with x or y represents an x or y value. Depth is always as described above. The portion of a parameter name that is "upper_l", "lower_r", or "lower_l" refers to the upper left, lower right, or lower left corner of one of the imaginary boundaries described in the preceding paragraph. "ul" stands for upper left, "lr" for lower right, "ch" for chamfer, "chout" for chamfer_out, "chin" for chamfer_in, and "cr" for corner radius.

The rules each feature must obey are written in English, but it is constrained English since it must be read by the VWS2 automatic feature verifier generation system, which reads English only as a second language. In the verification rules, the following usages are observed:

1. "min_thick" is a small positive number representing a minimum wall thickness.

2. "The x-value of the plus-x side" of a feature means the x-value of a point on the top view of the outline of the feature where x is largest. Similar phrases involving "plus-y", "minus-x", and "minus-y" are defined analogously.

3. "maximum corner radius" means the corner radius of the feature plus any increment that may result from countersinking or chamfering.

4. The "vertical rise" of a feature is the length of that portion of the wall of the feature which is vertical. In a hole, for example, some of the depth may be accounted for by a conical bottom, and some may be accounted for by a chamfer at the top. The remainder of the depth of the hole is its vertical rise.

Table 4. Feature Parameters and Subfeatures

Figure 7. Locating a Pocket, a Groove, and Text

2.15.2. Chamfer_out

2.15.2.1. Comments

There are three uses of chamfer in the system: chamfer_out as a primary feature, and chamfer_out and chamfer_in as subfeatures. A chamfer_out is a primary feature only if it is to be a chamfering of the original block. As a subfeature, chamfer_out is only used to chamfer the island left inside a groove. Chamfer_in is used to chamfer the inside of the outline of a groove, hole, pocket, or straight_groove.

All the chamfering routines expect the feature being chamfered to look like a rectangle with rounded corners (which may degenerate either to a rectangle with square corners or to a circle). A straight groove may be chamfered only if it is horizontal or vertical. All chamfers are 45 degree bevels on an edge between a horizontal surface and a vertical surface. Chamfers do not require much specification information because the information about the edge being chamfered is contained either in the dimensions of the block or in parameters of the feature being chamfered. The only specification needed is the "chamfer_out_depth". The bevel extends to the depth given by this parameter. Since the bevel is at 45 degrees, the width of the chamfer is equal to its depth.

Except where specifically noted otherwise, this remainder of this section describes the chamfer feature, not the subfeature.

2.15.2.2. Appearance

45 degree bevel on the outside top edge of the original block, when used as a primary feature. 45 degree bevel on the parent feature when used as a subfeature.

2.15.2.3. Required Parameters

feature_type	Must be "chamfer_out".
chamfer_out_depth	Must be a positive number less than the height of the block if a primary feature. Must be less than the vertical rise of the parent feature prior to chamfering when used as a subfeature.

2.15.2.4. Optional Subfeatures - None.

2.15.2.5. Optional Parameters - None.

2.15.2.6. Enhancement Parameters

All the enhancers for features which can be chamfered look for the presence of a chamfering depth. If one is found, enhancement parameters are added to specify the corners and corner radius of the chamfer.

z_surf	Set to a number.

chout_ulx	Set to zero.
chout_uly	Set to the width of the block.
chout_lrx	Set to the length of the block.
chout_lry	Set to zero.
chout_cr	Set to zero.

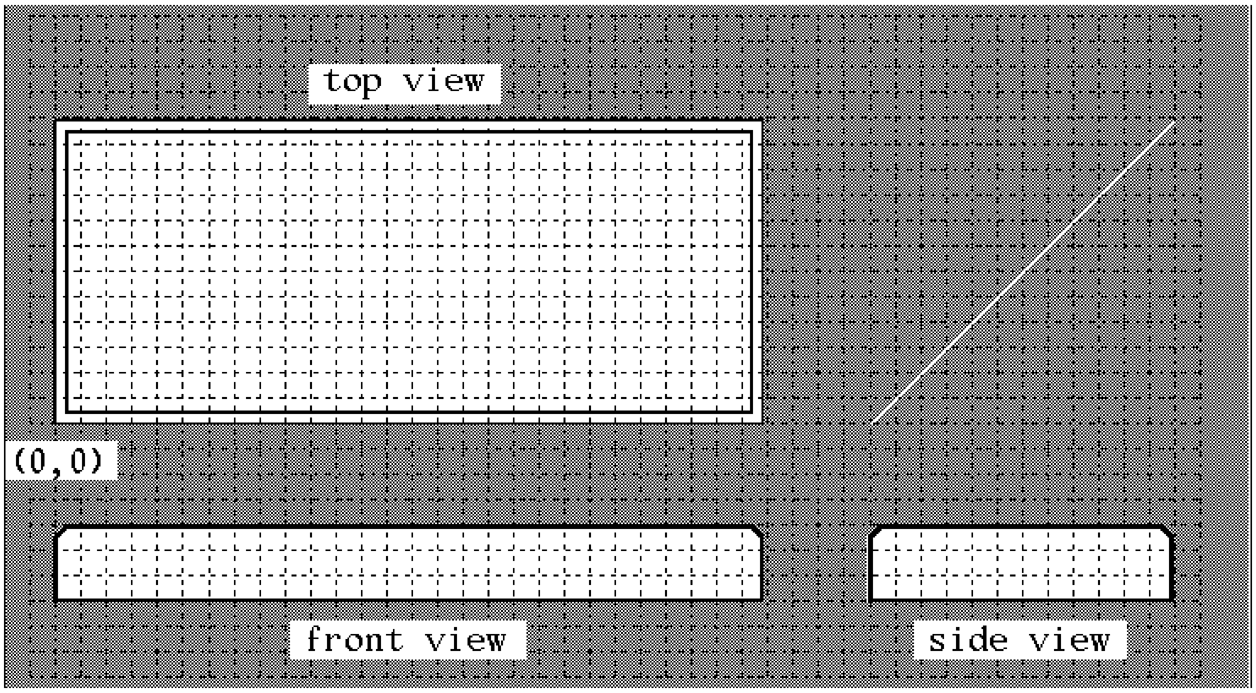
2.15.2.7. Verification Rules

The chamfer_out_depth should be less than the height of the block.

2.15.2.8. Example Design

This is a simple chamfer of a block. In examples for a number of other feature types there are additional chamfers.

Figure 8. Chamfer_Out



```
(setplist 'chamfer_out
          '(header (header  design_id   chamfer_out
                            description  "block chamfer"
                            block_size  (block_size   length 6.95   width 2.975   height 0.735))
            features  (features 1
                       (1 feature_type  chamfer_out
                          chamfer_out_depth  0.1))))
```

2.15.3. Groove

2.15.3.1. Comments

A groove may be thought of as the volume removed when a tool which is either flat or rounded on the end is passed through the block in a path shaped like a rectangle with (optionally) rounded corners. Because of the radius of the tool, the exterior outline of a groove always has rounded corners if the corners lie within the block. The tool path may pass outside the block at some points. A groove may not go through the bottom of the block. If it did, the island of material surrounded by the groove would fall out, which is not safe, since the island might jam and break the tool.

2.15.3.2. Appearance

Round-bottomed grooves have bottoms that are circular arcs in cross-section. The walls of grooves are vertical unless the groove is shallow with a round bottom. The straight parts of the outline of a groove are parallel to the x or y axes. If the corner radius of the groove is large enough, the rectangle may degenerate to a circle. There is always an island or infield left inside the groove. If a groove passes through the sides of the block, it must be the straight portion of the groove that passes through. If a groove passes through two opposite sides of the block, it degenerates to two parallel straight cuts through the block.

2.15.3.3. Required Parameters

feature_type	Must be "groove".
upper_l_x	Must be a number or "thru". Represents the x-coordinate of the upper left hand corner of the bounding box.
upper_l_y	Must be a number or "thru". Represents the y-coordinate of the upper left hand corner of the bounding box.
lower_r_x	Must be a number or "thru". Represents the x-coordinate of the lower right hand corner of the bounding box.
lower_r_y	Must be a number or "thru". Represents the y-coordinate of the lower right hand corner of the bounding box.
depth	Must be a positive number.
width	Must be a positive number. Represents the width of the cut.
bottom_type	Must be "flat" or "round".
corner_radius	Must be a positive number. Represents the radius of the outermost circular arc at the corners of the groove.

2.15.3.4. Optional Subfeatures

Chamfer_in and chamfer_out are optional subfeatures. See the discussion under chamfer_out above.

2.15.3.5. Optional Parameters

reference_feature	Must be a positive integer.
chamfer_in_depth	Must be a positive number.
chamfer_out_depth	Must be a positive number.

2.15.3.6. Enhancement Parameters

For several uses (especially writing nc-code), it is handy to have a second imaginary box around the groove that touches the center line of the groove rather than the outside of the groove. Parameters associated with this center line bounding box are put in place by the enhancement subsystem. These are the parameters that end in "nc" in the following list. The parameters that begin with "chin" or "chout" in the list are to assist in chamfering. The "chin" parameters are inserted only for a chamfer-in, the "chout" parameters only for a chamfer-out.

z_surf	Set to a number.
upper_l_x	If the given value is "thru", it is replaced by a negative number.
upper_l_y	If the given value is "thru", it is replaced by a positive number.
lower_r_x	If the given value is "thru", it is replaced by a positive number.
lower_r_y	If the given value is "thru", it is replaced by a negative number.
ulx_nc	A number representing the x-coordinate of the upper left hand corner of the center line bounding box.
uly_nc	A number representing the y-coordinate of the upper left hand corner of the center line bounding box.
lrx_nc	A number representing the x-coordinate of the lower right hand corner of the center line bounding box.
lry_nc	A number representing the y-coordinate of the lower right hand corner of the center line bounding box.
cr_nc	A non-negative number representing the corner radius of the center line of the groove.
tool_diam	A positive number representing the diameter of the tool needed to make the groove. This is the same as the width of the groove unless the groove is round-bottomed and its depth is less than half the width.
chin_ulx	Set to the value of upper_l_x.
chin_uly	Set to the value of upper_l_y.
chin_lrx	Set to the value of lower_r_x.
chin_lry	Set to the value of lower_r_y.
chin_cr	Set to the value of corner_radius.
chout_ulx	Set to the value of upper_l_x plus width.
chout_uly	Set to the value of upper_l_y minus width.
chout_lrx	Set to the value of lower_r_x minus width.
chout_lry	Set to the value of lower_r_y plus width.
chout_cr	Set to the value of corner_radius minus width (or zero if that is negative).

2.15.3.7. Verification Rules

If the x_value of the plus_x side of the groove is not greater than the length of the block, then the x_value of the plus_x side of the groove should not be greater than the length of the block minus min_thick.

If the x_value of the plus_x side of the groove is greater than the length of the block, then the x_value of the plus_x side of the groove should be greater than the length of the block plus the maximum corner radius of the groove.

If the x_value of the minus_x side of the groove is not less than zero, then the x_value of the minus_x side of the groove should not be less than min_thick.

If the x_value of the minus_x side of the groove is less than zero, then the x_value of the minus_x side of the groove should be less than zero minus the maximum corner radius of the groove.

If the y_value of the plus_y side of the groove is not greater than the width of the block, then the y_value of the plus_y side of the groove should not be greater than the width of the block minus min_thick.

If the y_value of the plus_y side of the groove is greater than the width of the block, then the y_value of the plus_y side of the groove should be greater than the width of the block plus the maximum corner radius of the groove.

If the y_value of the minus_y side of the groove is not less than zero, then the y_value of the minus_y side of the groove should not be less than min_thick.

If the y_value of the minus_y side of the groove is less than zero, then the y_value of the minus_y side of the groove should be less than zero minus the maximum corner radius of the groove.

The y_value of the plus_y side of the groove plus 0.00001 minus the y_value of the minus_y side of the groove should be greater than 2.0 times the maximum corner radius of the groove.

The y_value of the plus_y side of the groove minus the y_value of the minus_y side of the groove should be greater than 2.0 times the width of the groove plus the chamfer_in_depth of the groove plus the chamfer_out_depth of the groove.

The y_value of the plus_y side of the groove minus the y_value of the minus_y side of the groove should be greater than min_thick plus 2.0 times the width of the groove plus the chamfer_in_depth of the groove.

The x_value of the plus_x side of the groove plus 0.00001 minus the x_value of the minus_x side of the groove should be greater than 2.0 times the maximum corner radius of the groove.

The x_value of the plus_x side of the groove minus the x_value of the minus_x side of the groove should be greater than 2.0 times the width of the groove plus the chamfer_in_depth of the groove plus the chamfer_out_depth of the groove.

The x_value of the plus_x side of the groove minus the x_value of the minus_x side of the groove should be greater than min_thick plus 2.0 times the width of the groove plus the chamfer_in_depth of the groove.

The z_value of the top of the groove should be greater than 0.0.

The z_value of the bottom of the groove should be greater than min_thick.

The tool_diameter of the groove should be greater than 0.0624.

The width of the groove should be less than 0.00001 plus 2.0 times the corner radius of the groove.

The chamfer_in_depth of the groove should not be greater than the vertical rise of the groove.

The chamfer_out_depth of the groove should not be greater than the vertical rise of the groove.

2.15.3.8. Example Design

Figure 9 shows four grooves on a block, two round-bottomed and two flat-bottomed. The groove in the upper right has a chamfer_in. The round groove has a chamfer_out.

Figure 9. Grooves

2.15.4. Hole

2.15.4.1. Comments

The bottom of a hole may either be conical, like a drill hole, or flat. A hole is either (i) a "clean through hole" (one that goes cleanly through the bottom of the part), (ii) a "partly through hole" (one with a conical bottom where only the tip of the cone sticks through the bottom of the part), or (iii) a "blind hole" (one that does not go through the bottom at all). A clean through hole needs no bottom type. The bottom type of a partly through hole is always conical. The bottom type of a blind hole is either flat or conical.

A hole may be chamfered or countersunk (but not both). It may also be threaded. In the case of a thread, either all three parameters describing a thread (thread_depth, thread_diameter, and threads_per_inch) should be present or they should all be absent.

A hole is the only feature type that can be used as a reference feature even if it is not flat_bottomed. A conical_bottomed hole may be used as the reference feature for another conical_bottomed hole if the two are concentric.

2.15.4.2. Appearance

A hole is a circular recess. If the bottom of the hole is conical, the tip angle is 118 degrees (the tip angle of a standard drill). If the hole is chamfered or countersunk, the top edge of the hole will be bevelled, the only difference being the angle of the bevel (45 degrees from the horizontal for a chamfer, 49 degrees for a countersink). If the hole is threaded, a picture of the threading will be shown on the side views on the picture of the hole. The thread depth will be shown correctly in the picture. The number of threads per inch on the picture will be shown correctly unless the threads would be too close together to distinguish one from the next. In that case, the threads_per_inch on the picture will be fewer than in reality, so that the threads are discernable on the picture.

2.15.4.3. Required Parameters

feature_type	Must be "hole".
center_x	Must be a number. Represents the x-coordinate of the center of the hole.
center_y	Must be a number. Represents the y-coordinate of the center of the hole.
depth	ust be a positive number or "thru". If the depth is "thru", the hole is a clean through hole.
diameter	Must be a positive number. Represents the diameter of the hole.
bottom_type	This parameter is actually required in only some cases, as follows: (i) blind hole -- bottom_type must be either "flat" or "conical". (ii) partly thru hole -- bottom_type must be "conical". (iii) clean thru hole -- there should be no bottom_type. If there is one, it will be ignored by the system.

2.15.4.4. Optional Subfeatures

A hole may be countersunk or chamfered, but not both. A hole may be threaded.

2.15.4.5. Optional Parameters

reference_feature	Must be a positive integer.
thread_diameter	Must be a positive number greater than the hole’s diameter. Represents the maximum diameter of the thread.
thread_depth	Must be a positive number or "thru".
threads_per_inch	Must be a positive integer.
chamfer_in_depth	Must be a positive number.
countersink_diameter	Must be a positive number greater than the hole’s diameter.

2.15.4.6. Enhancement Parameters

Several of the enhancement parameters are used so that a hole and a pocket, when enhanced, will have similar sets of parameters.

z_surf	Set to a number.
depth	If the value of depth is "thru", it will be replaced by a number.
thread_depth	If the value of thread_depth is "thru", it will be replaced by a number.
upper_l_x	Set to center_x minus radius.
upper_l_y	Set to center_y plus radius.
lower_r_x	Set to center_x plus radius.
lower_r_y	Set to center_y minus radius.
corner_radius	Set to half the hole diameter.
chin_ulx	Set to center_x minus radius for a chamfer.
chin_uly	Set to center_y plus radius for a chamfer.
chin_lrx	Set to center_x plus radius for a chamfer.
chin_lry	Set to center_y minus radius for a chamfer.
chin_cr	Set to half the hole diameter for a chamfer.

2.15.4.7. Verification Rules

The x_value of the plus_x side of the hole should not be greater than the length of the block minus min_thick.

The x_value of the minus_x side of the hole should not be less than the x_value of the minus_x side of the block plus min_thick.

The y_value of the plus_y side of the hole should be less than the width of the block minus min_thick.

The y_value of the minus_y side of the hole should be greater than min_thick.
The z_value of the top of the hole should be greater than 0.0.

If the bottom_type of the hole is equal to flat and the z_value of the bottom of the hole is greater than 0.0, then the z_value of the bottom of the hole should be greater than min_thick.

If the countersink_diameter of the hole is greater than zero, then the chamfer_in_depth of the hole should not be greater than zero.

If the countersink_diameter of the hole is greater than zero, then the countersink_diameter of the hole should be greater than the diameter of the hole.

The countersink_diameter of the hole should not be greater than the diameter of the hole plus (1.7385 times the vertical rise of the hole).

The chamfer_in_depth of the hole should not be greater than the vertical rise of the hole.

It should be that the thread_diameter of the hole is equal to zero and the threads_per_inch of the hole is equal to nil and the thread_depth of the hole is equal to nil, or the thread_diameter of the hole is greater than zero and the threads_per_inch of the hole is not equal to nil and the thread_depth of the hole is not equal to nil.

If the thread_diameter of the hole is greater than zero and the vertical rise of the hole is less than the z_value of the top of the hole, then the thread_depth of the hole should be less than the vertical rise of the hole.

If the thread_diameter of the hole is greater than zero, then 8 should be equal to the threads_per_inch of the hole or 12 should be equal to the threads_per_inch of the hole or 16 should be equal to the threads_per_inch of the hole or 20 should be equal to the threads_per_inch of the hole or 24 should be equal to the threads_per_inch of the hole or 32 should be equal to the threads_per_inch of the hole.

If the thread_diameter of the hole is greater than zero, then the threads_per_inch of the hole times (the thread_diameter of the hole minus the diameter of the hole) should not be greater than 1.08.

If the thread_diameter of the hole is greater than zero, then the thread_diameter of the hole should not be less than the diameter of the hole.

If the chamfer_in_depth of the hole is greater than zero and the thread_depth of the hole is not equal to nil, then the thread_depth of the hole should be greater than the chamfer_in_depth of the hole.

If the countersink_diameter of the hole is greater than zero and the thread_depth of the hole is not equal to nil, then the thread_depth of the hole should be greater than 0.5752 times the countersink_diameter of the hole minus the diameter of the hole.

2.15.4.8. Example Design - See Figure 10.

Figure 10. Holes

2.15.5. Pocket

2.15.5.1. Comments

A pocket is a recess with a flat bottom and vertical walls. A pocket may go through the block at the bottom or at the sides (but not through two opposite sides and the bottom simultaneously, or the block would be cut in half). If a pocket passes through the side of the block, it must be a straight portion of the pocket that passes through.

In the design editor there are actually two kinds of pockets: pocket_corners and pocket_center. The rest of the VWS2 system only recognizes "pocket". This is accomplished by having the enhancement subsystem change the value of the feature_type parameter to "pocket" in both cases.

2.15.5.2. Appearance

The outline of a pocket is a rectangle with rounded corners. It may degenerate to a circle. Like a groove, a pocket may may be imagined as being surrounded by a bounding box whose sides are parallel to the x and y axes. Because a pocket may pass through the block several ways, there is quite a bit of variety in the appearance of pockets.

2.15.5.3. Required Parameters

A pocket may be specified in two ways, either by giving the coordinates of diagonally opposite corners of the bounding box or by giving the coordinates of the center of the box and its length and width. Thus the first two sets of five parameters given below are alternatives. The remaining parameters are the same in either case.

Use these five

feature_type	Must be "pocket_corners".
upper_l_x	Must be a number or "thru". Represents the x-coordinate of the upper left hand corner of the bounding box.
upper_l_y	Must be a number or "thru". Represents the y-coordinate of the upper left hand corner of the bounding box.
lower_r_x	Must be a number or "thru". Represents the x-coordinate of the lower right hand corner of the bounding box.
lower_r_y	Must be a number or "thru". Represents the y-coordinate of the lower right hand corner of the bounding box.

or these five

feature_type	Must be "pocket_center".
center_x	Must be a number. Represents the x-coordinate of the center of the bounding box.
center_y	Must be a number. Represents the y-coordinate of the center of the bounding box.
length	Must be a positive number. Represents the length of the bounding box.
width	Must be a positive number. Represents the width of the bounding box.

and always these

depth	Must be a positive number or "thru".
corner_radius	Must be a positive number.

2.15.5.4. Optional Subfeatures

chamfer_in

2.15.5.5. Optional Parameters

reference_feature	Must be a positive integer.
chamfer_in_depth	Must be a positive number less than the depth of the pocket.

2.15.5.6. Enhancement Parameters

Regardless of whether the pocket starts as a pocket_center or a pocket_corners, it looks the same when enhanced.

feature_type	Set to "pocket".
z_surf	Set to a number.
upper_l_x	If given as "thru" or not given, it is changed to a number.
upper_l_y	If given as "thru" or not given, it is changed to a number.
lower_r_x	If given as "thru" or not given, it is changed to a number.
lower_r_y	If given as "thru" or not given, it is changed to a number.
center_x	If not given, it is calculated as a number.
center_y	If not given, it is calculated as a number.
depth	If given as "thru", it is changed to a number.
chin_ulx	Set to the value of upper_l_x if there is a chamfer_in.
chin_uly	Set to the value of upper_l_y if there is a chamfer_in.
chin_lrx	Set to the value of lower_r_x if there is a chamfer_in.
chin_lry	Set to the value of lower_r_y if there is a chamfer_in.
chin_cr	Set to the value of corner_radius if there is a chamfer_in.

2.15.5.7. Verification Rules

If the x_value of the plus_x side of the pocket is not greater than the length of the block, then the x_value of the plus_x side of the pocket should not be greater than the length of the block minus min_thick.

If the x_value of the plus_x side of the pocket is greater than the length of the block, then the x_value of the plus_x side of the pocket should be greater than the length of the block plus the maximum corner radius of the pocket.

If the x_value of the minus_x side of the pocket is not less than zero, then the x_value of the minus_x side of the pocket should not be less than min_thick.

If the x_value of the minus_x side of the pocket is less than zero, then the x_value of the minus_x side of the pocket should be less than zero minus the maximum corner radius of the pocket.

If the x_value of the plus_x side of the pocket is greater than the length of the block, and the total depth of the pocket is greater than the height of the block, then the x_value of the minus_x side of the pocket should be greater than 0.5.

If the x_value of the minus_x side of the pocket is less than zero and the total depth of the pocket is greater than the height of the block, then the x_value of the plus_x side of the pocket should be less than the length of the block minus 0.5.

If the y_value of the plus_y side of the pocket is not greater than the width of the block, then the y_value of the plus_y side of the pocket should not be greater than the width of the block minus min_thick.

If the y_value of the plus_y side of the pocket is greater than the width of the block, then the y_value of the plus_y side of the pocket should be greater than the width of the block plus the maximum corner radius of the pocket.

If the y_value of the minus_y side of the pocket is not less than zero, then the y_value of the minus_y side of the pocket should not be less than min_thick.

If the y_value of the minus_y side of the pocket is less than zero, then the y_value of the minus_y side of the pocket should be less than zero minus the maximum corner radius of the pocket.

If the y_value of the plus_y side of the pocket is greater than the width of the block, and the total depth of the pocket is greater than the height of the block, then the y_value of the minus_y side of the pocket should be greater than 0.5.

If the y_value of the minus_y side of the pocket is less than zero and the total depth of the pocket is greater than the height of the block, then the y_value of the plus_y side of the pocket should be less than the width of the block minus 0.5.

The length of the pocket should be greater than (the corner radius of the pocket times 2.0) minus 0.00001.

The width of the pocket should be greater than (the corner radius of the pocket times 2.0) minus 0.00001.

The z_value of the top of the pocket should be greater than 0.0.

The corner radius of the pocket should be greater than 0.0624.

If the total depth of the pocket is less than the height of the block, then the z_value of the bottom of the pocket should be greater than min_thick.

The chamfer_in_depth of the pocket should not be greater than the vertical rise of the pocket.

2.15.5.8. Example Design - See Figure 11.

Figure 11. Pockets

2.15.6. Straight_groove

2.15.6.1. Comments

A straight_groove may be thought of as the volume removed when a tool is passed in a straight line from one point to another. The straight_groove is defined by giving the coordinates of the endpoints of this removed volume. A straight_groove may pass through the bottom of the block. Unlike grooves and pockets, which have the sides of their bounding boxes parallel to the x and y axes, a straight_groove may be oblique. A straight_groove which is parallel to the x or y axis may be chamfered and may pass through the side of the block, but an oblique straight_groove may not.

2.15.6.2. Appearance

The outline of straight_groove is a rectangle with semicircular ends. The bottom may be either flat or round.

2.15.6.3. Required Parameters

feature_type	Must be "straight_groove".
x1	Must be a number or "thru". If x1 is "thru", it is through the minus-x side of the block. It represents the x-coordinate of one end of the straight_groove.
y1	Must be a number or "thru". If y1 is "thru", it is through the minus-y side of the block. It represents the y-coordinate of one end of the straight_groove.
x2	Must be a number or "thru". If x2 is "thru", it is through the plus-x side of the block. It represents the x-coordinate of the other end of the straight_groove.
y2	Must be a number or "thru". If y2 is "thru", it is through the plus-y side of the block. It represents the y-coordinate of the other end of the straight_groove.
depth	Must be a positive number or "thru".
width	Must be a positive number.
bottom_type	Must be "flat" or "round".

2.15.6.4. Optional Subfeatures

chamfer_in

2.15.6.5. Optional Parameters

reference_feature	Must be a positive integer.
chamfer_in_depth	Must be a positive number less than the vertical rise of the straight_groove.

2.15.6.6. Enhancement Parameters

z_surf	Set to a number.
x1	If x1 is "thru" it is set to a negative number.
y1	If y1 is "thru" it is set to a negative number.
x2	If x2 is "thru" it is set to a positive number.
y2	If y2 is "thru" it is set to a positive number.
x1_nc	Set to the x-coordinate of the center point of the tool at the "1" end of the straight_groove.
y1_nc	Set to the y-coordinate of the center point of the tool at the "1" end of the straight_groove.
x2_nc	Set to the x-coordinate of the center point of the tool at the "2" end of the straight_groove.
y2_nc	Set to the y-coordinate of the center point of the tool at the "2" end of the straight_groove.
tool_diam	A positive number representing the diameter of the tool needed to make the groove. This is the same as the width of the groove unless the groove is round-bottomed and its depth is less than half the width.
chin_ulx	Set if there is a chamfer_in and the straight_groove is not oblique.
chin_uly	Set if there is a chamfer_in and the straight_groove is not oblique.
chin_lrx	Set if there is a chamfer_in and the straight_groove is not oblique.
chin_lry	Set if there is a chamfer_in and the straight_groove is not oblique.
chin_cr	Set if there is a chamfer_in and the straight_groove is not oblique.

2.15.6.7. Verification Rules

If the x_value of the plus_x side of the groove is less than the length of the block, then the x_value of the plus_x side of the groove should not be greater than the length of the block minus min_thick.

If the x_value of the plus_x side of the groove is greater than the length of the block, then the x_value of the plus_x side of the groove should be greater than the length of the block plus the maximum corner radius of the groove.

If the x_value of the minus_x side of the groove is greater than zero, then the x_value of the minus_x side of the groove should not be less than min_thick.

If the x_value of the minus_x side of the groove is less than zero, then the x_value of the minus_x side of the groove should be less than zero minus the maximum corner radius of the groove.

If the x_value of the plus_x side of the groove is greater than the length of the block, and the total depth of the groove is greater than the height of the block, then the x_value of the minus_x side of the groove should be greater than 0.5.

If the x_value of the minus_x side of the groove is less than zero and the total depth of the groove is greater than the height of the block, then the x_value of the plus_x side of the groove should be less than the length of the block minus 0.5.

If the y_value of the plus_y side of the groove is less than the width of the block, then the y_value of the plus_y side of the groove should not be greater than the width of the block minus min_thick.

If the y_value of the plus_y side of the groove is greater than the width of the block, then the y_value of the plus_y side of the groove should be greater than the width of the block plus the maximum corner radius of the groove.

If the y_value of the minus_y side of the groove is greater than zero, then the y_value of the minus_y side of the groove should not be less than min_thick.

If the y_value of the minus_y side of the groove is less than zero, then the y_value of the minus_y side of the groove should be less than zero minus the maximum corner radius of the groove.

If the y_value of the plus_y side of the groove is greater than the width of the block, and the total depth of the groove is greater than the height of the block, then the y_value of the minus_y side of the groove should be greater than 0.5.

If the y_value of the minus_y side of the groove is less than zero and the total depth of the groove is greater than the height of the block, then the y_value of the plus_y side of the groove should be less than the width of the block minus 0.5.

The length of the groove should not be less than the width of the groove minus 0.00001.

The z_value of the top of the groove should be greater than 0.0.

The width of the groove should be greater than 0.0624.

If the total depth of the groove is less than the height of the block, then the z_value of the bottom of the groove should be greater than min_thick.

The chamfer_in_depth of the groove should not be greater than the vertical rise of the groove.

If the chamfer_in_depth of the groove is greater than zero, then the x1 of the groove should be close to the x2 of the groove, or the y1 of the groove should be close to the y2 of the groove.

If the x1 of the groove is not close to the x2 of the groove, and the y1 of the groove is not close to the y2 of the groove, then the x_value of the plus_x side of the groove should be less than the length of the block and the x_value of the minus_x side of the groove should be greater than zero and the y_value of the plus_y side of the groove should be less than the width of the block and the y_value of the minus_y side of the groove should be greater than zero.

2.15.6.8. Example Design - See Figure 12.

Figure 12. Straight_grooves

2.15.7. Text

2.15.7.1. Comments

Text is characters that may be machined into the block. The text is made as a series of round-bottomed grooves that are either straight line segments or circular arcs. The characters currently in the system are the 26 letters (upper case only), the digits 0 through 9, and the space and period characters. It is feasible to add characters to the system.

A default font called "plain" is stored in the "fonts" database of the VWS2 system. A new font may be designed and stored in the database in a few minutes with the "new_font" command. This command is not accessible to the user of the system from the design editor; access is directly from LISP. Documentation of the new_font command and the workings of the font-making system is given later in this paper (section 2.16).

The system has been run with four additional fonts called "broad", "round", "italic", and "angular". These are made by running the "new_font" command once for each additional font when the system is loaded.

Horizontal spacing of characters may be changed by making a new font, by inserting "space" characters between the visible characters, or by making each character a separate feature. Vertical spacing of lines of text is not an attribute of the system. Each line of text must be independently located by the user.

2.15.7.2. Appearance

The outline of text looks like the upper case of the letters and numbers being made. The base of the characters is always horizontal, although the characters may be tilted to the right or left. Unlike most other features, the outline of text on the top view of the part covers up any previously drawn features. The profile is that of a series of round-bottomed grooves.

Since the height and line_width of characters are independent of the font, a great deal of variety in the appearance of a given font is available. Compare the characters of the word "THIS" with the same characters of the words "IS THE" on Figure 13 to get some appreciation of the effect of height and line_width on characters of the same font.

2.15.7.3. Required Parameters

feature_type	Must be "text".
text	Must be a string made up of characters that are in the system. Letters in the string may be either lower or upper case, but will be machined in upper case, regardless.
lower_l_x	Must be a positive number. Represents the x-coordinate of the lower left hand corner of the bounding parallelogram shown in Figure 7.
lower_l_y	Must be a positive number. Represents the y-coordinate of the lower left corner of the parallelogram.
depth	Must be a positive number. Represents the depth of the grooves.
height	Must be a positive number. Represents the difference between the maximum and minimum values of y that occur on the center lines of the tallest characters. The total height of a tall character is equal to the value of this parameter plus the value of the line_width parameter.
line_width	Must be a positive number. Represents the width of the grooves.

2.15.7.4. Optional Subfeatures - None.

2.15.7.5. Optional Parameters

reference_feature	Must be a positive integer.
font	Must be an atom which is the name of a font that is stored in the system. If this parameter is not present, or if its value is nil, the plain font is used.

2.15.7.6. Enhancement Parameters

z_surf	Set to a number.
text	Text is converted from a string to a list. This is done according to the table of ascii equivalents in the fonts data base. Currently, lower case letters are converted to upper case letters during this process.
tool_diam	A positive number representing the diameter of the tool needed to make the text. This is the same as the line_width of the text, unless the depth is less than half the width.

2.15.7.7. Verification Rules

The set of text characters should be contained in the set of all machinable characters.

The x_value of the plus_x side of the text should not be greater than the length of the block minus min_thick.

The x_value of the minus_x side of the text should not be less than the x_value of the minus_x side of the block plus min_thick.

The y_value of the plus_y side of the text should be less than the width of the block minus min_thick.

The y_value of the minus_y side of the text should be greater than min_thick.

The z_value of the top of the text should be greater than 0.0.

The total depth of the text should not be greater than the height of the block minus min_thick.

2.15.7.8. Example Design - See Figure 13.

Figure 13. Text

2.15.8. Contour_groove

2.15.8.1. Comments

This is the first of the three contour features. The other two are contour_pocket and side_contour. The three contour features share a common method of specifying a "defining line". The defining line is the outer outline of a contour_pocket, the inner outline of a side_contour, or the center line, right edge, or left edge of a contour_groove. The defining line of a contour_pocket or a side_contour must be closed. The defining line of a contour_groove may be either open or closed.

The objective in creating the notion of a "defining line" as part of the design protocol was to allow the specification of a continuous line consisting of any combination of arcs of circles and straight line segments, as long as the line never doubled back on itself. It was also desired to facilitate the imposition of the requirement that a circular arc be tangent to an adjoining straight line segment at the point of contact, if the designer wanted to impose that requirement. This kind of defining line, with additional requirements, as discussed below, insures that the feature which is described will be machinable. The following method of specifying the defining line meets the objective and VWS Design Editor makes the creation of defining lines a straightforward matter.

The method of specifying a defining line described here does not allow the user to specify the center of a circular arc. This is a drawback that can be worked around, but often only awkwardly. Allowing the user to specify the center of an arc, without sacrificing the automatic calculation of tangent points would be a high-priority item in any revision of this protocol.

To specify a defining line the user gives the x and y coordinates of a number of points. The points are numbered sequentially starting with 1, and are connected temporarily with straight lines, 1 to 2 to 3, etc. to form a "frame". If the defining line is closed, the last point is connected to the first point. At each of these points (which may be thought of as corners), the user specifies a radius. If the radius is zero, the corner is not rounded off. If the radius is a positive number, the corner is rounded off with that radius. Corner rounding is done with circular arcs which are tangent to the two lines which meet at the corner. This process produces the defining line for the feature. Figure 14 shows the frame and the defining line for a simple contour feature.

Instead of specifying a number for a corner radius, the user may specify either "join_back" or "join_ahead" as a corner radius. If join_back is specified as the corner radius at corner n, then the arc which rounds off corner n is taken to be tangent to the line between point n and point n-1 at exactly the point where the arc that rounds off corner n-1 is tangent to that line. If the radius at corner n-1 is zero, then the arc which rounds off corner n is taken to be tangent to that line right at corner n-1. See Figure 15 to make sense of this.

Clearly, in order to interpret "join_back", corner n-1 must be rounded before corner n. If corner n-1 also has the radius "join_back", then corner n-2 must be rounded first, and so on. Thus, a sequence of join_back's at successive corners must eventually be stopped at a corner where the radius has been assigned as a number, or at the point 1 in the case of an open defining line.

Join_ahead is analogous to join_back, except that the arc rounding corner n starts at the tangent point of arc n+1 rather than arc n-1. In the case of closed defining lines, if there are N corners, corner 1 is the n+1'th corner with respect to corner N, and corner N is the n-1'th corner with respect to corner 1.

The one additional limitation on the use of join_back and join_ahead is that join_back may never follow join_ahead as a radius. In this case the actual value of the radius for each of them would have to be determined before the other.

Calculation of the actual radii at the corners is handled by the enhancement subsystem. Because of the possible use of join_ahead and join_back, the enhancer needs to be fairly clever about sorting out the generation of corner radii.

The main pitfall of describing defining lines for contour features (as shown in Figure 16) is that between any two corners, n and n+1, the point of tangency of arc n+1 must fall between corner n+1 and the point of tangency of arc n. If the radius of arc n+1 is too big, its point of tangency may fall between corner n and the point of tangency of arc n, or it may lie completely off the line segment between point n and point n+1, making an illegal defining line in either case. This pitfall is particularly easy to fall into when using either of the join options. The enhancer checks for this problem every time it generates tangent points.

Problems may also arise if three successive points are colinear, and the enhancer may complain if it runs into this situation.

If a contour_groove is open, there is no corner radius at the end points. The value "nil" must be assigned to the corner radius at those points. No other corner radius may be nil. The use of nil tells the system when a defining line is open. The VWS Design Editor handles the assignment of radius nil to the endpoints of an open defining line automatically, giving the user no opportunity to do it wrong.

A contour_groove may be made to the left of the defining line by using the optional parameter "offset", and having the value of offset be "left". By setting offset to "right", the contour_groove is made to the right of the defining line. If the offset parameter is not used, the defining line is the center line of the contour_groove. The offset parameter makes it easy to cut a groove around some desired shape. To deal with the offset parameter, the enhancement subsystem, when it is enhancing a contour_groove, first calculates a new defining line, and then enhances the new line.

Figure 14. Defining Line of a Contour Feature

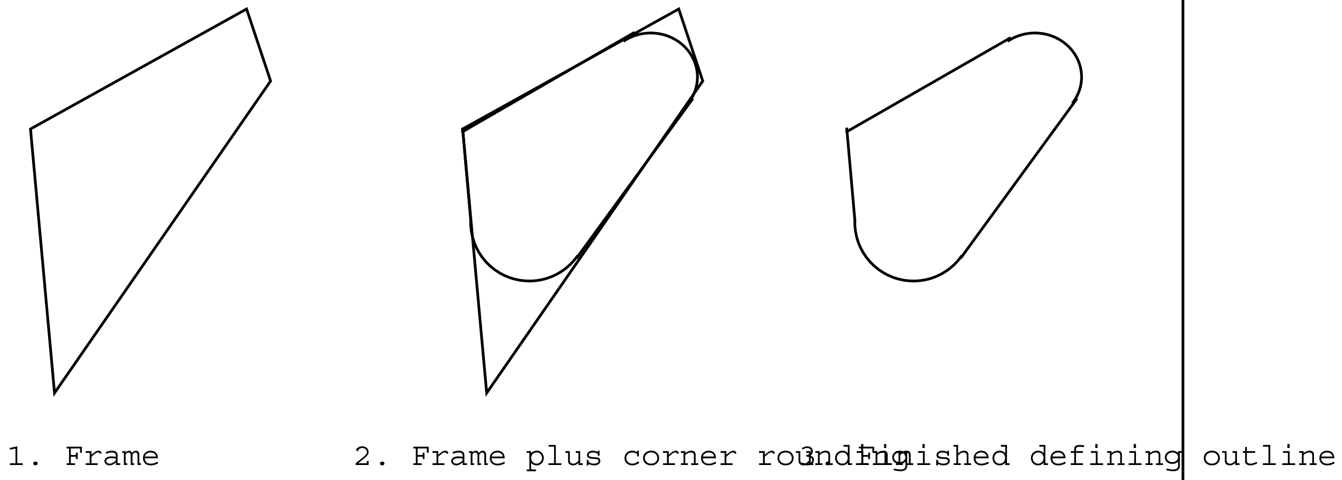
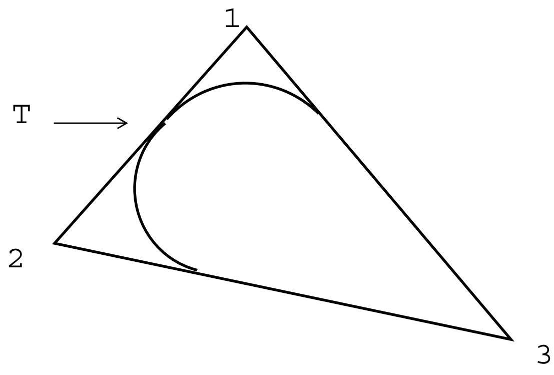
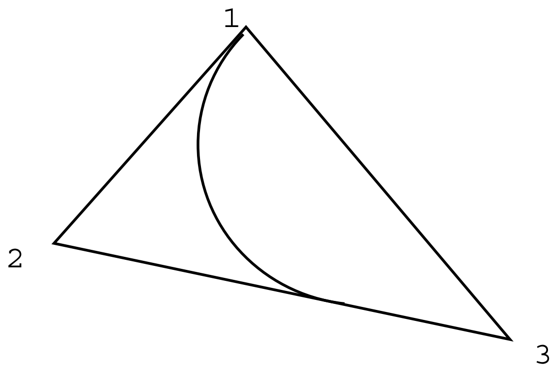


Figure 15. Join_back in a Contour Outline



CASE 1 - Positive rounding at corner 1 and "join_back" radius at corner 2.

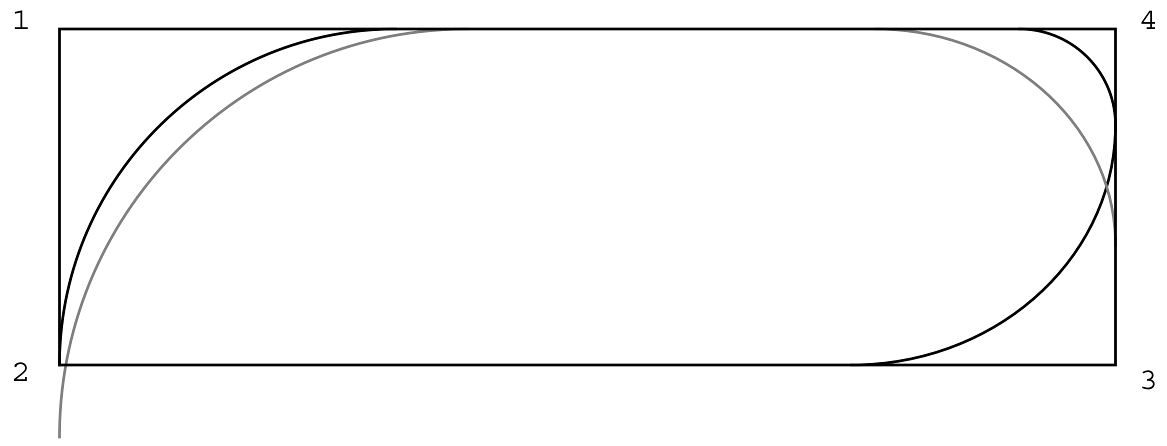
The arc at corner 1 is drawn first, fixing the tangent point T. The arc at corner 2 is fitted tangent at T and also tangent to the line from corner 2 to corner 3.



CASE 2 - No rounding at corner 1 and "join_back" radius at corner 2

The arc at corner 2 is fitted tangent to the line from corner 1 to corner 2 right at corner 1 and is also made tangent to the line from corner 2 to corner 3.

Figure 16. Radius Too Large in a Contour Feature



Suppose that the radius at corner 2 is zero and the radius at corner 3 is drawn before the radius at corner 4.

The solid line in corner 1 shows an arc of the largest radius that can be drawn in that corner. The dotted line at the left of the figure shows an arc whose radius is too large to fit in corner 1.

The solid line in corner 4 shows an arc of the largest radius that can be drawn in that corner. The dotted line at the right of the figure shows an arc whose radius is too large to fit in corner 4.

2.15.8.2. Appearance

The outline of a contour_groove is formed when a tool follows the enhanced defining line as the center line of the cutting path. Where the defining line is straight, the outline is two parallel lines on either side of the defining line. Where the defining line curves, the outline is an arc of larger radius on the outside of the curve and an arc of smaller radius (possibly a single point) on the inside. At any corners of the defining line where the radius is zero, the outline will include arcs whose radius is half the groove width. If a contour_groove crosses over itself, the outline may become arbitrarily complex, but it will be correctly drawn.

The bottom of a contour_groove may be round or flat.

In drawing the profile of a contour_groove, not all the lines which would appear in a silhouette are shown, both to ease the computational and drawing burden and to avoid bewildering the viewer. On the front profile, there are lines showing the maximum and minimum extent of the contour_groove in the x-direction. In addition, if there are straight sections of the groove parallel to the y-axis, the profiles of those sections are shown on the front view. Analogous rules govern drawing the right side view. With this drawing convention, if a contour_groove happens to be the same as an ordinary groove, the pictures of the two will be the same.

2.15.8.3. Required Parameters

feature_type	Must be "contour_groove".
depth	Must be a positive number.
width	Must be a positive number representing the width of the cut.
bottom_type	Must be "flat" or "round"
corners	Conceptually, "corners" is a list of corners, where each corner is represented by a sublist. In the VWS2 system, the concept is realized by having "corners" be a LISP disembodied property list in which the properties are the corner numbers (1, 2, 3, etc.) and the values are disembodied property lists with three properties: x, y, and radius. The values of x and y must be numbers representing the coordinates of the corner. Acceptable values of radius are discussed above. See the examples in section 3.15.8.8. and elsewhere.

2.15.8.4. Optional Subfeatures - None

2.15.8.5. Optional Parameters

offset	Must be "right", "left" or "nil".
reference_feature	Must be a positive integer.

2.15.8.6. Enhancement Parameters

Several parameters are added to the feature description at the top level of the list. In addition, each corner in the set of corners is enhanced.

top level enhancement

z_surf	Set to a number.
tool_diam	A positive number representing the diameter of the tool needed to make the contour_groove. This is the same as the width of the contour_groove unless the groove is round-bottomed and its depth is less than half the width.
max_x	A number representing the maximum value of x reached on the enhanced defining line (i.e. the center line) of the groove.
max_y	A number representing the maximum value of y.
min_x	A number representing the minimum value of x.
min_y	A number representing the minimum value of y.

corner enhancement

radius	The value of a radius which is either "join_back" or "join_ahead" is converted to a number.
x1	Set to the x-value of the corner if the corner radius is zero or nil. Otherwise set to the x-value of the point at which the arc at the corner is tangent to the line between the corner and the preceding corner.
y1	Set to the y-value of the corner if the corner radius is zero or nil. Otherwise set to the y-value of the point at which the arc at the corner is tangent to the line between the corner and the preceding corner.
x2	Set to the x-value of the corner if the corner radius is zero or nil. Otherwise set to the x-value of the point at which the arc at the corner is tangent to the line between the corner and the following corner.
y2	Set to the y-value of the corner if the corner radius is zero or nil. Otherwise set to the y-value of the point at which the arc at the corner is tangent to the line between the corner and the following corner.
max_x	If x reaches a maximum at a point on the interior of the arc at a corner, the value of x is saved as the value of max_x for that corner.
min_x	If x reaches a minimum at a point on the interior of the arc at a corner, the value of x is saved as the value of min_x for that corner.

max_y	If y reaches a maximum at a point on the interior of the arc at a corner, the value of y is saved as the value of max_y for that corner.
min_y	If y reaches a minimum at a point on the interior of the arc at a corner, the value of y is saved as the value of min_y for that corner.
center_x	If the radius at a corner is not zero or nil, the x-value of the center of the arc is saved as the value of center_x.
center_y	If the radius at a corner is not zero or nil, the y-value of the center of the arc is saved as the value of center_y.
turn	Imagine you are sitting at corner n on the point (x,y) and are pointing in the direction you travel to get from corner n-1 to corner n. The number of radians through which you must now turn to point at corner n+1 is the value of turn. The absolute value of turn is always less than pi. Turn is positive for going counterclockwise and negative for going clockwise.
inangle	Inangle at corner n is the counterclockwise angle between a line drawn horizontally to the right at corner n-1 and the line from corner n-1 to corner n.

2.15.8.7. Verification Rules

The x_value of the plus_x side of the contour groove should not be greater than the length of the block minus min_thick.

The x_value of the minus_x side of the contour groove should not be less than the x_value of the minus_x side of the block plus min_thick.

The y_value of the plus_y side of the contour groove should be less than the width of the block minus min_thick.

The y_value of the minus_y side of the contour groove should be greater than min_thick.

The z_value of the top of the contour groove should be greater than 0.0.

The z_value of the bottom of the contour groove should be greater than min_thick.

The tool_diameter of the contour groove should be greater than 0.0624.

2.15.8.8. Example Design

In the following example, feature 1 (the feature on the right in Figure 17) is very simple to describe and is easily entered using the design editor in less than five minutes. Feature 2, which looks like a script "S", required over an hour to perfect because of the use of a long series of "join_back" radii.

Figure 17. Contour_grooves

2.15.9. Contour_pocket

2.15.9.1. Comments

See the comments for contour_groove. There are three additional limitations on the defining line of a contour_pocket beyond those given above. First, the defining points for a contour_pocket must be traversed in the counterclockwise direction (with the inside of the contour on the left). Second, the defining line of the contour_pocket may not intersect itself. Third, no convex (i.e. counterclockwise) corner of the defining line may have zero radius.

In contour_pocket, all the material inside the defining line is removed to constant depth. Thus, the defining line is the outline of a contour_pocket.

The parameters of a contour_pocket are the same as those of a contour_groove, excluding width and bottom_type, which a contour_pocket does not have.

2.15.9.2. Appearance

A flat_bottomed recess with vertical walls whose outline is given by the defining outline.

2.15.9.3. Required Parameters

same as contour_groove, excluding width and bottom_type.

2.15.9.4. Optional Subfeatures - None.

2.15.9.5. Optional Parameters

Note that offset is NOT included.

reference_feature	Must be a positive integer.

2.15.9.6. Enhancement Parameters

same as contour_groove, excluding tool_diam.

2.15.9.7. Verification Rules

The outline of the contour_pocket should not intersect itself.

The x_value of the plus_x side of the contour_pocket should not be greater than the length of the block minus min_thick.

The x_value of the minus_x side of the contour_pocket should not be less than min_thick.

The y_value of the plus_y side of the contour_pocket should not be greater than the width of the block minus min_thick.

The y_value of the minus_y side of the contour_pocket should not be less than min_thick.

The z_value of the top of the contour_pocket should be greater than 0.0.

If the total depth of the contour_pocket is less than the height of the block, then the z_value of the bottom of the contour_pocket should be greater than min_thick.

2.15.9.8. Example Design - See Figure 18.

Figure 18. Contour_pockets

2.15.10. Side_contour

2.15.10.1. Comments

See the comments for contour_groove. There are three additional limitations on the defining line of a side_contour beyond those given for a contour_groove. First, the defining points for a side_contour must be traversed in the clockwise direction (with the inside of the contour on the right). Second, the defining line of a side_contour may not intersect itself. Third, no concave (i.e. counterclockwise) corner of the defining line may have zero radius.

The notion of reference feature is different for side_contour than for the other features. If A and B are both side_contours and B is the reference feature for A, that means that the top of A is at the bottom of B (as usual), but the outline of B must lie entirely inside the outline of A (the opposite of the normal situation). If B is the reference feature for any other type of feature, the outline of the other feature must lie entirely outside of the outline of B (and not enclose B), except that the outline of the island inside a groove may enclose B. Look at Figure 19 if this sounds confusing. A groove and a pocket have been included with three side_contours to help clarify the situation.

Note that if a side_contour has a reference feature, it must be another side_contour.

2.15.10.2. Appearance

A side_contour looks like a plateau with vertical walls whose outline is given by the defining line.

2.15.10.3. Required Parameters

The parameters of a side_contour are the same as those of a contour_groove, excluding width and bottom_type, which a side_contour does not have.

2.15.10.4. Optional Subfeatures - None.

2.15.10.5. Optional Parameters

Note that offset is NOT included.

reference_feature	Must be a positive integer representing the feature number of another side_contour.

2.15.10.6. Enhancement Parameters

same as contour_groove, excluding tool_diam. In addition, a side_contour has the following:

ulx_nc	0.0
uly_nc	A number equal to the width of the block.
lrx_nc	A number equal to the length of the block.
lry_nc	0.0

2.15.10.7. Verification Rules

The outline of the side_contour should not intersect itself.

The x_value of the plus_x side of the side_contour should not be greater than the length of the block plus 0.00001.

The x_value of the minus_x side of the side_contour should not be less than -0.00001.

The y_value of the plus_y side of the side_contour should not be greater than the width of the block plus 0.00001.

The y_value of the minus_y side of the side_contour should not be less than -0.00001.

The z_value of the top of the side_contour should be greater than 0.0.

If the total depth of the side_contour is less than the height of the block, then the z_value of the bottom of the side_contour should be greater than min_thick.

2.15.10.8. Example Design - See Figure 19.

Figure 19. Side_contours

2.16. Text System

2.16.1. Overview

The text system may be thought of as a variable part of the design protocol since it is a simple matter to define and use new fonts. There are currently five fonts in use in the VWS2 system. The same characters are available in all five fonts. The basic font of the system is the "plain" font. All other fonts are derived by applying mixtures of six parametric transformations to the plain font.

2.16.2. Characters

As discussed in section 2.15.7, the text system is based on characters. Each character in each font has a template for making the character stored in the fonts database. The template for making a character consists of two lists: nc_points and nc_path. In the plain font the letter R, for example, has the following nc_points and nc_path. The letter R is shown in Figure 20.

For R, nc_points is: ((0.0 0.0) (0.0 0.5) (0.0 1.0) (0.5 1.0) (0.6666666666666666 0.8333333333333334) (0.6666666666666666 0.6666666666666666) (0.5 0.5) (0.25 0.5) (0.6666666666666666 0.0) (1.0 0.0)))

For R, nc_path is: ((s 3) (s 4) (w 5) (s 6) (w 7) (s 2) (j 8) (s 9))

Each entry on the nc_points list (except the last one) is a pair of numbers which represent the x and y coordinates a point on the character.

The last entry on the nc_points list is the point to go to when the character is finished. The x-value of the last point is always larger than the x-value of the lower right-hand corner of an imaginary parallelogram that just fits around the character by an amount which is the spacing for the particular font. The y-value of the last entry is always zero.

The machining of a character always starts at the first point on the nc_points list.

Each entry on the nc_path is a pair in which the first item represents a type of path to mill to the next point, and the second item tells which point on the nc_points list is the next point.

The letter codes found on the nc_path list mean the following:

s = straight line
w = clockwise arc
ccw = counterclockwise arc
j = jump without milling

The use of nc_points and nc_path is explained most easily by example.

Figure 20. The Letter R

The nc_path for R is interpreted as follows (if will help if you look at Figure 20).

Put a pencil down at the first point on the nc_points list, (0.0 0.0).

Now look at the first entry on the nc_path list, (s 3). It means draw a straight line to the third point on the nc_points list (0.0 1.0).

Now look at the second entry on the nc_path list, (s 4). It means draw straight line to the fourth point (0.5 1.0).

Now look at the third entry on the nc_path list, (w 5). It means draw a clockwise arc to the fifth point (0.6666666666666666 0.8333333333333334).

You should be getting the hang of this by now. Keep going down the nc_path.

Draw a straight line to the sixth point (0.666666666666666 0.666666666666666).

Draw a clockwise arc to the seventh point (0.5 0.5).

Draw a straight line to the second point (0.0 0.5).

Jump to the eighth point (0.25 0.5), and draw a straight line to the ninth point (0.666666666666666 0.0).

This completes the character. Now jump to the last point on the nc_points list (1.0 0.0).

With this system for representing characters, the characters may be translated so that the (0 0) point of the character is at the point (X Y) by adding X to the x-value of every point on the nc_points list and Y to every y-value. The characters may be magnified or shrunk by multiplying the x and y values of every point by a constant. The VWS2 nc-coding and drawing functions apply both of these transformations. The same nc_path is used regardless of how the points have been transformed. When several characters are to be milled, the ending point of each character serves as the starting point for the next.

The plain font used in this system was originally designed by Alton Quist. Modifications were made by the authors. The representation scheme and all software currently in the system which deals with text were developed by the authors.

Adding a character to the default font is feasible. All that is necessary is to add an entry in the "plain" portion of the fonts data base with nc_points and nc_path for the character and add an entry for the character in the "ascii_table" of the fonts database. Any arc in the character, however, must be a quarter circle of radius 1/6 and must be tangent to any connecting arc or straight line at the point of connection. The tangents to the end points of any arc must be vertical or horizontal. If the character is to be transformed, there should be a horizontal separation between arcs that would otherwise be horizontally adjacent, of at least 1/6.

2.16.3. Making New Fonts Automatically

2.16.3.1. Overview

The five fonts currently in the system are: plain, angular, italic, round, and broad. The last four are derivatives of the plain font. It would be possible to make a new font by hand that was unrelated to the plain font, but it would take several days. A new font may be created from the plain font in a minute or so, however, by using the VWS2 automatic font-making facility, "new_font". New_font generates a set of character templates and stores them in the fonts database along with other information about the font being created.

The six aspects of a font which may be changed are:
1. hw_ratio (the height-to-width ratio of an average character such as S),
2. roundness (how much of the character is made up of arcs as opposed to straight lines),
3. curve (the proportional radius of arcs),
4. spacing (the amount of space between characters),
5. tilt (a slanting of the character), and
6. mirror (making a mirror image of the character).

With this selection of aspects an enormous variety of fonts may be created.

The height of the characters in the new font is always 1 (same as the default font). The scale of the text may be varied when the font is used. The functions that deal with text all require the user to specify "height" at some point. This "height" argument specifies the scale. Changing scale is simple enough that it is not necessary to make a new font to do it.

The new_font function has only one required argument. That is the name of the font. The rest of the arguments to new_font are optional and may be thought of as comprising a list that is like a property list. The list includes the names of aspects of the font, each of which (with one exception) is followed by a number which quantifies the aspect.

Each of the aspects listed above is quantified by a single number, except mirror, which has no quantifier. Each aspect may be included or omitted from the arguments, and the order of the aspects is irrelevant. Thus the command (new_font ’f1 ’hw_ratio 3 ’mirror) makes the same font as (new_font ’f1 ’mirror ’hw_ratio 3). In both cases, a mirror image font named "f1" is made which has a height-to-width ratio of 3 for the average character.

Table 5 summarizes the aspects of a font that are used in the new_font command.

Table 5. Summary of Font Aspects

Name	**Bounds of Quantifier**	**Default Value**
mirror	(no quantifier)	unmirrored
hw_ratio	0 < hw_ratio	1.5
roundness	0 <= roundness <=1	2/3
curve	1 <= curve	1
tilt	no bound, but may be limited by roundness	0
spacing	0 < spacing	0.5/hw_ratio

2.16.3.2. Mirror

If ’mirror appears in the argument list, a mirror image font is made. Mirror needs no numerical quantifier. If ’mirror is not in the argument list, the font will not be a mirror font.

2.16.3.3. Hw_ratio

Hw_ratio is the height-to-width ratio of an average character. It is quantified by any positive number. The default characters have an hw-ratio of 3/2 for average characters (seventeen of them). If ’hw_ratio is not in the argument list, or if its quantifier is out of bounds, the hw_ratio of the font will be 3/2.

Characters with an hw_ratio greater than about 2 look skinny; larger than 4 is very skinny. Characters with an hw_ratio less than 1 look wide. Less than 0.5 is very wide.

2.16.3.4. Roundness

Roundness is the proportion of the maximum possible roundness that the font will have. It is quantified by a number between 0.01 and one. If the roundness is one (the maximum), the character 8, for example, will have semicircles at either the right and left sides or the top and bottom, or both, depending on the hw_ratio. If the roundness is 0.01, the characters will be entirely composed of straight lines. At roundness 0.01, 8 is one rectangle on top of another. If ’roundness is not in the argument list, or is not a number, the roundness of the font will be 2/3. If roundness is a number less than 0.01, it will be set to 0.01. If roundness is a number greater than 1, it will be set to 1.

The appearance of roundness depends upon the hw_ratio. A character that is skinny will not look very round even when roundness is at the maximum, because the semicircles at the top and bottom of the rounded characters will be small compared to the straight sides. To get a maximally rounded character, the hw_ratio should be 2. At this ratio the character 8, for example, consists of one circle on top of another (which explains why the ratio is 2).

NOTE -- The descriptions of roundness just given assume that "curve" is set to 1 and "tilt" is zero. Now we will see what happens if they are not set this way.

2.16.3.5. Curve

Curve is the number of times larger than the minimum that the radius of arcs should be. It is quantified by a number greater than or equal to 1. The minimum radius is the radius at which the arcs flow into the straight portions of the characters tangentially. If curve is set to anything greater than 1, the arcs flatten out and there are angles between the arcs and the straight portions. If curve is set to anything greater than 10, the arcs appear to be flat. With curve set to 10, hw_ratio at 2, and roundness at 1, for example, the character 8 looks like one diamond on top of another. If ’curve is not in the argument list, or if its quantifier is out of bounds, the curve of the font will be set to 1.

Fonts with curve set to anything greater than one look angular since corners appear at the end of every arc.

2.16.3.6. Tilt

Tilt is a slanting of the character. It is quantified by any number. If the number is positive, the slant is to the right. If the number is negative, the slant is to the left. The number represents the distance that the top of a character is moved to the right by the slanting. The slant is accomplished in a way that keeps horizontal lines horizontal, and is not a rotation of the character. Adding "mirror" to the definition of a font will reverse the direction of slant. If ’tilt is not in the argument list, or if its quantifier is out of bounds, the tilt of the font will be set to zero.

Characters with tilt set between about .1 and .4 look like italics. A tilt setting greater than about five makes the font almost unreadable.

The amount a character can be tilted is limited by the roundness of the character. When new_font is at work, it will not make the tilt greater than this limit. Thus, the setting of the tilt may be less than requested. In general, the greater the roundness, the less the tilt. Characters of roundness 1 cannot be tilted at all. Characters of roundness zero can be tilted almost any amount (there is actually a limit around 100, but that is well beyond the limit of legibility).

2.16.3.7. Spacing

Spacing is the amount of space, in inches, between characters. It is quantified by a positive number. The spacing of default characters is 1/3, which is half the width of the average character. The space specified between characters is the space that would be there if lines had no thickness. Real lines have thickness, and, as a result, the actual space between characters is less than the value of spacing by the thickness of a line. If thick lines are used, it may be desirable to increase the spacing from the default value. If characters are tilted they tend to merge into one another (again because of line thickness), so larger values of spacing are desirable with tilted characters. If ’spacing is not in the argument list, or if its quantifier is out of bounds, the spacing of the font will be set to half the width of an average character, which is 0.5/hw_ratio.

2.16.4. The Fonts Database

The fonts database will be described in detail elsewhere. Briefly, it is a LISP property list in which all but one of the properties is the name of a font. The value of the each property which is a font name is a sublist, as described below.

The one property which is not a font name is "ascii_table". The value of this property is an equivalence table which gives the character which should be milled that corresponds to the ascii code for each allowable character which may appear in the text string of a text feature. The ascii_table is what the system uses to know that if either "n" or "N" appears in the text string, an "N" should be milled.

The sublist of the "fonts" property list which represents a font has seven properties: mirror, spacing, radius, roundness, tilt, hw_ratio, and characters.

The value of the "characters" property is a list of characters with nc_points and nc_path for each. The values of hw-ratio, tilt, mirror, spacing, and roundness, are as determined by the new_font function. Radius is the actual radius of the arcs of characters. It is assigned by the new_font function through a complex formula involving hw_ratio, curve, and roundness.

The VWS2 system uses the characters, radius, spacing, and tilt information stored with each font. No use is made of the stored mirror, roundness, and hw_ratio information.

The actual values of mirror, spacing, radius, roundness, tilt, and hw_ratio for the five fonts currently in use, along with the new_font command used to generate the non-plain fonts, are shown in Figure 21. Figure 21 uses the actual fonts to display this information.

Figure 21. Aspects of VWS2 Fonts

III. PART DESIGN EDITOR

1. OVERVIEW

The VWS2 Part Design Editor (PDE) is a friendly, computer-aided design system that runs on a Sun microcomputer. It accepts commands from a user and engages the user in dialogs to find out what the user wants and produces an internal data structure representing the design of a part. New designs may be created or old designs edited. The internal data structure is as described in Chapter II of this paper. A schematic diagram of the Design Editor is shown in Figure 22.

PDE runs in a LISP window on the Sun black-and-white monitor when the monitor is being used in the "suntools" multi-window mode. PDE may be called directly from LISP or through the VWS_CADM friendly front end. At the top level, PDE uses a second window to show the user a menu of options. PDE prompts the user to choose an option from the menu by typing in the choice on the keyboard. When an option is chosen, it is either carried out immediately or a routine is entered in which PDE engages the user in a question and answer session, after which the main menu and the prompt reappear. When a drawing is being made, PDE creates a large graphics window in the upper left of the terminal screen and uses it for the drawing of the part being edited.

PDE is largely keyboard driven, using mouse input for only a few things, such as finding the coordinates of a point on the drawing of the part being edited.

A 60-page users manual for PDE is given as Appendix B of [KRA4].

2. CAPABILITIES OF THE SYSTEM

2.1. Save Design Documents

At the user's option, the designs prepared using PDE may be saved as design documents on a disk file system, from where they may be printed out on paper. PDE prepares design documents in two formats, one which is LISP-readable (Table 1, for example), and one which is more easily human-readable (Table 2, for example). The LISP-readable format is not hard for a human to read.

2.2. Automatic Mechanical Drawing

Another option is to have a mechanical drawing of the part being designed drawn automatically on the Sun black-and-white display screen. The drawing is a standard three-view mechanical drawing of the top and front and right sides of the part being designed. Each view is two-dimensional. The drawing is updated whenever a feature is added, deleted, or changed. It usually takes ten seconds to a minute to update the picture of a feature, which is quick enough not to frustrate the user.

The drawing system does not have all capabilities frequently found in mechanical drawing systems. It lacks:

Figure 22. Part Design Editor Module Configuration

1. dimensional labelling on the drawing,
2. tolerance information on the drawing,
3. the possibility of drawing other views, and
4. hidden line removal or representation.

Although the drawing is optional, it is unusual for a user to choose not to have it done. The situation in which it is desirable to leave drawing off is when an existing design is to be changed in a minor way, and the user does not want to wait for the drawing to be done. When PDE starts up, automatic drawing is off. Turning it on is done using a system command from the PDE menu.

2.3. Feature Verification

A third option is to have the features of the design verified. The verifier checks that a feature conforms to the rules for the feature given in Chpater II and checks that if there is a reference feature, the feature being checked fits inside the reference feature. The verification option may be "off", "on hard", or "on soft". Off means no verification. On hard means that if an error is encountered, PDE will delete the feature from the design and (in most cases) return to the top level menu and prompt the user for new input. On soft means that if an error is encountered, the system will ask the user if it should attempt to continue handling the feature and will continue or quit, according to the user’s choice.

It is usually best to keep the verifier "on soft". When PDE starts up, its verification mode is "on soft". In this mode the user usually has the opportunity to see any mistake that has been made and to use the PDE "change" command to correct it after the initial drawing of the feature. Some violations of the rules cause a LISP error that prevents a picture of the feature from being drawn. If PDE runs into one of these, it notifies the user of the problem, deletes the feature, and prompts the user for new input.

3. GENERAL APPROACH

In designing PDE a good deal of thought was given to making it easy to use. Emphasis was also given to making it difficult to generate improper features. This was done without constraining the variety of designs that can be created. The general approach to interacting with the user in PDE is to give the user a free choice among proper choices at each step in the editing process but not to offer improper choices. What the user is allowed to do next depends upon what has been done in the recent past, and some of the trees of choices are both tall and wide.

If the user makes an invalid response at any point, PDE gives brief but informative feedback describing what is wrong with the response and prompts for a new response.

The top level function, pde, simply initializes the editor, and then goes into a read-evaluate loop that stops when the user says quit. At the top level the user is allowed to choose from a menu of 33 items. If the user enters anything readable not on the menu, PDE gives the user brief feedback "invalid pde command" and prompts for a new entry.
Each choice on the menu results either in immediate action or in the user being prompted for more input. At this second level and succeeding levels, the user is either given a menu of acceptable

choices and is constrained to pick of one of the choices on the menu, or is asked to enter data of a particular type at the keyboard. When data is asked for, some preliminary checks of the data type are made. A reference feature, for example must be the number of an existing feature. If the type constraints are violated, the user is notified and asked to enter something else. Entering the number of a non-existent reference feature, for example causes the system to give the feedback:

```
That reference_feature does not exist.
Enter "reference_feature" (numeral/n) ?
```

4. SCREEN LAYOUT

The normal screen layout of PDE for the Sun black-and-white monitor is shown in Figure 23. Three windows are used:
1. The command window (at the lower left)
2. The text window (on the right)
3. The graphics window (at the upper left).

In Figure 23 the text window is displaying the main menu, the command window shows the interactive text of a few steps in an editing session, and the graphics window shows the usual three views of a block with no features on it. The "loc" function has been used, so there is a cross-hair drawn on one of the views of the block, and position information is being displayed in the upper right corner of the graphics window, indicating where the cross-hair is located.

The user must set up the command window and the text window in suntools. This may be done manually with the mouse, or by using a suntools setup file. A LISP process must be brought up in the command window. This may be either a stored LISP environment which includes all the functions and data required by PDE, or a clean LISP into which the functions and data are loaded. In order to cause messages to be sent to the text window, the value of the global LISP variable pp_device must be set to the tty name of the text window, for example by the command (setq pp_device "/dev/ttyp3").

The graphics window appears automatically when the graphic mode is "on" and a design is being edited. It is under control of the LISP process running in the command window. At the same time, however, the graphics window is an ordinary suntools window, so it may be manipulated by the user with the mouse. The most common mouse manipulations are to redisplay it or to change its size for making a paper copy of a design via a screen dump.

The graphics window is formatted automatically after the user has given the dimensions of the block which serves as the starting point for a design. The three views of the block are always in the same relative positions, but the scale and the number of grid squares are varied so that the block takes up as much of the space available for the picture as possible.

Figure 23. Part Design Editor Screen Layout

5. MENU

5.1. Overview

The PDE menu offers the user 33 commands from which to choose. They are in three groups: part design editor commands, system commands, and graphics and verification commands.

The 18 part design editor commands drive the editing of designs, the 5 system commands control the editor itself, and the 10 graphics and verification commands control the graphics and verification subsystems.

In most cases, after the user has selected a menu item, if there are further choices for the user to make, the user will have the option of aborting the menu item rather than having it carried out.

PDE designates at most one design as the currently active design. It is possible to have no design currently active. Any editing operations are automatically applied to the currently active design. Whenever a design is first made to be the currently active design, the first four features in the design are displayed in the PDE text window, if drawing is on, a picture is drawn of the design, and if verification is also on, each feature will be verified before it is drawn.

PDE maintains a list called the "PART_DESIGN_LIST" of designs which are active during the current editing session. Normally, any design which has been edited will be on this list.

PDE assumes that files to be written to disk should go in a subdirectory named "design" of the directory which is currently the host directory for the LISP process in which PDE is running. PDE also assumes that existing designs will be found there. In the design subdirectory there are normally two copies of each design in the directory. One has the suffix ".pd" and is LISP-readable. The other has the suffix ".prt" (which is short for "pretty"), is more easily human-readable, and is not LISP-readable.

PDE maintains a file in the design subdirectory called "part_designs.ndx", which is a sort of index of part designs. This file does not necessarily contain all the designs in the design subdirectory, but only those that a user has asked to have placed in the index. This file is not like most things called indexes. It is simply a list of load instructions, each of which loads one design. The list is not ordered.

In sections 5.2 through 5.4, except as noted, no further user input is required once the menu item is chosen, and the user will be prompted to select another item from the main menu.

5.2. Design Editor Commands

5.2.1. "clp" Clear All Designs

This deletes all entries from the PART_DESIGN_LIST, and if there is a currently active design but no header or features in the design, eliminates the currently active design.

5.2.2. "dpd" Delete a Design

This prompts the user to select a specific design to delete from the PART_DESIGN_LIST. The selected design is deleted from the list. If the selected design is currently active, it is deactivated. The selected design is not removed from disk storage if it has been stored there.

5.2.3. "e" Select a Design to Edit

This displays the entries on the PART_DESIGN_LIST and asks the user to choose one to edit. The chosen design is made to be the currently active design.

5.2.4. "l" Load a Design File

This prompts the user to enter a design name and checks that the design (with the .prt suffix added) is in the "design" subdirectory. If it is there, the design is added to the PART_DESIGN_LIST and made to be the currently active design.

5.2.5. "lff" Load Designs from Index File

This loads all the designs in the file "part_designs.ndx" from the "design" subdirectory into the LISP environment and adds them to the PART_DESIGN_LIST.

5.2.6. "ls" List Designs

This lists the design_id's and the descriptions of all the designs on the PART_DESIGN_LIST. The listing is displayed in the PDE command window.

5.2.7. "n" Rename a Design

This copies the current design into a new data structure and prompts the user for a new design_id and new description. The new design is made the currently active design.

5.2.8. "new" Generate a New Design

This allows the user to generate a new design from scratch. Choosing this option invokes a very large tree of possible options before returning the user to the main menu.

5.2.9. "p" Display the Current Design

This prints the first four features of the current design in the PDE text window. If graphics is on, the existing picture (if any) is removed and the current design is drawn.

5.2.10. "save" Save the Current Design to Disk

This writes the current design to two disk files, as described in section 5.1.

5.2.11. "feat" Display Design Features

This prompts the user for a feature number and prints four features of the current design in the PDE text window, starting with the feature number entered by the user.

5.2.12. "stf" Store Designs to Index File

This overwrites the part_designs.ndx file in the design subdirectory so that the file will cause only the designs on the PART_DESIGN_LIST to be loaded.

5.2.13. "c" Change the Design

This allows the user to change a feature or the header of the currently active design. It causes PDE to enter into a substantial dialog with the user.

5.2.14. "d" Delete a Feature

This prompts the user to enter a feature number. If the number is the number of a feature in the currently active design, that feature is deleted from the design (and from the picture of the design, if there is one).

5.2.15. "i" Insert a Feature

This allows the user to insert a new feature in the currently active design. It causes PDE to enter into a substantial dialog with the user.

5.2.16. "array" Make a Feature Pattern

This allows the user to generate an array of features identical to an existing feature except for location. It causes PDE to enter into a substantial dialog with the user. Any feature may be duplicated in an array, but it makes little sense to have an array of side_contours or chamfer_outs. The array may be either rectangular or circular and there are several options within each kind.

When "array" is used, the new features are inserted immediately after the feature being duplicated. After the insertions, the design is resequenced.

5.2.17. "group" Duplicate a Group of Features

This allows the user to duplicate a group of features (such as a pocket with a set of holes at the bottom of the pocket) in another location. It causes PDE to enter into a substantial dialog with the user. It is sensible in its treatment of reference features. Any feature may be duplicated in a group, but it makes little sense to repeat the chamfer_out feature.

When "group" is used, the new features are inserted immediately after the largest feature number in the group. After the insertions, the design is resequenced.

5.2.18. "rseq" Resequence Design Features

If features are deleted from a design, gaps in the sequence of feature numbers may appear. This command resequences the feature numbers so that they once again are 1, 2, 3, etc. In doing this, it often has to change reference feature numbers, as well.

5.3. System Commands

5.3.1. "b" Break

This command puts the user into the LISP command interpreter, where any valid LISP command may be given. To return to PDE, the user enters "?ret".

5.3.2. "cls" Clear this Screen

This clears the PDE command screen.

5.3.3. "elem" Print a List of Features

This prints a list of feature types known to PDE.

5.3.4. "m" Print a List of Commands

This displays the main menu in the PDE text window.

5.3.5. "q" Quit from Part Design Editor

This gets the user out of the Part Design Editor and back to wherever the editor was called from (usually either LISP or VWS_CADM). Since the editor is no longer running, there is no prompt from the editor. If the editor has drawn a picture, it will disappear. The command window is cleared, but the text window is not cleared.

5.4. Graphics and Verification Commands

5.4.1. "block" Draw the Block with no Features

This erases any existing picture, and, if there is a currently active design, draws a picture of the block which is the starting point for the design, but without any features. This command is available so that the user can draw a picture of any subset of the features in a design. Such a picture is created by first calling this command, and then using the "draw" command as often as desired.

5.4.2. "draw" Draw a Feature

If there is a picture, this command prompts the user to enter the number of the feature to be drawn and draws the picture of that feature. If verification is on, the feature is verified first. This command does nothing except print a message if there is no picture on the screen.

5.4.3. "flash" Flash a Feature

This command prompts the user to enter the number of the feature that should be flashed. If a picture of that feature is on the screen, the picture is turned off and on three times. This command is used to identify the picture of a given feature.

5.4.4. "goff" Set the Graphic Mode to OFF

This sets the PDE graphics mode to "off" and makes any existing picture disappear. When the graphics mode is "off", no picture will be drawn if a new design is made to be the currently active design, and commands that otherwise will have some effect on a picture will not have those effects.

5.4.5. "gon" Set the Graphic Mode to ON

This sets the PDE graphics mode to "on". It does not automatically draw a picture of the currently active design; the "p" command will do this once graphics is "on". Other commands which have graphics effects will have those effects enabled when the graphics mode is "on".

5.4.6. "loc" Use Mouse to Pick a Position

If there is a picture, this command prompts the user to use the mouse to point at the picture and press the right mouse button. When the right button is pressed, a cross-hair is displayed at the position of the mouse cursor, and the location of the point picked is shown in the upper right corner of the picture, given in part coordinates. Position information is rounded off to the nearest eightieth of an inch.

The user may then locate another point by moving the mouse and pressing the right button, or may get out of "loc" by pressing the middle or left button. The position information and the crosshair remain on the screen.

5.4.7. "pick" Use Mouse to Pick a Feature

If there is a picture, this command prompts the user to use the mouse to point at a feature on the picture and press the left button. When the left button is pressed, the number of the feature is displayed on the command screen. If the pick fails the user is offered several options.

5.4.8. "rdraw" Redraw the Screen

This redisplays the picture, if there is one. This command produces a better picture than if the picture is redisplayed via suntools, since "rdraw" redisplays pictures of features and masks in the correct order. This command also deletes any pictures of fonts and any position information that may have been drawn during the process of making a text feature or using the "loc" command. Because "rdraw" simply turns existing images off and on, it is much faster than a call to "p", which regenerates all the images.

5.4.9. "verif" Verify a Design

The entire currently active design is verified. The user is notified feature by feature whether the feature passes or fails verification. If any feature fails, a list of all the features that failed is presented to the user at the end.

5.4.10. "vset" Set the Verification Mode

The user is prompted to choose one of the three verification modes, "off", "on_soft", or "on_hard", from a menu in the PDE text window. The PDE verification mode is set to the chosen level.

6. SOFTWARE

The software for PDE, like all the VWS2 software, is written in Franz LISP. About 70 functions, gathered in the "design2" subdirectory of the usr2/kramer/vws2 directory, make up PDE. The functions in this subdirectory are solely for the use of PDE. If PDE is run without graphics or verification, a few additional functions are needed from other directories. When graphics and verification are operating, many of the 160 functions in those subsystems may additionally be required.

One of the PDE files, "pde_plist", does not define a function, but rather establishes a large property list in which the properties are the names of commands from the PDE main menu, and the values are sections of LISP code which are executed when the corresponding command is chosen by a user. Thus, this file is equivalent to an additional 33 functions.

PDE makes use of a dozen or so global variables, unlike much of the rest of the VWS2 software, in which global variables are avoided.

The PDE software makes extensive use of the LISP "errset" function to catch errors and handle them intelligently before the errors can propagate, throw the user out of PDE, or corrupt the system.

7. ADVANTAGES AND DISADVANTAGES

PDE allows a user to create or change a design quickly and easily. With PDE it is easy to create a design which conforms to all requirements of the VWS2 design protocol. The authors have tried occasionally to edit a design using a general purpose text editor and have found that design errors are nearly unavoidable in this circumstance. Thus PDE provides the only reasonable way to generate designs which the VWS2 system can use.

The graphics and verification capabilities incorporated in PDE are powerful tools for insuring that the design is correct and is what the user intends. Both subsystems operate quickly enough that the user is unlikely to become impatient with the system.

PDE is fun to use, so that the user is likely to enjoy working with it. This is a critical factor since any system which is unpleasant to use is not likely to be used effectively. PDE is not tiring.

PDE is easy to learn to use. A person familiar with computers could learn to use all the capabilities of PDE effectively in a few afternoons of personal instruction. A tutorial program to help a user to learn the system without personal instruction is being prepared. A user unfamiliar with computers but familiar with machining (a machinist, for example) should be able to learn the system only slightly more slowly.

Aside from the limitations of the design protocol itself, which were discussed in section 2.5 of Chapter II, the only known significant disadvantage of PDE is the limitations of the drawing system, particularly its inability to show feature intersections. It is also a drawback not to have the option of viewing a three-dimensional picture of a design. The blocks to adding these capabilities to the drawing system are: (1) the time required to write the software, and (2) the likelihood that these drawing capabilities would slow the system down unacceptably.

Users familiar with systems that use icons and mouse input heavily might find the almost exclusive reliance on keyboard input a cumbersome feature of PDE. In future versions of PDE (if any are written), it would be nice to give the user a choice of keyboard or mouse input.

IV. <u>GEOMETRY LIBRARY</u>

1. OVERVIEW

1.1. <u>Introduction</u>

The VWS geometry library consists of 65 functions which perform geometric calculations in the VWS2 system. Some 21 of these, such as "pi_times", are elementary functions that are not available in Franz Lisp. Fifteen of them operate solely when a feature is being enhanced, and another 27 have to do with manipulations of contour features.

These functions have been gathered into a single directory (~/vws2/geom2) because most serve the purposes of more than one subsystem. The drawing, nc-coding, and verification subsystems, for example, all make use of the enhanced version of a feature.

Many of the contour manipulating functions serve only the verification subsystem, but rather than split the contour feature functions into two groups, they have all been gathered together in the geom2 directory.

None of the functions in the geometry library is intended for use directly by a user of the VWS2 system. Rather, the functions are called automatically as needed by the Design Editor, the VWS control system, or the VWS_CADM friendly front end. A knowledgeable user, however, might use some functions in the library directly, and the elementary functions are available to any programmer doing geometric calculations for inclusion in some other system.

1.2. <u>Significant Figures</u>

Franz Lisp on a Sun computer automatically performs calculations giving 16 significant decimal figures, for example 2.000000000000001, which has an integer and fifteen places after the decimal point. This is slightly better than adequate for the purposes of the VWS2 system. Roughly speaking, in order to ensure accuracy to 0.0001 inch, which is about the limit of accuracy of a good milling machine, calculations should be accurate to five decimal places.

Since part dimensions are in inches, never are more than 99, and usually are less than 10, only one or two significant figures are needed for the integer portion of numbers representing part dimensions. Angles are usually in the range of zero to two pi, so the integer portion requires only one digit. Thus there are usually 14 or 15 decimal places available to express numbers.

Some algorithms require two more decimal places to prepare for contingencies, and about half a dozen decimal places may be lost in certain inverse functions (such as inverse sine) employed by the system. That makes a total of about 13 decimal places that are important for calculation, two fewer than come automatically with Franz Lisp on a Sun computer. Thus, no special computational algorithms have been needed to ensure sufficient accuracy. It is likely that a few special routines would have to be written to adapt the VWS2 system to a computer maintaining accuracy to significantly fewer decimal figures -- but not many. A precise analysis of this has not been done because it has not been necessary.

1.3. Colinearity and Tangency

Two computational gremlins, colinearity and tangency, have had to be accounted for repeatedly in the geometry library, the verification subsystem, and the nc-coding subsystem.

The appearance of these gremlins in the feature verifiers is typical. For example, suppose, as is shown in Figure 24, that one pocket is made inside another. The bottom of the inner pocket may lie directly over a section of the bottom of the outer pocket, as shown. Depending upon the mood of the roundoff algorithm, and regardless of how many decimal places of accuracy are kept, a computational algorithm for determining if line AD intersects line BC may show no intersection, may find intersection at one point, or may show what the eye sees, namely that BC lies on top of AD all the way from B to C.

At point B the tangency problem arises. A calculation of whether circular arc EB intersects line AD is apt to give no intersection, one point, or two points in the same place, again depending on the vagaries of roundoff.

The approach taken to these gremlins in the VWS2 system is to try to identify when either of the two might arise, and either: 1. Shrink or expand one of the figures in question by 0.00001 before performing the calculation, or 2. When checking for equality of two numbers in certain situations, check instead whether the two numbers are within a certain tolerance of one another (where the tolerance may be one- or two-sided, depending on the situation).

Both ploys are physically irrelevant but mathematically helpful. For example, if the inner pocket of Figure 24 is shrunk by 0.00001, the nc-code generated to make the pocket will probably be unaltered, since only four decimal places are used in the nc-code. On the other hand, any decent algorithm for determining if AD intersects BC will show no intersection, and a calculation of whether arc EB intersects line AD will show no intersection. This is the situation that is desired since we would like to permit close fits of this sort.

The shrinking algorithms vary with the type of feature. In the case of contour features, the shrinking algorithm is complex.

2. ELEMENTARY FUNCTIONS

The 21 elementary functions include three for dealing with pi, four for tolerances, and a few miscellaneous. Most of the rest are for making basic geometric determinations, such as the distance between two points, or if and where one straight or curved line segment intersects another.

3. FEATURE ENHANCERS

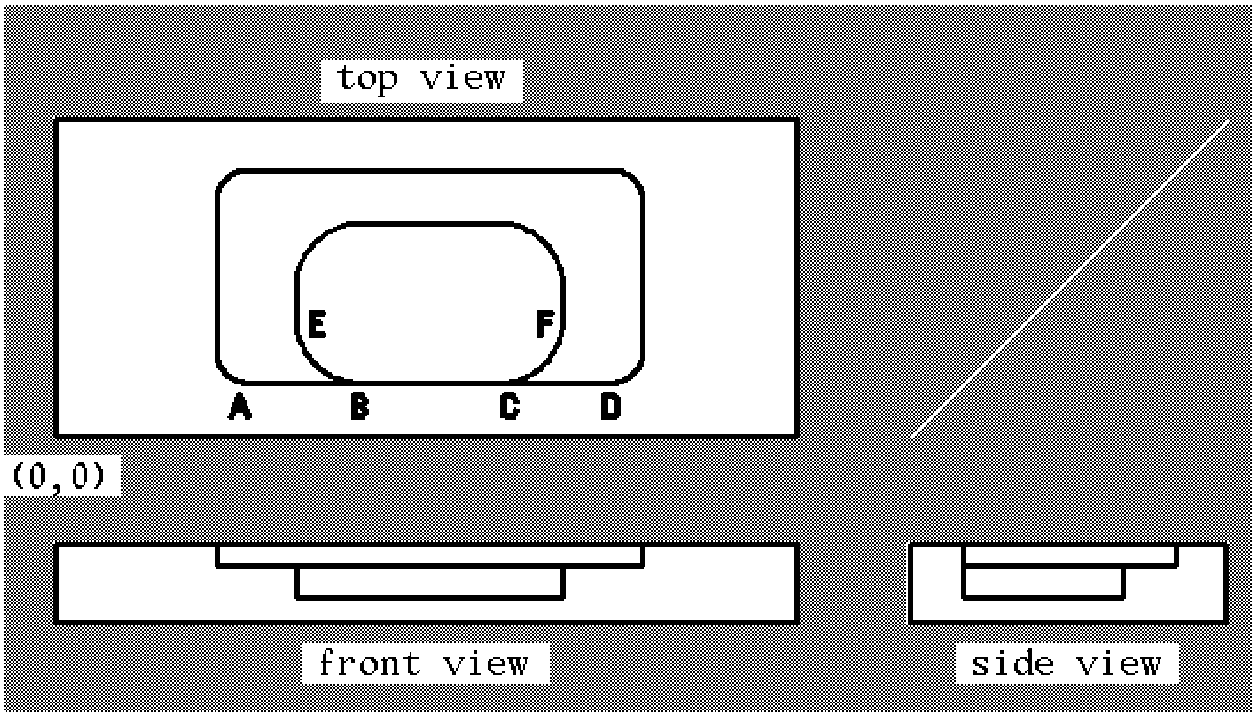
The 15 feature enhancers are called into use only to produce the enhanced version of a feature description. At the top of the hierarchy of these functions is "enhance_design", with "enhance_feature", which is the real workhorse, immediately below it. Enhance_feature makes a copy of the unenhanced feature, finds the z_surf and then consults the features data base to find the proper enhancer for the given feature type. The enhancer which is selected alters the copy in place, and returns it to enhance_design, which collects all the enhanced features together.

Eight of the enhancers deal with contour features, and might equally well have been included with the set of contour functions described next.

4. CONTOUR FUNCTIONS

The 27 contour functions in the geometry library include 6 for producing "virtual" contour outlines used in the verification subsystem, 8 for determining the intersection of contour features, 7 for generating new contour outlines from old, and 6 miscellaneous.

Figure 24. Colinearity and Tangency



REFERENCES

[HOPP]
Hopp, Ted; AMRF Database Report Format -- Part Model; AMRF report; October 22, 1986.

[JUN]
Jun, Jau-Shi; "The Vertical Machining Workstation Systems"; to be published as an NBSIR; 1988.

[KRA1]
Kramer, Thomas R.; "Process Planning for a Milling Machine from a Feature-Based Design"; Proceedings of Manufacturing International Meeting; Atlanta, Georgia; April 1988; ASME; 1988; Vol. III, pp. 179 -189.

[KRA2]
Kramer, Thomas R.; "The Graphics Subsystem of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; to be published as an NBSIR; 1988.

[KRA3]
Kramer, Thomas R.; "Data Handling in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3763; National Bureau of Standards; 1988; 62 pages.

[KRA4]
Kramer, Thomas R.; "The vws_cadm User Interface in the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3738; National Bureau of Standards; 1988; 110 pages.

[KRA5]
Kramer, Thomas R.; "Process Plan Expression, Generation, and Enhancement for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3678; National Bureau of Standards; 1987; 56 pages.

[KR&J]
Kramer, Thomas R.; and Jun, Jau-Shi; "Software for an Automated Machining Workstation"; Proceedings of the 1986 International Machine Tool Technical Conference; Chicago; Illinois; September 1986; pp. 12-9 through 12-44.

[K&S1]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention in Data Preparation for a Numerically Controlled Milling Machine"; Proceedings of 1987 ASME Annual Meeting; ASME; 1987; PED-Vol. 25, pp. 195 - 213.

[K&S2]
Kramer, Thomas R.; and Strayer, W. Timothy; "Error Prevention and Detection in Data Preparation for the Vertical Workstation Milling Machine in the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 87-3677; National Bureau of Standards; 1987; 61 pages.

[KR&W]
Kramer, Thomas R., and Weaver, Rebecca E.; "The Data Execution Module of the Vertical Workstation of the Automated Manufacturing Research Facility at the National Bureau of Standards"; NBSIR 88-3704; National Bureau of Standards; 1988; 58 pages.

[LOVE]
Lovett, Denver; "Equipment Controllers of the Vertical Workstation"; NBSIR 88-3769; National Bureau of Standards; 1988.

[NA&J]
Nakpalohpo, Ibrahim; and Jun, Jau-Shi; "Automated Equipment Program Generator and Execution System of the AMRF Vertical Workstation"; not yet published; 17 pages.

[REQU]
Requicha, A. A. G.; "Representations of Solid Objects"; Lecture Notes in Computer Science; Goos, G. and Hartmanis, J.; New York; Springer-Verlag; 1980, pp. 2 - 78.

[RUDD]
Rudder, Frederick; (a paper in preparation describing the VWS hardware and software for the HP-9000 workstation supervisor), to be published as an NBSIR; 1988.