Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

AUTOMATIC FEATURE EXTRACTION FROM CONVENTIONAL CAD MODEL TO SUPPORT FEATURE-BASED DESIGN APPROACH FOR THE SHEET METAL STAMPING INDUSTRIES

by

Dipak P. Thakar

Despite the continuing improvement in computer aided design (CAD) systems and improvements in computer aided manufacturing (CAM), the process planning activity has still not been completely integrated into the CAD/CAM cycle. Particularly in sheet metal stamping industries human interpretation of CAD data is required to extract the geometry and technological information of a component. As a result most CAD systems are used as advanced drafting and drawing management tools by designers. Thus the responsibility for interpreting the design data required for extracting the manufacturing part features still resides with the process planner. Which has increase possibilities of entering errors with design data. A need, therefore, exists to develop expert system for automatic features extraction from a CAD database. An application software was developed for automatic feature extraction from conventional CAD model database to impliment feature-based design approach for the sheet metal stamping industries.

Key word: CAD, CAM, Feature, Feature-based design, Feature-based process planning.

AUTOMATIC FEATURE EXTRACTION FROM CONVENTIONAL CAD MODEL TO SUPPORT FEATURE-BASED DESIGN APPROACH FOR THE SHEET METAL STAMPING INDUSTRIES

by
Dipak P. Thakar

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Manufacturing Systems Engineering

Manufacturing Engineering Division

May 1994

APPROVAL PAGE

AUTOMATIC FEATURE EXTRACTION FROM CONVENTIONAL CAD MODEL TO SUPPORT FEATURE-BASED DESIGN APPROACH FOR THE SHEET METAL STAMPING INDUSTRIES

Dipak P. Thakar

Dr. Nouri Levy, Thesis Advisor — Date
Associate Professor of Mechanical Engineering, NJIT

Dr. Raj Sodhi, Committee Member — Date
Director of Manufacturing Engineering Programs
and Associate Professor of Mechanical Engineering, NJIT

Dr. Meng-Chu Zhou, Committee Member — Date
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Dipak P. Thakar

Degree: Master of Science in Manufacturing Systems Engineering

Date: May 1994

Undergraduate and Graduate Education:

- Master of Science in Manufacturing Systems Engineering
  New Jersey Institute of Technology
  Newark, New Jersey, 1994

- Master of Science in Mechanical Engineering
  The M. S. University of Baroda
  Baroda, India, 1988

- Bachelor of Science in Mechanical Engineering
  The M. S. University of Baroda
  Baroda, India, 1984

Major: Manufacturing Engineering

This thesis is dedicated to
my parents, wife and other family members.

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his advisor, Dr. Nouri Levy, for his guidance, constant encouragement, and moral support throughout the cource of the thesis.

Special thanks to Dr. Raj Sodhi and Dr. Meng-Chu Zhou for serving as members of the committee.

The author is grateful to professor R. Reddy, for his expert assistance at various stages of software development.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Features have become a popular way of modeling the geometry of engineering components because they allow engineering significance and engineering data to be associated with geometry. Feature-based CAD/CAM systems have demonstrated some potential in creating attractive design environments and in automating geometric reasoning related to design function, performance evaluation, manufacturing and inspection process planning, NC programming, and other engineering tasks. Feature-based design is regarded as a key factor towards CAD/CAM integration from process planning point of view. From a design point of view, feature-based design offers possibilities for supporting the design process better than current CAD systems do. Feature technology, therefore, is expected to be able to provide for a better approach to integrate design and applications following design such as engineering analysis, process planning, and inspection.

There have been several different approaches to associating the engineering significant with a geometric model. Usually, the engineering information are attached to the faces of a boundary representation (B-rep) model or to the primitives of a constructive solid geometry (CSG) model. Several groups of researchers in the fields of information modeling and artificial intelligence have suggested better methods for representing different aspects of the design information in CAD systems. A result of this effort has been the development of intelligent CAD systems that can reason about designs. The intelligence of a CAD system can be measured by the system's ability to understand higher-level concepts and to execute tasks defined in term of these concepts. However, purely geometric representation using the available solid modeling programs are unable to provide the information

necessary for reasoning about the nongeometric aspects of design. Research in the area of feature has resulted in many promising techniques for combining engineering data and knowledge with geometric information. Good reviews on research in feature-based design are found in Reference 4.

Almost all the research on feature has been in the domain of mechanical design for machining components, largely because the primary goal of the mechanical CAD systems has been to provide concise, accurate representations of mechanical components along with their corresponding machining process. Furthermore, the need for integrated design environment and automated information processing techniques for complex mechanical designs has prompted extensive research for better representation of geometry and topology of both completed and in-process designed components. During past few years a growing number of researchers has investigated integration and automation issues for mechanical component design problems. However, most of the current research has concentrated on various machining processes such as turning operations, and milling operations without much emphasis on press working techniques, which are mainly used in the sheet metal industries (References 1, 6, 9, and 13).

In this thesis we propose a feature-based design approach for the sheet metal stamping industries, and developed program for automatic feature extraction from conventional CAD model to integrate CAD/CAM. This requires reading and interpreting conventional CAD design database information and generating output file in terms of feature list.

Such system can be considered as a step forward in the direction of implementing feature-based design approach into the sheet metal stamping industries and achieving CAD/CAM integration. This software was developed in C programming language running under UNIX[TM-1] based computer system.

---

1. UNIX is a registered trademark of AT & T Bell Labs.

CHAPTER 2

FEATURE FUNDAMENTALS

2.1 Definition of a feature

During the past several years many researchers have proposed using features as a natural form of communication among designers, analysts, and manufacturers about the topology and geometry of designed artifacts. Definitions proposed by many researchers tend to be either very general or very specific (References 2, 3, 4, 5, and 8). Examples of these more general definitions of features are:

- A prominent part or characteristic of an entity of interest.
- Any entity used in reasoning of design, engineering, and manufacturing.
- Recurring patterns of information related to a part description.
- A stigmatic grouping used to describe a part and its assembly. It groups in a relevant manner functional, design and manufacturing information.
- A geometric form or entity whose presence or dimensions are required to perform at least one Computer Integrated Manufacturing (CIM) function and whose availability as a primitive permits the design process occurs.
- A carrier of product information between other engineering and manufacturing.
- A region of interest.
- Any named entity with attributes of both form and function.
- A set of faces grouped together to perform a functional purpose.

- A physical constituent of a component, be mappable to a generic shape, have engineering significance, and predictable properties.

The more specific definitions usually deal with interaction between boundary faces and apply to limited domains, such as machining. Features were first introduced for combining machining information with solid models of mechanical components. The most commonly known type of features used for this purpose are form feature. Examples of some specific definitions of process planning related form features are:

- A specific geometric configuration formed on the surface, edge or corner of a workpiece intended to modify or to aid in achieving a given function.
- A distinctive or characteristic part of a workpiece, defining a geometrical shape, which is either specific for a machining process or can be used for fixturing and/or measuring purposes.
- A stereotypical portion of a shape or a portion of a shape that fits a pattern or stereotype.
- Any geometric form or entity uniquely defined by its boundaries, or any uniquely defined geometric attribute of a part that is meaningful to any life cycle issue.

Thus, there is no universally accepted definition and representation for object features. However, it is generally agreed that features are generic shape with the engineering significance. Features also have attributes that deal with the function they serve in the component. In general, feature representation and classification are domain-specific. As the general definitions implies, different users of a model will

see different aspects of the object as the features of that objects. For material removal processes the cavities being removed from the raw stock. For manufacturability analysis of extrusions the features can be represented simply as the walls and intersections of the extruded shape. Thus, a feature-based model is not unique. The same component can be described with different sets of features depending on the designers' and clients' purposes and point of view. Figure 2.1 shows an example component in terms of its shape features for machining component and Figure 2.2 shows an example of its shape features for sheet metal stamping component. The shape of feature may be expressed in terms of dimension parameters, enumeration of geomentric/topological entities and geometric/ topological relations between composing entities, or in terms of a geometry construction procedure. The engineering meaning may involve the formalization of the function the feature serves, or how it can be produced, or what actions must be taken in the presence of this feature if one is performing some kind of evaluation, or how the feature behaves in various situations, and so on.

2.2 Feature Types

Since features are application-dependent, the types of features must bedependent on the domain of application. A lot of different types of features have been proposed. Some feature types are:

- Form feature — Portions of nominal geometry; recurring shapes
- Precision features — Deviations from nominal form/size/ location(tolerances, finish)

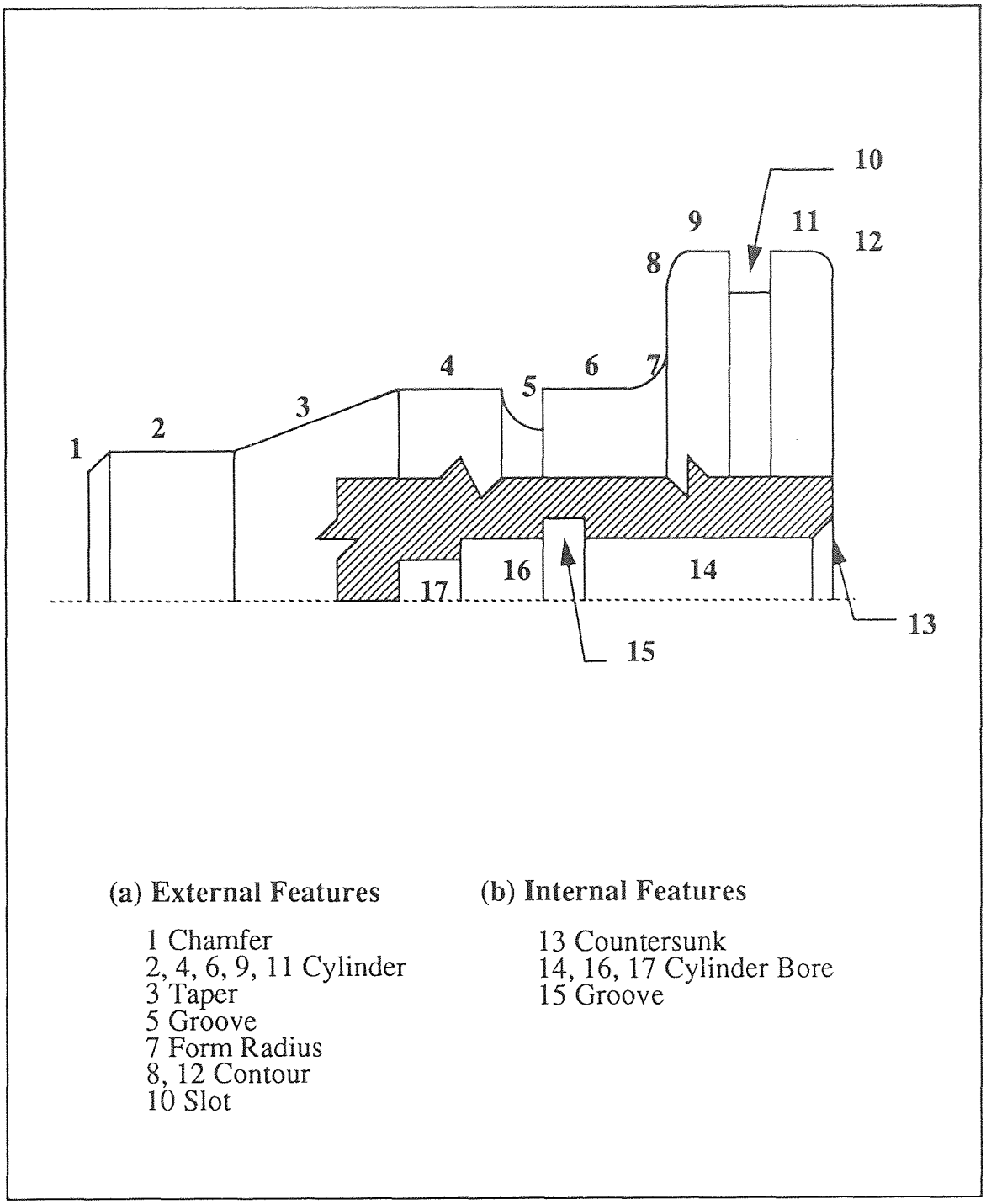
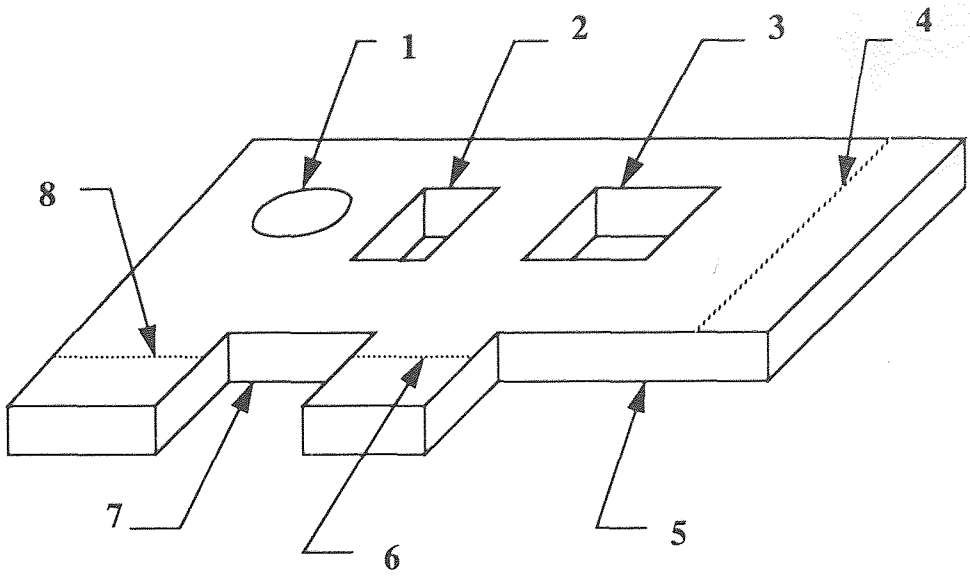


Figure 2.1

Example of external and internal features for machining components



(a) Inner Features

1 Hole
2, 3 Block
4, 6, 8 Bend

(b) Outer Features

5, 7 Block

Figure 2.2

Example of inner and outer features for sheet metal stamping comonents

- Technological features — Nongeometric parameters related to function, performance, and so on
- Material features — Material composition, treatment, condition, and so on
- Assembly features — Part relative orientations, interaction surfaces, fits, kinematic relations

The number of features in not finite but it may be possible to categorize features into group or classes. Several schemes have been proposed for classification based entirely on shape, rather than the application.

A scheme was developed by Pratt and Wilson for CAM-I and adopted for the form features information model of Product Data Exchange Specifications (PDES). PDES classified features as follows:

- Passages — Subtracted volumes that intersect the preexisting shape at both ends
- Depressions — Subtracted volumes that intersect the preexisting shape at one end
- Protrusions — Added volumes that intersect the preexisting shape at one end
- Transitions — Regions involved in smoothing of intersection regions
- Area features — Dimensionality two elements defined on faces of preexisting shape
- Deformations — Shape changing operations such as bending and stretching

Other schemes were developed by Kang and Nnaji (Reference 5) for Mechanical assembly and Sheet metal fabrication based on a face-oriented feature. A generalized feature classification systems are presented by Shah, J. J (Reference 7) for mechanical assembly and sheet metal fabrication applications are as shown in Figure 2.3 and Figure 2.4 respectively.

2.3 Feature Properties

There are growing consensus about some of the properties that features should have. A good feature implementation should support multiple functional views and allow features to aid in the design and engineering process. From a survey of features used in various applications, the following list of feature properties has been compiled:

- Generic shape
- Dimension parameters
- Location method
- Location parameters
- Orientation method
- Orientation parameters
- Tolerances
- Construction procedure for geometric model
- Recognition algorithm, if applicable
- Validation rules
- Parameters inherited from other features
- Inherited rules and procedures
- Nongeometric attributes

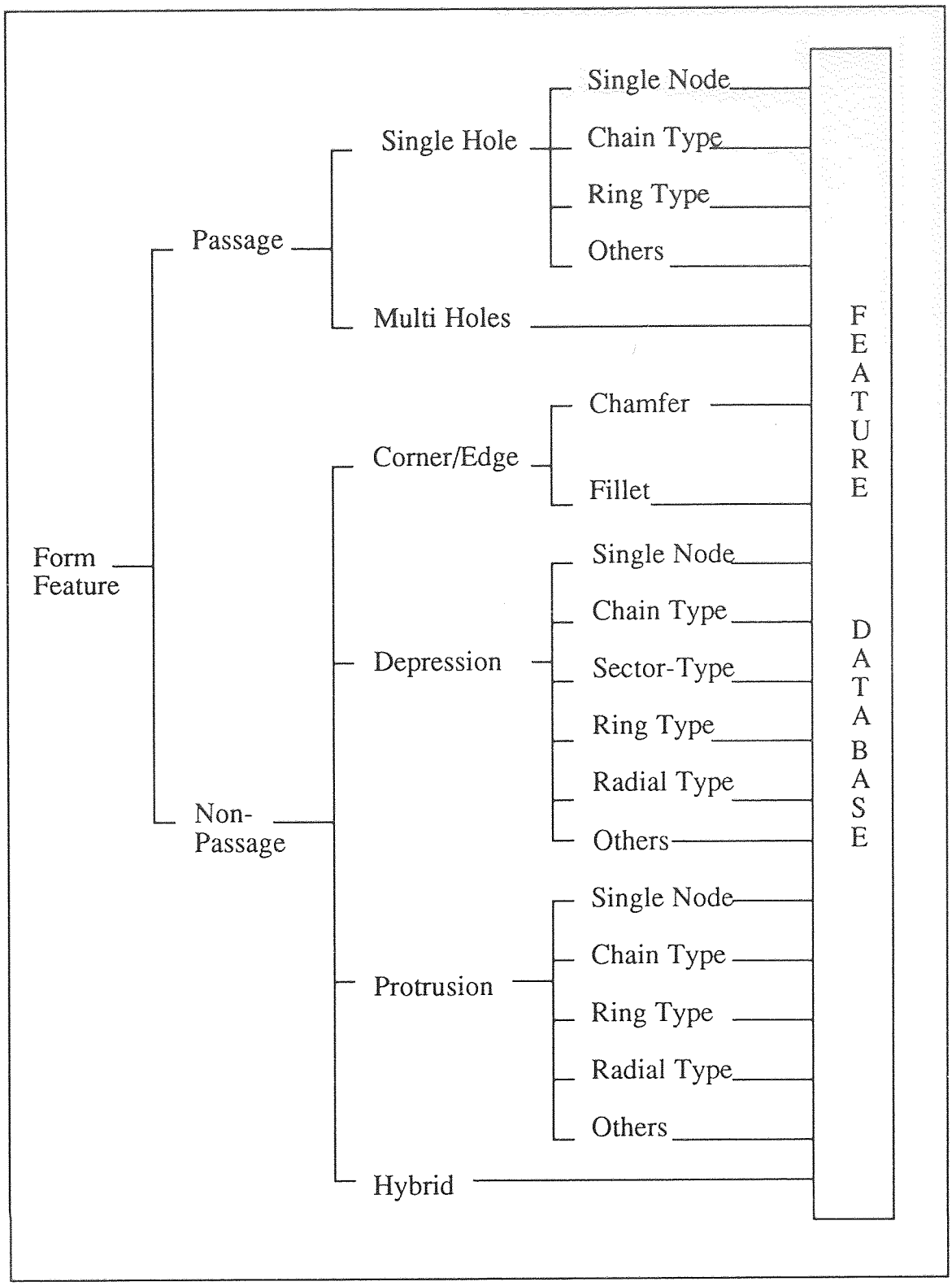


Figure 2.3

Generalized system for features classification for mechanical assembly

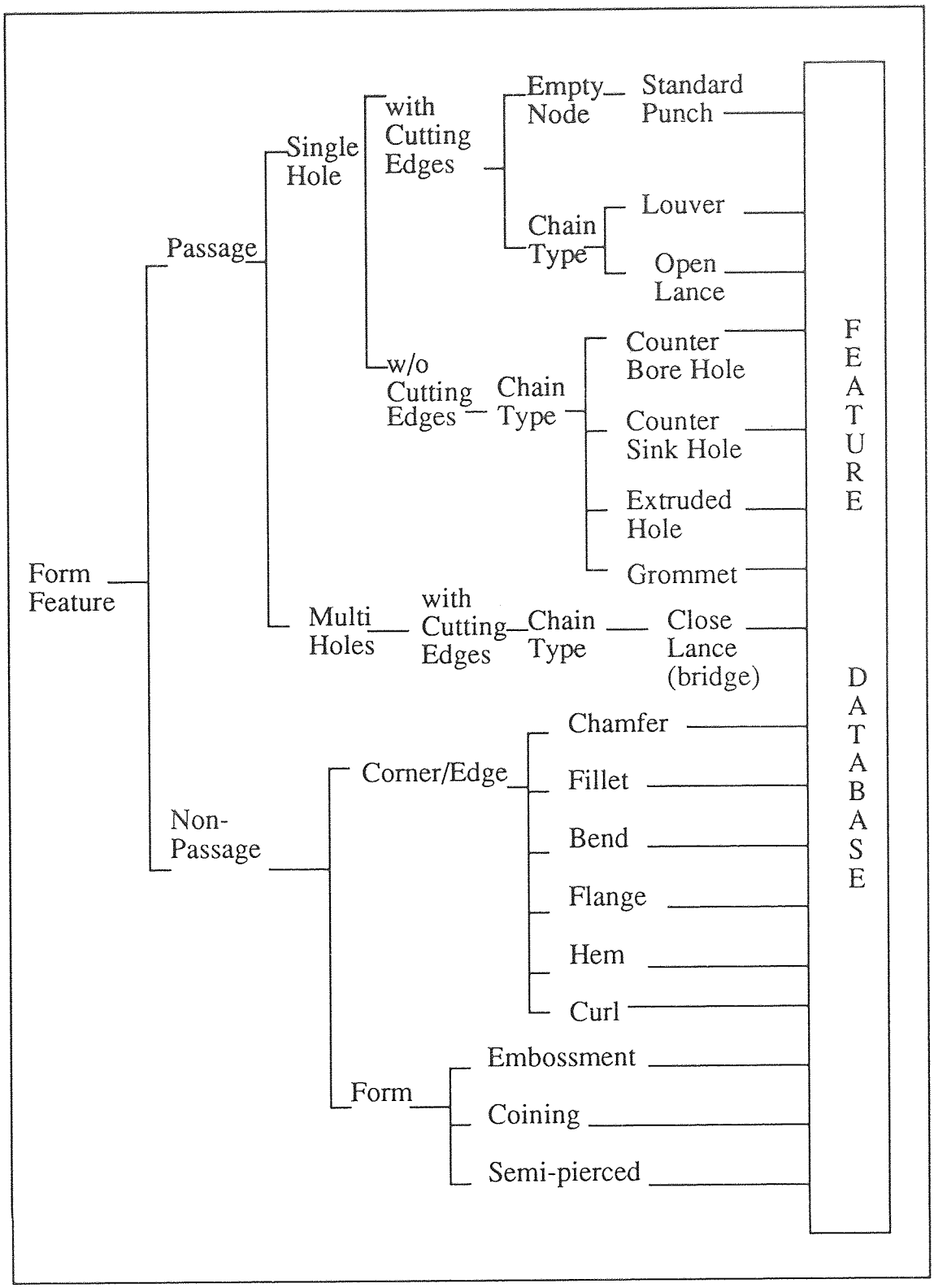


Figure 2.4
Feature classification for sheet metal fabrication

Feature modelers usually provide a library of generic features, which have been formalized in terms of some of the properties listed above. In creating model a user needs only instance a feature from the library and it will automatically take on all the generic properties of it class. The user needs to complete the definition by specifying values for dimensions, location, and so on. Although it is recognized that a feature-based CAD system should meet these requirements, no system has achieved them.

CHAPTER 3

METHODOLOGIES FOR CREATING FEATURE MODELS

There are many alternatives for creating feature models in geometric modeling context. If there were a universal set of features that satisfied all needs, clearly it would be better to design with those features and put them into the model at its creation. But because different domains and analyses view different regions of the geometry as significant, it is necessary either to extract the features from a generic feature format or to translate between different feature models. It is not yet clear which method is the best approach to the problem. In order to provide a framework for comparison, many researchers have proposed to classify these methods into three broad groups (References 4, 7, and 8):

- Interactive Feature Definition
- Automatic Feature Recognition
- Design by Feature

3.1 Interactive Feature Definition

A geometric model is created first, then features are defined by human users by picking entities on an image of the part. The block diagram of this process is as shown in Figure 3.1.

This methodology involves predefinition of the geometric model. Therefore the data structure of the geometric model is a major factor in the design of the definition procedure. A 2D/3D wireframe or B-rep solid model is created using a conventional CAD package. The database created is then read by a program that

render an image of the component on a CRT to allow the user to interactively pick topological entities (i.e. edges, faces) needed to define a feature. This information can be augmented with attributes such as tolerances, surface finish, or high level

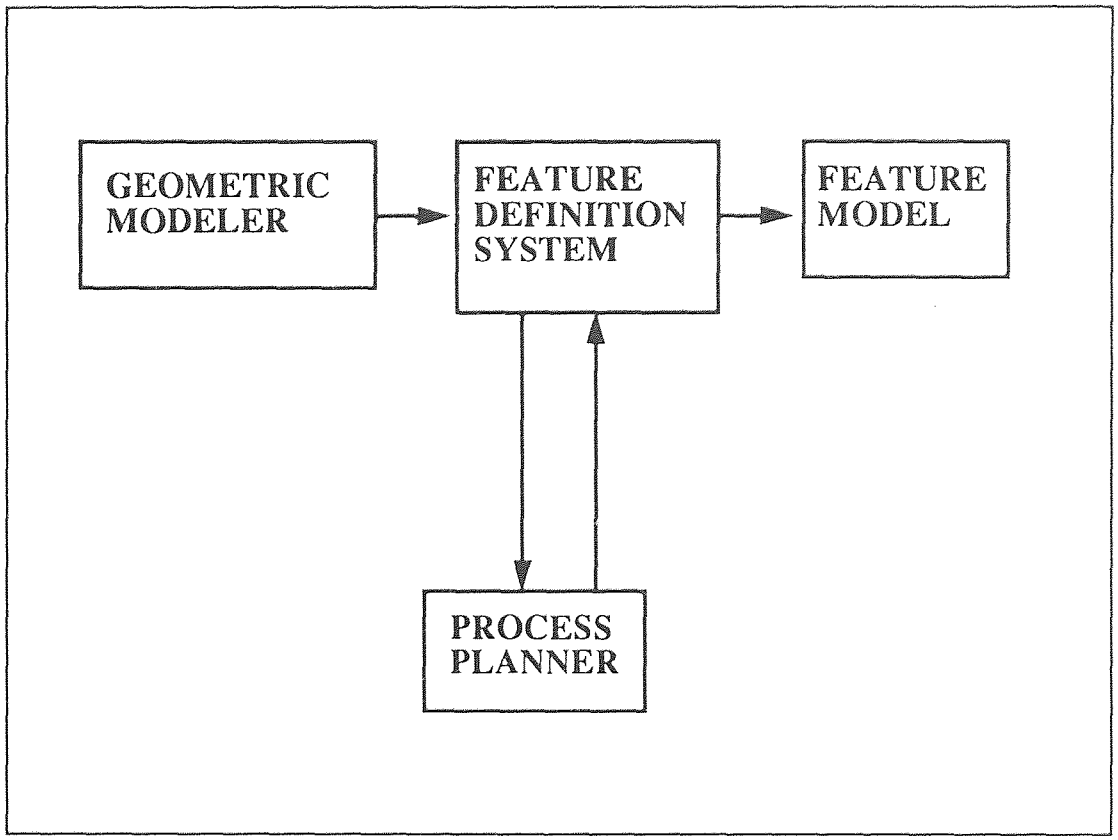


Figure 3.1
Interactive feature definition system

parameters (i.e. hole diameter, hole location etc.). This approach has been used largely for inputting data to programs for process planning and NC tool path generation.

The human assisted definition is easy to implement and it can work off Initial Graphic Exchange Specifications (IGES) or modeler specific database of contemporary systems. Only features needed for an application (i.e. process

planning) need be identified. For models containing a large number of feature, this method can be time consuming. In many current implementations the burden of picking valid entities lies on the user. The job can be made easier with some changes; for example, if the user indicates to identify a flat-bottom hole, then the system prompts that user must pick a cylinder face and a plane face; when entities are picked the system will check if they are of the appropriate type. Procedures must exit for automatically deriving the diameter and depth of the hole from the geometric model. The number of topological entities is arbitrary and often depends on intersections performed in construction of the model.

3.2 Automatic Feature Recognition

A geometric model is created first, then a computer program process the database to automatically extract features. The block diagram of this process is as shown in Figure 3.2.

The most important principle in recognizing a feature is avoiding ambiguity. Feature recognition is inherently a domain dependent process. A geometric model is searched for the particular features of interest for a given process, such as process planning, NC programming, and Group Technology coding. Various techniques have been developed in order to obtain the particular features of interest for a given process, directly from a geometric modeling database. This is popularly referred to as feature recognition. Most feature recognition has been directed at the machining domain so that the majority of research involves searching for cavities in the component model. Although the output of some techniques is not in the form of features but rather as machining volumes, some examples of these techniques are Sectioning techniques, Convex hull algorithm, Cell decomposition, and Artificial intelligence (AI)/Geometric reasoning. They differ from Feature Recognition, and

known as Machining Region Recognition. These methods typically assume that all machining will be done by one machining process (such as Milling) so it is not necessary to know the specific feature other than its boundaries corresponding to final machined surfaces. For example, it does not matter if a machining volume is a rectangular pocket or an L-shaped slot because tool path can be generated without

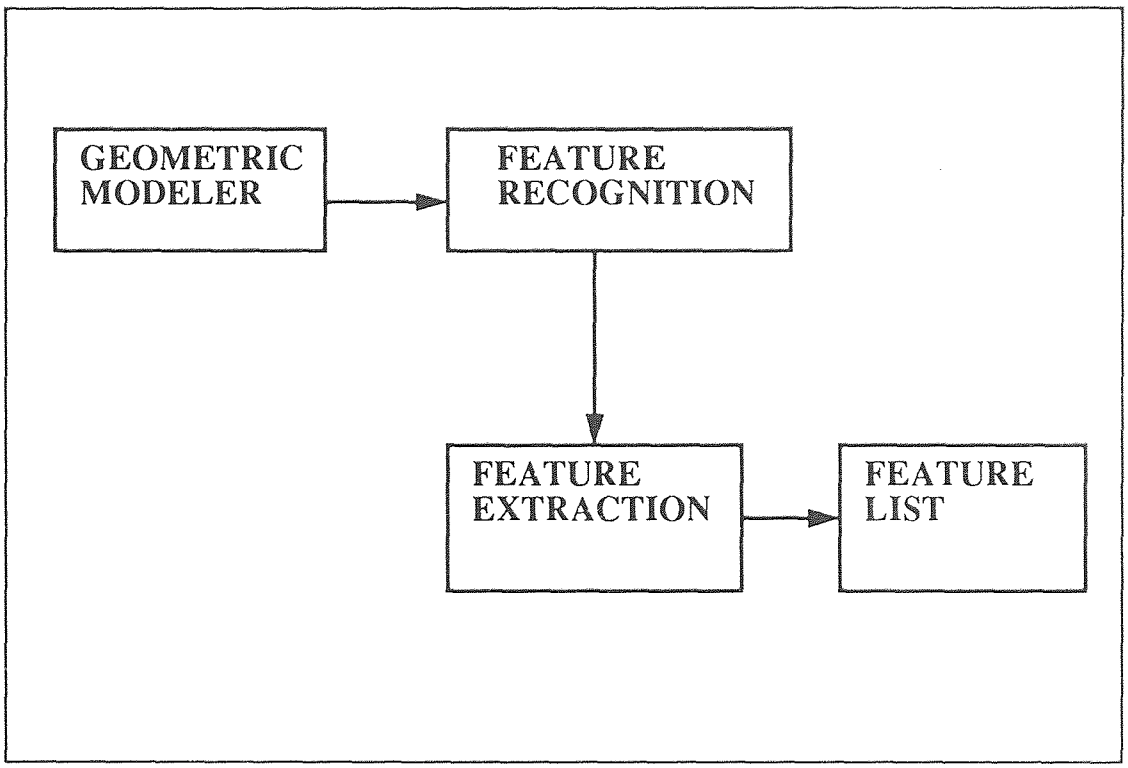


Figure 3.2
Automatic feature recognition system

this distinction. Thus feature recognition differs from machining region recognition. In feature recognition portions of the geometric model are compared to predefined generic features in order to identify instances that match the predefined ones. Specific tasks in feature recognition may include the following:

- Searching the database to match topologic/geometric patterns
- Extracting recognized feature from the database (remove a portion of the model associated with the recognized feature)
- Determining feature parameters (i.e. hole diameter, pocket depth)
- Completing the feature geometric model (i.e. edge/face growing, closure)
- Combing simple features to get higher level features

There are many feature recognition methods have been developed. It is difficult to classify recognition methods into clear taxonomy because there is considerable overlap between the various techniques. These methods can be classify into two broad groups, they are:

- Boundary-based matching methods
- Volume-based decomposition methods.

3.2.1 Boundary-Based Matching

Generic features are first formalizes in terms of their geometric and/or topologic characteristics. Then search algorithms are devised to determine which of these characteristics are present in the geometric model. There are two techniques more in use, they are graph matching, and syntactic pattern recognition.

Since solid model data structures are usually graph structures, graph matching has been a popular method for feature recognition. Pure graph matching done on unargumented solid models amounts to topological matching. The characteristics are based on the number of entities, topological type, connectivity, and adjacency. If matching is done in this way, features of very different semantics

would be classified as being same. Therefore some subclassification using geometric relationships is necessary. An entity classification method, developed by Kyprianou, has been used widely (Reference 6). It is based on the magnitude of the angle of intersection. For example in this method, edges are classified either as convex, concave, smooth convex, and smooth concave as shown in Figure 3.3.

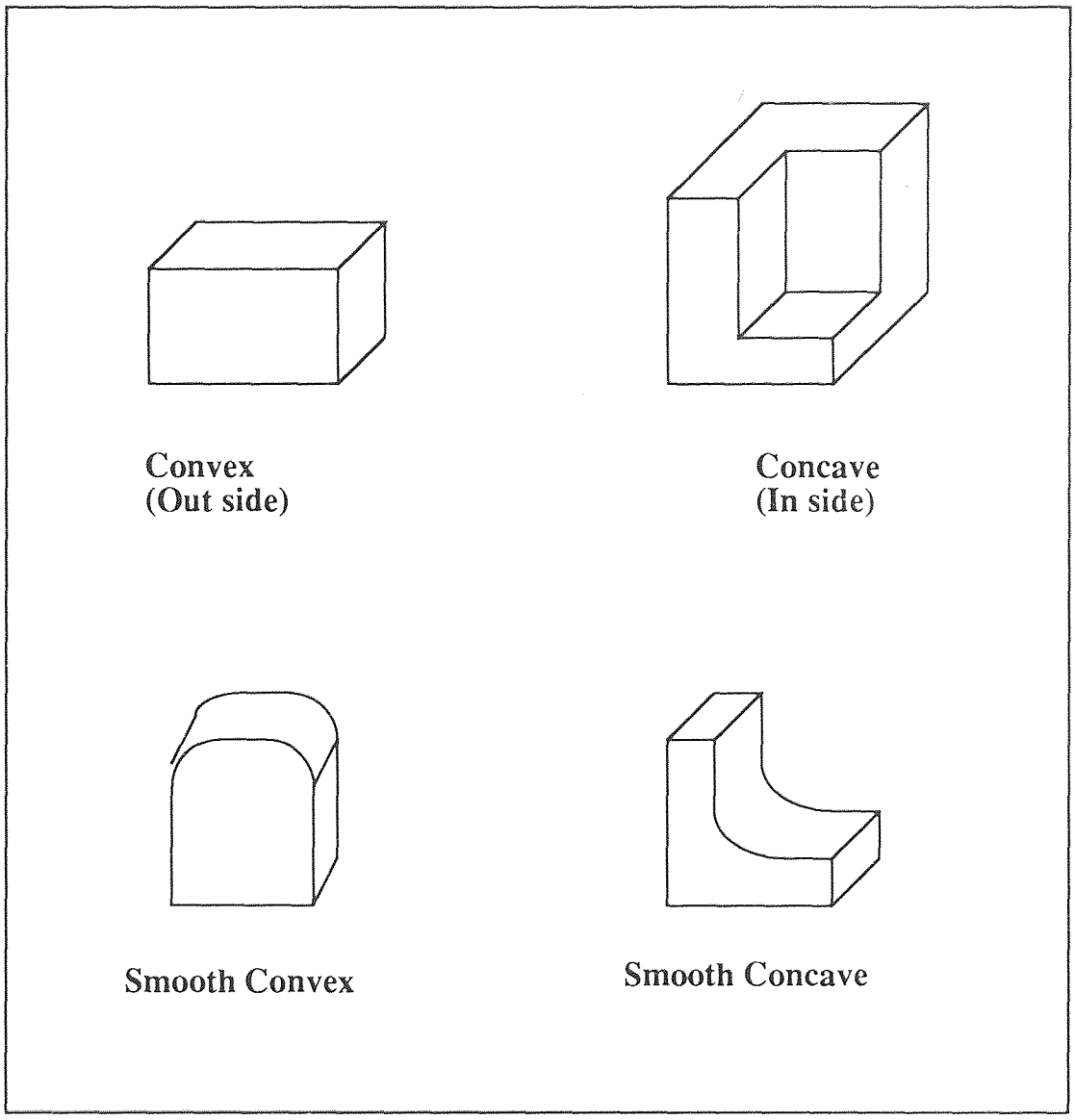


Figure 3.3
Kyprianou's edge classification criteria

Second technique syntactic pattern recognition adopted from vision systems. In these systems geometric patterns are described by a series of typically straight, circular, or other curved line segments. Simple patterns can be link together to give compound patterns. Languages have been developed for describing these sequences algebraically and manipulating them with operators that form a grammar. Feature can be recognized by parsing the feature against the object's description in the grammar. Graph grammars and shape grammars have been developed for matching feature shapes. The most common method is based on rules. Features are formalized by templates that consisted of pattern rules. Templates are defined for both general features (i.e. holes) and specific features (i.e. flat bottomed hole). Rules are expressed as a set of both geometric and topologic conditions, each of which had to be tested separately. In order for the rule to be satisfied, all conditions have to be satisfied.

3.2.2 Volume-Based Decomposition

The purpose of volume decomposition is to identify material to be removed from a base stock and break down this volume into units corresponding to distinct machining processes. First, the total material to be removed by machining is found by a boolean difference between the stock and the finished part. This volume must then be decomposed into units that corresponded to practical machining processes that match machining features as shown in Figure 3.4 and Figure 3.5 for an example components made by machining processes and sheet metal stamping process respectively. A well-known work on volume decomposition is done by General Dynamics for CAM-I. The purpose of the project was to achieve a high degree of automation for generating NC program for components defined by noncomplex surfaces (planner, quadric, and cylindrical). An algorithm was developed for

operating on B-rep model of the total volume to be removed, augmented with tool accessibility codes for each face. A library of generic delta volumes existed in the system.

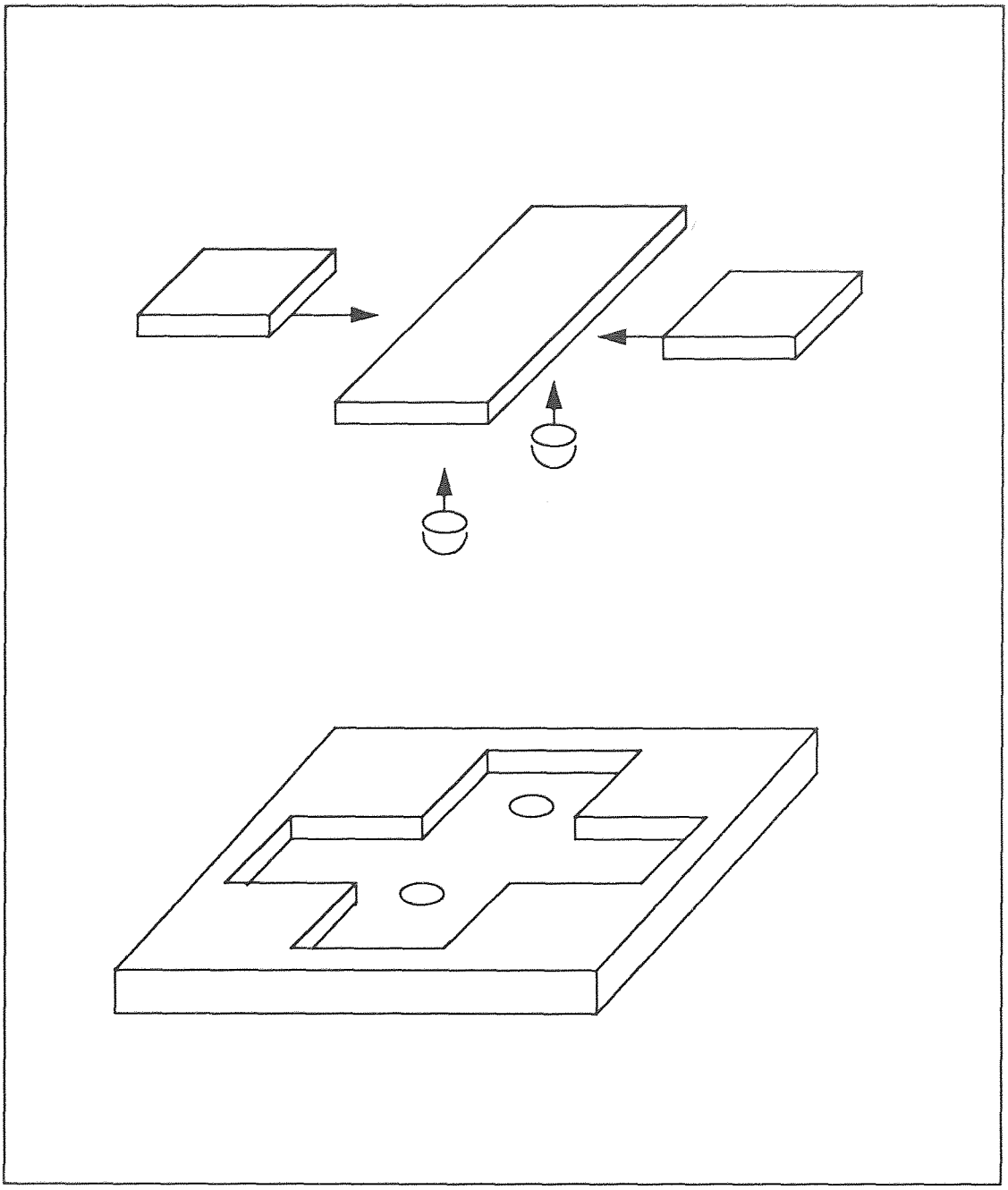


Figure 3.4
Example part volume decomposed into its delta volumes
for machining part

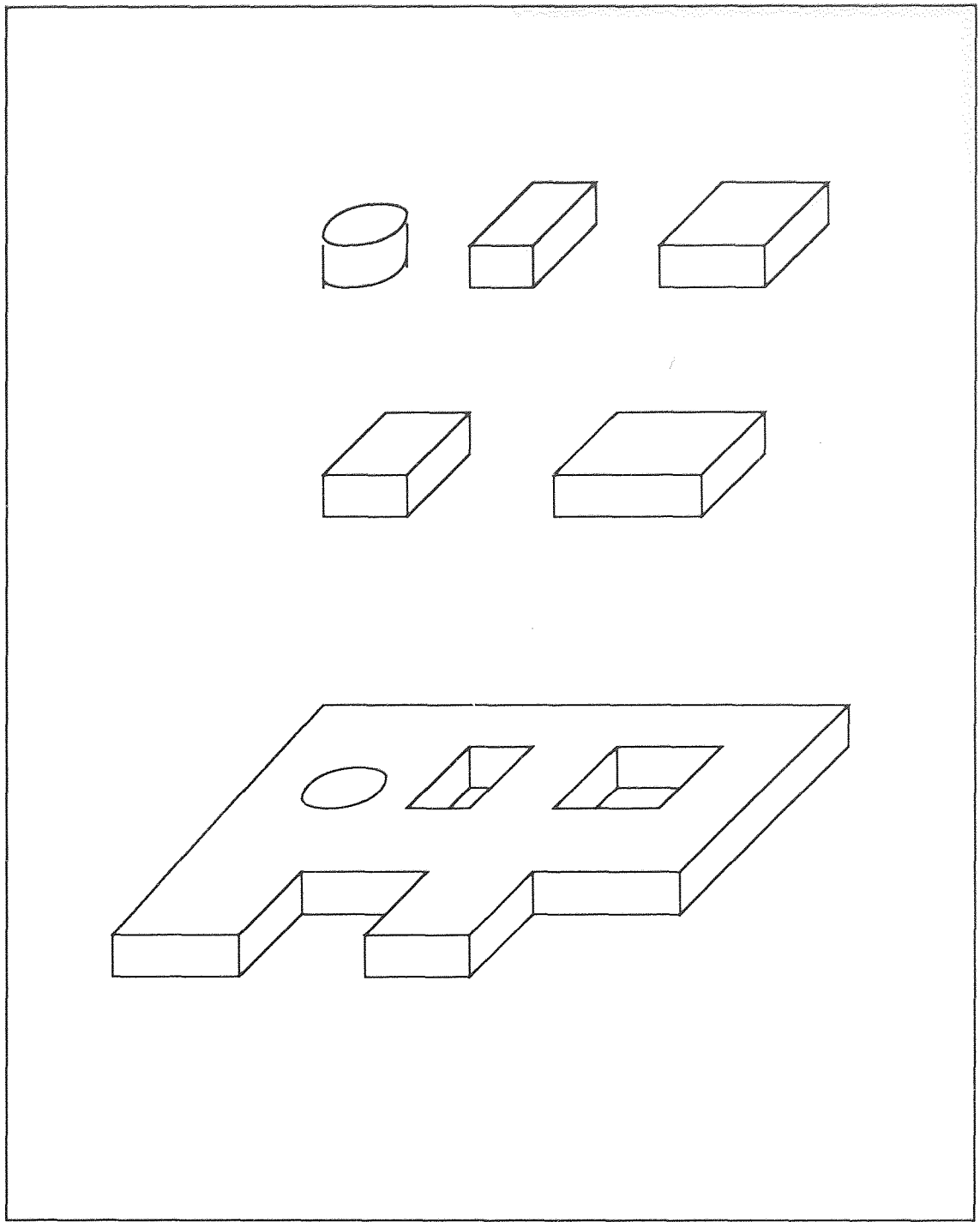


Figure 3.5
Example part volume decomposed into its delta volumes for sheet metal stamping part

Considerable progress in feature recognition has been made. Principal among the advantages of feature recognition is the use of current geometric modeler database or even IGES. Another advantage is that recognition can be made application specific, allowing each application program to have its own recognition program.

3.3 Design by Feature

The component geometry is defined directly in terms of features by the CAD modeler, thus geometric models are created from the features. The block diagram of this process is as shown in Figure 3.6.

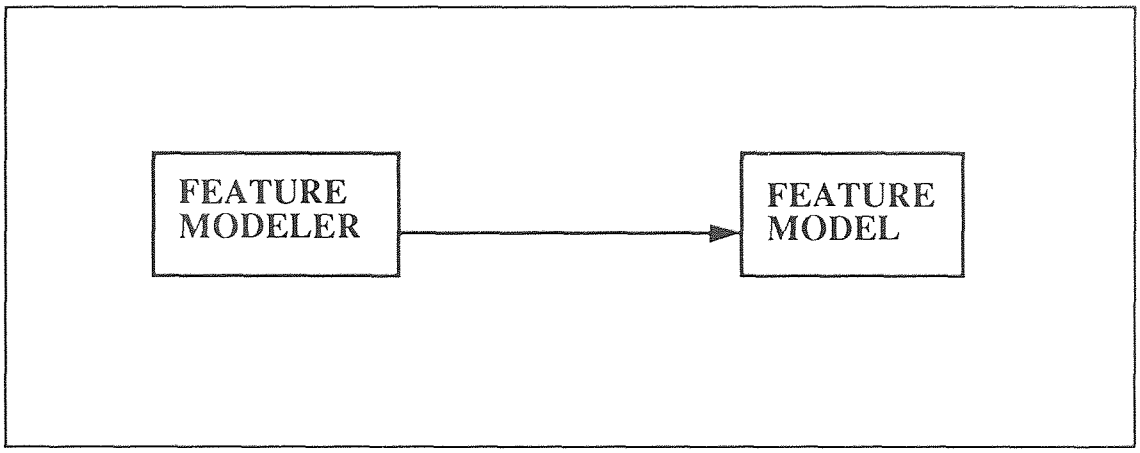


Figure 3.6
Design by feature system

In this approach, features are incorporated in the part model from the beginning. Generic feature definitions are placed in a library from which features are instanced by specifying dimension and location parameters and various attributes. There are two common methodologies in use, they are as follows:

- Destruction by Machining Features
- Synthesis by Design Features

3.3.1 Destruction by Machining Features

Destruction by Machining Features method also known as destructive solid geometry, and deforming solid geometry. In this approach a component model is created by boolean subtracting features from a base stock model. The designs and manufacturing plans are concurrently developed by transforming a base stock model into the final part model through the application of operations that correspond to stock removal. Prototype systems using this approach have been demonstrated at Standford and Purdue. The commercial system Pro-Engineer also supports this approach. All these systems use a set of predefined features that are subtracted from the base solid. Features are defined by attribute slots encompassing dimensions, tolerances, tool finish and starting face. In the Purdue and Stanford systems, process plans are generated and tested with each design change. In the Stanford system various expert systems work concurrently to generate, simulate, and verify plans. These expert systems are feature machining experts, fixturing experts, tooling experts, and collision checkers. Thus, when design is complete, it means the process plans, tool designs, and NC programs are also complete.

3.3.2 Synthesis by Design Features

In this approach a component model is created by adding or subtracting features without a starting base stock, unlike in the Destruction by Machining Features approach. Many researches and commercial systems belongs to this category. Generic features are predefined in term of rules and procedures. Procedures may include methods for instancing, modifying, coping, deleting features, generating

solid models, deriving certain parameters, and validating feature operations.

Design by features has the advantage that it allows to transfer to the database much of the information available at the design stage. This richer and higher level database is available for use by downstream applications. It is even possible to implement real-time manufacturabilityy evaluation and concurrent design and process planing. However, the set of features used in design is not finite. One needs to determine how many features should be contained in the feature library and at what level of abstraction. Also, since features are application specific, the need for feature recognition by application does not go away when one design by features. Finally interactions between features can result in nongeneric shapes that do not exist in the database or they could make some generic dimension values obsolete.

3.4 Feature Representation

Feature may be represented at various levels. For example, one could represent them by the process by which they may be created or by the resultant geometric model. Feature may be defined more abstractly as a neutral description without any specification of how the feature is to become part of the geometric model. Explicit representations have commonly been used in interactive and automatic feature recognition and explicit representations in design by features. Some of the structures used for geometric representations of features are:

- Augmented graphs
- Algebraic, syntactic
- Delta volumes
- Constraint-based B-rep

Augmented graphs are usually based on face adjacency. The arcs of the graph are attributed with information on edge classification and geometric relationships. Syntactic languages have also been devised that encapsulate adjacency, connectivity, geometric orientation, and convexity/concavity of feature entities, though their use has been limited to 2D. Delta volumes are complete B-rep models of closed spaces associated with tool accessibility codes for faces and connectivity codes for the delta volumes are indicated by solid and dashed arrows, respectively. Pratt, M. J. shown that features are best modelled in B-rep structure (Reference 10).

3.5 Feature Validation

There are no universally applicable methods for checking the validity of features. It is up to the users defining a feature to specify what is valid or invalid for a given feature. This should not be confused with geometric or topological validity, which is based on rigorous mathematics. Features are invalid if any of the conditions declared in the generic definitions are violated. Such conditions could be based on size limits, shape, location, and so on. Therefore it is possible that some operations may result in valid (physically realizable) solids but may product invalid features. Typical checks that need to be done are inadvertent interference with other features. There are situations in which the resulting features may be invalid. Intersection between feature volumes could:

- Make a feature nonfunctional
- Create nongeneric feature(s) from two or more generic ones
- Render feature parameters obsolete
- Give nonstandard topology

- Delete a feature by subtraction of larger feature
- Delete a feature by addition of larger feature
- Close an open feature

At present time most feature modelers do not perform automatic feature validation. It is user's responsibility to do so. However, research systems have demonstrated the feasibility of automatic validation. Apart from checks on parent entities, parameter range, and position constraints, they can:

- Detect intersections between feature entities using geometric modeler functionality
- Classify the type of interaction, when detected (i.e. the classes given above)
- Consult the rules/procedures specified by user or application to determine what action to take, which may include (i) disallow interaction, (ii) send message to other features to alter affected parameters, (iii) take no action, and (iv) change feature type to match its current state.

3.6 Feature Standardization

Feature standardization research has been initiated by CAM-I and US Air Force. CAM-I has been working on the classification of form features for process planning. The US Air Force has been involved in PDDI project, which has been concerned with the representation of form features. Also, the US Department of Energy reports on a Product Definition Initiative (PDI) project in which a Form Features Centered Architecture (FFCA) is proposed. Later, a Form Feature Integration Model (FFIM)

developed within PDES/STEP and a Neutral File Format developed within ESPRIT (CAD-I). FFIM is meant for defining the two lowest data levels of the STEP model representation. They are the physical level, and the logical level. At the logical level, the Express language has been developed to define features and constraints.

As the feature technology is a fast developing field, the development and acceptance of standards for features could adversely affect research. For that reason, many researchers voted that standardization should not restrict research in features.

3.7 Feature-Based Application

Feature-based applications must transform feature models from a design viewpoint into a manufacturing or analysis view. This transformation may include feature extraction, decomposition into lower level entities, reconstruction by geometric reasoning, and augmentation with new data or entities. Many research work have concentrated on developing application packages using feature-based design approach for the various areas, such as, Process planning for Machineing componenets, Structeral design, Tolerancing model, and Tool cost estimation etc. (References 1, 3, 11, 12, and 14).

A basic requirement is access to the feature-geometry database. Sample queries for NC are (i) list all features associated with face 2, and (ii) list the stock size, and (iii) list all pockets.

Applications should be able to create one to augment the feature model with data private to them to create one or more secondary models. This includes attaching attributes to features or faces, for example, surface finish to a face or tool approach for a slot. For finite element analysis, the augmented model may include loads and restraints. Several applications may use the same data.

To support the database queries, a static interface to the feature modeler and

solid modeler is required. A dynamic interface is also needed for applications. For example, verification of NC programs includes tests gouging of a part and machining a fixture, both of which require a solid modeler command interface for the boolean operations.

As the transformation occurs from the decision model to the feature-based application secondary model, a number of interdependencies arise. Design changes must propagate to the application, flagging situation that need human resolution. If an application changes the geometry of a component, these alterations will not propagate up. Such situations should be noted as an attribute of the application feature model. Example of the need to alter component geometry include building an idealized model for finite element analysis or changes needed for work holding.

CHAPTER 4

FEATURE-BASED DESIGN APPROACH FOR THE SHEET METAL STAMPING INDUSTRIES

The feature-based design approach is a new approach to integrate CAD with the following application such as process planning, cost estimating, and analysis etc., for the sheet metal stamping industries. During past few years, use of CAD systems have been tremendously increased in every industry also in sheet metal stamping industries. At present, most of the CAD systems in use are not based on feature-based design approach. A need, therefore, exists either to modify existing CAD system or change to new feature-based CAD system to implement feature-based design approach. Second option is not acceptable by most of industries for various reasons. Some of them are as follows:

- Higher cost of the new feature-based CAD system.
- It may be rework to transfer existing conventional CAD model to the new feature-based CAD system.
- Operator training required for the new feature-based CAD system.

Thus, either way, it is required to convert existing conventional CAD model to feature-based CAD model. A need, therefore, exists to develop automatic feature extraction software to implement feature-based design approach in sheet metal stamping industries without major changes in existing industrial set-up. This will also, increase flexibility to transfer data between the companies, those having and not having feature-based design approach in practice.

Figure 4.1 shows the generalized feature-based modular structure for the

sheet metal stamping industries. The conventional CAD module provides design information to the feature recognition module through the IGES processor, as the format used by the various CAD systems differ from each other. Though, most of the commercially available CAD software systems allow writing design database information to an ascii file in the IGES format. Once the component is designed in the CAD system, such as ascii file in the IGES format can be prepared by selecting appropriate options. The IGES processor is designed to read IGES formatted file. Conversion algorithms were designed and coded to map the IGES file entities into the software database. After conversion, the processor will generate '.dat' file with groups of entities, their corresponding coordinates' values, and other particular information. A sample of the IGES file generated by the Pro-engineer CAD software system and the converted '.dat' files are shown in Appendix.

The feature recognition module read '.dat' files generated by the IGES processor and provide model data in terms of feature list. This module is discussed in detail in the following chapter. Features are considered as a subroutines to create solid model using feature list and some other usage of the feature list are as discussed in the following modules.

The sequencing module is interactive graphic package, which reads feature list and interactively prepare various alternative sequences of features and display their die-layouts graphically on the screen. Which helps users to choose one or more finalist from the various sequenced feature lists for further analysis such as a process planning, cost estimating, and so on.

The process planning module matches machine capabilities with the requirement for specific sequenced feature list. The machine database, withmachine process capability data like permissible minimum and maximum workpiece sizes, maximum power capacity etc., was created for this purpose.

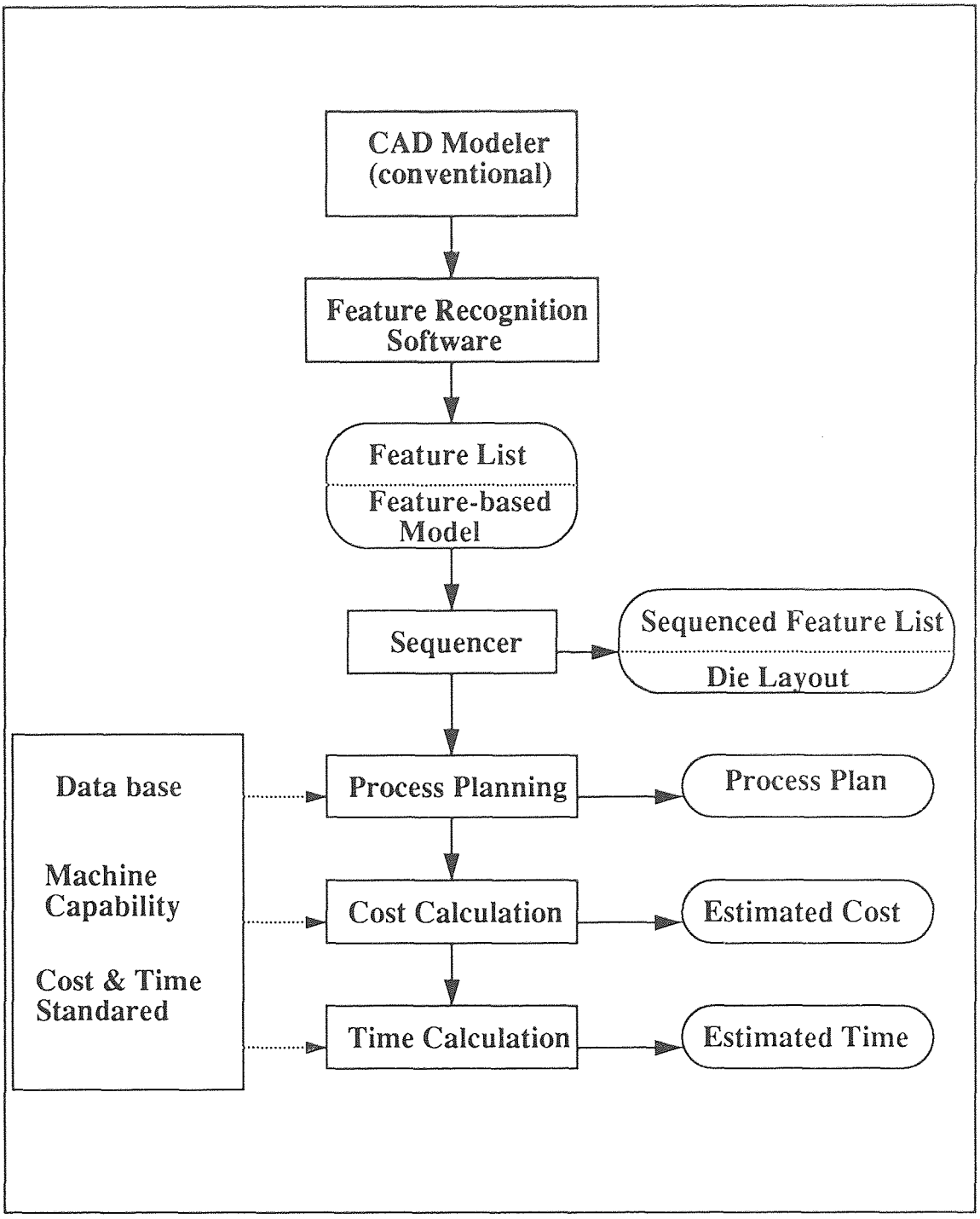


Figure 4.1

Generalized feature-based modular structure for the sheet metal stamping industries

The strategy of this module should be, such that, it selects the efficient machines, which will give minimum cost of production with the capability requirement of the job. In the case of operations which can be done on more than one machine the cost data such as machine-hr-rate, machiniest-hr-rate, and setting time of the machine can use to make selections of the machine. The cost calculation and time calculation modules provide estimated cost and esimated time respectively for various sequenced feature lists.

CHAPTER 5

AUTOMATIC FEATURE EXTRACTION SOFTWARE

The primary purpose of the automatic feature extraction software to provide feature list by extracting features from the conventional CAD database of the sheetmetal stamping product. This software was developed in C programming language running under UNIX based computer system. The software is designed with modular approach and its flow is controlled by the answers responded by the users for the questions asked by the program.

The block diagram of the software is as shown in Figure 5.1. In put data file to the software is generated by the IGES processor, by converting the IGES formatted conventional CAD file into the software data file ('.dat' file). The format of the '.dat' file is as shown in the Table 5.1.

In the question module few questions to be responded by users. The first question is to enter input data file name without extension, which will be the name of the file generated by the IGES processor. The second question is related to the CAD model data type (what type of CAD model data is?). The user has to choose one of the option out of three. The three options are as follows:

(1) 2D unfolded data: This option selected, if the input data file is generated from the 2D unfolded component layout CAD model. If this option selected, then the 2D unfolded layout data module will read the data file.

(2) 3D unfolded data: This second option selected, if the input data

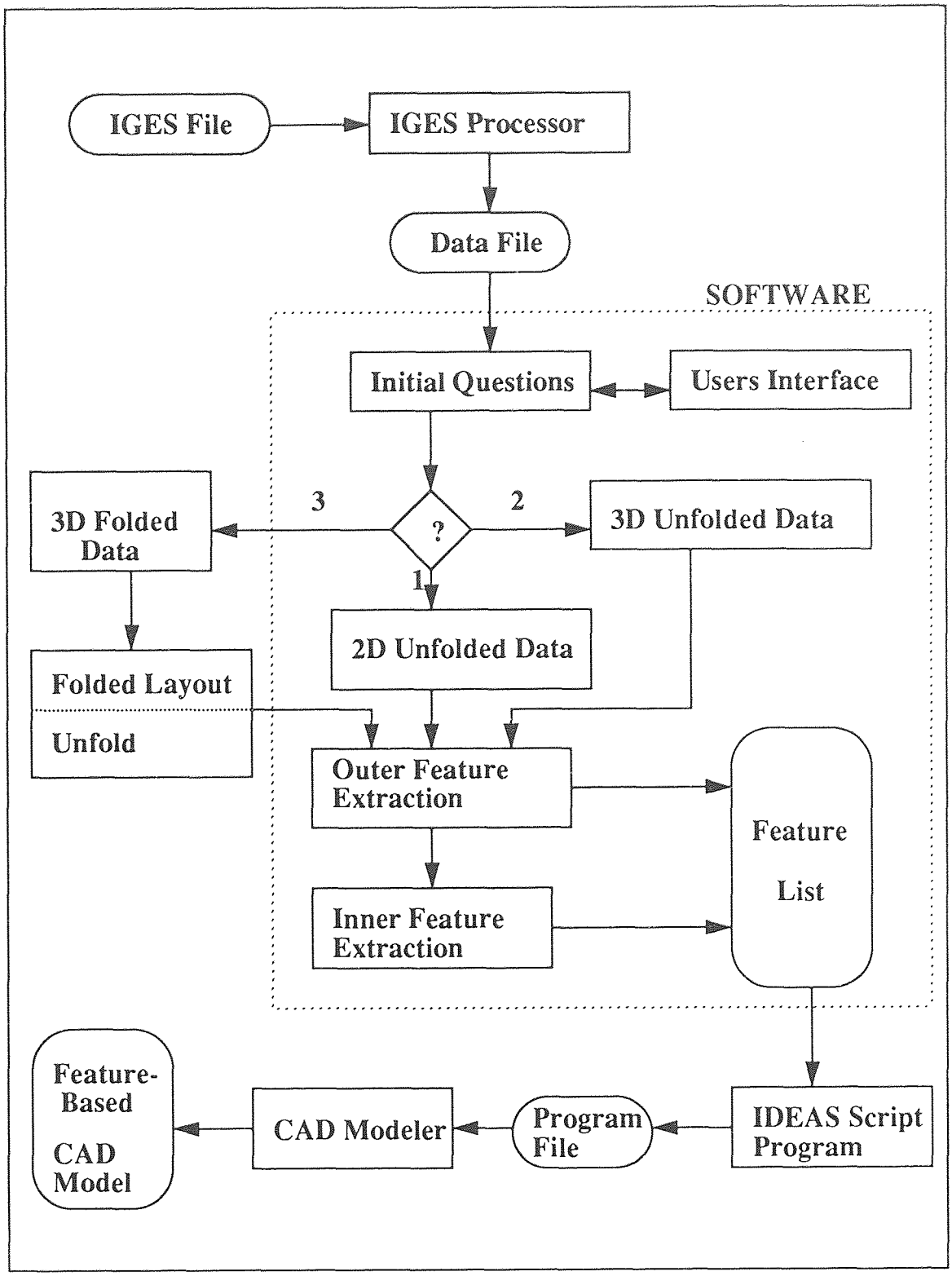


Figure 5.1
Automatic feature extraction software

Table 5.1

Input data file format (generated by IGES processor - '.dat' file)

FORMAT	DESCRIPTION
XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX	Minimum and maximmum coordinates values. (i. e., 1.00, 5.00, 1.000,9.00, 0.00, .25)
N	Total number of the point data. (i.e., 1)
X, Y, Z	Following N number of lines will give X, Y, and Z coordinates values of the all N number of points. (i.e., 3.50, 8.00, 4.00)
M	Total number of the line data. (i.e., 1)
X, Y, Z, X1, Y1, Z1	Following M number of lines will give end points coordinates values for the all M number of lines. (i.e., 2.875, 1, 4.00, 5.0, 1.0, 4.0)
C	Total number of the circle data. (i.e., 1)
X, Y, Z, D, P	Following C number of lines will give center point coordinates values, diameter and plane code data for the all C number of circles. (i.e., 3.5, 8.0, 4.0, 0.5, 1)
A	Total number of the arc data. (i.e., 1)
X,Y, Z, X1, Y1, Z1, X2, Y2, Z2 CX, CY, CZ, D, P	Following 2 times A number of lines will give data for all A number of arcs. For each arc data will be in two lines, first line will give two end points, and middle point coordinates values, and second line will give the center point, diameter and plane code data. (i.e., 3.5, 8.0, 4.0, 0.5, 1)
BX	Total number of the bending @ x-axis. (i.e., 1)
X, Y, Z, X1, Y1, Z1, R	Following BX number of lines will gives both end points coordinates values, and rotation in deg.. (i.e., 2.875, 2.5, 4.0, 5.0, 2.5, 4.0, 90)
BY	Total number of the bending @ y-axis. (i.e., 1)
X, Y, Z, X1, Y1, Z1, R	This data will be in same format as the data of the bending @ x-axis (i.e.,1.5, 5.05, 4.0, 1.5, 9.0, 4.0, 90)

file is generated from the 3D unfolded component CAD model. If this option selected, then the 3D unfolded data will interpret in terms of the 2D unfolded data by 3D unfolded module.

(3) 3D folded data: This third option is selected, when the input data file is generated from the 3D folded component CAD model. As scope of the present work is limited up to first two options, this option is not available.

So, the software is handling first two options and will provide feature list for those selections. The next question program will asked, is related to the thickness of the component. User has to input thickness of the component.

Feature extraction is processed in two parts. In the first part the outer feature extraction module extracts all features located on the boundary of the component. Then, in second part all features located inside the boundary of the component are extracted by inner feature extraction module. The basic features, we are considering in this work are listed in Table 5.2. Here, various shapes of opening in components are considered using two basic types of features, they are Hole and Square. Here, Square feature is not mean real life square shape. All real life square and rectangular shapes are consider as a Square features. Using these all basic features the software is recognizing component's shapes as one or combinations of the basic features. If component's inner shape is combinatoin of basic features, then the software will recognize it and list those features under one station in the feature list. Table 5.2 also shows the format of the features listed in feature list file.

Table 5.2

Output feature format (generated by feature extraction software - '.ext' file)

FEATURE	FORMAT	DESCRIPTION
BASE	BASE H L T X Y Z	This feature gives dimensions of the blank (i.e., height, length, thickness, and values of the center point coordinates).
HOLE	HOLE D X Y Z	This feature gives dimensions of the circular or arc shape opening. (i.e., hole diameter, and the coordinates values of the center point)
SQUARE	SQUARE H L X Y Z	This feature gives dimensions of the rectangular or square shape opening. (i.e., height, length, and the coordinates values of the center point)
BENDX	BENDX Y A	This feature gives dimensions of the bending about the axis parallel with the x-axis. (i.e., y distance, and bending angle in deg.)
BENDY	BENDY X A	This feature gives dimensions of the bending about the axis parallel with the y-axis. (i.e., x distance, and bending angle in deg.)

For example consider some shapes as a combination of one or more basic features as shown in Figure 5.2 and Figure 5.3. For the example #1 (as shown in Figure 5.2) the 'L' shape of opening will be recognize as combination of two Square features and the shape of the internal opening will be recognize as a combination of three basic features (i.e., two Hole features and one Square features). For the example #2 (as shown in Figure 5.3) the open loop shape 1 and shape 2 will be recognize as combination of the Square feature as explained by Figure 5.3 (b) and Figure 5.3 (c). This example shows the software's capability to extract features from the stepped shapes.

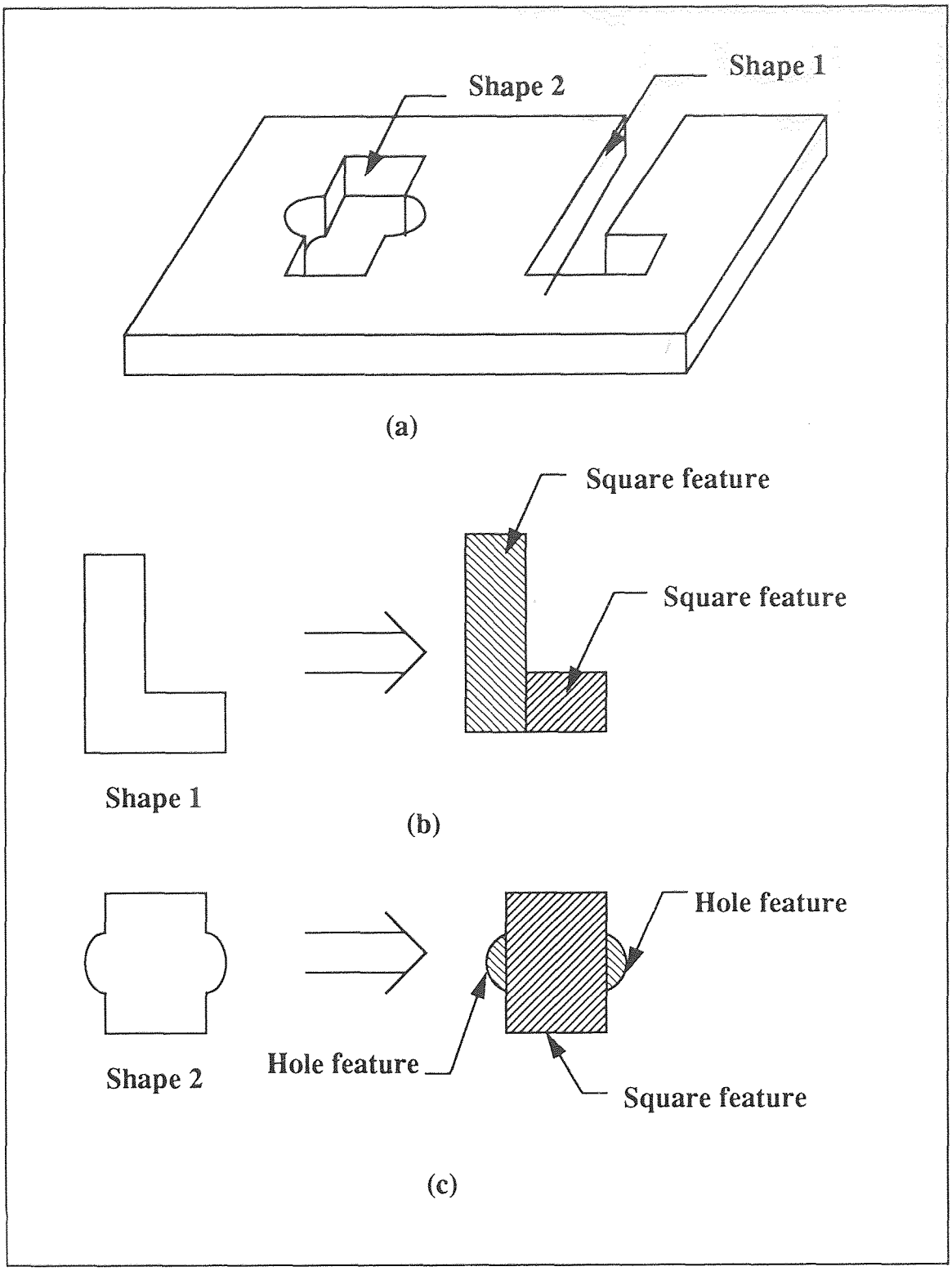


Figure 5.2
Example #1 with some shapes as a combination of one or more basic features

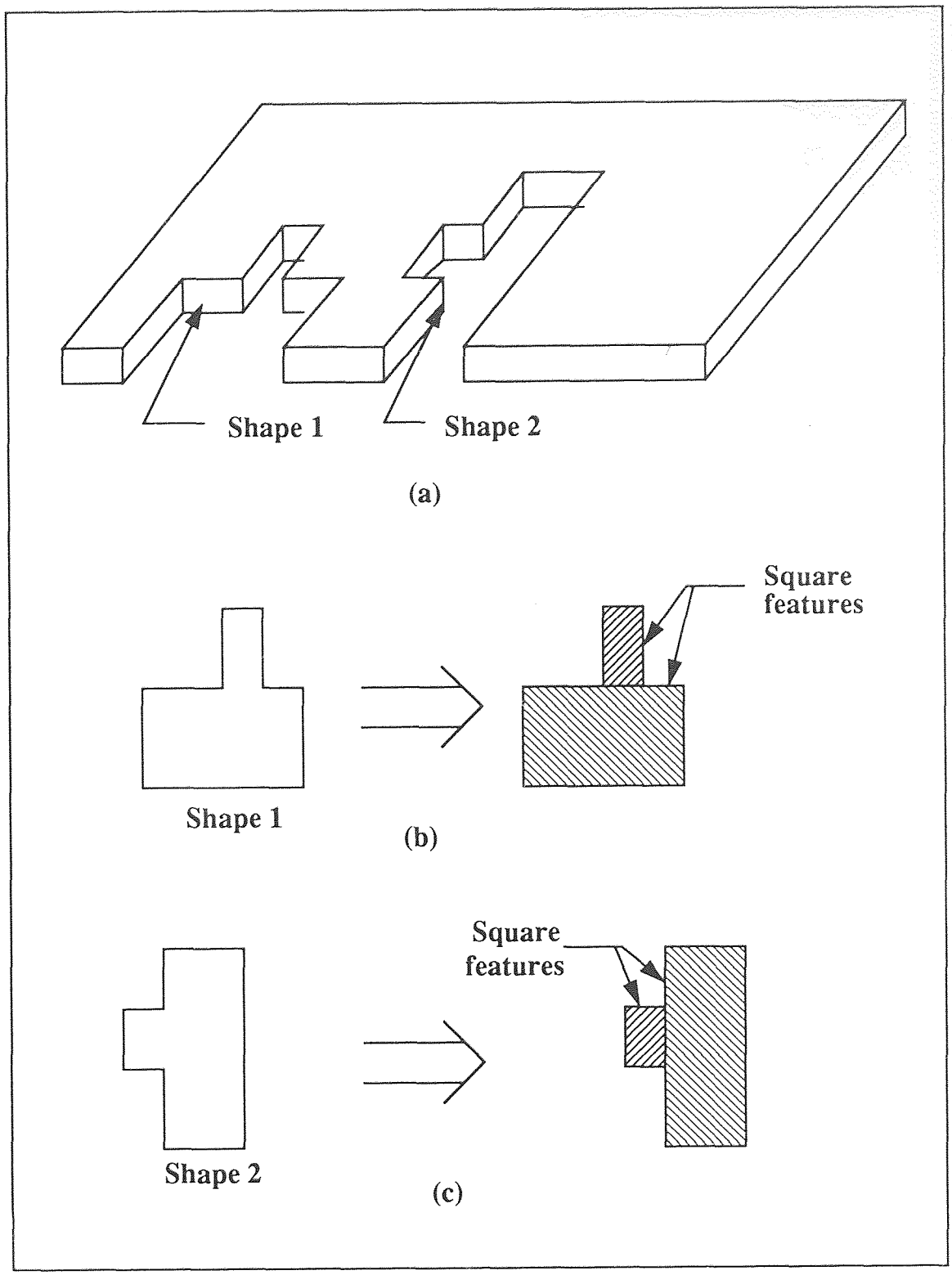


Figure 5.3
Example #2 with some open loop stepped shapes opening

The format of the feature list file is as follows. In the feature list file the first line will always provide dimensions of the component's blank size using the BASE feature and it's format as shown in Table 5.2. Then, all extracted features for the shapes located on the border of the component (in other word open loop shapes) are grouped under the title outer feature. After that, all extracted features for the shapes located inside the component's boundary (in other word the closed loop shapes) are grouped under the title inner features. Furthermore, for each inner shape, its extracted feature/s is/are listed under the subgroup title station. In appendix, two components with various shapes are shown with their CAD model figure and respectively with their input data file, and output feature list files.

5.1 Software Usage

Before invoking the software, write the design database information in an ascii file in the IGES format. The IGES file should be written for the wireframe model only as the software cannot interpret surface model information. The software can be invoked by typing 'feature' at the command line. After initializing the environment, the software will asked few questions to be responded by users, as discussed in the above section. Then, the software will give output feature list file with same as input file name except the extension. The extension of the output file name will be '.ext'. Here, we are presenting one of the application of the feature list is to create a solid model in the SDRC-IDEAS CAD system. For this, first the output file required to process through the script program, which will generate part program file with extension '.prg' for the SDRC-IDEAS application. Then using SDRC-IDEAS CAD system, solid model can be created using the part program file.

CHAPTER 6

SUMMARY

6.1 Conclusion

As technologies advance and new developments occur, it is always the situation wherein a revolutionary marriage of the new technology with old technology has to be established, to implement the new technology. During the course of this thesis, an effort is made to develop an automatic feature extraction software to implement the new feature-based design approach for the sheet metal industries.

This software can be consider as a step forward, in implementing the new feature-based design approach for the sheet metal industries having conventional CAD systems. It can provide an important link in integrating CAD with the other applications (such as Computer Aided Process Planning, Cost estimation etc.) by utilizing feature-based design approach. The software was tested for various components. The software provided a quick and efficient way to generate consistent feature list from the conventional CAD database, which can be utilized by the other applications. Thus, the feature list is complete specifications of the design component in term of the geometrical information as well as information required by the other applications (i.e. manufacturing, cost estimation, and time estimation etc.). The features are more natural and concise way to represent the design, and it reduces the required level of expertise to interpret the CAD data for the applications uses the CAD database.

6.2 Future Work

The work from this thesis gives a start for implementing the new feature-based

design approach for the sheet metal industries using the conventional CAD systems. More basic features can be added to the tool to make it more attractive, versatile, and complete (Such as features to recognize angular opening, and angular edge). The software has been designed and developed using modular approach. The modularity of the software makes possible to add new modules where necessary in future. Some of the area in which future improvement can be considered are as follows:

- Develop a module to expand the capability of this program to process three dimentinal folded data (i.e. develop 3rd option).
- Develop a graphic interactive package to arrange features in required sequence, and generate various process sequences for comparison.
- Develop cost estimation module and time estimation module to generate cost and time estimates for comparison.

APPENDIX A

AUTOMATIC FEATURE EXTRACTION PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <math.h>

        int j, k, n, jl, ji, stl[50], junk, c, sk[50];
        int lcount, b, bl, flage, ul[50], uc[50], ua[50], ansr;
        float t, sum, XMIN, XMAX, YMIN, YMAX, YLOW, CONX,
CONY, Dsh, D[50];
        float XLOW, ZLOW, th;
        struct ppath {
        float x;
        float y;
        float z;
        };
        struct ppath points[500];
        struct lpath {
        float x;
        float y;
        float z;
        float x1;
        float y1;
        float z1;
        };
        struct crlpath {
        float x;
        float y;
        float z;
        float d;
        int  p;
        };

        struct bpath{
        char st1[5];
        float x;
        float y;
        float th;
```

```
float x1;
float y1;
};
struct blpath{
float x;
float y;
};

struct spath{
char st2[7];
float l;
float h;
float x;
float y;
};
struct arcpath{
float x;
float y;
float z;
float x1;
float y1;
float z1;
float x2;
float y2;
float z2;
float cx;
float cy;
float cz;
float d;
int  p;
};
struct cylpath{
char st1[7];
float d;
float x;
float y;
};
struct bnpath{
float x;
float y;
float z;
 float x1;
float y1;
```

```
                float z1;
                int a;
                };
                struct bnpath bendx[50];
                struct bnpath bendy[50];
                struct cylpath cyl;
                struct arcpath arc[500];
                struct spath square[500];
                struct crlpath circle[500];
                struct lpath line[500];
                struct lpath ln[1000];
                struct ppath oript;
   char buffer[80];      /* character buffer */
   char file_buffer[80];  /* buffer holding full path of data file name */
   int i, maxpts, totlns, maxlns, maxarc, maxcrl, maxbnx, maxbny;          /* index
varaiable */
   char indata[10], *dat=".dat", outdata[10], *ext=".ext", quit[4];
                FILE *fpl, *fpw, *fpr, *blw, *blr, *jfw;

                main()

{

run_again:
                question();

      fpl = fopen(indata, "r");
                fpw = fopen("temp", "w");
                fpr = fopen("temp", "r");
                blw = fopen(outdata, "w");
                blr = fopen(outdata, "r");
                jfw = fopen("junk", "w");

                getdata();
                n = strlines();
                fprintf (jfw, "n = %d\n", n);
                if (n > 1)
                { getlowx(n);
                  getzlow(n); }
                fprintf (jfw, "ZLOW %f\n", ZLOW);
                c = lowxlns(n);
      fprintf (jfw, "c = %d, sk1-%d, sk2-%d xlow %f\n", c, sk[1], sk[2], XLOW);
```

```
                if (ansr == 3 )
                { fprintf (jfw, "This part of the program is underdevelopment\n");
                fldata(); }

                else { if (c == 3)
                { maxlns = skin(); tdstrlnum(); }

                else { maxlns = totlns; lndata();
                                strlnum();}
                base();
                xmaxfeature();
                ymaxfeature();
                xminfeature();
                yminfeature();
                fprintf (jfw, "End of the data. %d\n", lcount);
                fprintf (jfw, "lcount %d\n", lcount);
                inblock();
                incyl();
                bend();
                }
reconfirm:
                confirm();
                if (quit[0] == 'Y' || quit[0] == 'y')
                { goto run_again; }
                else if (quit[0] == 'N' || quit[0] == 'n')
                { bye(); }
                else { goto reconfirm; }

/************* End of main*********/
}
/**************End of main *********/

/*************User input************/

question()
{

   system("clear");
   printf("\n\n\nFeature Extraction> Give input file name wihout extention\n");
   printf("Feature Extraction> ");

      scanf("%s",indata);
      getchar();
```

```
        for(i = 0;i<10;i++) outdata[i] = indata[i];
        strcat(indata,dat);
        strcat(outdata,ext);
        printf("Feature Extraction> %s is your in put file\n",indata);
get_selection:
        printf ("Feature Extraction> Select the option\n");
        printf ("\n\n\n");
        printf ("   1 Model is 2D unfolded model\n");
        printf ("   2 Model is 3D unfolded model\n");
        printf ("   3 Model is 3D folded model\n");
        printf ("\n\n\nFeature Extraction> ");
        scanf("%d", &ansr);
        getchar();
                if (ansr != 1 && ansr != 2 && ansr != 3 )
                { system("clear"); goto get_selection; }
                printf("\n\n\nFeature Extraction> What is the
thichness(XX.XX)?\nFeature Extraction> ");
        scanf("%f", &th);
        getchar();
}

/***************confirm to quit the program *********/
confirm()
{
                system("clear");
                printf("\n\n\nFeature Extraction> Do you want to extract featurs from
another file\n");
                printf("                (Yes or No):\n\n\n");
                printf("Feature Extraction> ");

                scanf("%s",quit);
                getchar();

}
/***************quit the program*****/
bye()
{
system("clear");
printf ("\n\n\n\n\n\n\n\nFeature Extraction> Thank you for using Feature Extraction
program\n");

}
```

```
/***************get data from the input file*********/
getdata()
{
                    memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%f,%f,%f,%f", &XMIN, &XMAX, &YMIN, &YMAX);
      fprintf (jfw, "Xmin and Xmax Values are %f, %f, %f\n", XMIN, XMAX, th);

   memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &maxpts);
      fprintf (jfw, "Maximum number of points are %d\n", maxpts);
   for (i = 0; i < maxpts; i++)
{
       memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
       sscanf(buffer, "%f,%f,%f", &points[i].x, &points[i].y, &points[i].x); }

   memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &totlns);
      fprintf (jfw, "Total numbers of lines are %d\n", totlns);
   for (j = 0; j < totlns; j++)
 {
      memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
      /** sscanf(buffer, "%f,%f,%f,%f,%f,%f", &line[j].x, &line[j].y, &line[j].z,
&line[j].x1, &line[j].y1, &line[j].z1); ***/

      sscanf(buffer, "%f,%f,%f,%f,%f,%f", &ln[j].x, &ln[j].y, &ln[j].z, &ln[j].x1,
&ln[j].y1, &ln[j].z1);
  }
   memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &maxcrl);
      fprintf (jfw, "Maximum numbers of circles are %d\n", maxcrl);
   for (j = 0; j < maxcrl; j++)
 {
      memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
      sscanf(buffer, "%f,%f,%f,%f,%d", &circle[j].x, &circle[j].y, &circle[j].z,
&circle[j].d, &circle[j].p);
  }
```

```
   memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &maxarc);
      fprintf (jfw, "Maximum numbers of arcs are %d\n", maxarc);
   for (j = 0; j < maxarc; j++)
 {
      memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
     sscanf(buffer, "%f,%f,%f,%f,%f,%f,%f,%f,%f", &arc[j].x, &arc[j].y, &arc[j].z,
&arc[j].x1, &arc[j].y1, &arc[j].z1, &arc[j].x2, &arc[j].y2, &arc[j].z2);

       memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
     sscanf(buffer, "%f,%f,%f,%f,%d", &arc[j].cx, &arc[j].cy, &arc[j].cz, &arc[j].d,
&arc[j].p);

}

  memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &maxbnx);
      fprintf (jfw, "Maximum numbers of xbend are %d\n", maxbnx);
   for (j = 0; j < maxbnx; j++)
 {
      memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
      sscanf(buffer, "%f,%f,%f,%f,%f,%f,%d", &bendx[j].x, &bendx[j].y,
&bendx[j].z, &bendx[j].x1, &bendx[j].y1, &bendx[j].z1, &bendx[j].a);
}

  memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
   sscanf(buffer, "%d", &maxbny);
      fprintf (jfw, "Maximum numbers of ybend are %d\n", maxbny);
   for (j = 0; j < maxbny; j++)
 {
      memset(buffer, '\0', 80);
      fgets (buffer, 80, fpl);
      sscanf(buffer, "%f,%f,%f,%f,%f,%f,%d", &bendy[j].x, &bendy[j].y,
&bendy[j].z, &bendy[j].x1, &bendy[j].y1, &bendy[j].z1, &bendy[j].a);
}
```

```
}
/**************line data ***********/
lndata()
{
        for (j = 0; j < maxlns; j++)
                { line[j].x = ln[j].x;
                line[j].y = ln[j].y; line[j].z = ln[j].z;
                line[j].x1 = ln[j].x1; line[j].y1 = ln[j].y1;
                line[j].z1 = ln[j].z1;
                }
}
/**************bending information *********/
bend()
{
        struct bendpath{
        char st[7];
        float l;
        int a;
        };

        struct bendpath bend[50];
        for (j = 0; j < maxbnx; j++)
        {
        bend[j].l = bendx[j].y;
        strcpy (bend[j].st, "BENDX");
        bend[j].a = bendx[j].a;
        fprintf (blw, "%s %f %d\n", bend[j].st, bend[j].l, bend[j].a);
        }
        j = 0;
        fprintf (jfw, "maxbny %d\n", maxbny);
    for (j = 0; j < maxbny; j++)
    {
    bend[j].l = bendy[j].x;
    strcpy (bend[j].st, "BENDY");
    bend[j].a = bendy[j].a;
    fprintf (blw, "%s %f %d\n", bend[j].st, bend[j].l, bend[j].a);
    }


}
/**************inner cylinder data*********/
incyl()
{
```

```
                char sst[9];
                for (j =0; j < maxcrl; j++)
            {
            strcpy (sst, "SLOT");
            fprintf (blw, "%s\n", sst);
                strcpy (cyl.st1, "HOLE");
      cyl.d = circle[j].d;
      cyl.x = circle[j].x;
      cyl.y = circle[j].y;
 fprintf (jfw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);
      fprintf (blw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);

                }
}
/**************inner block data*********************/
inblock()
{
      int j, jj;
                char sst[9];
      j = 0;
                while(j < maxlns)
      { jj = 0;
            while(jj < lcount)
            { if (ul[jj] == j)
                  {flage = 1; break;}
               else {++jj; flage = 0;}
                }

                 if (flage == 0 && jj >= lcount)
       {
            jl = j;
            ul[lcount] = jl;
            ++lcount;
                              strcpy (sst, "SLOT");
                              fprintf (blw, "%s\n", sst);
                     if (line[jl].x == line[jl].x1)
            {
                               if (line[jl].y < line[jl].y1)
               { CONX = line[jl].x1;
                                                  CONY = line[jl].y1;}
             else { CONX = line[jl].x; CONY = line[jl].y;}
             }
            else
```

```
                                {
            if (line[jl].x < line[jl].x1)
              { CONX = line[jl].x1; CONY = line[jl].y1;}
            else { CONX = line[jl].x; CONY = line[jl].y; }
                                }
          block();
                                fprintf (jfw, "%s %f %f %f %f\n", square[b].st2,
square[b].l, square[b].h, square[b].x, square[b].y);
                                fprintf (blw, "%s %f %f %f %f\n", square[b].st2,
square[b].l, square[b].h, square[b].x, square[b].y);
       ++b; ++j; }
              else ++j;
               }
}
/*******************/
block()
{
              struct blpath inblock[5];
     int bj;
     bj = 1;
     fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);
              blockconect();
              ++lcount;
              while (bj < 4)
     {
     inblock[bj].x = CONX;
     inblock[bj].y = CONY;
     fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);
              if (bj == 3 && ( inblock[1].x == CONX || inblock[1].y == CONY))
              {break;}
              getblock();
     ++bj;
     }
              if (CONX == line[jl].x && CONY == line[jl].y)
                { CONX = line[jl].x1; CONY = line[jl].y1;}
              else { CONX = line[jl].x; CONY = line[jl].y; }
              inblock[bj].x = CONX;
              inblock[bj].y = CONY;

     strcpy (square[b].st2, "SQUARE");
              fprintf(jfw, "inblockx1 %f, inblocky1 %f, inblockx2 %f, inblocky2
%f\n", inblock[1].x, inblock[1].y, inblock[2].x, inblock[2].y);
```

```
            fprintf(jfw, "inblockx3 %f, inblocky3 %f, inblockx4 %f, inblocky4
%f\n", inblock[3].x, inblock[3].y, inblock[4].x, inblock[4].y);

            for (j = 0; j <= lcount; j++)
    { fprintf (jfw, "j %d, ul %d\n", j, ul[j]);}

            if (inblock[1].x == inblock[2].x)

    { if ((inblock[1].x - inblock[3].x) > 0)
    square[b].l = (inblock[1].x - inblock[3].x);
    else square[b].l = (inblock[3].x - inblock[1].x);

    if ((inblock[1].y - inblock[2].y) > 0)
    square[b].h = (inblock[1].y - inblock[2].y);
    else square[b].h = (inblock[2].y - inblock[1].y);


    if (inblock[1].x < inblock[4].x)
    square[b].x = inblock[1].x + (square[b].l / 2.0);
    else
    square[b].x = inblock[1].x - (square[b].l / 2.0);

    if (inblock[1].y < inblock[2].y)
    square[b].y = inblock[1].y + (square[b].h / 2.0);
    else
    square[b].y = inblock[1].y - (square[b].h / 2.0);
    }

    else { if ((inblock[1].x - inblock[2].x) > 0)
    square[b].l = (inblock[1].x - inblock[2].x);
    else square[b].l = (inblock[2].x - inblock[1].x);

    if ((inblock[1].y - inblock[3].y) > 0)
    square[b].h = (inblock[1].y - inblock[3].y);
    else square[b].h = (inblock[3].y - inblock[1].y);


    if (inblock[1].x < inblock[2].x)
            square[b].x = inblock[1].x + (square[b].l / 2.0);
    else
    square[b].x = inblock[1].x - (square[b].l / 2.0);
```

```
        if (inblock[1].y < inblock[4].y)
        square[b].y = inblock[1].y + (square[b].h / 2.0);
        else
        square[b].y = inblock[1].y - (square[b].h / 2.0);
        }
                return(b);
}

/
*************************************************************************
*******
                                        Find the line length and shortest line #
*************************************************************************
*******/
shortl()
{
                int ln;
                float L, M, N;
                double d, pow();

                if (maxlns >0 )
                for (j = 0; j < maxlns; j++)
                {
                L = (line[j].x - line[j].x1)*(line[j].x - line[j].x1);
                M = (line[j].y - line[j].y1)*(line[j].y - line[j].y1);
                N = (line[j].z - line[j].z1)*(line[j].z - line[j].z1);

                d = (L + M + N);

                D[j] = pow (d, .5);
                fprintf (fpw, "%f\n", D[j]);
                }
                fprintf (jfw, "Shortest length is at line #");
                Dsh = D[0];
                                for ( j = 0; j < maxlns; j++)
                                { memset(buffer, '\0', 80);
                fgets (buffer, 80, fpr);
                                sscanf(buffer, "%f", &D[j]);
                                if (Dsh > D[j] )
                                {Dsh = D[j];
                                ln = j + 1;
                                fprintf (jfw, "%d,", ln);}
                                }
```

```
                fprintf (jfw, " Shortest length is %f\n", Dsh);
                return(Dsh);
}

/*****************
                Find the feature around the border of the part
******************/
strlines()
{
                n = 0;
                for (j = 0; j < totlns; j++)
                {
                 if (ln[j].y == YMIN || ln[j].y1 == YMIN )

                                { fprintf (jfw, "%d\n", j);
                                n = n +1;
                                stl[n] = j;
                                }
                }
                fprintf (jfw, "n = %d\n", n);
                return(n);
}


/***********get lowx*********/
getlowx(n)
int n;
{
                n = 1;
                j = stl[n];
                if (ln[j].x < ln[j].x1)
                                XLOW = ln[j].x;
                else XLOW = ln[j].x1;

                                for (n = 2; stl[n] != NULL; n++)
                { j = stl[n];
                if (ln[j].x <= ln[j].x1)
                                {if (XLOW > ln[j].x)
                                XLOW = ln[j].x;
                                }
                else {if (XLOW > ln[j].x1)
                                XLOW = ln[j].x1;
```

```
                    }
                }
                fprintf (jfw, "lowx is %f \n", XLOW);
                return(n);
}


/***********get lowz*********/
getzlow(n)
int n;
{
      n = 1;
      j = stl[n];
      if (ln[j].z < ln[j].z1)
            ZLOW = ln[j].z;
      else ZLOW = ln[j].z1;

            for (n = 2; stl[n] != NULL; n++)
      { j = stl[n];
      if (ln[j].z <= ln[j].z1)
            {if (ZLOW > ln[j].z)
            ZLOW = ln[j].z;
            }
      else {if (ZLOW > ln[j].z1)
            ZLOW = ln[j].z1;
            }
      }
      fprintf (jfw, "lowz is %f \n", ZLOW);
      return(n);
}

/****************lowx line numbers **************/
lowxlns(n)
int n;
{
                int jj;
                fprintf (jfw, "n = %d\n", n);
                c = 0;
                for (jj = 1; jj <= n; jj++)
                {
                              j = stl[jj];
                if (ln[j].z == ZLOW || ln[j].z1 == ZLOW)
                   {
```

```
                fprintf (jfw, "ln[j].x %f, x1 %f\n", ln[j].x, ln[j].x1);
            if (ln[j].x == XLOW || ln[j].x1 == XLOW)
                                { if (ln[j].y == YMIN || ln[j].y1 == YMIN)
                                                { c = c + 1;
                                   sk[c] = j; }
                                }
                    }
                }
                                fprintf (jfw, "c = %d\n", c);
                 return (c);
}

/******* starting line number**************/

strlnum()
{
                                for (n = 1; n <= c; n++)
                                {
                                j = sk[n];
                                fprintf (jfw, "j.x %f, j.x1 %f sk1 %d, sk2 %d\n",
line[j].x, line[j].x1, sk[1], sk[2]);
                                if (line[j].x != line[j].x1 && line[j].z == line[j].z1 &&
line[j].z == ZLOW )
                                                { jl = j; return(jl); }
                                }

}
/************starting line number for 3D part data*******/
tdstrlnum()
{
      int jj;
      c = 0;
                n = 0;
      for (j = 0; j < maxlns; j++)
       { if (line[j].y == YMIN || line[j].y1 == YMIN )
            {  fprintf (jfw, "%d\n", j);
            n = n +1;
            stl[n] = j; }
      }
      fprintf (jfw, "n = %d\n", n);
      for (jj = 1; jj <= n; jj++)
        { j = stl[jj];
      fprintf (jfw, "line[j].x %f, x1 %f\n", line[j].x, line[j].x1);
```

```
        if (line[j].x == XLOW || line[j].x1 == XLOW)
         { if (line[j].y == YMIN || line[j].y1 == YMIN)
              {  c = c + 1;
                 sk[c] = j; }
         }
              }
            strlnum();
}
/*******************base data *************/
base()
{
            struct bpath base;
            strcpy (base.st1, "BASE");
     base.x = (XMAX - XMIN);
     base.y = (YMAX - YMIN);
     base.th = th;
            base.x1 = (XMIN + (base.x/2.0));
            base.y1 = (YMIN + (base.y/2.0));
    fprintf (blw, "%s %f %f %f %f %f\n", base.st1, base.x, base.y, base.th, base.x1,
base.y1);
}

/***************
                        Get started to find all featurs around border
****************************/

xmaxfeature()

{
                        if (jl != NULL)
                        fprintf(jfw, "jl %d\n", jl);
                        ji = jl;
                                if (line[jl].x < line[jl].x1)
                        { CONX = line[jl].x1; CONY = line[jl].y1;}
                        else { CONX = line[jl].x; CONY = line[jl].y; }
            lcount = 0;
            b = 1;
            ul[lcount] = jl;
            fprintf(jfw, "ul %d, lcount %d jl %d\n", ul[lcount], lcount, jl);
            ++lcount;

            conect ();
            ++lcount;
```

```
                fprintf (jfw, "you have return value of jl is %d, ji is %d\n", jl, ji);
                fprintf (jfw, "line[jl].x %f, line[jl].x1 %f\n", line[jl].x, line[jl].x1);
                if (CONX == XMAX)
                return(jl);
/**********starting to find block on line from xmin to xmax*****/

      while ((lcount < maxlns) && (XMAX != CONX))
      {
      fprintf (jfw, "you have return value of jl is %d, ji is %d\n", jl, ji);
                bl = 0;
                if (jl != ji)
                {
                comparx ();
                fprintf (jfw, "value of junk %d\n", junk);
                if (junk == 1 )
                {
                getconxy();
                  }
                else if (junk == 2)
                 { cadblockxmax ();
                  fprintf (blw, "%s %f %f %f %f\n", square[b].st2, square[b].l,
square[b].h, square[b].x, square[b].y);
                  ++b; }

                else fprintf (jfw, "There is error in the data file\n");
                }
                }
                fprintf (jfw, "conx %f and cony %f jl %d\n", CONX, CONY, jl);
                fprintf (jfw, "end of block on line from xmin to xmax\n");
}
/*********** other end ********/
getconxy()
{               if ( CONX == line[jl].x && CONY == line[jl].y)
                                {CONX = line[jl].x1; CONY = line[jl].y1; }
                else {CONX = line[jl].x; CONY = line[jl].y; }
                conect();
                ++lcount;
}

 /*********** other end for inner block line********/
getblock()
{       if ( CONX == line[jl].x && CONY == line[jl].y)
            {CONX = line[jl].x1; CONY = line[jl].y1; }
```

```
      else {CONX = line[jl].x; CONY = line[jl].y; }
      blockconect();
      ++lcount;
}
/*** Get connected line para for inner block********/
blockconect()
{
      int jk, con;
            con = 0; fprintf(jfw, "jl %d\n", jl);
       for (j = 0; j < maxlns; j++)
            {
                  if (j == jl)
                  j = j +1;
                  else j = j;

      if ((line[j].x == CONX && line[j].y == CONY) || (line[j].x1 == CONX &&
line[j].y1 == CONY))
                  { jl = j; con = 1;
                  break;}
            }
                  if (con == 0)
                  { blarconect(jl);
                                          jk = ul[lcount - 1];
                              if (line[jk].x != line[jk].x1)
                  { if (line[jl].x != line[jl].x1)
                  { ul[lcount] = jl;
                                                      ++lcount; getblock();
return (jl);}
                   else { ul[lcount] = jl; return(jl); }
                                          }
                              else { if (line[jl].x == line[jl].x1)
                  { ul[lcount] = jl; ++lcount;
                                                      getblock(); return (jl); }
                                    else { ul[lcount] = jl; return(jl); }
                                          }
                                    }
                  else if (con == 1)
                                          ul[lcount] = jl;
      fprintf (jfw, "lcount %d ul %d jl %d\n", lcount, ul[lcount], jl);
                  return (jl);
}

/*************************************/
```

```
blarconect(jl)
{
      struct ppath midp;

      for (j =0; j < maxarc; j++)
            {
            if (arc[j].x == CONX && arc[j].y == CONY || arc[j].x1 == CONX &&

 arc[j].y1 == CONY || arc[j].x2 == CONX && arc[j].y2 == CONY)
      break;
      }
      fprintf (jfw, "jl %d, j %d\n", jl, j);

      if (line[jl].x == line[jl].x1)
        {
        if (arc[j].x == arc[j].x1)
          { midp.x = arc[j].x2; midp.y = arc[j].y2;midp.z = arc[j].z2;}
        else if (arc[j].x == arc[j].x2)
        { midp.x = arc[j].x1; midp.y = arc[j].y1;midp.z = arc[j].z1;}
        else
         { midp.x = arc[j].x;
           midp.y = arc[j].y; midp.z = arc[j].z; }

        if ((line[jl].y == CONY && line[jl].y < line[jl].y1) || (line[jl].y1

 == CONY && line[jl].y1 < line[jl].y))
          { if(midp.x > CONX)
             strcpy (cyl.st1, "CYL");
            else strcpy (cyl.st1, "HOLE");
          }

        else { if(midp.x < CONX)
                 strcpy (cyl.st1, "CYL");
             else strcpy (cyl.st1, "HOLE"); }
         if (CONY != arc[j].y && midp.y != arc[j].y)
           { CONX = arc[j].x;
            CONY = arc[j].y;
           }
         else if (CONY != arc[j].y1 && midp.y != arc[j].y1)
           {  CONX = arc[j].x1;
             CONY = arc[j].y1; }
         else {  CONX = arc[j].x2;
```

```
                                CONY == arc[j].y2; }
                    }
        else {
            if (arc[j].y == arc[j].y1)
            { midp.x = arc[j].x2; midp.y = arc[j].y2;midp.z = arc[j].z2;}
           else if (arc[j].y == arc[j].y2)
            { midp.x = arc[j].x1; midp.y = arc[j].y1;midp.z = arc[j].z1;}
           else if (arc[j].y1 == arc[j].y2)
            { midp.x = arc[j].x; midp.y = arc[j].y; midp.z = arc[j].z;}

           if ((line[jl].x == CONX && line[jl].x < line[jl].x1) || (line[jl].x1

== CONX && line[jl].x1 < line[jl].x))
             { if(midp.y < CONY)
               strcpy (cyl.st1, "CYL");
              else strcpy (cyl.st1, "HOLE");
             }
           else { if(midp.y > CONY)
                  strcpy (cyl.st1, "CYL");
                 else strcpy (cyl.st1, "HOLE"); }

           if (CONX != arc[j].x && midp.x != arc[j].x)
              { CONX = arc[j].x;
                CONY = arc[j].y;
              }
           else if (CONY != arc[j].x1 && midp.x != arc[j].x1)
              {  CONX = arc[j].x1;
                 CONY = arc[j].y1; }
           else {  CONX = arc[j].x2;
              CONY = arc[j].y2; }
            }
        fprintf(jfw, "conx = %f, cony = %f\n", CONX, CONY);
        cyl.d = arc[j].d;
        cyl.x = arc[j].cx;
        cyl.y = arc[j].cy;
    fprintf (jfw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);
        fprintf (blw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);
        blockconect();
        return(jl);


}
```

```
/**********************
                starting to find block on line start at xmax end at ymax
**********************/
ymaxfeature()
{
                getconxy();
                while ((lcount < maxlns) && (CONY != YMAX))
                {
                fprintf (jfw, "you have return value of jl is %d, ji is %d\n", jl, ji);
                bl = 0;
      if (jl != ji)
      {
      compary ();
      fprintf (jfw, "value of junk %d\n", junk);
      if (junk == 1 )
      {
       getconxy();
                  }
      else if (junk == 2)
       { cadblockymax ();
                     fprintf (blw, "%s %f %f %f %f\n", square[b].st2, square[b].l,
square[b].h, square[b].x, square[b].y);
        ++b;}
                else fprintf (jfw, "There is error in the data file\n");
                }
                }
      fprintf (jfw, "end of block on line from xmax to ymax\n");
}



/****************
                starting to find block on line start at ymax end at xmin
****************/

xminfeature()
{
                getconxy();
                while ((lcount < maxlns) && (XMIN != CONX))
                {
                fprintf (jfw, "you have return value of jl is %d, ji is %d\n", jl, ji);

                bl = 0;
```

```
        if (jl != ji)
        {
        comparx ();
        fprintf (jfw, "value of junk %d\n", junk);
        if (junk == 1 )
        {
                getconxy();
                 }
        else if (junk == 2)
                { cadblockxmin ();
                fprintf (blw, "%s %f %f %f %f\n", square[b].st2, square[b].l,
square[b].h, square[b].x, square[b].y);
         ++b; }

                else fprintf (jfw, "There is error in the data file\n");
        }
        }
        fprintf (jfw, "end of block on line from ymax to xmin\n");
}

/****************
        starting to find block on line start at xmin end at ymin
****************/
yminfeature()
{
                int j;
                getconxy();
                while ((lcount <= maxlns) && (CONY != YMIN))
                {
        fprintf (jfw, "you have return value of jl is %d, ji is %d\n", jl, ji);
                bl = 0;
        if (jl != ji)
        {
        compary ();
        fprintf (jfw, "value of junk %d\n", junk);
        if (junk == 1 )
        {
                getconxy();
                }
        else if (junk == 2)
                { cadblockymin ();
                 fprintf (blw, "%s %f %f %f %f\n", square[b].st2, square[b].l,
square[b].h, square[b].x, square[b].y);
```

```
        ++b;}

                else fprintf (jfw, "There is error in the data file\n");
        }
        }
                if (CONY == YMIN)
                  lcount = lcount - 1;
                else lcount = lcount;
        fprintf (jfw, "end of block on line from xmin to ymin\n");
                fprintf (jfw, "lcount %d, ul %d, jl %d\n", lcount, ul[lcount], jl);
                for (j = 0; j <= lcount; j++)
                { fprintf (jfw, "j %d, ul %d\n", j, ul[j]);}
}

/*********************************/

/
**********************************************************************
**
                Find the origin pt

**********************************************************************
****/

originpt()

{
        if (maxlns > 0)
           for (j = 0; j < maxlns; j++)

{
/****
fprintf (jfw,
"%f,%f,%f,%f,%f,%f\n",line[j].x,line[j].y,line[j].z,line[j].x1,line[j].y1,line[j].z1);
**********/


                if (line[j].x == 0.0000 && line[j].y == 0.0000 && line[j].z == 0.0000
)
                { oript.x = line[j].x;
                oript.y = line[j].y;
                oript.z = line[j].z;
                fprintf (jfw, "Origin is at %f, %f, %f \n", oript.x, oript.y, oript.z);
```

```
                flage = 1;}
                else if  (line[j].x1 == 0.0000 && line[j].y1 == 0.0000 && line[j].z1
== 0.0000 )
                { oript.x = line[j].x1;
        oript.y = line[j].y1;
        oript.z = line[j].z1;
                fprintf (jfw, "Origin is at %f, %f, %f \n", oript.x, oript.y, oript.z);
                flage = 1;}
}
                return(flage);
}

/*** Get connected line para ********/
conect()
{
                int con;
                                con = 0; fprintf(jfw, "jl %d\n", jl);
                 for (j = 0; j < maxlns; j++)
                                {
                                                 if (j == jl)
                                                j = j +1;
                                                else j = j;

                if ((line[j].x == CONX && line[j].y == CONY) || (line[j].x1 ==
CONX && line[j].y1 == CONY))
                                                { jl = j;
                                                con = 1;
                                                break;}
                                }
                                                if (con == 0)
                                                { arconect(jl); }

                                ul[lcount] = jl;
                fprintf (jfw, "lcount %d ul %d jl %d\n", lcount, ul[lcount], jl);
                                                return (jl);
}

/*****************************************/

arconect(jl)
{
                struct ppath midp;
```

```
                for (j =0; j < maxarc; j++)
                                {
                                if (arc[j].x == CONX && arc[j].y == CONY || arc[j].x1
== CONX && arc[j].y1 == CONY || arc[j].x2 == CONX && arc[j].y2 == CONY)

                break;
                }
                fprintf (jfw, "jl %d, j %d\n", jl, j);

                if (line[jl].x == line[jl].x1)
                  {
                   if (arc[j].x == arc[j].x1)
                     { midp.x = arc[j].x2; midp.y = arc[j].y2;midp.z = arc[j].z2;}
                   else if (arc[j].x == arc[j].x2)
           { midp.x = arc[j].x1; midp.y = arc[j].y1;midp.z = arc[j].z1;}
                   else
           { midp.x = arc[j].x;
                                midp.y = arc[j].y; midp.z = arc[j].z; }

                  if ((line[jl].y == CONY && line[jl].y < line[jl].y1) || (line[jl].y1 ==
CONY && line[jl].y1 < line[jl].y))
                    { if(midp.x > CONX)
                       strcpy (cyl.st1, "HOLE");
                                else strcpy (cyl.st1, "CYL");
                    }
                   else { if(midp.x < CONX)
                  strcpy (cyl.st1, "HOLE");
              else strcpy (cyl.st1, "CYL"); }
             if (CONY != arc[j].y && midp.y != arc[j].y)
               { CONX = arc[j].x;
                 CONY = arc[j].y;
               }
             else if (CONY != arc[j].y1 && midp.y != arc[j].y1)
               {  CONX = arc[j].x1;
                  CONY = arc[j].y1; }
             else {  CONX = arc[j].x2;
               CONY = arc[j].y2; }
                   }
                  else {
                     if (arc[j].y == arc[j].y1)
           { midp.x = arc[j].x2; midp.y = arc[j].y2;midp.z = arc[j].z2;}
          else if (arc[j].y == arc[j].y2)
           { midp.x = arc[j].x1; midp.y = arc[j].y1;midp.z = arc[j].z1;}
```

```
        else if (arc[j].y1 == arc[j].y2)
         { midp.x = arc[j].x; midp.y = arc[j].y; midp.z = arc[j].z;}

                    if ((line[jl].x == CONX && line[jl].x < line[jl].x1) || (line[jl].x1 ==
CONX && line[jl].x1 < line[jl].x))
           { if(midp.y < CONY)
             strcpy (cyl.st1, "HOLE");
            else strcpy (cyl.st1, "CYL");
           }
        else { if(midp.y > CONY)
               strcpy (cyl.st1, "HOLE");
             else strcpy (cyl.st1, "CYL"); }

                    if (CONX != arc[j].x && midp.x != arc[j].x)
           { CONX = arc[j].x;
             CONY = arc[j].y;
           }
        else if (CONY != arc[j].x1 && midp.x != arc[j].x1)
           {  CONX = arc[j].x1;
              CONY = arc[j].y1; }
        else {  CONX = arc[j].x2;
           CONY = arc[j].y2; }
                    }
                fprintf(jfw, "conx = %f, cony = %f\n", CONX, CONY);
                cyl.d = arc[j].d;
                cyl.x = arc[j].cx;
                cyl.y = arc[j].cy;
 fprintf (jfw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);
     fprintf (blw, "%s %f %f %f\n", cyl.st1, cyl.d, cyl.x, cyl.y);
                conect();
                return(jl);

}

/****** comperision of connected & new para ***************/
comparx()

{
                                if (line[jl].y == line[jl].y1)
                                                { junk = 1;
                                                }

                                else if ( line[jl].y != line[jl].y1 )
```

```
                { junk = 2;
                                        }

                        else junk = 0;
                        return (junk);
}

/*********************************

*********************************/

cadblockxmax ()

{
                struct blpath blpoint[50][4];

                ++bl;
                fprintf(jfw, "you reached up to cadblock %d\n", b);
                fprintf (jfw, "Block # %d is started\n", b);
                fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

                while (bl < 4)
                {
                blpoint[b][bl].x = CONX;
      blpoint[b][bl].y = CONY;

                fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);
                getconxy();
                ++bl;
                fprintf (jfw, "value of bl %d, lcount %d, blpt3 %f\n", bl, lcount,
blpoint[b][3].x );

                if ( CONX == XMAX)
                break;

                if (line[jl].x < CONX || line[jl].x1 < CONX)
                        { cadsubxmax (b, bl, 0);
       fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
                        if (flage == 3)
                        { blpoint[b][bl].x = blpoint[b][bl-1].x;
            blpoint[b][bl].y = CONY;
                        ++bl; flage = NULL;}
```

```
                        flage = NULL; }
                if (bl != 4 && (line[jl].y > CONY || line[jl].y1 > CONY))
                        { cadsubxmax (b, bl, 0);
                fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY,
bl, blpoint[b][2].x);
                        if (flage == 3)
                        { blpoint[b][bl].x = CONX;
                          blpoint[b][bl].y = blpoint[b][bl-1].y;
                          ++bl; flage = NULL;}
                        flage = NULL;
                        }
                }

                if (CONX == XMAX)
                {
                if (blpoint[b][1].x != 0.0 && blpoint[b][2].x != 0.0 && bl == 3)
                        {
                        blpoint[b][3].x = CONX;
                        blpoint[b][3].y = CONY;
                        blpoint[b][4].x = blpoint[b][3].x;
                        blpoint[b][4].y = blpoint[b][1].y;
                        }
                else if (bl == 2)
                        {
                        blpoint[b][2].x = blpoint[b][1].x;
                        blpoint[b][2].y = CONY;
                        blpoint[b][3].x = CONX;
                        blpoint[b][3].y = CONY;
                        blpoint[b][4].x = CONX;
                        blpoint[b][4].y = blpoint[b][1].y;
                        }
                }
                else {
                        blpoint[b][4].x = CONX;
                        blpoint[b][4].y = CONY;
                        }
                fprintf (jfw, "bl1.x %f, bl2.x %f, bl3.x %f, bl4.x %f\n",
blpoint[b][1].x, blpoint[b][2].x, blpoint[b][3].x, blpoint[b][4].x);
                fprintf (jfw, "end of the block # %d\n", b);

                strcpy (square[b].st2, "SQUARE");

                if ((blpoint[b][1].x - blpoint[b][4].x) > 0)
```

```
                square[b].l = (blpoint[b][1].x - blpoint[b][4].x);
                else square[b].l = (blpoint[b][4].x - blpoint[b][1].x);

                if ((blpoint[b][1].y - blpoint[b][2].y) > 0)
                square[b].h = (blpoint[b][1].y - blpoint[b][2].y);
                else square[b].h = (blpoint[b][2].y - blpoint[b][1].y);


                if (blpoint[b][1].x < blpoint[b][4].x)
                square[b].x = blpoint[b][1].x + (square[b].l / 2.0);
                else
                square[b].x = blpoint[b][1].x - (square[b].l / 2.0);

                if (blpoint[b][1].y < blpoint[b][2].y)
                square[b].y = blpoint[b][1].y + (square[b].h / 2.0);
                else
                square[b].y = blpoint[b][1].y - (square[b].h / 2.0);
                return (jl);
}

/*******************cadsubxmax**************/

cadsubxmax (sb, bl, sbl)

int sb, bl, sbl;

{
        struct blpath blpt[50][100][4];

                if (sbl == 0)
                sbl = 1;
        fprintf(jfw, "you reached up to subcadblock %d\n", sb);
        fprintf (jfw, "Block # %d - %dis started\n", b, sb);
        fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

        while (sbl < 4)
        {
        blpt[b][sb][sbl].x = CONX;
        blpt[b][sb][sbl].y = CONY;
                getconxy();
        ++sbl;

        fprintf (jfw, "value of sbl %d, lcount %d, blpt3 %f\n", sbl, lcount,
```

```
blpt[b][sb][3].x );

      if ( CONX == XMAX )
      break;
      }

      if (CONX == XMAX && sbl == 4)
                                 {
                                 blpt[b][sb][4].x = blpt[b][sb][1].x;
          blpt[b][sb][4].y = blpt[b][sb][3].y;
                                 flage = 1;}


               else if (CONX == XMAX && sbl ==3)
        {
          blpt[b][sb][3].x = CONX;
          blpt[b][sb][3].y = CONY;
          blpt[b][sb][4].x = CONX;
          blpt[b][sb][4].y = blpt[b][sb][1].y;
                                 flage = 1;
               fprintf (jfw, "bl %d, flage %d\n", bl, flage);
                  }

                else if (CONX != XMAX && sbl == 4)
                {
                 if (blpt[b][sb][1].y == blpt[b][sb][2].y)
                 {
                  if (blpt[b][sb][1].x > CONX)
                   {
          blpt[b][sb][4].x = CONX;
          blpt[b][sb][4].y = CONY;
                                 blpt[b][sb][0].x = blpt[b][sb][1].x;
                                 blpt[b][sb][1].x = CONX;

      fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
      fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

      strcpy (square[sb].st2, "SQUARE");

      if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
      square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
      else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);
```

```
if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

if (blpt[b][sb][1].x < blpt[b][sb][2].x)
square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
else
square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

if (blpt[b][sb][1].y < blpt[b][sb][4].y)
square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
else
square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

 fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
            fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l,
square[sb].h, square[sb].x, square[sb].y);

                        ++sb;
        blpt[b][sb][1].x = blpt[b][sb-1][0].x;
                        blpt[b][sb][1].y = blpt[b][sb-1][1].y;
                        blpt[b][sb][2].x = blpt[b][sb-1][1].x;
                        blpt[b][sb][2].y = blpt[b][sb-1][1].y;
                        sbl = 3;
                        getconxy();
                        cadsubxmax(sbl);
                        return(bl);
                        }
           else if (CONX == blpt[b][sb][1].x && CONY == blpt[b][sb][3].y)
                        {
                         blpt[b][sb][4].x = CONX;
            blpt[b][sb][4].y = CONY;
                        getconxy();
                        }
           else  { blpt[b][sb][4].x = blpt[b][sb][1].x;
           blpt[b][sb][4].y = blpt[b][sb][3].y;
                        flage = 3;
                        }
                        }

           else {
```

```
                    if (blpt[b][sb][1].y < CONY)
          {
           blpt[b][sb][4].x = CONX;
           blpt[b][sb][4].y = CONY;
           blpt[b][sb][0].y = blpt[b][sb][1].y;
           blpt[b][sb][1].y = CONY;

     fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
     fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

     strcpy (square[sb].st2, "SQUARE");

     if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
     square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
     else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);

     if ((blpt[b][sb][1].y - blpt[b][sb][2].y) > 0)
     square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);
     else square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);

     if (blpt[b][sb][1].x < blpt[b][sb][4].x)
     square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
     else
     square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

     if (blpt[b][sb][1].y < blpt[b][sb][2].y)
     square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
     else
     square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
          ++sb;
          blpt[b][sb][1].x = blpt[b][sb-1][1].x;
          blpt[b][sb][1].y = blpt[b][sb-1][0].y;
          blpt[b][sb][2].x = blpt[b][sb-1][1].x;
          blpt[b][sb][2].y = blpt[b][sb-1][1].y;
          sbl = 3;
                         getconxy();
                         cadsubxmax(sbl);
```

```
                        return(bl);
                        }

                else if (CONX == blpt[b][sb][3].x && CONY == blpt[b][sb][1].y)
        {
         blpt[b][sb][4].x = CONX;
         blpt[b][sb][4].y = CONY;
        getconxy();
                        }
                else  {
                        blpt[b][sb][4].x = blpt[b][sb][3].x;
        blpt[b][sb][4].y = blpt[b][sb][1].y;
                        flage =3;}
            }
            }
    fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
    fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

    strcpy (square[sb].st2, "SQUARE");

            if (blpt[b][sb][1].y == blpt[b][sb][4].y)
            { if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
    square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
    else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);

    if ((blpt[b][sb][2].y - blpt[b][sb][1].y) > 0)
    square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);
    else square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);


    if (blpt[b][sb][1].x < blpt[b][sb][4].x)
    square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
    else
    square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

    if (blpt[b][sb][1].y < blpt[b][sb][2].y)
    square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
    else
    square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
            }

     else { if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
```

```
        square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
        else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);

        if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
        square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
        else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);


        if (blpt[b][sb][1].x < blpt[b][sb][2].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][4].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
                }

 fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

                return(bl);
}

/****** comperision of connected & new para ***************/
compary()

{
        if (line[jl].x == line[jl].x1)
                {  junk = 1;
                }

        else if ( line[jl].x != line[jl].x1 )
                 { junk = 2;
                 }

        else junk = 0;
        return (junk);
}
```

```
/************************************/

cadblockymax ()
{
      struct blpath blpoint[50][4];

      ++bl;
                  fprintf(jfw, "you reached up to cadblock %d\n", b);
                  fprintf (jfw, "Block # %d is started\n", b);
                  fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);


      while (bl < 4)
      {
      blpoint[b][bl].x = CONX;
      blpoint[b][bl].y = CONY;

      fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

      getconxy();
                  ++bl;

     fprintf (jfw, "value of bl %d, lcount %d, blpt3 %f\n", bl, lcount, blpoint[b][3].y);

                  if ( CONY == YMAX)
      break;

                  if (line[jl].y < CONY || line[jl].y1 < CONY)
            { cadsubymax (b, bl, 0);
       fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
            if (flage == 3)
            { blpoint[b][bl].y = blpoint[b][bl-1].x;
            blpoint[b][bl].x = CONX;
            ++bl; flage = NULL;}
            flage = NULL; }
      if (bl != 4 && (line[jl].x < CONX || line[jl].x1 < CONX))
            { cadsubymax (b, bl, 0);
      fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
            if (flage == 3)
            { blpoint[b][bl].y = CONX;
              blpoint[b][bl].x = blpoint[b][bl-1].x;
```

```
        ++bl; flage = NULL;}
      flage = NULL;
      }
}

if (CONY == YMAX)
        {
        if (blpoint[b][1].y != 0.0 && blpoint[b][2].y != 0.0 && bl == 3)
  {
    blpoint[b][3].x = CONX;
    blpoint[b][3].y = CONY;
    blpoint[b][4].x = blpoint[b][1].x;
    blpoint[b][4].y = blpoint[b][3].y;
    }
      else if (bl == 2)
    {
    blpoint[b][2].y = blpoint[b][1].y;
    blpoint[b][2].x = CONX;
    blpoint[b][3].x = CONX;
    blpoint[b][3].y = CONY;
    blpoint[b][4].y = CONY;
    blpoint[b][4].x = blpoint[b][1].x;
    }
  }
      else          {
                    blpoint[b][4].x = CONX;
                    blpoint[b][4].y = CONY;
                    }

fprintf (jfw, "bl1.x %f, bl2.x %f, bl3.x %f, bl4.x %f\n", blpoint[b][1].x,
blpoint[b][2].x, blpoint[b][3].x, blpoint[b][4].x);
fprintf (jfw, "end of the block # %d\n", b);

strcpy (square[b].st2, "SQUARE");

if ((blpoint[b][1].x - blpoint[b][2].x) > 0)
square[b].l = (blpoint[b][1].x - blpoint[b][2].x);
else square[b].l = (blpoint[b][2].x - blpoint[b][1].x);

if ((blpoint[b][1].y - blpoint[b][4].y) > 0)
square[b].h = (blpoint[b][1].y - blpoint[b][4].y);
else square[b].h = (blpoint[b][4].y - blpoint[b][1].y);
```

```
                if (blpoint[b][1].x > blpoint[b][2].x)
        square[b].x = blpoint[b][1].x - (square[b].l / 2.0);
        else
        square[b].x = blpoint[b][1].x + (square[b].l / 2.0);

        if (blpoint[b][1].y < blpoint[b][4].y)
        square[b].y = blpoint[b][1].y + (square[b].h / 2.0);
        else
        square[b].y = blpoint[b][1].y - (square[b].h / 2.0);
        return (b);
}

/************************/
cadblockxmin ()
{
        extern b;
        struct blpath blpoint[50][4];

        ++bl;
        fprintf(jfw, "you reached up to cadblock %d\n", b);
        fprintf (jfw, "Block # %d is started\n", b);
        fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

        while (bl < 4)
        {
        blpoint[b][bl].x = CONX;
        blpoint[b][bl].y = CONY;

        getconxy();
        ++bl;
      fprintf (jfw, "value of bl %d, lcount %d, blpt3 %f\n", bl, lcount, blpoint[b][3].x );

        if (CONX == XMIN)
        break;

        if (line[jl].x > CONX || line[jl].x1 > CONX)
              { cadsubxmin (b, bl, 0);
         fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
              if (flage == 3)
              { blpoint[b][bl].x = blpoint[b][bl-1].x;
              blpoint[b][bl].y = CONY;
              ++bl; flage = NULL;}
```

```
            flage = NULL; }
        if (bl != 4 && (line[jl].y < CONY || line[jl].y1 < CONY))
            { cadsubxmin (b, bl, 0);
        fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
            if (flage == 3)
            { blpoint[b][bl].x = CONX;
              blpoint[b][bl].y = blpoint[b][bl-1].y;
              ++bl; flage = NULL;}
            flage = NULL;
            }
        }

        if (CONX == XMIN)
        {
        if (blpoint[b][1].x != 0.0 && blpoint[b][2].x != 0.0 && bl == 3)
          {
            blpoint[b][3].x = CONX;
            blpoint[b][3].y = CONY;
            blpoint[b][4].x = blpoint[b][3].x;
            blpoint[b][4].y = blpoint[b][1].y;
            }
                    else if (bl == 2)
            {
            blpoint[b][2].x = blpoint[b][1].x;
            blpoint[b][2].y = CONY;
            blpoint[b][3].x = CONX;
            blpoint[b][3].y = CONY;
            blpoint[b][4].x = CONX;
            blpoint[b][4].y = blpoint[b][1].y;
            }
          }

        else {
            blpoint[b][4].x = CONX;
            blpoint[b][4].y = CONY;
            }
        fprintf (jfw, "bl1.x %f, bl2.x %f, bl3.x %f, bl4.x %f\n", blpoint[b][1].x,
blpoint[b][2].x, blpoint[b][3].x, blpoint[b][4].x);
        fprintf (jfw, "end of the block # %d\n", b);

        strcpy (square[b].st2, "SQUARE");
```

```
        if ((blpoint[b][1].x - blpoint[b][4].x) > 0)
        square[b].l = (blpoint[b][1].x - blpoint[b][4].x);
        else square[b].l = (blpoint[b][4].x - blpoint[b][1].x);

        if ((blpoint[b][1].y - blpoint[b][2].y) > 0)
        square[b].h = (blpoint[b][1].y - blpoint[b][2].y);
        else square[b].h = (blpoint[b][2].y - blpoint[b][1].y);


        if (blpoint[b][1].x < blpoint[b][4].x)
        square[b].x = blpoint[b][1].x + (square[b].l / 2.0);
        else
        square[b].x = blpoint[b][1].x - (square[b].l / 2.0);

        if (blpoint[b][1].y < blpoint[b][2].y)
        square[b].y = blpoint[b][1].y + (square[b].h / 2.0);
        else
        square[b].y = blpoint[b][1].y - (square[b].h / 2.0);
        return (b);
}
/************************/
cadblockymin ()
{
        struct blpath blpoint[50][4];

        ++bl;
        fprintf(jfw, "you reached up to cadblock %d\n", b);
        fprintf (jfw, "Block # %d is started\n", b);
        fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);


        while (bl < 4)
        {
        blpoint[b][bl].x = CONX;
        blpoint[b][bl].y = CONY;

        fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

        getconxy();
        ++bl;

       fprintf (jfw, "value of bl %d, lcount %d, blpt3 %f\n", bl, lcount, blpoint[b][3].y);
```

```
if (CONY == YMIN)
break;

        if (line[jl].y > CONY || line[jl].y1 > CONY)
    { cadsubymin (b, bl, 0);
 fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
    if (flage == 3)
    { blpoint[b][bl].y = blpoint[b][bl-1].y;
    blpoint[b][bl].x = CONX;
    ++bl; flage = NULL;}
    flage = NULL; }
if (bl != 4 && (line[jl].x > CONX || line[jl].x1 > CONX))
    { cadsubymin (b, bl, 0);
fprintf (jfw, "conx = %f, cony %f, bl %d, blp2%f\n", CONX, CONY, bl,
blpoint[b][2].x);
    if (flage == 3)
    { blpoint[b][bl].y = CONY;
      blpoint[b][bl].x = blpoint[b][bl-1].x;
      ++bl; flage = NULL;}
    flage = NULL;
    }
}

if (CONY == YMIN)
{

if (blpoint[b][1].y != 0.0 && blpoint[b][2].y != 0.0 && bl ==3)
  {
    blpoint[b][3].x = CONX;
    blpoint[b][3].y = CONY;
    blpoint[b][4].x = blpoint[b][1].x;
    blpoint[b][4].y = blpoint[b][3].y;
    }
        else if (bl == 2)
    {
    blpoint[b][2].y = blpoint[b][1].y;
    blpoint[b][2].x = CONX;
    blpoint[b][3].x = CONX;
    blpoint[b][3].y = CONY;
    blpoint[b][4].y = CONY;
    blpoint[b][4].x = blpoint[b][1].x;
    }
```

```
                }
        else    {
                blpoint[b][4].x = CONX;
                blpoint[b][4].y = CONY;
                }

        fprintf (jfw, "bl1.x %f, bl2.x %f, bl3.x %f, bl4.x %f\n", blpoint[b][1].x,
blpoint[b][2].x, blpoint[b][3].x, blpoint[b][4].x);
        fprintf (jfw, "end of the block # %d\n", b);

        strcpy (square[b].st2, "SQUARE");

        if ((blpoint[b][1].x - blpoint[b][2].x) > 0)
        square[b].l = (blpoint[b][1].x - blpoint[b][2].x);
        else square[b].l = (blpoint[b][2].x - blpoint[b][1].x);

        if ((blpoint[b][1].y - blpoint[b][4].y) > 0)
        square[b].h = (blpoint[b][1].y - blpoint[b][4].y);
        else square[b].h = (blpoint[b][4].y - blpoint[b][1].y);

        if (blpoint[b][1].x > blpoint[b][2].x)
        square[b].x = blpoint[b][1].x - (square[b].l / 2.0);
        else
        square[b].x = blpoint[b][1].x + (square[b].l / 2.0);

        if (blpoint[b][1].y < blpoint[b][4].y)
        square[b].y = blpoint[b][1].y + (square[b].h / 2.0);
        else
        square[b].y = blpoint[b][1].y - (square[b].h / 2.0);

        return (b);
}
/*********************cad subblock for ymax*********/
cadsubymax (sb, bl, sbl)

int sb, bl, sbl;

{
        struct blpath blpt[50][100][4];

        if (sbl == 0)
        sbl = 1;
        fprintf(jfw, "you reached up to subcadblock %d\n", sb);
```

```
fprintf (jfw, "Block # %d - %dis started\n", b, sb);
fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

while (sbl < 4)
{
blpt[b][sb][sbl].x = CONX;
blpt[b][sb][sbl].y = CONY;
getconxy();
++sbl;

fprintf (jfw, "value of sbl %d, lcount %d, blpt3 %f\n", sbl, lcount,
blpt[b][sb][3].x );

if ( CONY == YMAX )
break;
}

if (CONY == YMAX && sbl == 4)
      {
      blpt[b][sb][4].x = blpt[b][sb][3].x;
      blpt[b][sb][4].y = blpt[b][sb][1].y;
      flage = 1;}


else if (CONY == YMAX && sbl ==3)
   {
      blpt[b][sb][3].x = CONX;
      blpt[b][sb][3].y = CONY;
      blpt[b][sb][4].y = CONY;
      blpt[b][sb][4].x = blpt[b][sb][1].x;
      flage = 1;
fprintf (jfw, "bl %d, flage %d\n", bl, flage);
    }
 else if (CONY != YMAX && sbl == 4)
 {
  if (blpt[b][sb][1].x == blpt[b][sb][2].x)
  {
   if (blpt[b][sb][1].y > CONY)
    {
      blpt[b][sb][4].x = CONX;
      blpt[b][sb][4].y = CONY;
      blpt[b][sb][0].y = blpt[b][sb][1].y;
      blpt[b][sb][1].y = CONY;
```

```
        fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
        fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

        strcpy (square[sb].st2, "SQUARE");

        if ((blpt[b][sb][1].x - blpt[b][sb][4].x) > 0)
        square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);
        else square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);

        if ((blpt[b][sb][1].y - blpt[b][sb][2].y) > 0)
        square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);
        else square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);

        if (blpt[b][sb][1].x < blpt[b][sb][4].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][2].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
        fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

                ++sb;
                blpt[b][sb][1].x = blpt[b][sb-1][1].x;
                blpt[b][sb][1].y = blpt[b][sb-1][0].y;
                blpt[b][sb][2].x = blpt[b][sb-1][1].x;
                blpt[b][sb][2].y = blpt[b][sb-1][1].y;
                sbl = 3;
                getconxy();
                cadsubymax(sbl);
                return(bl);
                }
        else if (CONX == blpt[b][sb][3].x && CONY == blpt[b][sb][1].y)
                {
                 blpt[b][sb][4].x = CONX;
```

```
            blpt[b][sb][4].y = CONY;
            getconxy();
            }
      else  { blpt[b][sb][4].x = blpt[b][sb][3].x;
            blpt[b][sb][4].y = blpt[b][sb][1].y;
            flage = 3;
            }
            }

      else {
            if (blpt[b][sb][1].x > CONX)
           {
            blpt[b][sb][4].x = CONX;
            blpt[b][sb][4].y = CONY;
            blpt[b][sb][0].x = blpt[b][sb][1].x;
            blpt[b][sb][1].x = CONX;

      fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
      fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

      strcpy (square[sb].st2, "SQUARE");

      if ((blpt[b][sb][2].x - blpt[b][sb][1].x) > 0)
      square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);
      else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);

      if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
      square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
      else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

      if (blpt[b][sb][1].x < blpt[b][sb][2].x)
      square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
      else
      square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

      if (blpt[b][sb][1].y < blpt[b][sb][4].y)
      square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
      else
      square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
```

```
fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
            ++sb;
            blpt[b][sb][1].x = blpt[b][sb-1][0].x;
            blpt[b][sb][1].y = blpt[b][sb-1][1].y;
            blpt[b][sb][2].x = blpt[b][sb-1][1].x;
            blpt[b][sb][2].y = blpt[b][sb-1][1].y;
            sbl = 3;
            getconxy();
            cadsubymax(sbl);
            return(bl);
            }

          else if (CONX == blpt[b][sb][1].x && CONY == blpt[b][sb][3].y)
            {
             blpt[b][sb][4].x = CONX;
             blpt[b][sb][4].y = CONY;
            getconxy();
            }
          else  {
            blpt[b][sb][4].x = blpt[b][sb][1].x;
            blpt[b][sb][4].y = blpt[b][sb][3].y;
            flage =3;}
       }
       }
       fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
       fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

       strcpy (square[sb].st2, "SQUARE");

       if (blpt[b][sb][1].y == blpt[b][sb][4].y)
       { if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
       square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
       else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);

       if ((blpt[b][sb][2].y - blpt[b][sb][1].y) > 0)
       square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);
       else square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);

       if (blpt[b][sb][1].x < blpt[b][sb][4].x)
       square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
       else
```

```
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][2].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
        }

         else { if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
        square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
        else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);

        if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
        square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
        else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

        if (blpt[b][sb][1].x < blpt[b][sb][2].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][4].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
        }

 fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

        return(bl);
}

/*******************cad sub block xmin*************/
cadsubxmin (sb, bl, sbl)

int sb, bl, sbl;
{
        struct blpath blpt[50][100][4];
```

```
if (sbl == 0)
sbl = 1;
fprintf(jfw, "you reached up to subcadblock %d\n", sb);
fprintf (jfw, "Block # %d - %dis started\n", b, sb);
fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

while (sbl < 4)
{
blpt[b][sb][sbl].x = CONX;
blpt[b][sb][sbl].y = CONY;
getconxy();
++sbl;

fprintf (jfw, "value of sbl %d, lcount %d, blpt3 %f\n", sbl, lcount,
blpt[b][sb][3].x );

if ( CONX == XMIN )
break;
}

if (CONX == XMIN && sbl == 4)
      {
      blpt[b][sb][4].x = blpt[b][sb][1].x;
      blpt[b][sb][4].y = blpt[b][sb][3].y;
      flage = 1;}


else if (CONX == XMIN && sbl ==3)
   {
      blpt[b][sb][3].x = CONX;
      blpt[b][sb][3].y = CONY;
      blpt[b][sb][4].x = CONX;
      blpt[b][sb][4].y = blpt[b][sb][1].y;
      flage = 1;
fprintf (jfw, "bl %d, flage %d\n", bl, flage);
    }

 else if (CONX != XMIN && sbl == 4)
 {
  if (blpt[b][sb][1].y == blpt[b][sb][2].y)
  {
   if (blpt[b][sb][1].x < CONX)
    {
```

```
            blpt[b][sb][4].x = CONX;
            blpt[b][sb][4].y = CONY;
            blpt[b][sb][0].x = blpt[b][sb][1].x;
            blpt[b][sb][1].x = CONX;

      fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
      fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

      strcpy (square[sb].st2, "SQUARE");

      if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
      square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
      else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);

      if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
      square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
      else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

      if (blpt[b][sb][1].x < blpt[b][sb][2].x)
      square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
      else
      square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

      if (blpt[b][sb][1].y < blpt[b][sb][4].y)
      square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
      else
      square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
      fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

            ++sb;
            blpt[b][sb][1].x = blpt[b][sb-1][0].x;
            blpt[b][sb][1].y = blpt[b][sb-1][1].y;
            blpt[b][sb][2].x = blpt[b][sb-1][1].x;
            blpt[b][sb][2].y = blpt[b][sb-1][1].y;
            sbl = 3;
            getconxy();
            cadsubxmin(sbl);
            return(bl);
```

```
        }
    else if (CONX == blpt[b][sb][1].x && CONY == blpt[b][sb][3].y)
        {
         blpt[b][sb][4].x = CONX;
         blpt[b][sb][4].y = CONY;
        getconxy();
        }
    else  { blpt[b][sb][4].x = blpt[b][sb][1].x;
        blpt[b][sb][4].y = blpt[b][sb][3].y;
        flage = 3;
        }
        }

    else {
        if (blpt[b][sb][1].y < CONY)
       {
        blpt[b][sb][4].x = CONX;
        blpt[b][sb][4].y = CONY;
        blpt[b][sb][0].y = blpt[b][sb][1].y;
        blpt[b][sb][1].y = CONY;

    fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
    fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

    strcpy (square[sb].st2, "SQUARE");

    if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
    square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
    else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);

    if ((blpt[b][sb][1].y - blpt[b][sb][2].y) > 0)
    square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);
    else square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);

    if (blpt[b][sb][1].x < blpt[b][sb][4].x)
    square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
    else
    square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

    if (blpt[b][sb][1].y < blpt[b][sb][2].y)
    square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
    else
```

```
      square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

 fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
            ++sb;
            blpt[b][sb][1].x = blpt[b][sb-1][1].x;
            blpt[b][sb][1].y = blpt[b][sb-1][0].y;
            blpt[b][sb][2].x = blpt[b][sb-1][1].x;
            blpt[b][sb][2].y = blpt[b][sb-1][1].y;
            sbl = 3;
            getconxy();
            cadsubxmin(sbl);
            return(bl);
            }

          else if (CONX == blpt[b][sb][3].x && CONY == blpt[b][sb][1].y)
             {
              blpt[b][sb][4].x = CONX;
              blpt[b][sb][4].y = CONY;
             getconxy();
             }
          else  {
             blpt[b][sb][4].x = blpt[b][sb][3].x;
             blpt[b][sb][4].y = blpt[b][sb][1].y;
             flage =3;}
       }
       }
       fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
       fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

       strcpy (square[sb].st2, "SQUARE");

       if (blpt[b][sb][1].y == blpt[b][sb][4].y)
       { if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
       square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
       else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);

       if ((blpt[b][sb][2].y - blpt[b][sb][1].y) > 0)
       square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);
       else square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);
```

```
        if (blpt[b][sb][1].x < blpt[b][sb][4].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][2].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
        }

         else { if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
        square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
        else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);

        if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
        square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
        else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

        if (blpt[b][sb][1].x < blpt[b][sb][2].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

        if (blpt[b][sb][1].y < blpt[b][sb][4].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
        }

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

        return(bl);
}
/*******************cad sub block ymin *************/
cadsubymin (sb, bl, sbl)
```

```
int sb, bl, sbl;

{
      struct blpath blpt[50][100][4];

      if (sbl == 0)
      sbl = 1;
      fprintf(jfw, "you reached up to subcadblock %d\n", sb);
      fprintf (jfw, "Block # %d - %dis started\n", b, sb);
      fprintf (jfw, "conx = %f, cony %f\n", CONX, CONY);

      while (sbl < 4)
      {
      blpt[b][sb][sbl].x = CONX;
      blpt[b][sb][sbl].y = CONY;
      getconxy();
      ++sbl;

      fprintf (jfw, "value of sbl %d, lcount %d, blpt3 %f\n", sbl, lcount,
blpt[b][sb][3].x );

      if ( CONY == YMIN )
      break;
      }

      if (CONY == YMIN && sbl == 4)
            {
            blpt[b][sb][4].x = blpt[b][sb][3].x;
            blpt[b][sb][4].y = blpt[b][sb][1].y;
            flage = 1;}


      else if (CONY == YMIN && sbl ==3)
         {
            blpt[b][sb][3].x = CONX;
            blpt[b][sb][3].y = CONY;
            blpt[b][sb][4].y = CONY;
            blpt[b][sb][4].x = blpt[b][sb][1].x;
            flage = 1;
      fprintf (jfw, "bl %d, flage %d\n", bl, flage);
         }
       else if (CONY != YMIN && sbl == 4)
       {
```

```
        if (blpt[b][sb][1].x == blpt[b][sb][2].x)
        {
         if (blpt[b][sb][1].y < CONY)
          {
           blpt[b][sb][4].x = CONX;
           blpt[b][sb][4].y = CONY;
           blpt[b][sb][0].y = blpt[b][sb][1].y;
           blpt[b][sb][1].y = CONY;

      fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
      fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

      strcpy (square[sb].st2, "SQUARE");

      if ((blpt[b][sb][1].x - blpt[b][sb][4].x) > 0)
      square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);
      else square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);

      if ((blpt[b][sb][1].y - blpt[b][sb][2].y) > 0)
      square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);
      else square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);

      if (blpt[b][sb][1].x < blpt[b][sb][4].x)
      square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
      else
      square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

      if (blpt[b][sb][1].y < blpt[b][sb][2].y)
      square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
      else
      square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
      fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);

           ++sb;
           blpt[b][sb][1].x = blpt[b][sb-1][1].x;
           blpt[b][sb][1].y = blpt[b][sb-1][0].y;
           blpt[b][sb][2].x = blpt[b][sb-1][1].x;
           blpt[b][sb][2].y = blpt[b][sb-1][1].y;
```

```
            sbl = 3;
            getconxy();
            cadsubymax(sbl);
            return(bl);
            }
        else if (CONX == blpt[b][sb][3].x && CONY == blpt[b][sb][1].y)
            {
             blpt[b][sb][4].x = CONX;
             blpt[b][sb][4].y = CONY;
            getconxy();
            }
         else  { blpt[b][sb][4].x = blpt[b][sb][3].x;
            blpt[b][sb][4].y = blpt[b][sb][1].y;
            flage = 3;
            }
            }

        else {
            if (blpt[b][sb][1].x < CONX)
           {
            blpt[b][sb][4].x = CONX;
            blpt[b][sb][4].y = CONY;
            blpt[b][sb][0].x = blpt[b][sb][1].x;
            blpt[b][sb][1].x = CONX;

        fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
        fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

        strcpy (square[sb].st2, "SQUARE");

        if ((blpt[b][sb][2].x - blpt[b][sb][1].x) > 0)
        square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);
        else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);

        if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
        square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
        else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

        if (blpt[b][sb][1].x < blpt[b][sb][2].x)
        square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
        else
        square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);
```

```
        if (blpt[b][sb][1].y < blpt[b][sb][4].y)
        square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
        else
        square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);

  fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
              ++sb;
              blpt[b][sb][1].x = blpt[b][sb-1][0].x;
              blpt[b][sb][1].y = blpt[b][sb-1][1].y;
              blpt[b][sb][2].x = blpt[b][sb-1][1].x;
              blpt[b][sb][2].y = blpt[b][sb-1][1].y;
              sbl = 3;
              getconxy();
              cadsubymax(sbl);
              return(bl);
              }

           else if (CONX == blpt[b][sb][1].x && CONY == blpt[b][sb][3].y)
              {
               blpt[b][sb][4].x = CONX;
               blpt[b][sb][4].y = CONY;
              getconxy();
              }
           else  {
              blpt[b][sb][4].x = blpt[b][sb][1].x;
              blpt[b][sb][4].y = blpt[b][sb][3].y;
              flage =3;}
        }
        }
        fprintf (jfw, "sbl1.x %f, sbl2.x %f, sbl3.x %f, sbl4.x %f\n", blpt[b][sb][1].x,
blpt[b][sb][2].x, blpt[b][sb][3].x, blpt[b][sb][4].x);
        fprintf (jfw, "end of the subblock # %d - %d\n", b, sb);

        strcpy (square[sb].st2, "SQUARE");

        if (blpt[b][sb][1].y == blpt[b][sb][4].y)
        { if ((blpt[b][sb][4].x - blpt[b][sb][1].x) > 0)
        square[sb].l = (blpt[b][sb][4].x - blpt[b][sb][1].x);
        else square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][4].x);
```

```
if ((blpt[b][sb][2].y - blpt[b][sb][1].y) > 0)
square[sb].h = (blpt[b][sb][2].y - blpt[b][sb][1].y);
else square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][2].y);


if (blpt[b][sb][1].x < blpt[b][sb][4].x)
square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
else
square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

if (blpt[b][sb][1].y < blpt[b][sb][2].y)
square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
else
square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
}

 else { if ((blpt[b][sb][1].x - blpt[b][sb][2].x) > 0)
square[sb].l = (blpt[b][sb][1].x - blpt[b][sb][2].x);
else square[sb].l = (blpt[b][sb][2].x - blpt[b][sb][1].x);

if ((blpt[b][sb][1].y - blpt[b][sb][4].y) > 0)
square[sb].h = (blpt[b][sb][1].y - blpt[b][sb][4].y);
else square[sb].h = (blpt[b][sb][4].y - blpt[b][sb][1].y);

if (blpt[b][sb][1].x < blpt[b][sb][2].x)
square[sb].x = blpt[b][sb][1].x + (square[sb].l / 2.0);
else
square[sb].x = blpt[b][sb][1].x - (square[sb].l / 2.0);

if (blpt[b][sb][1].y < blpt[b][sb][4].y)
square[sb].y = blpt[b][sb][1].y + (square[sb].h / 2.0);
else
square[sb].y = blpt[b][sb][1].y - (square[sb].h / 2.0);
}

 fprintf (jfw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);


fprintf (blw, "%s %f %f %f %f\n", square[sb].st2, square[sb].l, square[sb].h,
square[sb].x, square[sb].y);
    return(bl);
```

```
}

/*******unfolded 3D data input***********/
skin()
{               int k;
                k = 0;
                    for (j = 0; j < totlns; j++)
                                { if (ln[j].z == ln[j].z1 && ln[j].z == ZLOW)
                                                { line[k].x = ln[j].x;
            line[k].y = ln[j].y; line[k].z = ln[j].z;
            line[k].x1 = ln[j].x1; line[k].y1 = ln[j].y1;
            line[k].z1 = ln[j].z1;
                                k = k + 1;}
                                }
      fprintf (jfw, "k = %d, maxlines = %d\n", k, maxlns);
      fprintf (jfw, " Starting line is at jl # %d\n", jl);
      return(k);
}
/*****folded 3D data input******/
fldata()
{
                int n, j, k;
            for (n = 1; n <= c; n++)
            {
            j = sk[n];
           fprintf (jfw, "j.x %f, j.x1 %f sk1 %d, sk2 %d\n", line[j].x, line[j].x1, sk[1],
sk[2]);
            if (ln[j].x != ln[j].x1 && ln[j].z == ln[j].z1 && ln[j].z == ZLOW )
                  { break; }
            }
                if (ln[j].x < ln[j].x1)
                { CONX = ln[j].x1; CONY = ln[j].y1; }
                else { CONX = ln[j].x; CONY = ln[j].y; }
                k = 0;
                while ( k < totlns)
                {
                line[k].x = ln[j].x;
      line[k].y = ln[j].y; line[k].z = ln[j].z;
      line[k].x1 = ln[j].x1; line[k].y1 = ln[j].y1;
      line[k].z1 = ln[j].z1;
                k = k +1;}
}
/***********/
```

APPENDIX B

AUTOMATIC FEATURE EXTRACTION PROGRAM EXHIBITS

DIPAK>>feature

Feature Extraction> Give input file name wihout extention
Feature Extraction> part1
Feature Extraction> part1.dat is your in put file
Feature Extraction> Select the option

1 Model is 2D unfolded model
2 Model is 3D unfolded model
3 Model is 3D folded model

Feature Extraction> 2

Feature Extraction> What is the thichness(XX.XX)?
Feature Extraction> .2

Feature Extraction> Do you want to extract featurs from another file
(Yes or No):

Feature Extraction> y

Feature Extraction> Give input file name wihout extention
Feature Extraction> part2
Feature Extraction> part2.dat is your in put file
Feature Extraction> Select the option

1 Model is 2D unfolded model
2 Model is 3D unfolded model
3 Model is 3D folded model

Feature Extraction> 2

Feature Extraction> What is the thichness(XX.XX)?
Feature Extraction> .2

Feature Extraction> Do you want to extract featurs from another file
(Yes or No):

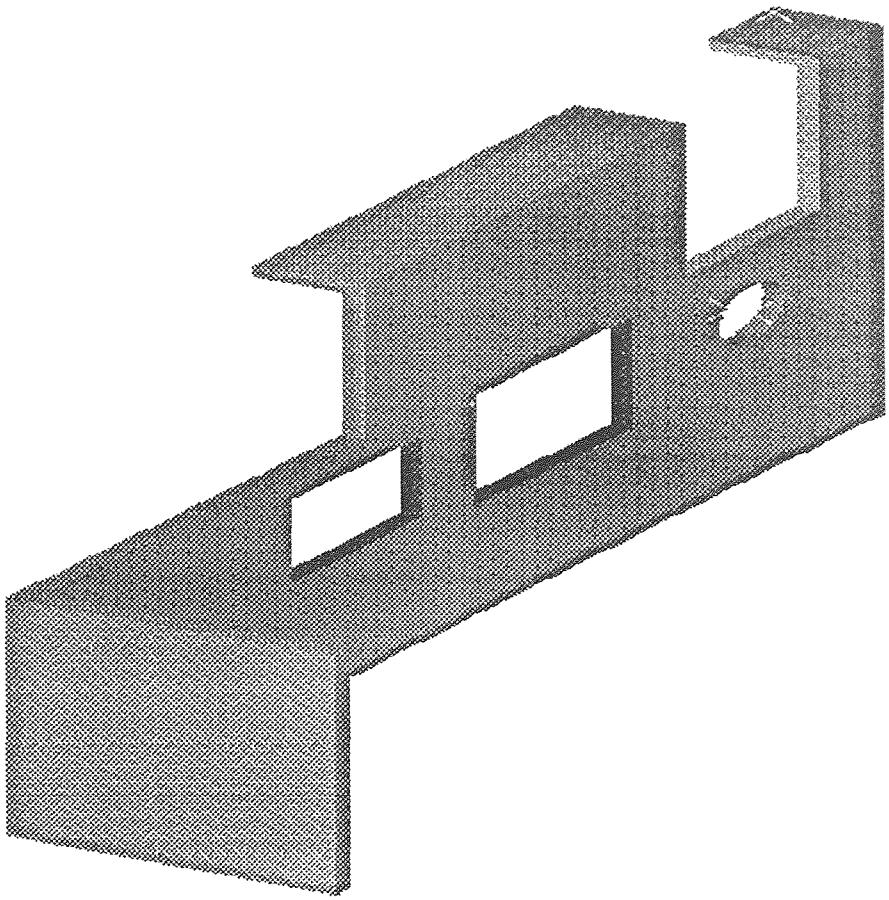
Feature Extraction> n

Feature Extraction> Thank you for using Feature Extraction program
DIPAK>>

APPENDIX C

SAMPLE PART#1

C.1 Part#1 model created on Pro-Engineer CAD system



C.2 Input data file of the part#1 (generated by IGES processor) for the automatic feature extraction program

```
1.00, 5.00, 1.00,9.00,4.00,4.20
0
37
2.875,1.000,4.000,5.000,1.000,4.000
2.875,1.000,4.000,2.875,1.000,4.200
5.000,1.000,4.000,5.000,9.000,4.000
5.000,9.000,4.000,1.000,9.000,4.000
1.000,9.000,4.000,1.000,8.500,4.000
1.000,8.500,4.000,2.875,8.500,4.000
2.875,8.500,4.000,2.875,7.500,4.000
2.875,7.500,4.000,1.000,7.500,4.000
1.000,7.500,4.000,1.000,5.050,4.000
1.000,5.050,4.000,2.875,5.050,4.000
2.875,5.050,4.000,2.875,1.000,4.000
3.875,5.500,4.000,3.875,4.500,4.000
3.875,4.500,4.000,3.125,4.500,4.000
3.125,4.500,4.000,3.1250,5.50,4.000
3.125,5.500,4.000,3.875,5.500,4.000
3.000,7.100,4.000,4.000,7.100,4.000
4.000,7.100,4.000,4.000,5.900,4.000
4.000,5.900,4.000,3.000,5.900,4.000
3.000,5.900,4.000,3.000,7.100,4.000
2.875,1.000,4.200,5.000,1.000,4.200
5.000,1.000,4.200,5.000,9.000,4.200
5.000,9.000,4.200,1.000,9.000,4.200
1.000,9.000,4.200,1.000,8.500,4.200
1.000,8.500,4.200,2.875,8.500,4.200
2.875,8.500,4.200,2.875,7.500,4.200
2.875,7.500,4.200,1.000,7.500,4.200
1.000,7.500,4.200,1.000,5.050,4.200
1.000,5.050,4.200,2.875,5.050,4.200
2.875,5.050,4.200,2.875,1.000,4.200
3.875,5.500,4.200,3.875,4.500,4.200
3.875,4.500,4.200,3.125,4.500,4.200
3.125,4.500,4.200,3.125,5.500,4.200
3.125,5.500,4.200,3.875,5.500,4.200
3.000,7.100,4.200,4.000,7.100,4.200
4.000,7.100,4.200,4.000,5.900,4.200
4.000,5.900,4.200,3.000,5.900,4.200
3.000,5.900,4.200,3.000,7.100,4.200
```

```
1
3.500,8.000,4.000,0.500,1.000
0
1
2.875,2.500,4.000,5.000,2.500,4.000,90
1
1.500,5.050,4.000,1.500,9.000,4.000,90
```

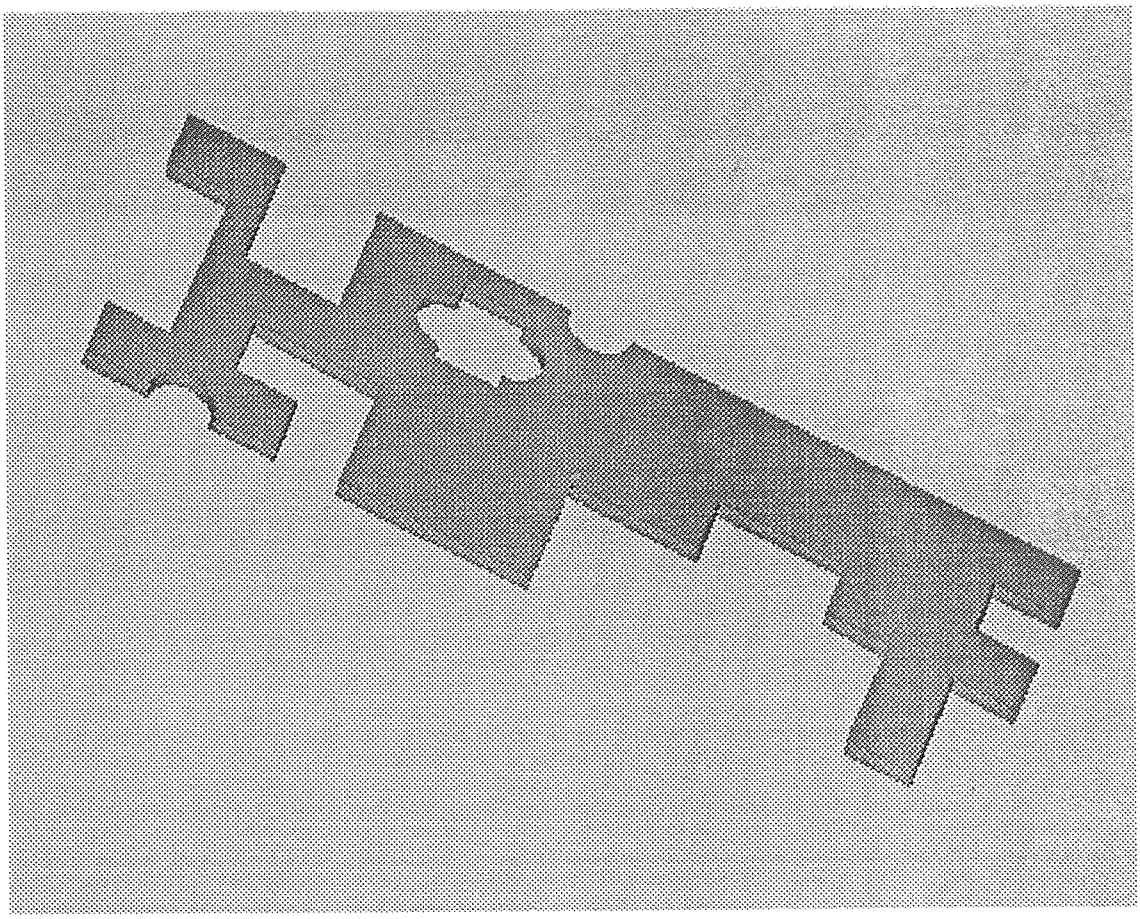
C.3 Output data (feature list) file of the part#1 generated by the automatic feature extraction program

```
BASE 4.000000 8.000000 0.200000 3.000000 5.000000 4.000000
OUTER FEATURES
SQUARE 1.875000 1.000000 1.937500 8.000000 4.000000
SQUARE 1.875000 4.050000 1.937500 3.025000 4.000000
INNER FEATURES
STATION
SQUARE 0.750000 1.000000 3.500000 5.000000 4.000000
STATION
SQUARE 1.000000 1.200000 3.500000 6.500000 4.000000
STATION
HOLE 0.500000 3.500000 8.000000 4.000000
STATION
BENDX 2.500000 90
STATION
BENDY 1.500000 90
```

APPENDIX D

SAMPLE PART#2

D.1 Part#2 model created on Pro-Engineer CAD system



D.2 Input data file of the part#2 (generated by IGES processor) for the automatic feature extraction program

1.000, 15.000, 1.000,6.000,4.000,4.200
0
81
1.000,1.000,4.000,2.000,1.000,4.000
1.000,1.000,4.000,1.000,1.000,4.200
3.000,1.000,4.000,4.000,1.000,4.000
4.000,1.000,4.000,4.000,2.000,4.000
4.000,2.000,4.000,3.000,2.000,4.000
3.000,2.000,4.000,3.000,3.000,4.000
3.000,3.000,4.000,5.000,3.000,4.000
5.000,3.000,4.000,5.000,1.000,4.000
5.000,1.000,4.000,8.000,1.000,4.000
8.000,1.000,4.000,8.000,3.000,4.000
8.000,3.000,4.000,10.000,3.000,4.000
10.000,3.000,4.000,10.000,4.000,4.000
10.000,4.000,4.000,12.000,4.000,4.000
12.000,4.000,4.000,12.000,3.000,4.000
12.000,3.000,4.000,13.000,3.000,4.000
13.000,3.000,4.000,13.000,1.000,4.000
13.000,1.000,4.000,14.000,1.000,4.000
14.000,1.000,4.000,14.000,3.000,4.000
14.000,3.000,4.000,15.000,3.000,4.000
15.000,3.000,4.000,15.000,4.000,4.000
15.000,4.000,4.000,14.000,4.000,4.000
14.000,4.000,4.000,14.000,5.000,4.000
14.000,5.000,4.000,15.000,5.000,4.000
15.000,5.000,4.000,15.000,6.000,4.000
15.000,6.000,4.000,10.000,6.000,4.000
5.000,6.000,4.000,4.000,6.000,4.000
4.000,6.000,4.000,4.000,4.000,4.000
4.000,4.000,4.000,2.500,4.000,4.000
2.500,4.000,4.000,2.500,6.000,4.000
2.500,6.000,4.000,1.000,6.000,4.000
1.000,6.000,4.000,1.000,5.000,4.000
1.000,5.000,4.000,2.000,5.000,4.000
2.000,5.000,4.000,2.000,2.000,4.000
2.000,2.000,4.000,1.000,2.000,4.000
1.000,2.000,4.000,1.000,1.000,4.000
5.500,4.000,4.000,6.500,4.000,4.000
6.500,4.000,4.000,6.500,4.250,4.000

6.500,5.250,4.000,6.500,5.500,4.000
6.500,5.500,4.000,5.500,5.500,4.000
5.500,5.500,4.000,5.500,5.250,4.000
5.500,4.250,4.000,5.500,4.000,4.000
1.000,1.000,4.200,2.000,1.000,4.200
3.000,1.000,4.200,4.000,1.000,4.200
4.000,1.000,4.200,4.000,2.000,4.200
4.000,2.000,4.200,3.000,2.000,4.200
3.000,2.000,4.200,3.000,3.000,4.200
3.000,3.000,4.200,5.000,3.000,4.200
5.000,3.000,4.200,5.000,1.000,4.200
5.000,1.000,4.200,8.000,1.000,4.200
8.000,1.000,4.200,8.000,3.000,4.200
8.000,3.000,4.200,10.000,3.000,4.200
10.000,3.000,4.200,10.000,4.000,4.200
10.000,4.000,4.200,12.000,4.000,4.200
12.000,4.000,4.200,12.000,3.000,4.200
12.000,3.000,4.200,13.000,3.000,4.200
13.000,3.000,4.200,13.000,1.000,4.200
13.000,1.000,4.200,14.000,1.000,4.200
14.000,1.000,4.200,14.000,3.000,4.200
14.000,3.000,4.200,15.000,3.000,4.200
15.000,3.000,4.200,15.000,4.000,4.200
15.000,4.000,4.200,14.000,4.000,4.200
14.000,4.000,4.200,14.000,5.000,4.200
14.000,5.000,4.200,15.000,5.000,4.200
15.000,5.000,4.200,15.000,6.000,4.200
15.000,6.000,4.200,10.000,6.000,4.200
5.000,6.000,4.200,4.000,6.000,4.200
4.000,6.000,4.200,4.000,4.000,4.200
4.000,4.000,4.200,2.500,4.000,4.200
2.500,4.000,4.200,2.500,6.000,4.200
2.500,6.000,4.200,1.000,6.000,4.200
1.000,6.000,4.200,1.000,5.000,4.200
1.000,5.000,4.200,2.000,5.000,4.200
2.000,5.000,4.200,2.000,2.000,4.200
2.000,2.000,4.200,1.000,2.000,4.200
1.000,2.000,4.200,1.000,1.000,4.200
5.500,4.000,4.200,6.500,4.000,4.200
6.500,4.000,4.200,6.500,4.250,4.200
6.500,5.250,4.200,6.500,5.500,4.200
6.500,5.500,4.200,5.500,5.500,4.200
5.500,5.500,4.200,5.500,5.250,4.200

5.500,4.250,4.200,5.500,4.000,4.200
0
4
2.000,1.000,4.000,3.000,1.000,4.000,2.500,1.500,4.000
2.500,1.000,4.000,1.000,1.000
10.000,6.000,4.000,5.000,6.000,4.000,7.500,3.500,4.000
7.500,6.000,4.000,1.000,1.000
5.500,5.250,4.000,5.500,4.250,4.000,5.000,4.75,4.000
5.500,4.750,4.000,1.000,1.000
6.500,5.250,4.000,6.500,4.250,4.000,7.000,4.750,4.000
6.500,4.750,4.000,1.000,1.000

D.3 Output data (feature list) file of the part#2 generated by the automatic feature extraction program

BASE 14.000000 5.000000 0.200000 8.000000 3.500000 4.000000
OUTER FEATURES
HOLE 1.000000 2.500000 1.000000 4.000000
SQUARE 1.000000 1.000000 3.500000 2.500000 4.000000
SQUARE 1.000000 2.000000 4.500000 2.000000 4.000000
SQUARE 2.000000 1.000000 11.000000 3.500000 4.000000
SQUARE 5.000000 2.000000 10.500000 2.000000 4.000000
SQUARE 1.000000 2.000000 14.500000 2.000000 4.000000
SQUARE 1.000000 1.000000 14.500000 4.500000 4.000000
HOLE 1.000000 7.500000 6.000000 4.000000
SQUARE 1.500000 2.000000 3.250000 5.000000 4.000000
SQUARE 1.000000 3.000000 1.500000 3.500000 4.000000
INNER FEATURES
STATION
HOLE 1.000000 6.500000 4.750000 4.000000
HOLE 1.000000 5.500000 4.750000 4.000000
SQUARE 1.000000 1.500000 6.000000 4.750000 4.000000

REFERENCE

1. Pande, S. S. and Walvekar, M. G., "A Computer Assisted Process Planning System for Prismatic Components (PC-CAPP)", Computer-Aided Engineering Journal, August 1989, pp. 133-137.

2. Luby, S. C. et al., "Creating and Using Feature Database", Computers in Mechanical Engineering, November 1986, pp. 25-33.

3. Zamanian, M. K. et al., "A Feature-Based Approach to Structural Design", Engineering with Computers, Vol. 7, 1991, pp. 1-9.

4. Salomons, O. W. et al., "Review of Research in Feature-Based Design", Journal of Manufacturing Systems, Vol. 12, No. 2, 1993, pp. 113-132.

5. Kang, Tzong-Shyan and Nnaji, B. O., "Feature Representation and Classification for Automatic Process Planning Systems", Journal of Manufacturing Systems, Vol. 12, No. 2, 1993, pp. 133-145.

6. Pande, S. S. and Prabhu, B. S., "An Expert System for Automatic Extraction of Machining Features and Tooling Selection for Automats", Computer-Aided Engineering Journal, August 1990, pp. 99-103.

7. Shah, J. J., "Features in Design and Manufacturing", Intelligent Design and Manufacturing, Edited by A. Kusiak, John Wiley & Sons, 1992, pp. 39-71.

8. Unruh, Vance and Anderson, D. C., "Feature-Based Modeling for Automatic Mesh Generation", Engineering with Computers, Vol. 8, 1992, pp. 1-12.

9. Shah, Jami et al., "Survey of CAD/feature-based process planning and NC programming techniques", Computer-Aided Engineering Journal, Vol. 8, February 1991, pp. 25-33.

10. Pratt, M. J., "Synthesis of An Optimal Approach to Form Feature Modeling", Vol. 1, 1988, pp. 263-274.

11. Karinthi, R. R. and Nau, D. S., "Geometric Reasoning As A Guide To Process Planning", Vol. 1, 1989, pp. 609-610.

12. Winbourne, J. P. and Toolsie, C. M., "Computer-Aided Tool Cost Estimating (CATE)", Vol. 1, 1989, pp. 617-621.

13. Shah, J. J. and Rogers, T. M., "Feature Based Modeling Shell: Design and Implimentation", Vol. 1, 1988, pp. 255-261.

14. Ranyak, P. S. and Fridshal, R., "Feature For Tolerancing A Solid Model", Vol. 1, 1988, pp. 275-280.