DESIGN AND EVALUATION OF A SHAPE RETRIEVAL SYSTEM

John P Eakins

PhD Thesis

Computing Laboratory
University of Newcastle upon Tyne
December 1990

ABSTRACT

While automated storage and retrieval systems for textual and numeric data are now commonplace, the development of analogous systems for pictorial data has lagged behind - not through the lack of need for such systems, but because their development involves a number of significant problems.

The aim of this project is to investigate these problems by designing and evaluating an information retrieval system for a specific class of picture, 2-dimensional engineering drawings. This involves consideration of the retrieval capabilities needed by such a system, what storage structures it would require, how the salient features of each drawing should be represented, how query and stored shapes should be matched, what features were of greatest importance in retrieval, and the interfaces necessary to formulate queries and display results.

A form of hierarchical boundary representation has been devised for stored shapes, in which each boundary can be viewed as a series of levels of steadily increasing complexity. A set of rules for boundary and segment ordering ensures that as far as possible, each shape has a unique representation. For each level at which each boundary can be viewed, a set of invariant shape features characterizing that level is extracted and added to the shape representation stored in the database. Two classes of boundary feature have been defined; *global* features, characteristic of the boundary as a whole, and *local* features, corresponding to individual fragments of the boundary. To complete the shape description, *position* features are also computed and stored, to specify the pattern of inner boundaries within the overall shape.

Six different types of shape retrieval have been distinguished, although the prototype system can offer only three of these - exact shape matching, partial shape matching and similarity matching. Complete or incomplete query shapes can be built up at a terminal, and subjected to a feature extraction process similar to that for stored drawings, yielding a query file that can be matched against the shape database. A variety of matching techniques is provided, including similarity estimation using global or local features, tests for the existence of specified local features in stored drawings, and cumulative angle *vs* distance matching between query and stored shape boundaries. Results can be displayed in text or graphical form.

The retrieval performance of the system in similarity matching mode has been evaluated by comparing its rankings of shapes retrieved in response to test queries with those obtained by a group of human subjects faced with the same task. Results, expressed as normalized recall and precision, are encouraging, particularly for similarity estimation using either global or local boundary features. While the detailed results are of limited significance until confirmed with larger test collections, they appear sufficiently promising to warrant the development of a more advanced prototype capable of handling 3-D geometric models. Some design aspects of the system would appear to be applicable to a wider range of pictorial information systems.

ACKNOWLEDGEMENTS

It is a pleasure to acknowledge the help and encouragement I have received from many sources over the six years of this project. Pride of place must go to my supervisor, Elizabeth Barraclough, for her enthusiasm to see the work succeed, her unfailing patience, and her consistent ability to ask seemingly-innocuous questions. Thanks are also due to Peter Hitchcock, my former supervisor, now at the University of York, for helping to define the initial shape of the project.

Discussions with colleagues too numerous to mention at Newcastle upon Tyne Polytechnic have provided me with valuable opportunities to clarify ideas and set them in context. Particular thanks are due to Tony McLeod of the Department of Mechanical Engineering, for providing a design engineer's view of the project, offering a wealth of constructive criticism, and identifying sources of test data. The advice of Mic Porter from the Department of Applied Social Science on the design of the human shape-matching experiments, and of Carolyn Craggs of the Department of Mathematics and Statistics on statistical analysis of the results, is also gratefully acknowledged.

Last but emphatically not least, I owe an enormous debt of gratitude to my wife Ann, both for her constant encouragement, and for her work in reading earlier drafts of this thesis, identifying areas which were inconsistent, illogical or just plain incomprehensible!

NOTE

A preliminary version of this research was presented at the British Computer Society 11th Information Retrieval Research Colloquium held at Huddersfield Polytechnic in July 1989, and appears in the proceedings of that meeting under the title "SAFARI - a shape retrieval system for engineering drawings".

CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Graphics as a communication medium

It can be argued that graphical presentation of information is one of the oldest means of communication known to man - prehistoric cave-dwellers painted pictures on the walls of their caves, the ancient Greeks and Egyptians drew lines in the sand while developing their ideas on geometry, and early navigators summarized their knowledge of the seas and coasts in the form of maps. Engineers and architects have used plans as a means of developing their designs and communicating their ideas to others for hundreds - maybe thousands - of years.

The use of computers in handling graphical information is still not fully developed. In the early days of computing, computers were regarded only as number-crunchers. Later their record-handling and text-processing ability was recognized, but their potential to display, manipulate and store images was not realized until the 'Sketchpad' project (Sutherland, 1965) came to fruition in the early 1960's. The high cost of hardware, and the difficulty of creating good graphics software, delayed the commercial acceptance of computer graphics for over a decade. It was not until hardware costs began to fall dramatically in the late 1970's - and a mass market developed for home computers capable of playing arcade games - that use became at all widespread. Since then, computer graphics have become widely used as a display medium in such apparently diverse fields as business (Ives, 1982), geography (Nagy and Wagle, 1979), and chemistry (Max, 1983), where graphically-presented information can often be more easily interpreted than tables of figures.

The main use of the computer's ability to process graphical information has been in the area of design, whether engineering (e.g. Gardan and Lucas, 1983), architecture (Rogers, 1980), fashion (Kunii et al, 1975), or graphic art (Dietrich, 1985). Engineers, in particular, have made substantial use of computers for years - purely as a calculating tool to start with, but increasingly as an interactive design aid, enabling them to create, modify, and store graphic representations of the object they were designing. Sophisticated draughting and geometric modelling packages have been developed, providing the designer with a far wider range of facilities than the old drawing-board could offer.

1.2 Computers in engineering design

Computers have been used in engineering design since the 1950's, though the use of interactive computer graphics to create and modify engineering drawings has a much shorter history. The earliest systems, growing out of research programmes in large engineering-based firms such as Lockheed and General Motors, were quite literally computerized draughting systems, allowing the designer to produce drawings on a screen with the help of a light pen, or stylus and tablet. These screen images could be selectively modified, stored for later use, or used to generate high-quality engineering drawings on a suitable plotter. Such two-dimensional (2-D) draughting packages are still in use today. They are best regarded as a direct replacement for the drawing-board; just as with traditional engineering drawings, a number of views or *projections* from different angles need to be drawn to describe an object to be manufactured. Each projection has to be the subject of a separate drawing.

A typical 2-D draughting package is DOGS (drawing office graphics system) from PAFEC Ltd of Nottingham. This package allows users to build up and edit drawings on a high-resolution graphics screen, allowing input of drawing elements via keyboard or graphics tablet. A relatively small set of primitives is used for constructing drawings - points, straight lines, circular arcs, and text characters. More complex shapes such as ellipses are represented by chains of short straight-line or circular segments. A variety of drawing aids is provided - users can define temporary construction lines, zoom in to magnify selected parts of a drawing, and copy one part of a drawing to another by translation, rotation, or mirroring. The facility is also provided to associate sets of line elements into user-defined symbols, which can be saved in a library file, and

retrieved and manipulated as a unit. Drawings can be dimensioned, and drawn in a variety of line styles. Hard copy can be provided as screen dumps or via a plotter, and drawings can be archived on disc for later editing. For the advanced user, facilities are provided for defining multiple views of a drawing, or defining drawings on different levels, which can be viewed separately or together. A typical drawing generated by DOGS is shown in Fig 1.1.

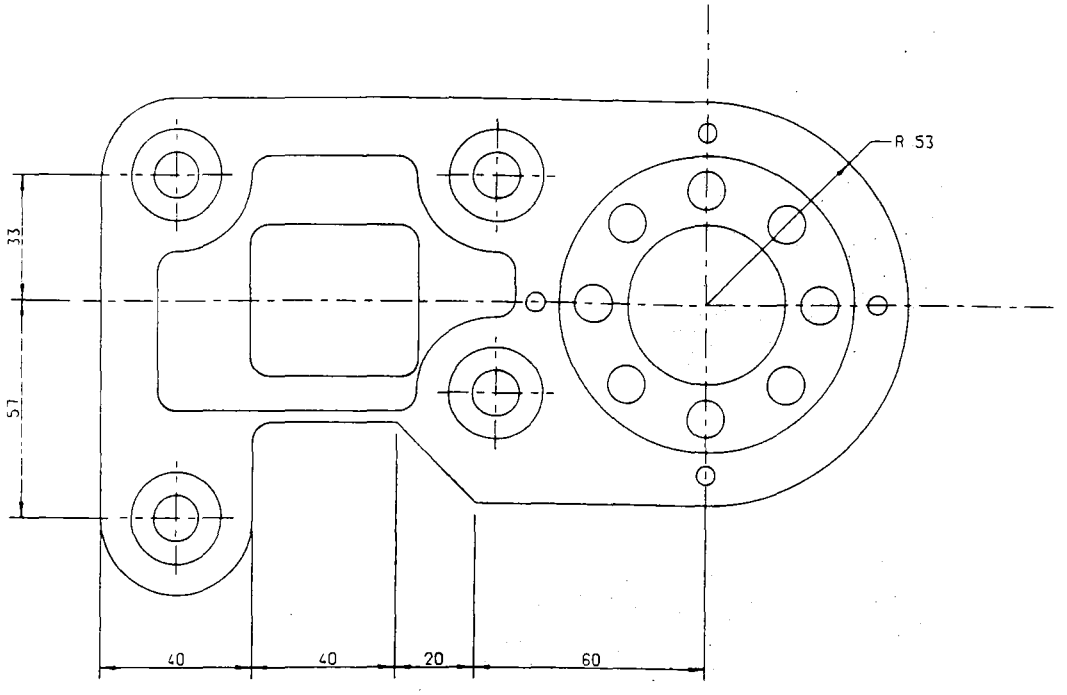


Fig 1.1 Typical engineering drawing produced using DOGS 2-D draughting system

DOGS faithfully represents the geometry (i.e. the type and spatial coordinates of each element) of any drawing it is used to construct. Each element (point, line, arc, etc.) in the drawing is represented by an individual record in the drawing file, containing an indicator of the element type, its defining (x,y) coordinates, and ancillary information such as line type or (for circular arcs) angle subtended. However, there is no specific indication of the topology of the object represented (i.e. how elements are connected), unless the designer chooses to make this explicit when the drawing is first created. Though four lines may trace the boundary of a square object, this relationship will in general have to be inferred from the drawing(s) by the user.

Following the pioneering work of Braid (1973) and Voelker et al (1978), interest has grown in developing software which is capable of modelling design objects directly, not via 2-D drawings. The motivation behind Braid's original BUILD system was that since a relatively small set of machining operations was sufficient to manufacture components of any desired complexity, it should be possible to carry out design in a similar manner. BUILD thus allowed the user to construct, store and display a solid model of an object such as that shown in Fig 1.2 by performing Boolean operations on shapes constructed from a set of six solid primitives (cube, tetrahedron, cylinder, wedge, segment and fillet). Most subsequent geometric modelling systems have adopted this basic principle, though the range of operations provided has been considerably enhanced. Two main forms of object representation are in common use in such systems - *constructive solid geometry* (CSG), which represents solid objects as binary trees, indicating how they were "built up" from a small set of primitives by repeated application of union, intersection, difference or transformation operators (Fig 1.3), and *boundary representation*, in which the geometry and topology of each bounding surface, edge and vertex are explicitly specified (Fig 1.4).

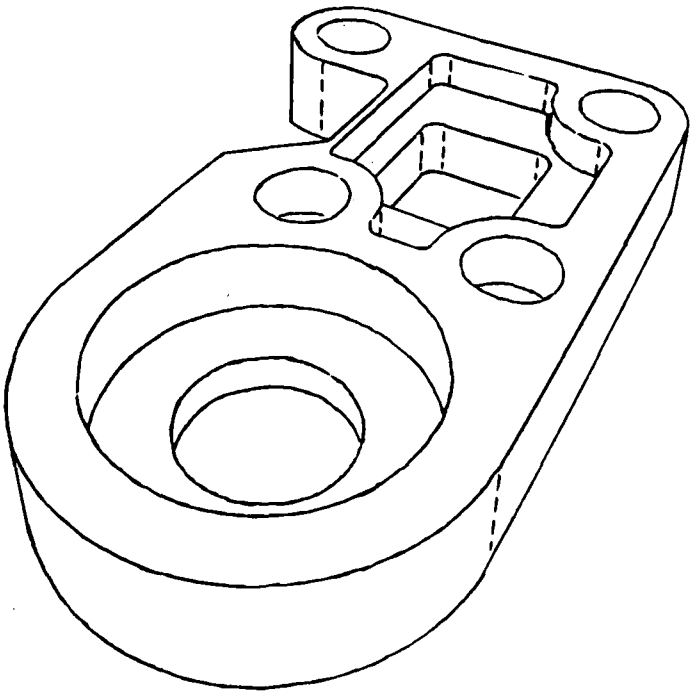


Fig 1.2 The part illustrated in Fig 1.1, as modelled by a 3-D system

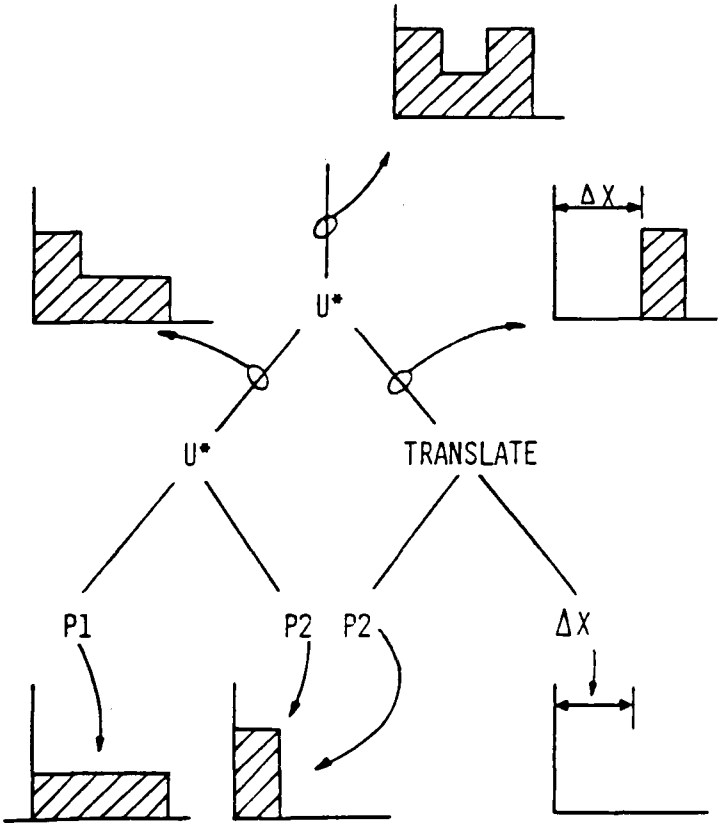


Fig 1.3 Representation of a solid object (shown in cross-section) as a CSG tree. The solid primitives P1 and P2 are combined by the union operator U, and the resultant solid combined again with the result of translating primitive P2 the specified distance along the x-axis. Figure reproduced from (Requicha, 1980) by permission of the Association for Computing Machinery. Copyright ACM 1980*

```
%BODY      ABC %NAME_GENERATOR 8
 %SURFACE $18  %PLANAR %POINT_ON_PLANE ( 0.0,0.0 ) @
             %NORMAL_TO_PLANE ( 0.0,0.0,-1.0 )
           $48  %PLANAR %POINT_ON_PLANE ( 0.0,0.0,10.0 ) @
             %NORMAL_TO_PLANE ( 0.0,0.0,-1.0 )
           $21  %CYLINDRICAL %AXIS_POINT ( 0.0,0.0 )
                %DIRECTION ( 0.0,0.0,1.0 ) @
                %RADIUS 5.0
 %TRACK    $13  %CIRCULAR %CENTRE ( 0.0,0.0 )
                %NORMAL_TO_PLANE ( 0.0,0.0,-1.0 ) @
                %RADIUS 5.0
           $11  %CIRCULAR %CENTRE ( 0.0,0.0,10.0 ) @
                %NORMAL_TO_PLANE ( 0.0,0.0,-1.0 ) %RADIUS 5.0
           $46  %LINEAR
 %POINT     P1 ( -2.23607,4.47214 )
           P6 ( -2.23607,4.47214,10.0 )
 %FACE      F0  %FOLLOWING %SURFACE $18 %HA 0.0  0.0  0.0
 %LOOP
 %VERTEX P1 %EDGE     E2  %FOLLOWING %TRACK   $13
 %FACE      F3  %OPPOSING %SURFACE $48 %HA 0.0  0.0  0.0
 %LOOP
 %VERTEX P6 %EDGE     E5  %OPPOSING %TRACK   $11
 %FACE      F4  %FOLLOWING %SURFACE $21 %HA 0.0  0.0  0.0
 %LOOP
 %VERTEX P1 %EDGE     E2  %OPPOSING %TRACK   $13
          P1 %EDGE     E7  %FOLLOWING %TRACK   $46
          P6 %EDGE     E5  %FOLLOWING %TRACK   $11
          P6 %EDGE     E7  %OPPOSING %TRACK   $46
%END       ABC
```

Fig 1.4 An example of the boundary representation of a cylindrical object, as generated by the ROMULUS geometric modelling system, specifying both the geometry of each surface, track (path followed by an edge) and point, and the topological relationships between faces, edges, and vertices.

A typical boundary representation-based geometric modelling package is ROMULUS, from Shape Data Ltd, Cambridge. This allows users to create, modify, and store solid models, and generate 2-D views of such models from any desired angle. The user builds up a geometric model of the design object by specifying appropriate combinations of solid primitives (cube, rectangular block, regular prism, cylinder, cone, sphere or torus). He can create instances of any of these primitives at any given location in 3-space, with specified size and orientation, and can modify their surface or edge geometry, reposition them by scaling, translation, rotation, or mirroring, and combine them into more complex objects by using suitable Boolean operators (for example, a block with four cylindrical holes would be generated by specifying a solid block and four cylinders of suitable size and position, then subtracting the cylinders from the block). "Sweep" operators are also provided, allowing 2-D shapes to be "extruded" into solid form, either along a specified linear path, or around a given axis, generating a solid of revolution. ROMULUS can display orthographic (top and side) views of the design object, or perspective views from any desired angle, with all hidden edges removed. The user can also generate cross-sectional views. Again, all or part of the model can be stored on disc for later use. Facilities are provided for the user to specify additional associations by defining sets of faces which make up a common feature, share a common type of geometry, or are traversed by a given ray (a vector with specified direction and starting-point), though the use of this facility is purely optional.

Solid modelling systems of this kind demand new ways of working from the design engineer, who has to think in three dimensions rather than two. They also need to handle more information about the object. In order to describe the shape and size of a three-dimensional (3-D) object fully, it is essential to include information about both its geometry and its topology. Concave objects such as those shown in Fig 1.5 cannot be represented unambiguously by geometry alone. Some information - explicit or implicit - needs to be provided on the topological relationships between faces, edges, and vertices. However, solid modellers do provide considerably more

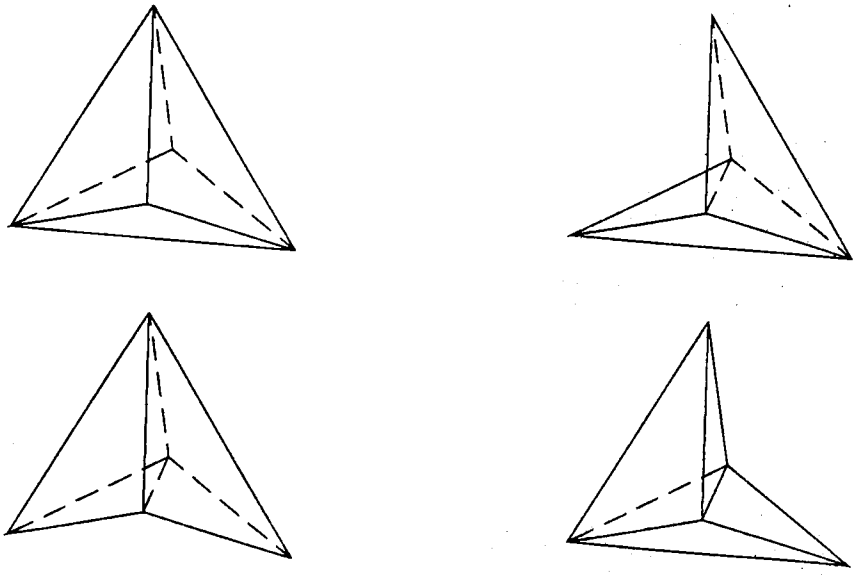


Fig 1.5 Ambiguity of shape described by vertices alone

functionality than 2-D packages - their ability to calculate mass properties such as centres of gravity or moments of inertia, to perform interference analysis and to generate finite element meshes for stress analysis means that they can genuinely be regarded as computer-aided *design* (not just draughting) tools. Further developments such as automatic code generation for NC machine tools and automatic process planning are actively being researched, offering the real prospect of linking design and manufacture into an integrated whole (Requicha, 1988; Voelker, 1990).

1.3 Data exchange in computer-aided design (CAD)

As CAD systems have grown in popularity, the need to exchange data between different CAD systems has grown in importance. Organizations working on cooperative projects, who in the past would have exchanged design drawings, now wish to do the same with their machine-readable equivalents. Since the majority of commercially-available CAD systems store data in quite different formats, direct exchange of data between independent CAD systems is virtually impossible. To allow such data exchange requires either (a) the development of interface software to translate directly between each pair of systems (a task which rapidly becomes impracticable with more than a handful of systems; to create all required interfaces between n systems requires $n(n\text{-}1)$ separate pieces of software), or (b) the definition of a standard interchange format, and the provision of interface software between each CAD system and this standard format (requiring no more than $2n$ pieces of software to interface n CAD systems).

The attractiveness of the second option has led to several attempts at standardization, of which the most successful to date has been IGES, the Initial Graphics Exchange Specification (National Bureau of Standards, 1983). This was developed largely out of standardization efforts made by

the Boeing Corporation and other large American organizations during the 1970's, and attempts to define a common format for exchange of data between any pair of 2-D draughting systems. It has some limited 3-D capabilities, but was not designed with 3-D solid modellers in mind. Standardization for data exchange needs to address three main areas (Liewald & Kennicott, 1982): *format* (file structures required, and bit representations to be used for integer, floating-point number and character definitions); *representation* (the set of geometric primitives chosen to define an object's shape), and *meaning* (how to preserve structural and other relationships present in the original drawing).

The IGES standard addresses the first two of these issues in considerable detail. An IGES file describes a single drawing, and consists of five separate sections, each containing one or more 80-character records (a relic from the days of punched cards) in ASCII character format. These five sections are:

- A **start** section, providing a "human-readable" file header which may be displayed by the receiving system.

- A **global** section, defining standard parameters for system use, including file name, date and time of creation, number of bits used to represent integers and floating point numbers in the sending system, and reduction scale used.

- A **directory** section, containing two records for each geometric entity (point, line, surface, etc) in the drawing, which define the entity type, point to its associated parameter value records, and indicate line type and weight, and entity status.

- A **parameter** section, containing a variable number of records for each entity, giving values for whatever parameters are required to define that particular entity.

- A **terminator** section, consisting of a single record, holding record counts for all preceding sections.

The IGES standard allows a wide variety of entity types to be defined; some of the commoner types are shown in Table 1.1. As can be seen from the table, there is considerable variety in the choice of defining parameters for different entities. It is also possible to define a given geometric entity in more than one way, e.g. straight lines as entity type 106 or 110, curves as type 102, 112 or possibly 104. This allows great flexibility in representation, and minimizes the danger of distorting complex entities. It also unfortunately makes it virtually impossible to define a "standard" IGES representation of a drawing, as so many valid alternatives exist. In general, there will be several thousand, if not million, valid IGES representations of any drawing of average complexity. To recognize that any two of these actually represent the same drawing is a non-trivial task.

This task is made considerably more difficult because IGES makes no serious attempt to address the third main standardization issue - how to preserve the meaning implicit in the original drawing. A limited amount of semantic information can be passed from sending to receiving system by the use of the "associativity" and "property" entities (types 402 and 406). These allow the user of the sending system to make explicit the topology of the object described, to specify that certain geometric entities are grouped together to form a structural unit, or to specify certain non-geometric properties such as materials or surface finish. There is nothing mandatory about using this feature, however, and it remains completely up to the designer what associations, if any, are specified. The user of the receiving system cannot assume that any such information has been supplied.

Implementation of IGES has been patchy. A number of CAD systems now incorporate modules for reading drawing files in IGES format. A smaller number are capable of generating IGES files - though these often turn out to be extremely limited in the range of entity types they can handle. DOGS, for example, is capable of reading IGES files, generated by other CAD systems,

containing entity types 100 (circular arc), 106 (copious data), 110 (line), and 112 (spline curve). IGES files generated by DOGS contain only two types of geometric entity - 100 (circular arc) and 106 (copious data).

IGES was never intended as more than an interim standard for data exchange; the original IGES committee proposed that it should be replaced within two or three years by a new standard to be known as PDES (Product Definition Exchange Standard), and further extensions, involving interchange of both 2-D drawings and 3-D solid modelling data, have been proposed in the intervening years (Wilson, 1987). Recent standardization activity has been directed towards drawing together as many disparate standards as possible within a new international standard known as STEP, or STandard for Exchange of Product data (Wilson, 1989). At the time of writing, however, IGES remains the only officially-adopted ISO standard (Wilson, 1990).

1.4 Data management for CAD

The concept of database, where data are held in applications- independent form and accessed only via specific database management software, has gained widespread acceptance in the commercial field. It has yet to make a great deal of impact in the engineering area - proprietary CAD systems such as DOGS store drawing representations in a completely application-specific form, making it impossible to use their drawing files for any other purpose without writing interface software. While such systems remain standalone draughting packages whose sole function is to produce drawings, this is an acceptable situation. However, with the rise in importance of computer-integrated manufacturing (CIM), where CAD systems are directly interfaced with numerically-controlled machine tools, and bill of materials and production-scheduling applications, interest in database management for engineering has grown.

A unified engineering database, holding information on materials, manufacturing methods, and design specifications as well as shape information, can play a central role in integrating engineering design and manufacture (Kimura et al, 1982). There are, however, fundamental problems in applying conventional database models to this kind of application (Eberlein and Wedekind, 1982; Staley and Anderson, 1986). These include the need to be able to cope with transactions lasting for several hours, and the ability to extend database schema definitions dynamically as the structure of the object being designed gradually emerges.

In theory, conventional DBMS can cope with the schema definition problem if one chooses simple enough building blocks for one's database (see chapter 5 below). In practice, however, this approach is of limited use, although at least one CAD-oriented database management system (DBMS) designed along these lines has been described in the literature (Ulfsby et al, 1982). Its main limitation is that it fails to capture any of the *semantic* information conveyed by the original drawing, such as the relationships between assemblies and their constituent parts, and the presence within such components of recognizable structural features.

One way to capture part-assembly relationships is to construct a database which shows such relationships explicitly from the start. This in essence was the "Product structured data base" approach described by Johnson and Dewhirst (1982) and adopted for their COMCAD system, which allowed complex products such as diesel engines to be specified as a hierarchy of assemblies and components, each identified by name, and containing a geometric description of each named component. A more general approach to the problem, proposed by a group from Rensselaer Polytechnic Institute, NY (Spooner et al, 1985), lies in the creation of an integrated design environment, in which boundary representations, CSG models, finite element meshes and part-assembly hierarchies are represented as high-level abstract data types whose detailed implementation need not concern the user.

Table 1.1 - some common IGES entity types

Type no.	Name	Defining parameters
Geometric entities:		
100	Circular arc	(x,y) coordinates of centre of circle, start and end point
102	Composite curve	Pointers to constituent curve segments (lines or arcs)
104	Conic arc	Coefficients $a..f$ of defining equation, (x,y) coordinates of start & end points
106	Copious data	Sets of (x,y) or (x,y,z) coordinates forming a set of line segments
108	Plane	Coefficients $a..d$ of defining equation
110	Line	(x,y,z) coordinates of start and end points
112	Parametric spline	Coefficients of cubic polynomials defining x, y and z coordinates
Non-geometric entities:		
202	Angular dimension	Pointers to general note, leader and witness lines, (x,y) coordinates of vertex point
212	General note	ASCII characters forming text string, (x,y,z) coordinates of start point, text angle & size
214	Leader	(x,y) coordinates of arrowhead, and each segment of tail
216	Linear dimension	Pointers to general note, leaders and witness lines
402	Associativity	Various
404	Drawing	Various
406	Property	Various
410	View	Various

1.5 The need for retrieval by feature

A system such as that described above can be of enormous help to design teams working on large and complex assemblies. Yet it has at least one significant limitation. It provides no means of helping the designer with questions such as "What standard components do we have similar to the one shown on this sketch, or with such-and-such a combination of design features?". Such questions can be important to designers, production engineers, and their managers. Designers have no wish to waste their time reinventing standard parts, production engineers may well wish to take advantage of feature similarity among parts to machine a whole family of parts in a single run - and any cost-conscious management needs to discourage the proliferation of almost-identical parts, each of which has to be separately manufactured and stocked, when a single part would suffice. Surveys such as that reported by Schaffer (1981) have suggested that a typical company could cut the number of new drawings it generates by anything up to 30% by adopting such standardization.

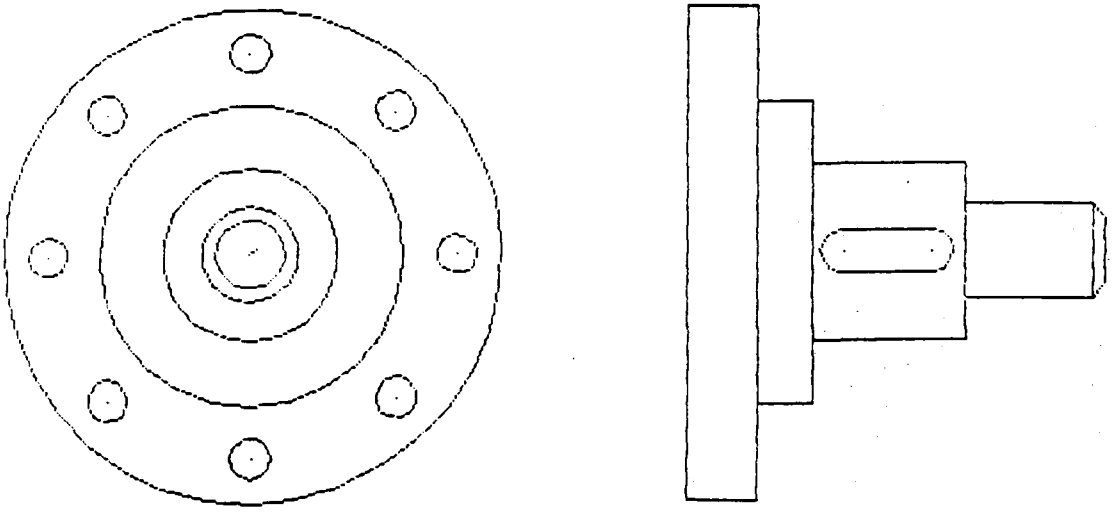
A database that represents components purely in terms of points, lines and surfaces is of minimal use in such a situation. Design engineers tend to think primarily in terms of shape features, such as slots, grooves or pockets. (Similarly, architects think in terms of features such as architraves, columns, and gables). This is the language they use when talking about objects to be machined, and there is strong evidence of a connection between the way people use language and the way they think (Stamper, 1973). However, language is notoriously ambiguous in the way objects are denoted - an identical object can be known by several different names. Kyprianou (1980) quotes an illuminating example of a simple cylindrical part separately designed, made and stocked under a dozen different names, including "arbor", "boss", "cotter", "mandrel", "pin", "pivot", "rod", and "spindle".

Engineers have in the past responded to the need to recognize components by structural feature through classifying parts into families on the basis of shape features. A number of standard parts classification schemes have been put forward, the main impetus being the rise during the 1960s and 1970s in the importance of group technology, which involved grouping together small batches of parts requiring similar machining steps, in order to increase batch size and reduce unit costs. One of the best known coding schemes is the Opitz code (Opitz et al, 1969). This was devised, according to the author, to provide a widely usable coding system capable of providing a means for quick retrieval of drawings and working plans, the selection of similar groups of components, and surveying the types of components suitable for a given machine tool. It classifies parts, first on overall shape (rotational/non-rotational; length/width ratio, etc), then on the presence, type and orientation of machined features such as slots, grooves, or holes. Fig 1.6 shows an example of the Opitz code for a simple machined part.

While codes of this kind have proved successful in use, they have significant limitations. Firstly, there is the overhead of manually classifying each part. The need to make manual encoding as easy as possible imposes severe limitations on the number and specificity of features that can be coded. Secondly, human classifiers will not always agree on the coding to apply to a given part (some categories involve an element of subjective judgement), and will on occasion inevitably make mistakes. Thirdly, manual classification systems lack flexibility - they take account only of those features their original authors considered important. This problem has been tackled in many classification systems by making provision for local extensions and variations. Unfortunately, this brings its own problems if firms wish to share their design data - classification codes assigned in one centre may be meaningless in another. There is thus a demonstrable need for more economical, reliable and flexible systems.

1.6 Automatic shape recognition in CAD

The question of how shape features can best be defined and represented is central to this whole investigation. What constitutes an important shape feature, how can it be recognized, and what form of representation is most suitable? At the simplest level, the problem could be thrown back



Classification number:- 1 2 1 3 2

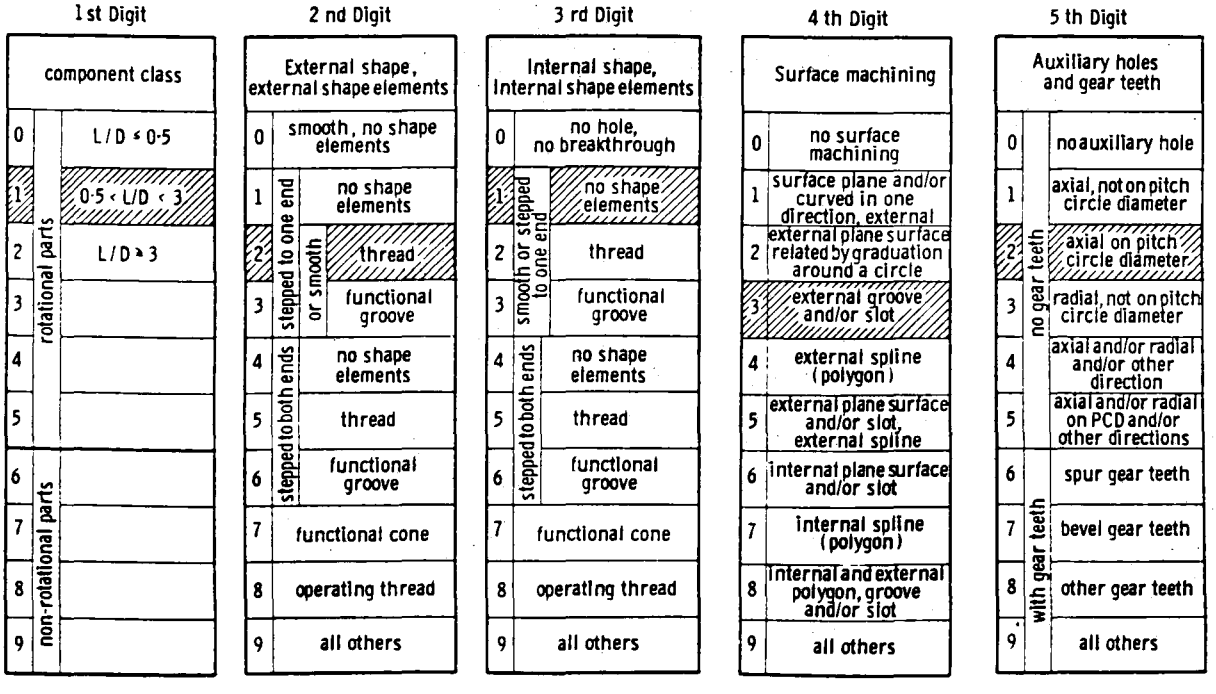


Fig 1.6 Opitz code as applied to a typical machined part (from Opitz, 1969)

on the designer. It would be quite feasible to ask designers to indicate on their completed designs the main shape features possessed by each major component ("part XYZ1000 is cylindrical, with a centre hole, and one longitudinal groove"); such text descriptions could readily be used as the basis of a retrieval system. Without careful training, however, it is most unlikely that designers even in a single location would agree on the naming of standard shape features. The chances of such a system achieving any widespread application are minimal. Designers are far too proud of their individuality.

A more feasible variant of this idea is the concept of feature-based design (Patel, 1985). Here, the designer is provided with a "front-end" to a CAD system, which allows him to select desired structural features from a menu. A design drawing or geometric model is then prepared, incorporating the desired features. Details of the features chosen could readily be extracted from such a system to provide the basis for feature retrieval, with the advantage that the "front-end" imposes a large measure of standardization on the name and type of features chosen, thus overcoming one of the main problems highlighted above. Unfortunately, such systems are still experimental, and it is too early to tell whether they will achieve any widespread use among designers. The choice of "standard" features and how they are defined is also likely to vary from system to system. They are in any case designed for ease of use rather than with retrieval usefulness in mind, so that it is generally possible to design the same resulting shape in more than one way. This problem is discussed in more detail in the next chapter.

The two approaches outlined above are thus severely limited in their applicability. To provide a shape retrieval system capable of widespread application, it is necessary to use the power of the computer to seek out and identify features in each design object, using the only representation that one can guarantee to be available - engineering drawings or geometric models from existing CAD systems. Since these in general provide no means of identifying structural features, each drawing requires a considerable amount of preprocessing by suitable shape recognition software. The design of such software represents a considerable challenge.

Kyprianou (1980) has made a detailed study of this problem. The aim of his investigation was to devise an automatic parts classification system, free from subjective bias or error. To do this, he needed an objective way of recognizing important morphological features, such as protrusions and depressions. As his starting point, he took geometric models created using the Cambridge University BUILD system referred to in Section 1.2. These were systematically analysed in order to uncover major structural features. The structural primitives available in BUILD (the *loop*, defining the boundaries of a surface, *edge* and *vertex*) defined too small a portion of an object to denote structural features of any importance on their own. They could, however, be used as the building-blocks of a feature language which did. Kyprianou was able to define this language in terms of a regular *shape grammar*, which could be shown to belong to the general class of context-free grammars, and could therefore be parsed without undue difficulty. For a detailed discussion of the underlying theory and its application to picture parsing, see Fu (1982).

The fundamental building-blocks of Kyprianou's shape grammars were *facesets*, connected sets of faces defining important morphological features, which could be recognized locally either as protrusions or as depressions. He constructed a feature recognizer which systematically "parsed" the BUILD boundary representation of each object, one face at a time, in order to identify all facesets. The process of generating facesets involved two main stages. Firstly, each loop, defining a face boundary, was marked as convex or concave. Then each loop was examined in turn, starting with the loop enclosing the largest number of interior loops, and faces aggregated into facesets (Fig 1.7) according to criteria defined by the shape grammar. The overall structure of the object could thus be represented as a graph (normally a tree), with nodes representing facesets, and arcs the relations between facesets. The shape of the part itself could then be found by matching each faceset in turn with a suitable template - planar parts against regular prisms, rotational parts against cylinder, cone and sphere. Once the nature of each faceset was known, the whole object could be recognized and classified.

Unfortunately the project never came completely to fruition. The shape recognizer was unable to recognize facesets under all circumstances. The proposed classification method could be applied

only to one small class of parts. The major importance of Kyprianou's work is probably the concept of the faceset, an objective means of defining and representing shape features of importance. Facesets do seem to coincide remarkably well with a subjective impression of structural features - in the majority of cases. There are, however, important exceptions, particularly where a depression or protrusion overlaps two or more faces (Fig 1.8). According to Kyprianou's rules, a single faceset describes this object: the depression is not recognized as a separate feature.

Further work on feature recognition has been reported by a number of groups. Staley et al (1983) have taken the 2-D shapes resulting from manual cross-sectioning of holes of different types from geometric models produced by ROMULUS, and developed a chain code representation similar to that first described by Freeman (1974). They then devised separate shape grammars to describe each of 13 possible classes of hole - flat, tapered, chamfered, and so on. A string parser was then used to classify "unknown" hole descriptions into one of the 13 classes. Choi et al (1984) have developed a series of PASCAL programs to recognize instances of circular holes, rectangular pockets and similar machined features in ROMULUS boundary representations of simple parts. Although their method is not based formally on the use of a shape grammar, shape features are defined according to a rigid syntax. Henderson and Anderson (1984) have used a rule-based approach to feature recognition. Their FEATURES system, designed to generate feature descriptions for process planning from ROMULUS part descriptions, represents shape features such as holes and slots as Horn clauses in the artificial intelligence programming language PROLOG. Lee and Fu (1987) have devised algorithms to transform CSG trees into a standard form for feature recognition, and Dong and Wozny (1988) have described a frame-based system, linked to the PADL solid modeller (Voelker et al, 1978), which will recognize and characterize named features such as pockets, bosses or slots. While these systems have automation of manufacturing process planning, not database construction, as their ultimate aim, the techniques used are clearly of wider relevance.

1.7 Graphical databases in other areas

1.7.1 What constitutes a graphical database?

Increasing interest has been shown over the last decade in what are variously called *pictorial information systems* or *image database systems*. Such systems have in common that they hold machine-readable representations of pictorial data of some kind, available for retrieval, display or further processing. Unfortunately they have remarkably little else in common. Image records may be large (over 20 MB for LANDSAT images) or small (a simple line drawing requiring only 100 bytes or so of storage). The numbers of images held in such "databases" may range from less than 10 to thousands (though large databases of this kind are extremely rare). Each image may describe a single object, or hundreds of objects of different types. Data retrieval facilities may range from the primitive (simple retrieval by name or record number) to the sophisticated (use of relational query languages). While earlier reviews (e.g. Tamura, 1980) attempted to create a taxonomy of such systems, more recent reviews have suggested that such an exercise is not helpful, and in fact question whether more than a small fraction of so-called "image databases" deserve the name:

> "It does not seem that the significant concepts of IDB (image data base) have been established yet, because there exist too few systems that we can call a true IDB" (Tamura and Yokoya, 1984)

> "Our own impression is that much of the work appearing under the heading 'image database' describes either image nondatabase systems ... or nonimage database systems" (Nagy, 1985)

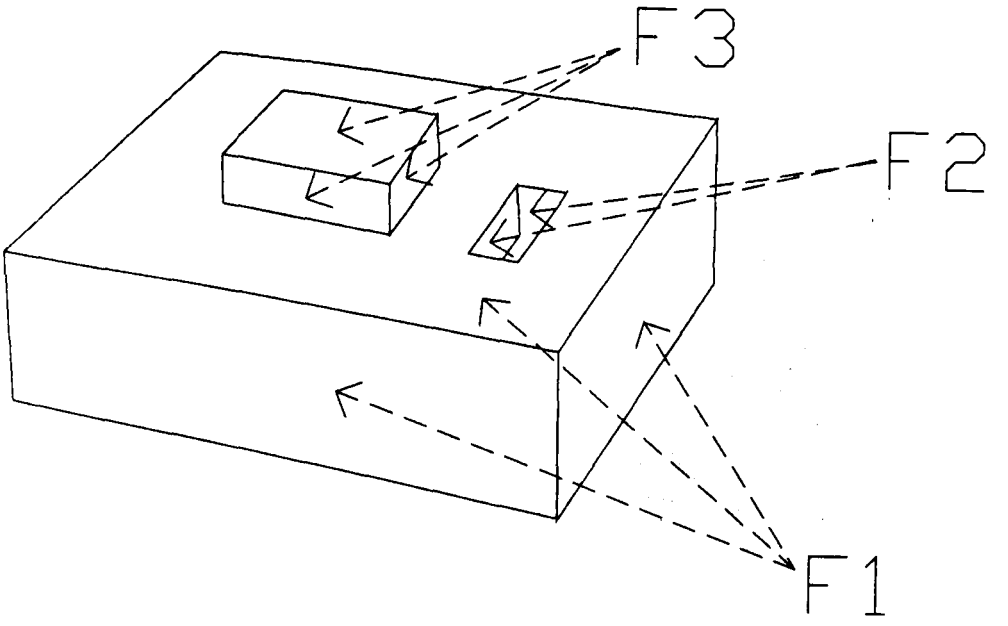


Fig 1.7 Three facesets (F1, F2, F3) generated from a simple shape by Kyprianou's method, corresponding well to intuitively recognized features.

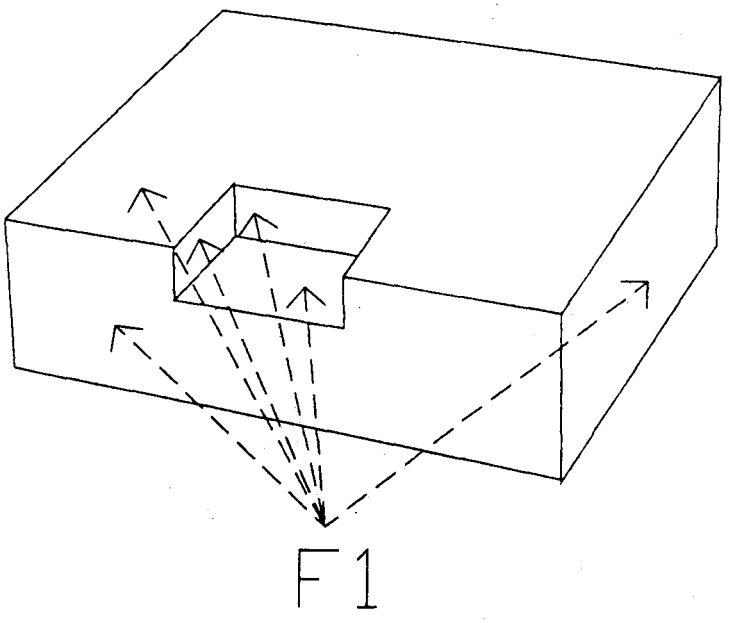


Fig 1.8 A case where a morphological feature is not distinguished as a separate faceset by Kyprianou's method. The faceset F1 covers the entire surface of the object.

To attempt a strict definition of an image database system under such circumstances is clearly pointless. It is, however, important to form some idea of the distinguishing characteristics of an image database system. A true image database system can be regarded as a collection of images, together with descriptive data *derived directly from those images*, maintained in applications-independent form, and organized for efficient storage and retrieval. This excludes specialized collections of test images used in developing pattern-recognition algorithms, databases of manually-assigned text descriptors of images, and data compression systems which provide compact storage of large or complex images, but no retrieval facility.

Some examples of image database systems which meet at least some of these criteria are described below. The review deliberately omits consideration of the many "image databases" which retrieve images purely on identifier or manually-assigned key words.

1.7.2 Geographical information systems

Much of the pioneering work in image database development has been in the geographical area. One early system of note was IBIS (Image-Based Information System), developed at California Institute of Technology's Jet Propulsion Laboratory during the mid-1970's (Zobrist and Bryant, 1980). Its aim was to integrate data from satellite photographs with census and land use data, generating a single spatially-oriented database which could handle a wide range of queries, such as the likely health impact of a new chemical plant. All data were stored as two-dimensional arrays of cells, each of which could contain image data (brightness, colour, etc. of a single image pixel), physical variables (rainfall, population or pollution levels), or identifiers (codes for geographic region or land use, or identifiers for the nearest of a set of specified points or lines - which could be named cities or roads). Any individual cell could be directly addressed from its spatial coordinates, and different categories of information coordinated. A command-based query language was provided, allowing users to aggregate or cross-tabulate different data types, and report the results of such operations either in tabular or graphical form. Queries put to actual IBIS databases have included correlating land use with highway provision, population density with pollution levels, and even rooftop area with electric power consumption (part of a solar power study).

Much of the development effort in the IBIS project was directed towards automating data entry - a major problem for any large database. The problems of aligning administrative boundaries with LANDSAT satellite photographs were successfully overcome, though human intervention was required with non-geometric information such as district names and highway numbers. The use of a single grid format for referencing all types of data was a key feature of the system, ensuring the compatibility needed to allow complex cross-tabulations. From the examples quoted, one suspects that the system was somewhat cumbersome to use by present-day standards, and required users to understand the structure of the database in considerable detail. However, the general principle of reducing all data to a common grid-based format has clearly been found useful by designers of later systems (e.g. Chock et al, 1984).

Another system dating from a similar period was GRAIN (Graphics-oriented Relational Algebraic INterpreter), which aimed to separate out image features and the images themselves into separate data stores (Chang et al, 1977). This was achieved by coupling together two systems: ISMS (Image Store Management System), which held digitized representations of the images themselves, and RAIN (Relational Algebraic INterpreter), a relational database holding a series of relations which together formed a logical description of each picture. In its original version, RAIN held three types of relation - *picture object tables*, naming each significant object in a picture and showing its relationships with other objects; *picture contour tables*, giving coordinates of each object's bounding polygon; and *picture page tables*, indicating where in the physical image file each object could be found. All queries were handled by RAIN; the resulting pictures could then be displayed by ISMS. The system's query language (also called GRAIN) provides the user with commands to display a named picture, sketch a line drawing or paint an image of an object retrieved by name, spatial attribute or similarity with another picture object. Some examples of GRAIN queries are shown in Fig 1.9.

(i) draw a line diagram of all highways in the picture

```
sketch picture; name equal 'highway'
```

(ii) draw a line diagram of all railways with gauge over 120 cm

```
sketch railroad; rgage greater than '120'
```

(iii) display a digitized image of all forests in a given area

```
paint picture; name equal 'forest';
    forest.x less than '40'
    and forest.x greater than '20'
    and forest.y less than '70'
    and forest.y greater than '30'
```

(iv) display all cities similar to a given city (according to a set user-defined procedure)

```
paint city; similar (city.name equal 'Detroit')
     using 'PROC1'
```

Fig 1.9 Examples of pictorial queries in the GRAIN language

While GRAIN - also known as DIMAP (Chang, 1980) - clearly embodied some sophisticated ideas, it is far from clear whether any of the information in GRAIN was derived directly from the pictures themselves. If the picture descriptions were indeed laboriously extracted by hand (and eye!), and entered into RAIN just like text descriptors, one could argue that all that was achieved, in Nagy's terms, was to link a nonimage database system to an image nondatabase system.

One system which, according to its designers, did successfully extract semantic information from satellite images into a data base was REDI (RElational Database for Images), also known as IMAID (Chang and Fu, 1980). Like GRAIN, it incorporated a separate image store and relational database; in addition, it contained powerful feature recognition software capable of processing LANDSAT images using syntactic pattern recognition techniques. The software identified specified objects in LANDSAT images, including roads, rivers, cities and meadows, creating database entries showing the position and orientation of each instance of these objects found, and producing line drawings showing the results of this operation in graphical form. Human intervention was still necessary to add identifying names for cities, roads and rivers - otherwise the degree of automation was almost complete.

Another striking feature of REDI was the query language chosen. Rather than use a conventional command-based language, Chang and Fu used an extension of the menu-based QBE (Query-By-Example) language (Zloof, 1975), which they designated QPE (Query-by-Pictorial-Example). The reasoning behind this was that unskilled users would find this easier to use than a command-based language, particularly for complex queries. Its mode of operation was very similar to QBE; the user was presented with a screen displaying a blank table, on which the name of the relation(s) likely to contain the answers to the query could be entered. The system then filled in the column headings on the screen table(s) with domain names from the specified relations, and the user typed in an example of the answer required (Fig 1.10). The system then responded by displaying actual answers in the format specified by the user - a table of results, a sketch, or a display of part of the original image. The range of queries handled was quite extensive: as well as retrieving named entities such as cities, the availability of additional spatial operators (such as DISTANCE, SLOPE, AREA, PERIMETER, and INTERSECTION) meant that questions such as "find the length of highway 65 within Lafayette city" could easily be answered. As with GRAIN, a similarity retrieval operator was provided, though nothing was reported about how (or even whether) this feature had in fact been implemented.

While the ease of use of QPE cannot readily be assessed (from the examples given, complex queries appear just as difficult to express in QPE as in any other command language), REDI was definitely the most impressive of the early systems. It provided automated processing of image data, generating a database capable of answering a wide range of queries. Clearly its feature-recognition capabilities were limited to those few features for which recognition algorithms were constructed. It remains, however, the closest approach to a true "image data base" of any of the geographic information systems reviewed here.

Although not a geographic information system in the strict sense, the recently-reported "intelligent image database system" of Chang et al (1988) falls into the same general category in that it was designed primarily to answer spatial queries of the type "find me a picture with a car to the east of a house". In the present version of the system, pictures are encoded (manually!) as 2-D character strings indicating the position and type of each significant picture object. The resulting *iconic index* can be queried in a variety of ways, and pictorial representations of retrieved scenes displayed on the screen. Future versions of the system are planned to have the capability to extract 2-D indexing strings directly from the images themselves. Not until then will this (admittedly ingenious) system qualify as a true image database.

(i) print the names of roads in the same frame as the city of Lafayette

CITYNAME	FRAME	CITY ID	NAME
	10		Lafayette

ROADNAME	FRAME	ROAD ID	NAME
	10		P.X99

(ii) sketch the part of interstate highway 65 falling within a given geographical area

ROADNAME	FRAME	ROAD ID	NAME
	4	17	H65

ROADSEG	FRAME	ROAD ID	X1	X2	Y1	Y2
S.	4	17	22	32	41	49

Fig 1.10 Examples of queries formulated using QPE. The user enters search terms in the appropriate columns of the query tables ("Lafayette" in the first query, "H65" and (x,y) coordinate values in the second), and then gives dummy examples (underlined) of the type of answer he requires, together with the command "P" (print) or "S" (sketch), depending on the form of output required

1.7.3 Medical image database systems

Recognition of abnormal features has always been a key aspect of medical diagnosis: this century has seen rapid growth in the importance of techniques such as radiology (the use of X-ray photography and other scanning techniques) and cytopathology (identification of abnormal cell types). The advent of techniques such as computerized tomography (CT), where a computer-guided scanner produces a series of cross-sectional images of a patient's anatomy, has generated large numbers of digitized images suitable for automatic processing. While many so-called medical IDBs are really no more than collections of images retrievable by patient name and date, some of the systems described in the literature do automatically derive descriptive information from the images themselves. Huang et al (1980) describe a system to extract geometric features of body segments or internal organs from CT scans, and save these for future retrieval. Toriwaki et al (1980) have presented details of a feature extraction system for chest X-rays, which derives annotated sketches from each image, showing the rib cage, lung borders, and any suspicious shadows (abnormal blood vessels or tumour fragments). Medical staff can use such sketches for quick review, allowing them to select radiographs of interest. The system can also compute certain information about the image, such as the size of the heart shadow, and select images on that basis. Yokoya and Tamura (1982) describe a cytopathology database, where slides can be retrieved by a combination of image and non-image features. It uses a relational approach similar to that of GRAIN. Frasson and Er-Radi (1986) describe a partly-implemented icon-based query system designed to identify and characterize examples of diseased organs. Like most of the geographical systems described in the previous section, however, few of these systems can be called true image databases, as their ability to derive indexing features directly from stored images is in most cases very limited.

1.7.4 Other pictorial information systems

Some of the most sophisticated pictorial information systems described to date are the fingerprint matching and retrieval systems developed for use by police forces in the UK and America. Systems of this kind are among the very few that qualify unequivocally as image databases, as retrieval features are extracted automatically from images without human intervention. Though few details are available on the inner workings of such systems, a brief description of one such system (IEEE, 1985) indicates that pattern recognition techniques are used to identify the number, position and orientation of *minutiae*, parts of the fingerprint where ridges begin and end. A similarity measure can then be computed between an "unknown" print and each print in the database, and the ten most similar prints displayed for human examination. The search process is slow, as the entire database is searched sequentially, though the use of parallel processors is expected to improve system performance substantially.

A further example of what appears to be a true image database came to the author's attention shortly before the completion of the present project. This is the GRIM_DBMS system reported by Rabitti and Stanchev (1987b, 1989), which aims to provide retrieval of specific types of diagram (such as charts and graphs in business reports, or office floor plans) on the basis of the objects contained within it. Although the system is claimed to be of general applicability, any given instance of the database can handle only a limited domain of drawings. Drawing elements (lines, arcs, points, text strings) are analysed, and used to infer the probability that the drawing contains instances of certain specified objects, such as titles from bar charts in a business graphics context, or desks and chairs from office floor plans. These probability values are then used in two ways; firstly, the techniques of cluster analysis (Everitt, 1980) are used to group together images with similar content, and secondly, indexes are created for each object type, indicating the images in which they are most likely to be found. A relational-type text-based query language is provided for retrieval. While no details of system effectiveness are provided, the concepts involved seem sound, even if the system as described lacks flexibility in that the data base designer would have to specify to the system in advance what types of object were of interest, and provide the system with detailed inference rules for recognizing that object from primitive picture elements.

1.8 Scope of the present project

There appears to be evidence, both from the literature, and from discussions with designers and users of CAD systems, that data management for CAD is likely to play an increasingly important role in the future. The ability to retrieve by feature is at present in its infancy - yet there is a growing body of evidence that it would prove a useful adjunct to future CAD/CAM systems, allowing designers and production engineers to recognize when a "new" design shared major structural features with existing parts without the need to resort to time-consuming manual parts classification. In other fields where pictorial information is handled, especially in geography and medicine, the feasibility of such systems has been demonstrated, though it has to be admitted that few such systems are operational outside the research laboratory. The fact that several systems have used pattern recognition techniques for feature identification, and relational database for data storage and retrieval, is noteworthy - though it is not immediately clear whether this is because of the overwhelming superiority of such techniques, or is just a bandwagon effect. The use of alternative data models deserves more investigation than it has so far received.

The design of an engineering database with shape feature retrieval capabilities is clearly a feasible proposition. However, it requires careful investigation of a number of alternative approaches before design decisions are made. The key issues to be faced in designing an engineering database capable of feature retrieval can be grouped under the following headings:

(a) Scope of the database: what type of model should be included (2-D or 3-D), and what should constitute a basic pictorial entity in such a database? In what input format(s) should drawings or geometric models be accepted?

(b) Storage structures: is the entire object representation to be stored, or just a description of its shape features? How exact a representation is required? To what extent is it necessary to represent each object in a strict canonical form? How does one choose the most suitable format to represent the geometry, topology and shape features of an object? What is the most appropriate database model to support such representations?

(c) Feature extraction: what types of feature are likely to be most useful for retrieval? How can they most effectively be extracted?

(d) Retrieval capabilities: what level of retrieval should be provided? Is a simple indication of part family sufficient, or are Boolean combinations of features or sophisticated similarity matching required? What kind of matching algorithms are necessary, and to what extent are they provided in existing database management or information retrieval systems? How should retrieval by geometric feature be combined with retrieval of non-geometric items such as text descriptors?

(e) Interface design: what kind of query language is most appropriate: something completely novel, or an adaptation of existing languages, such as QPE? How should graphical input of queries be managed? What level of interaction with the user is required? What constitutes the answer to a query - listing of tabular information, display of a drawing, highlighting key parts of a geometric model, or making available a copy of the original design file?

(f) Evaluation: what kinds of performance measure should be applied to a system such as this, and how does one construct appropriate benchmarks?

Although listed separately, these design problems all interact. The form of object representation chosen, the processes of feature extraction and similarity matching, the retrieval capabilities offered, and the type of interface provided are all mutually dependent decisions which cannot be taken in isolation from one another. The aim of this project is to investigate this complete range of problems, by designing, implementing and evaluating a database of engineering drawings capable of retrieving objects by shape feature, and allowing graphical formulation and

refinement of queries. Later chapters of this thesis examine these design issues, their interaction, and their influence on the eventual system design, in some detail.

As far as is known, this project, with its emphasis on the totality of the problem and the mutual interaction of its sub-problems, is unique. Most other reports of graphics-oriented databases in the literature have focussed on different issues. The engineering databases discussed in section 1.4 are generally aimed at improving security and integrity of data, not providing users with additional retrieval capabilities. Indeed, the trend seems to be to pass semantic retrieval problems back to the original designer, rather than to design systems capable of tackling them directly. The geographical databases outlined in section 1.7 provide retrieval on the basis of an item's position and function, not its shape, and most still rely heavily on text-based query languages. The question of generating canonical representations of stored objects has not been raised, and may not even be relevant in this field. Medical image databases such as that described by Toriwaki et al (1980) are closer in concept, in that graphical input and output are considered an integral part of the overall design. However, little consideration seems yet to have been given to shape retrieval, and none to the question of canonical shape representation (which one might expect to be more relevant in this field). None of the systems described have been subjected to any systematic evaluation, even the recent work of Rabitti and Stanchev (1987b, 1989), which is probably closest in concept to the present study.

The remainder of this thesis describes in detail the development and evaluation of a prototype shape retrieval system for engineering drawings, to be known as SAFARI (Shape Analysis For Automatic Retrieval of Images). Chapter 2 discusses the problem of drawing representation for shape retrieval, arriving at conclusions whose implementation is described in Chapter 3. Chapter 4 analyses the problems of feature selection and extraction, and chapter 5 discusses the applicability of different data models to support the required drawing and feature representations. Chapter 6 analyses the retrieval capabilities needed by any such system, and describes the matching processes adopted for the prototype. Chapter 7 examines the question of interface design. Chapter 8 presents an evaluation of the system's retrieval effectiveness; finally, chapter 9 discusses the implications of the project's findings for 3-D shapes, and for pictorial information systems in general.

CHAPTER 2. FORM AND SCOPE OF OBJECT REPRESENTATION

2.1 Introduction

This chapter explores two of the most fundamental issues in the design of a shape database: what is the most appropriate scope for such a database, and how can objects stored within the database best be represented? Like many design issues, these questions are interrelated. A database of sculptured parts such as body panels for cars or aircraft would require more powerful representation techniques than a database of simple machined parts. Essentially 2-D objects such as sheet metal parts or VLSI designs could be adequately handled by much simpler forms of representation. The scope most appropriate for a prototype database used as a vehicle to explore design problems is in any case unlikely to be appropriate for an operational database for use in a design office.

2.2 Scope of the database

2.2.1 Type of drawing

A fundamental - and universally relevant - distinction needs to be made at the outset between drawings (or models) on which the designer is still working, and completed drawings, describing objects which are past the design stage. (Libraries of standard parts as described by Ketabchi and Berzins (1987) fall into the second category). The database requirements of these two types of drawing are quite different. In the first, the designer's prime requirement is to retrieve a specified drawing or model in order to add, modify or delete drawing elements. He or she therefore needs easy access to each individual drawing or solid modelling primitive. Implicit shape features are of minimal importance to the designer at this stage, so shape retrieval is unlikely to be a worthwhile facility to offer. For such purposes, the database models outlined in Section 1.4 are likely to prove perfectly satisfactory. The main function of a DBMS here is to maintain the security and integrity of "live" design files, and aid in version control.

The second type of drawing (or, more properly, the second stage in the life of a drawing) poses different requirements. It is not expected that further design work will be necessary, so that rapid access to each individual line and point is of much less importance. The main requirement here is that the drawing should be retrievable from the archives (whether paper or computerized) whenever there is a need for a component with specified design features. A design database set up for these purposes thus needs above all to be able to provide retrieval by feature. Security is still important; completed drawing files should normally have 'read-only' status.

So far, most of the database development work reported (see Section 1.4) has been concerned with "live" drawing files, with the prime aim of improving ease of access to individual components in large drawings. The investigation of ways of meeting the special requirements of databases suitable for completed drawings ("consolidated design files" in IBM parlance) has received much less attention. This balance needs to be redressed.

There is an interesting parallel here between the needs of a design engineer and a software designer using an integrated programming support environment (IPSE). IPSE's have very similar requirements for access control and integrity maintenance for software modules under development. Just as the design engineer requires access to each drawing element in a "live" drawing file for modification, the software developer needs access to programs under development at the level of the individual line of code, via a text editor. If engineers feel a need for retrieval of completed drawings by feature, perhaps programmers might be interested in retrieval of completed program modules by overall

function. The question of how to define 'function' in this context is even harder than it is for drawings.

2.2.2 Two or three dimensions?

To be of real usefulness to the majority of design engineers, a database offering retrieval by feature needs to operate on 3-D models. Few, if any, genuine two-dimensional objects exist (the closest approach, as observed above, being sheet metal parts or VLSI layouts), though these are the only objects which can be directly represented in a single 2-D drawing. For the vast majority of objects, key aspects of feature similarity are not obvious from a single drawing, becoming apparent only when 3-D models are compared. Designers are clearly far more interested in similarity between two objects than between the drawings that represent them. However, the use of 3-D geometric model ling is still far from widespread; many engineering firms seem likely to continue using 2-D draughting packages for perhaps years to come. This suggests that a storage and retrieval system capable of handling input from both 2-D drawings and 3-D geometric models would be desirable - though (as discussed below) the difficulties of interpreting 3-D shapes from 2-D drawings are considerable.

For the present study, however, it was decided to restrict the database to 2-D drawings of essentially two-dimensional objects. This was done for two reasons. Firstly, the IGES standard (the only interchange format in which drawings were readily available at the outset of the project) provides an adequate description only for 2-D drawings. Any 3-D database would therefore have to be linked to a specific geometric modelling system such as ROMULUS, reducing its general usefulness (see section 2.2.3, below). Secondly, the complexities of casting 2-D object representations into standard form and devising suitable forms of query input were felt to be sufficient to provide a more than adequate vehicle for exploring the relevant design issues. The solution of the design problems for 2-D objects was felt to be a necessary precondition to tackling the more difficult case of 3-D objects.

The domain of objects to be included in the test database was thus limited to the simplest class of genuine CAD drawings - 2-D objects that could potentially have been stamped out of sheet metal. The set of picture processing algorithms developed to cast these into unique form is valid only for such objects. Each drawing consists of a continuous outer boundary made up of a sequence of one or more contiguous line segments. This boundary may enclose zero or more internal shape features, voids defined by continuous inner boundaries made up of one or more line segments. Each boundary line segment can be either a straight line or a circular arc, and is contiguous with precisely one neighbouring segment at each end. Voids cannot themselves enclose further shape features, and no two boundary segments may touch (other than as specified above) or intersect. The composition of such shapes can be defined in Backus-Naur form as follows:

```
            <shape> ::= <outer boundary> {<inner boundary>}
   <outer boundary> ::= <boundary segment> {<boundary segment>}
   <inner boundary> ::= <boundary segment> {<boundary segment>}
<boundary segment> ::= <straight line> | <circular arc>
```

where { } indicates zero or more repetitions of the enclosed symbols, and | indicates a choice between alternatives. Typical shapes conforming to these rules are illustrated in Fig 2.1.

Such shapes, while representing a very limited subset of those in engineering use, should provide a sound basis for subsequent generalization to boundary representations of 3-D objects. Generalization to "two-and-a-half"-dimensional (2.5-D) parts - objects where every principal face is either parallel or perpendicular to every other (Fig 2.2) - is simply

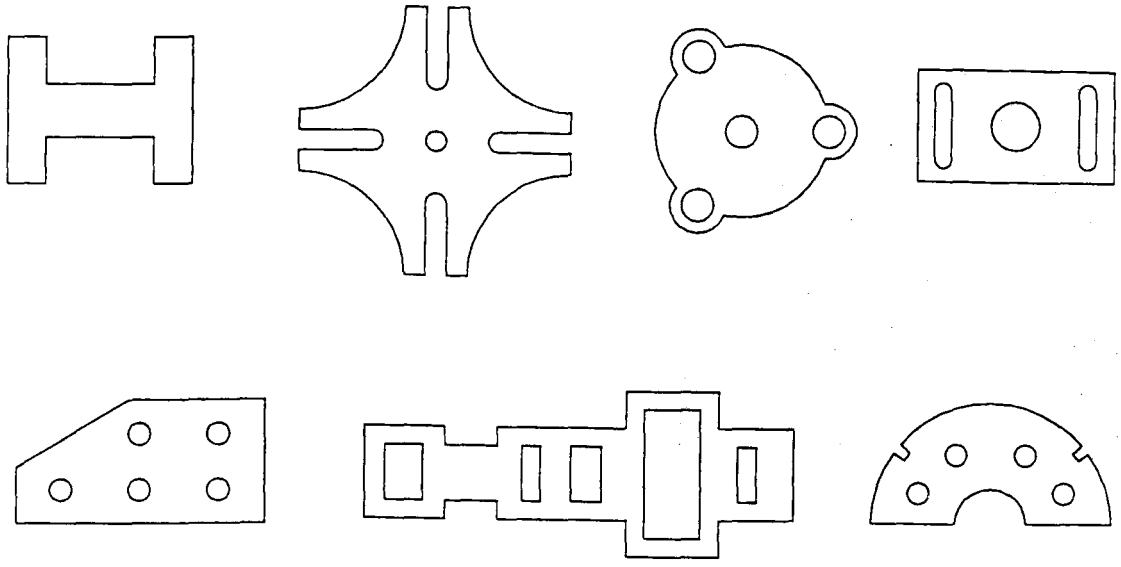


Fig 2.1 Examples of some of the 2-D shapes included in the scope of the test database. Note the restriction to objects (such as sheet metal stampings) that can be accurately represented by a single 2-D drawing

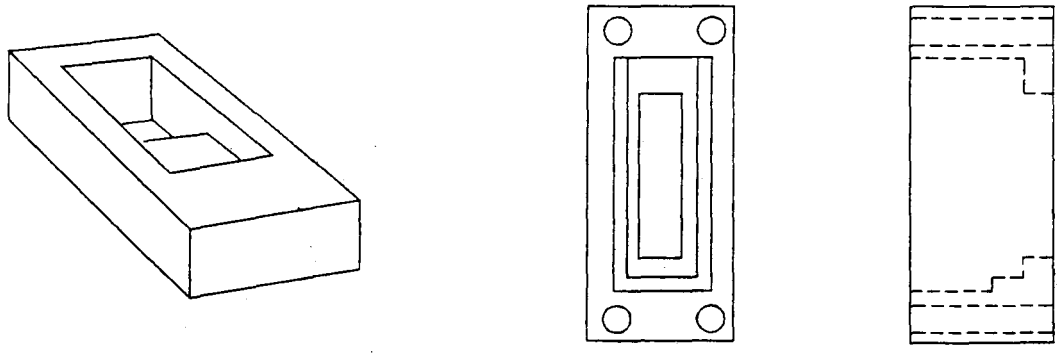


Fig 2.2 Examples of "two-and-a-half" dimensional objects

a matter of indicating the height or depth of each interior feature, and relaxing the restriction that inner boundaries cannot themselves enclose further shape features. Generalization to 3-D boundary representations can be achieved for most but not all true 3-D objects by defining each face of the object in the manner shown above, and indicating the topological relationships between faces. (The main exception to this is the class of sculptured parts, where the geometry of a face cannot be specified completely by defining its bounding edges). The validity of extending the approach developed here for 2-D shapes to 3-D objects is discussed further in section 9.3.

Generalization of these restricted 2-D shapes to conventional engineering drawings (2-D projections of 3-D objects), such as that shown in Fig 1.1, is simple at one level, but much more difficult at another. The only difference between drawings of the restricted set of shapes shown in Fig 2.1 and ordinary engineering drawings is that boundary lines can touch or intersect, and that inner boundaries can enclose further drawing features. Extending processing algorithms to cope with such drawings - and therefore to assess whether two *drawings* shared a given feature - would not be a difficult task (though the computational effort would be significantly greater). However, this would not necessarily uncover similarity or common shape features in the objects themselves. Shape retrieval of a 3-D object represented by 2-D orthographic projections is possible only if its 3-D geometry can be reconstructed automatically from the drawings. As discussed later (section 9.3), this is a major problem in itself.

2.2.3 Input format

The choice of input format for objects included in the database is also important, irrespective of the scope and type of object representation chosen. Three main alternatives are available: (a) input by scanning of existing (paper) drawings, (b) input from drawing files of proprietary CAD systems such as DOGS or ROMULUS, or (c) input in a standard exchange format such as IGES.

Input by scanning pen-on-paper drawings has the advantage that virtually all engineering drawings are available in this format, whether computer-produced or drawn by hand. But it hardly seems appropriate to make this the main form of input to a database of computer-generated drawings, particularly when it renders direct input of 3-D models impossible. (As indicated above, 3-D structures would have to be inferred from 2-D projections - a process still not well understood). An additional input module to accept pen-on-paper drawings could be of benefit to any operational system, but hardly warrants study as a research topic, as the tasks involved in converting 2-D drawings to machine-readable form are well-characterized, and several pieces of proprietary software capable of digitizing engineering drawings (though not capable of recognizing shape features) are already on the market.

Input from drawing files of a draughting or geometric modelling package has the advantage that full advantage could be taken of any structure present in that system's files. Thus a system taking its input from ROMULUS would have the advantage that all models would have been checked for structural integrity, and that the topology of the entire object was explicitly specified. This would make the identification of facesets or other features of structural significance a relatively straightforward task. However, it does limit the database to models generated on a single host system. There is also the problem that details of the internal data structures of systems such as ROMULUS are not fully-documented - and are in any case liable to change in the future.

To be of general usefulness, a computer-based drawing archive needs to be able to take input from a variety of draughting and geometric modelling systems. It can in fact be argued that the main justification for such a system lies in its ability to store and compare designs that have been produced over a long period of time, perhaps on several different CAD systems. This implies that it must accept input in a widely-used standard format.

The only existing format in widespread enough use to meet these criteria is IGES, described in detail in Section 1.3. Any system which accepts input in standard IGES format is capable of incorporating drawings from a wide variety of CAD systems.

Using IGES as principal form of input has its disadvantages. Its major problems are (a) lack of consistency in representing geometric information (see section 1.3 above), particularly where curved segments are concerned, and (b) failure to make the inclusion of topological information mandatory. Together, these mean that the receiving system has to standardize line representation, and infer topology, before processing to uncover shape features can begin. For future extensions to 3-D models, it needs to be remembered that although more recent versions of IGES have been extended to cope with 3-D object descriptions, its basic format was designed with 2-D drawings in mind, and has limited capacity to handle topological information. In particular, it has no way of specifying checks on the validity of the resulting object. As a standard, IGES is now obsolescent, though its influence can be clearly seen in some later standards proposals - for example, the XBF standard for solid model exchange shares IGES' five-section structure, its numeric identifiers for entity types, and its use of 80-character card image format (Mason, 1985). For these reasons, it was felt that IGES represented a better choice than any other alternative as the input medium to the prototype version of SAFARI. Adapting input routines in the future to cope with IGES' successor standards should not pose any major problems.

2.2.4 What constitutes a single object?

In the long run, a decision also has to be made on what constitutes a single retrievable entity in a database of drawings. In most CAD systems, a drawing or geometric model occupies an entire file, with a separate record for each drawing or modelling primitive. Thus storage and retrieval facilities are geared to finding individual lines in a given drawing. As discussed in Section 2.2.1, this meets the needs of the designer working on a "live" drawing - but is much too low a level for completed drawing files. Here, the unit of retrieval needs to be an entire drawing or geometric model; this implies that a drawing should occupy a single record, and that files should in general contain a number of drawings - though if a database management system is used, a wider range of structures is possible, and the distinction is less important. Even here, as the previous discussion on object-oriented database models makes clear, it is important for users of the system to have a way of identifying an entire drawing as a single object, rather than having to synthesize it afresh from its constituent lines, faces or shape features each time they wish to retrieve it.

The situation is complicated by the fact that there is not necessarily a one-to-one correspondence between drawings and the objects they represent. Two or more orthographic views of an object may well be provided - possibly on different drawings. The fine detail of part of a complex object may be shown separately on a larger scale. Again, the presumption must be that designers are primarily interested in knowing about objects themselves rather than their drawings. In a database of 2-D drawings of 3-D objects, drawings representing different views of the same object would need to be linked together, at least at the logical level, so that if a part has features of interest, all relevant drawings can be displayed.

The problem can be avoided by choosing a representation appropriate to the objects in the database. In collections of 2-D objects represented by 2-D drawings, and 3-D objects represented by 3-D geometric models, there is always a one-to-one correspondence between object and representation - one respect in which the restricted domain of 2-D shapes chosen for the present study forms a good model for eventual 3-D database design.

2.3 General principles of shape representation

One fairly fundamental consideration concerns the amount of information from the original drawing to include in the database. Is it necessary to store a representation of the entire object (either in its original or a condensed form), or is it sufficient to store a set of shape descriptors, together with a reference to the location of the original object or drawing? This closely parallels the decision made in a text retrieval system on whether to incorporate just the title of a document, an abstract summarizing its contents, or the entire text. Storage of a representation of the complete object (rather than a set of shape descriptors) could be valuable for two reasons. Firstly, it would allow an image of the object to be displayed readily on demand, a useful feature during an interactive query session. Secondly, it would allow the possibility of generating retrieval features at run time if this were required. While it is not known at present how important such a facility might be, the difficulty of anticipating all types of queries put to a pictorial retrieval system suggests at the very least that the possibility of run-time feature generation deserves investigation.

There is, of course, no compelling reason why the same form of representation should be chosen for the object itself and its feature set. The GRAIN and REDI systems both use two distinct data stores - a feature database and an image file, used respectively for query handling and picture display. Such a solution clearly makes sense when one considers that one is handling two quite distinct types of data, with different access requirements. The feature database consists of (relatively) well-structured data, rich in semantic information, with a requirement for access by a variety of routes. The image store contains largely unstructured data, and requires only a single access path. The same distinction can be applied to engineering drawings and their shape features, so a similar solution could well be appropriate for an engineering design database. One could for example retain IGES format files for display purposes, but link these to a shape feature database that allowed users to search for objects with desired feature characteristics. In practice, as shown below, far more compact representations than IGES exist; the general principle of separating display images and shape features is still valid. (Note that it could in any case be necessary to retain IGES-format files to allow regeneration of "working" drawings to use as the starting-point for a modified design).

The choice of representation adopted for design objects is crucial, as it affects both the processing required to extract shape information and the type of information that can be derived. The question of computer representation of design objects has been discussed in depth by Requicha (1980); while his discussion focuses on 3-D objects, much of it is also relevant to representations of objects that are essentially 2-D in nature. Mathematically, one can regard any representation scheme as the mapping from domain D, the set of representable objects, to range V, the set of valid representations of objects in D. The scheme is *unambiguous* or *complete* if each representation in V corresponds to a single object in D; it is *unique* if each object in D has but a single representation in V. (Such a unique representation is often referred to as *canonical* form). It is unique and unambiguous if there is a one-to- one mapping between every object in D and every representation in V.

Different representation schemes can be evaluated by these and other properties. The *domain* of a scheme is a measure of its descriptive power, denoting the variety of objects for which it can be used. The extent to which the *validity* of each representation can be ensured, by syntax checking or other means, is important when considering database integrity. *Completeness* or lack of ambiguity, at least within a restricted domain, is essential if a representation is to be of practical use in communicating an engineer or architect's ideas. *Conciseness* is clearly an important consideration if a large database is to be maintained, as is the *efficiency* with which representations can be created and manipulated.

Uniqueness is a different matter. For most purposes, there is no advantage in ensuring that a given object will always have an identical representation, as long as its different representations (drawings at different scales or orientations, for example) can always be recognized as referring to the same object. Most representation schemes in actual use are not in fact unique.

There are, however, cases where uniqueness might be important, such as when a database needs to be searched to identify whether two objects are identical. Only where it is certain that an object's representation is unique is this a computationally simple task. If multiple representations of the object are possible, such a question can in general be answered only if all possible object representations can be generated and matched in turn with the query representation. Similarly, the task of deciding whether two objects share a specified shape feature can be made much simpler if both are known to be uniquely represented. A database specifically designed to answer such queries might well depend for efficient operation on a unique representation scheme for stored shapes. Since no widely-used scheme has this property, this implies that a new form of representation may need to be developed.

There is however one problem with this approach. While it is perfectly valid to use the concept of canonical representation in the context of shapes that can be defined with complete precision, actual engineering parts can only be defined within a specified range of tolerances - for example, a given side may be 30 ± 0.01 cm long, an angle 45 ± 0.1^{o}. This renders uniqueness a somewhat elusive concept, since each stored drawing in fact represents the *set* of all objects falling within the specified tolerances. In this context, a truly unique object representation would imply that for every set of objects, identical except that their dimensions could vary within specified tolerances $\{t_o\}$, a corresponding set of representations exists, identical except that their defining parameters fall within correspondingly small tolerances $\{t_r\}$. This cannot in practice be fully achieved; some of the inherent difficulties approach are discussed in chapter 3 below, though a full treatment of the subject is beyond the scope of this thesis. (As indicated in Requicha & Chan (1986) and Turner & Wozny (1987), the question of tolerancing in CAD systems is a major research topic in its own right).

This does not however rule out the possibility that a more restricted definition of uniqueness could have value in the situations described above. If it is known that shape elements are always represented and stored in a standard order, the process of comparing shapes to determine whether they are the same is in principle a simple matter of sequentially comparing corresponding shape elements - an $O(n)$ process. If no such standard ordering exists, so that corresponding shape elements could appear anywhere within their stored representations, each shape element in one representation has to be compared with every corresponding element in the other, an $O(n^2)$ process at best. At the very least, then, a scheme which attempts to define a standard order for individual shape elements should be able to reduce the complexity of the matching process - at the risk of failing to match a certain (hopefully small) proportion of shapes for which the ordering rules give anomalous results. It can thus be regarded as a heuristic whose usefulness can be empirically tested.

2.4 Representation schemes for 2-D objects

As noted above, few, if any, genuine two-dimensional objects exist, though some have virtually no 3-D component, and others (principally geographic features such as coastlines, river basins, road and rail layouts) have a third dimension can safely be ignored, and the 2-D image of the feature treated as the prime object of interest. Two main representation schemes have been used for 2-D images: (a) *line* or *vector* format, where each object is represented by a series of line segments (straight or curved) forming its boundary, each line segment in turn being represented by the coordinates of its end-points, its defining equation or by an appropriate code; and (b) *raster* format, where each

display element or *pixel* forming the image is represented by an element of computer storage indicating its brightness and possibly colour. Raster format is of prime use where the main interest of the picture lies within each region of the image (for example, the overall colour, texture or brightness of a region are important), and for this reason it has been widely used in geographical information systems, from IBIS (Section 1.7.2) onwards. Its principal advantage is that it provides a simple means of representing surface texture, colour and shadow, and also allows ready correlation of different types of data (e.g. population density and land use) relating to a given point or area within a region. Its main disadvantages are its verbosity (pixel values have to be listed for an entire image, even for blank regions, though techniques such as run-length coding can alleviate the problem), and the difficulty in recognizing object shapes. Even to identify a straight line embedded in a raster image is far from simple; the identification of complex shape features from such images is a task that has been occupying the computer vision research community for many years. (It is interesting to note that the first step in picture processing for computer vision applications often involves extracting all boundary lines in the image, reducing it to a line diagram which can be more easily manipulated). An interesting variant on raster format is *quadtree* representation (Klinger and Dyer, 1976), which produces a hierarchical representation of an image by taking an n x n pixel array, and dividing it progressively into quarters until leaf nodes contain pixels which are all of the same colour or density. Such a format is generally much more compact than pixel enumeration, and it is claimed that object boundaries can easily be identified. However, neither this format nor its 3-D analogue, the *octtree*, have found any favour with designers of CAD systems. Its strength lies in its ability to represent irregular shapes such as those generated by computer vision systems. Faced with regular objects defined by smooth curves and straight lines, it generates unnecessarily complex and inelegant representations compared with those described below.

Vector format is the most appropriate representation where objects' boundaries are the main focus of interest - invariably the case with engineering drawings, where objects are defined by the set of line segments making up their boundaries. All CAD systems, as far as is known, use line format as the basis of their 2-D representations. Drawings in line format provide an unambiguous, relatively compact representation for parts which are essentially 2-D, though representations are not in general unique. (Coordinates of line end-points and coefficients of defining equations are both sensitive to translation, rotation and scaling). An ideal representation scheme for objects in a design database would almost certainly be based on line format; considerations of efficiency dictate that it should provide a unique representation for each object, and therefore for its constituent line segments. This is in fact a crucial design issue for both 2-D and 3-D objects, as line segments are key components of boundary representations of 3-D objects. It is therefore necessary to examine methods for representing line segments in some detail.

2.5 Representation of line segments

2.5.1 What is an edge?

At first sight, this may seem a trivial question, at least for 2-D object representations; an edge is a line segment forming a subset of the object's boundary. The union of all such segments thus defines the object's boundary. An edge may be a straight line, or an analytic or parametric curve. Thus a rectangular object can be defined by four straight lines, as in Fig 2.3. However, the same object could have been drawn using a CAD system's "mirroring" facility, and hence represented by six (Fig 2.4) or eight (Fig 2.5) line segments. Does each of these constitute an edge? Most observers would answer "no"; the rectangular object is bounded by four edges, however its drawing may have been constructed. It is not difficult to come up with an intuitively satisfactory definition of an edge in this case; an edge is the union of one or more continuous line segments. Two edges meet at a vertex, where there is always a discontinuity. Under this definition, it is

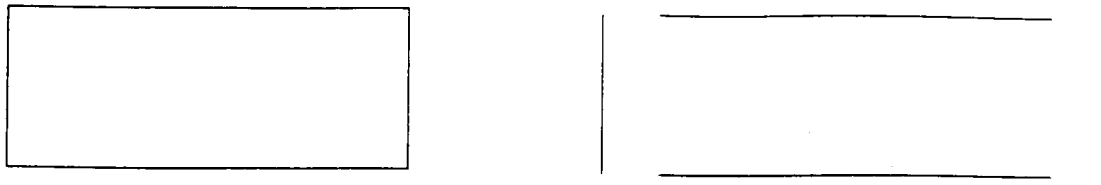


Fig 2.3 A rectangular shape, naturally segmented into four edges

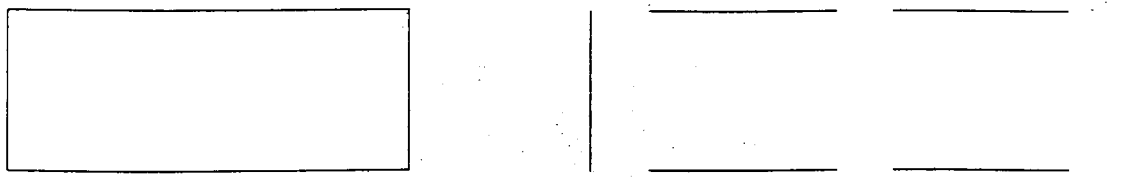


Fig 2.4 The same shape, produced by mirroring, now represented as six edges

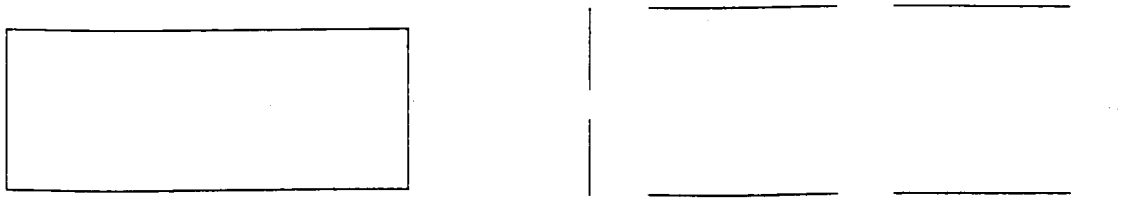


Fig 2.5 A rectangle produced by double mirroring, represented as eight edges. A unique representation scheme has to recognize all three forms of the rectangle as equivalent

immaterial how many line segments are used to construct the edges of the rectangle in Fig 2.3; it will always have four edges and four vertices.

Less straightforward is the task of defining an edge which includes curved segments. There are no discontinuities at all in the "canoe" shape of Fig 2.6, though most observers of such a shape intuitively feel that (like the rectangular object of Fig 2.3) it is made up from four edge segments (Fig 2.7). To cope with this situation, one has to narrow the definition of continuity; an edge consists of the union of one or more line segments which are continuous both in respect of direction and curvature. This would yield four edges for the shape in Fig 2.6 even if it had been produced by mirroring as in Fig 2.8.

There remains the problem of smooth curves of varying curvature. While these are not common in machined parts (as opposed to sculptured surfaces such as car body panels), any representation scheme which claims to be generally applicable must be able to handle them. An example of such a part is the rocker arm shown in Fig 2.9. How, if at all, should this boundary be segmented? Again following intuition, one might decide that the boundary should be segmented where curvature changes sign (Fig 2.10), and possibly where there are easily-observable step changes in curvature (Fig 2.11). In formulating a definition of an edge, however, one needs to be able to distinguish between situations where the designer has specified a step change in curvature and situations where a complex surface has been approximated by a series of straight lines or circular arcs. This is part of the wider issue of the extent to which curved segments can be represented in canonical form, which will be discussed in detail below. An essential prerequisite for this process is the need to ensure that a single edge is recognized as such, even though one designer may have represented it as a single elliptical segment, another as a series of circular arcs, and a third as a parametric spline. Changes of curvature that were clearly intended by the designer must be distinguished from those that are merely artefacts of the CAD system used.

The most straightforward way to achieve this is to establish a threshold value for curvature change, and assume that changes below this value do not signify discontinuities. The principle of a threshold accords well with empirical evidence that human observers find it difficult to perceive small changes in curvature, though no studies appear to have been carried out so far to establish the smallest change of curvature that can be perceived (Asada and Brady, 1986). One would then define an edge as the union of straight or curved line segments (whose curvature could be constant or continuously varying), containing no angular discontinuity or significant change of curvature. A significant change of curvature would be defined as either a change in the sign of curvature, or a change in its magnitude greater than a given threshold value. This issue is discussed in more detail in Chapter 3.

2.5.2 How should edges be represented?

No universally-accepted criteria exist for answering this question for any given application. Given the system objectives discussed above, it is possible to derive a reasonable set of criteria which any unique edge representation scheme should meet, as follows:

(a) It must faithfully preserve all information of interest; in particular, it should be possible to reconstruct the original object from its representation to a given level of accuracy.

This is clearly necessary if the system is to be able to support exact shape matching, display of retrieved structures, or reliable feature extraction.

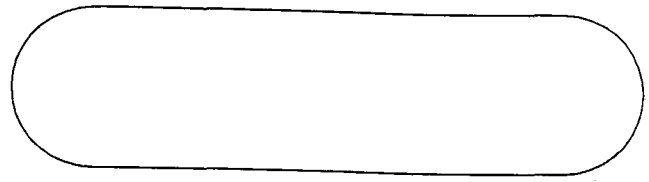


Fig 2.6 A "canoe" shape, containing no angular discontinuities

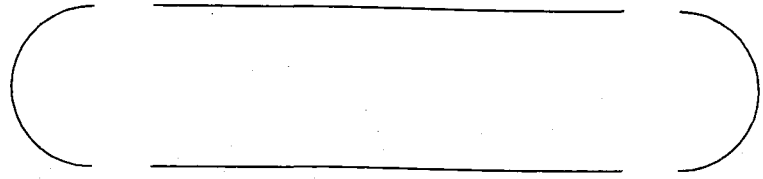


Fig 2.7 A "natural" segmentation of the shape in Fig 2.6

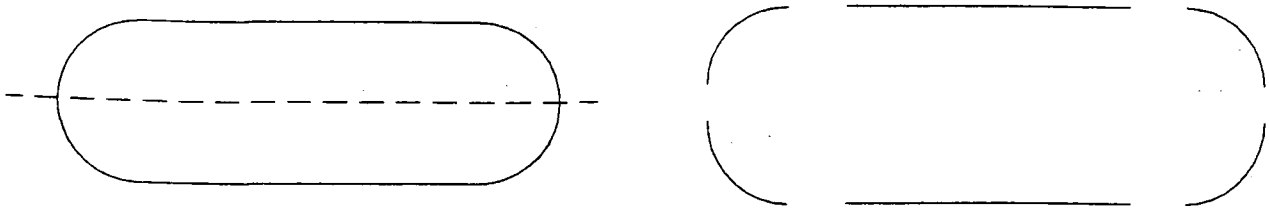


Fig 2.8 Additional edges produced by mirroring. Again, a unique representation scheme must recognize this as equivalent to the object in Fig 2.6

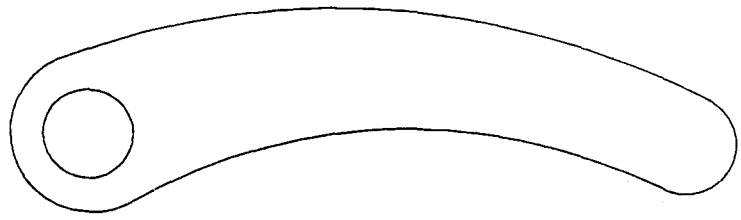


Fig 2.9 A rocker arm, whose boundary consists entirely of curved segments

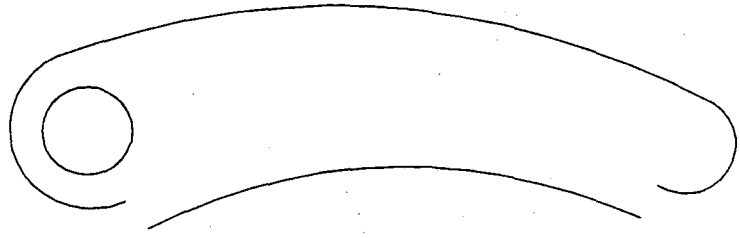


Fig. 2.10 Segmentation of the shape in Fig 2.9 at points where curvature changes sign

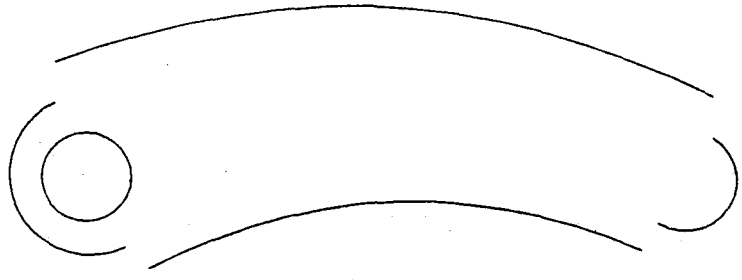


Fig 2.11 Additional segmentation of the object in Fig 2.9 at points where there is a significant step change in curvature. A unique representation scheme has to be able to define such points unequivocally

(b) It must generate a single canonical representation for all types of curve; such a representation should be invariant under translation, rotation, and scaling, and independent of the starting-point chosen.

This is essential for exact shape matching and highly desirable for similarity matching.

(c) It should promote ease of processing, in construction, manipulation, and display; in particular, it should be possible readily to derive local and global shape features from such representations.

Similarity matching, particularly of complex shapes, is likely to be a computationally expensive process. Simple forms of representation, particularly those which allow non-matching structures to be rejected quickly, are therefore at a premium.

(d) It should provide reasonably compact storage; the complexity of the representation should bear some relation to the complexity of the original object.

Not a crucial objective, except that compact representations can generally be retrieved more quickly from backing storage, and should therefore enhance system performance.

(e) It should ideally give representations that are reasonably robust; small changes to a machined object (addition or removal of a boss or keyway, for example) should produce correspondingly small changes in its representation.

Again, efficient similarity matching is possible only if the representations of objects sharing many common features are themselves very similar.

Most, if not all, representation schemes are approximations of the original object (which is in any case specified only within certain tolerances, as discussed above). The important point is that their accuracy is sufficient for the domain of objects and the set of applications in question. Most machined parts can in fact be represented by a small number of primitives, corresponding to the capabilities of commonly-used machine tools. In a survey carried out by the University of Rochester, as reported by Voelker & Requicha (1977) and Pratt (1984), it was found that over 90% of parts could be built up from just five solid primitives - rectangular blocks, cylinders, spheres, cones and tori. Over 40% could be built up from blocks and cylinders alone. As might be expected, workpieces made by forging, casting and moulding covered a much wider range of shapes.

One can thus represent the faces of most machined parts by planar or quadric surfaces. This implies that, at least when considering 2-D or 2.5-D objects, the vast majority of edges can be represented by straight lines or circular arcs. There is clearly a strong need for an edge representation scheme to provide essentially exact representations of straight lines and circular arcs; what is less certain is the need for an exact representation for edges in the form of conic arcs (which could naturally arise at the intersection of quadric surfaces) or parametric splines (normally to be found only with "sculptured" surfaces).

If a unique representation of curved edges is required, it is essential that the range of curve types permitted should be severely limited. While it is theoretically possible to compare cubic splines with elliptical arcs, or circular and parabolic arcs, to establish whether they represent the same curve within specified error limits, the task is computationally expensive. The use of a single type of representation for all curves provides a much more efficient solution to the problem. The questions to be resolved if this approach is taken centre on the domain of curves to be represented exactly; the method of approximation to be used for others; and the merits and disadvantages of representation schemes meeting these criteria. The principal options are set out below.

2.5.3 Representations based on straight-line segments

An obvious possibility is to choose the straight line as the sole curve primitive, and use polygonal approximation for all other curve types. This method has the advantage of simplicity; it is in fact one of the oldest methods of representing curves, and is still widely used in display graphics. (The ISO graphics standard GKS provides no functions for drawing circles or ellipses directly; they have to be approximated by user-specified functions drawing appropriate sequences of straight-line segments). Because of this, a wide selection of algorithms is available for generating and processing such displays. One recently-reported algorithm, for example, claims to provide an optimal linear approximation to any planar curve, in the sense of finding the minimal number of segments necessary to approximate the curve within a specified error bound (Dunham, 1986).

Straight-line approximations can be information-preserving (it is possible to specify as accurate an approximation as required by increasing the number of linear segments), and canonical representations can be devised for any shape, though few representations described in the literature are suitable as they stand. The main problem with representations based purely on straight-line segments is that they are unnecessarily verbose when used to describe engineering parts, unless very coarse approximations are used. (Perkins (1978) cites a case where the outline of a steering knuckle could be described adequately by 27 segments if both circular and straight-line segments were permitted, but needed 260 segments if all were straight lines). Such verbosity is wasteful both of storage space and processing time. It also seems inelegant to represent a conceptually simple structure like a circular arc (as we have seen, a very common feature of machined workpieces) in such a complex way.

It is worth noting that representations based on straight-line segments have been extensively used in geographic information processing, where the objects to be represented (height contours, rivers, coastlines) are often jagged and have no regular shape. The simplest way of representing such segments is to use their end-point coordinates, though this has limited usefulness because of their sensitivity to translation, rotation or scaling. One of the best-known schemes is Freeman's chain code (Freeman, 1974), which uses a grid of arbitrary fineness to quantize the contour to be represented into segments of unit length and restricted direction (Fig 2.12). Such a representation is insensitive to translation, and to some extent to scaling, though it remains sensitive to rotation (not a major problem in geographic information processing where the orientation of map features is normally known). Published algorithms are available for deriving shape features from chain-coded contours, matching segments of chain code, and computing properties such as chain length and moment of inertia.

More recent representation schemes have concentrated on providing hierarchical descriptions of contours (logical when one remembers that most objects of geographic interest, such as coastlines, are in fact fractals, and can therefore by definition be represented only to a specified level of approximation). These have included Burton's *Binary Searchable Polygonal Representation* (Burton, 1977), which constructs a binary tree to "index" a set of curve segments, based on recognizing local x and y minima and maxima, and Ballard's *Strip trees* (Ballard, 1981), overlapping rectangles which bound segments of a curve at progressively finer levels of detail. These representation schemes are of peripheral interest in the present context because of their sensitivity to transformation, rotation, scaling, and choice of starting point - and because their principal advantage, the ability to apply processes such as curve-matching at varying levels of detail, is of limited usefulness here. The only truly invariant representation scheme of this kind is that of Mokhtarian and Mackworth (1986), which derives a measure of curvature at each point on a polygonal curve, and transforms this by convolution with a one-dimensional Gaussian kernel. From this an invariant *scale space image* can be derived, which can be used for curve matching. The method is unsuitable for most machined parts

as it depends on identifying points where curvature changes sign - a concept of doubtful validity in objects with long straight edges.

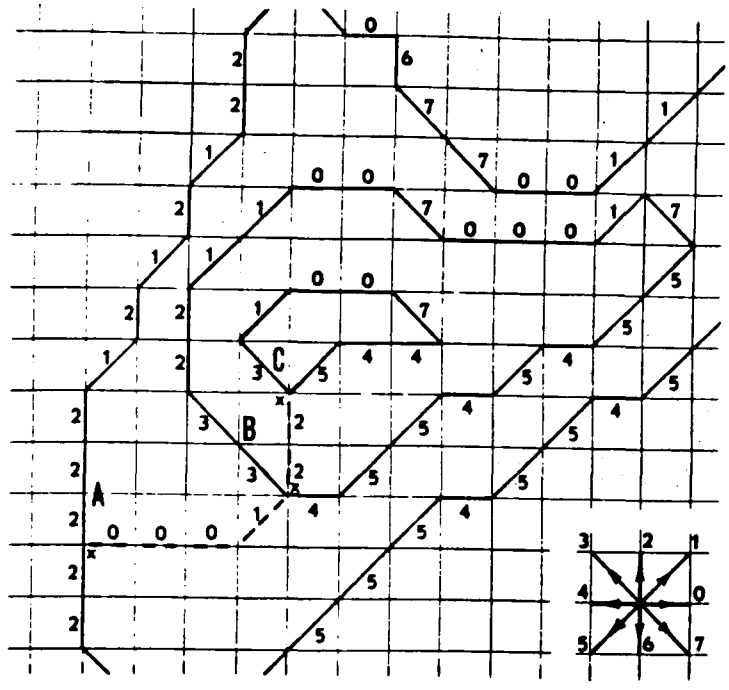


Fig 2.12 An example of the Freeman chain code (from Freeman, 1974), which quantizes all curves into unit segments which can take one of eight directions, as shown in the bottom right of the picture. Any curve can thus be approximated by a string of numbers indicating the direction of each unit segment in turn. The grid size can be chosen to give any required degree of accuracy.

2.5.4 Circular arc representations

Another option is to use circular arcs (with straight lines regarded as arcs of zero curvature) as a basis for describing edges. This has the advantage that an "exact" representation can be provided for the vast majority of edges found on simple workpieces; approximations will be needed only for more complex "sculptured" parts. Representations based on circular arcs are thus information-preserving to at least the same degree as those based on straight lines; in the majority of cases they will provide a more exact representation. It is not difficult to generate invariant representations both for simple arcs and (through approximation) segments of varying curvature. For most workpieces it is possible to generate extremely compact representations (each segment needs only three parameters, such as length, curvature, and angle made with next segment), and readily extract local and global shape features. As a method of representing simple workpieces it shows considerable promise.

An example of the use of this kind of representation is provided by Perkins (1978), whose recognition system for industrial parts analysed digitized images of workpiece outlines, identifying boundaries and fitting them to straight-line or circular segments which he named *concurves*. He was then able to match concurves derived from images with concurves for known objects.

2.5.5 Spline representations

A third option is to use spline representations for all curves. This has the advantage of permitting exact representations of all kinds of curves, though at some cost in complexity. Spline functions normally provide approximations to curves by dividing them at appropriate *knot points* into segments, which can be represented to any specified accuracy by polynomials or *basis functions* of any required degree (cubic polynomials are frequently used). Rational B-splines, curves $C(t)$ defined by the equation

$$C(t) = \frac{\sum_{1}^{n} B_{i,k}(t) h_i P_i}{\sum_{1}^{n} B_{i,k}(t) h_i}$$

where P_i are points known as *control points*, t is a parameter whose value lies between fixed limits a and b, and $B_{i,k}(t)$ are the basis functions, have a number of useful properties not shared by polynomial B-splines, including the ability to provide "exact" representations of circles and conics (Tiller, 1983). In many ways, this provides the ideal form for curve representation, as all types of curve can readily be cast into a single canonical form (for this reason, it has been adopted as the sole form of curve representation in the GEOMOD geometric modelling system from Structural Dynamics). Its disadvantages are its mathematical complexity, making it difficult to develop algorithms for feature extraction, and its lack of conciseness when compared to circular arc representations, at least for the majority of workpieces. Its high degree of generality make it a scheme worth exploring for families of parts where "sculptured" surfaces are common.

2.5.6 Boundary transformations

The final option is to use some transformation of the object boundary, taken as a whole. The best-known of these is the Fourier transformation (Zahn and Roskies, 1972). This defines the cumulative curvature around the object boundary as a function of curve length, and proceeds to expand this function as a Fourier series

$$\theta(t) = \mu_0 + \sum A_k \cos(kt - a_k)$$

in which the coefficients A_k and a_k, the kth harmonic amplitude and phase angle respectively, are known as the *Fourier descriptors* of the curve. This form of representation is information-preserving up to a point: if an infinite Fourier expansion is used, the curve can be reconstructed exactly; a truncated Fourier series will generate an approximation to the original curve. Fourier descriptors can provide an invariant description of the curve, and do appear to reflect the overall shape of the object fairly consistently - lower-order Fourier descriptors from similar objects are generally of similar magnitude, and can be used to provide an objective index of similarity between two shapes. The underlying theory has been well researched, and numerous algorithms have been developed for generating and processing Fourier descriptors. Fourier descriptors also have a number of useful properties - for example, they can be used directly to find axial or rotational symmetry in the object boundary. Storage requirements depend on the number of terms retained from the expansion; in many cases, adequate

representations can be achieved with a dozen terms or less; if so, representations can be very compact. The insuperable objection to their use in the present context is that they are properties of the boundary *taken as a whole*; there is no known way in which information about local shape features can be extracted (Pavlidis, 1980).

A similar objection rules out another recently-described method of shape description, the autoregressive model approach of Dubois and Glanz (1986). This takes the object radius at points where the boundary intersects a number of equispaced radius vectors, and forms a regression equation for the radius at the *k*th point in terms of the radius at the previous *j* points. The coefficients of this equation can again be used as invariant descriptors of the overall boundary shape - but are of no use in describing local shape features.

One feature common to nearly all applications using complex transformations appears to be the need to recognize shape features in object representations generated from inherently noisy digitized images. One of the principal reasons for using such transformations is clearly the need to distinguish between genuine shape features and noise, and the need to take account of the effects of digitization at different scales. The present study takes drawings generated on a CAD system as its starting point; such drawings should be (relatively) noise-free, and digitization errors should not affect measures of curvature to any marked extent. It is not therefore expected that complex transformation techniques will prove particularly useful. The only transformation which has been used, as it provides invariance to translation, rotation and scaling, is to represent all boundary lines in intrinsic coordinates, by plotting θ, the cumulative curvature, against s, the cumulative arc length. An advantage of this transformation is that circular arcs become straight lines, making the task of processing representations based on circular arcs much easier.

2.6 Representation schemes for 3-D objects

As discussed above, one of the aims of the present study is to lay the groundwork for the development of databases of 3-D objects. A brief examination of representation schemes for 3-D objects is thus in order. Numerous schemes have been proposed, including *primitive instancing*, where objects are classified as "cube", "prism", "cylinder", etc. and further specified by appropriate parameters; *spatial occupancy*, the 3-D equivalent of raster format, in which space is divided into unit cubes (sometimes known as *voxels*), each of which carries an indication of whether it is "inside" or "outside" a given object; and *cell decomposition*, where objects are divided up into tetrahedra by a solid triangulation process. Such schemes all have severe drawbacks (Requicha, 1980), and in practice only two methods have been used to any significant extent. These are the *boundary* and *constructive solid geometry* (CSG) representations referred to in Section 1.2.

CSG is on the face of it a promising form of representation for 3-D solids. The CSG trees representing how solid objects were built up from their constituent primitives provide unambiguous representations of the original object, and can be very concise. They are not unique, in that (a) even starting with a single set of primitives, an object could be built up in many different ways by varying the order in which operators were applied; (b) two CSG schemes based on different sets of primitives would inevitably generate a given object in different ways.

The first problem can in fact be successfully overcome in the majority of cases. Woo (1982) has shown that any CSG representation of a rigid solid can be transformed into a canonical form termed ASV (Alternating Sum of Volumes), by forming its *convex hull* (the smallest convex solid into which the object can be completely fitted), subtracting it from that convex hull to generate a new solid (representing the difference between the original solid and its convex hull), and repeating this pair of operations on the resultant solid until this difference becomes zero. The only limitation is that the process does not

terminate in all cases. For such solids, Woo's method cannot generate a unique representation.

The second problem above could be overcome in two ways. One way would be to adopt an agreed set of standard primitives out of which all systems could build geometric models. Such a set would need to be limited in such a way that a geometric model could be built only from a single set of primitives (so that, for example, cubes and cuboids would not be permissible as separate primitives because one designer might use cubes to build up a given part, while another used cuboids). Such standardization flies in the face of the current trend to provide designers with as wide a range of primitives as possible in order to make the design process easier, and cannot therefore be considered a practical proposition. Another way would be to adopt a small set of standard primitives into which all modelling systems could transform their models, possibly as the basis for an exchange format. Although such a possibility has been mentioned in connection with the development of standards for data exchange between solid modelling systems, no generally accepted set of primitives exists at present. The existing IGES standard is unable to handle CSG specifications, and extensions to the standard to enable it to do so are still the subject of debate. For these reasons, CSG seems an inappropriate representation method at present. Its many advantages suggest that it may prove a fruitful basis for future studies.

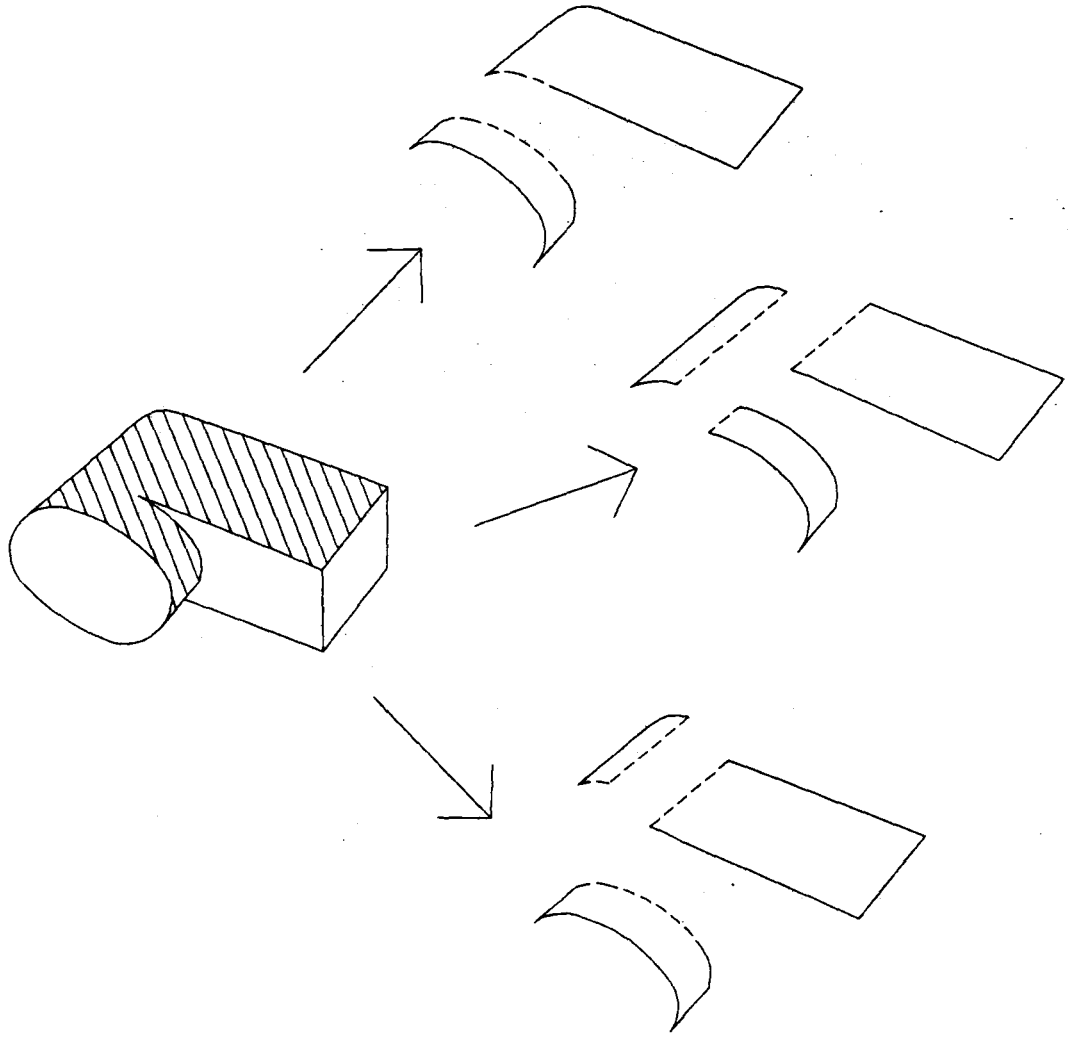


Fig 2.13 An example of the difficulty in defining a face. The shaded area could logically be partitioned in any of the alternative ways shown.

Boundary representation of solid objects is almost certainly the most common form encountered in practice, if only because of its use in computer graphics as well as geometric modelling. Solid objects are represented as a finite number of bounding faces which between them enclose the entire solid. In turn, each face is represented by its bounding loop of edges and vertices. To represent a solid unambiguously, both its topology and the geometry of each constituent face must be represented unambiguously - though, as shown by the investigations carried out by Weiler (1985), a number of alternative representations are topologically sufficient. Planar faces may be represented by their bounding edges, but curved faces require additional information (such as their surface equation) before they can be unambiguously specified.

To construct a boundary representation scheme which is also unique raises additional problems, as objects' bounding faces need to be uniquely defined, and their topological relationships uniquely represented. Intuitively, it is not always obvious what should constitute a face, particularly where planar and curved surfaces meet. The object in Fig 2.13, for example, contains a large smooth area which could be regarded as one, two, three or even more faces. The design of a unique boundary representation scheme therefore involves a number of far from trivial decisions.

2.7 Conclusions

Several possible methods of representing line segments have been outlined above. When judged against the criteria set out in paragraph 2.4.2, their merits and disadvantages can be summed up as follows:

(1) IGES parameters could in theory be used directly to specify line segments. They would certainly be information-preserving, but would fail on virtually every other count. There is no simple way in which they could be made invariant to scaling, translation or rotation; they are far from compact; and similar lines could have totally different IGES representations.

(2) Straight-line approximations can provide an invariant representation provided lines are expressed as normalized length and direction. But they cannot simultaneously provide information preservation and compact storage for curved segments (use of short straight-line approximations to prevent information loss inevitably increases the number of segments).

(3) Circular arcs can provide compact information-preserving representations for machined shapes, the vast majority of which are made up of straight lines and circular arcs. If expressed in intrinsic coordinates, they are invariant to translation, rotation and scaling, and relatively easy to process for feature extraction or similarity matching.

(4) Rational B-splines can provide information-preserving representations for all shapes, and can be cast into invariant form. Their large number of parameters makes compact storage difficult, and processing algorithms are necessarily complex.

(5) Fourier and similar transformations fail to preserve local shape information, even though they can represent an object's global shape accurately (in invariant form) provided enough terms are included. In this respect, they resemble the straight-line representations in that they force a "trade-off" between accuracy and compactness. Processing of such representations can be complex.

From the above discussion, it seems clear that the only two representations worth serious consideration are circular arcs and rational B-splines. Since the vast majority of

engineering parts are in fact machined shapes, circular arc representations seem preferable, as they provide much more compact storage and ease of processing at the expense of a very slight loss of generality. An exploration of the use of spline representations, particularly for sculptured parts, could form a useful extension of the present research.

CHAPTER 3. SHAPE REPRESENTATION FOR RETRIEVAL

3.1 Introduction

As discussed in section 2.3, the need for efficient shape matching and retrieval implies that all drawing features likely to be used in retrieval should be represented in as invariant a form as possible. While detailed discussion of the question of feature extraction for retrieval is deferred until chapter 4, it must be noted here that if there is to be any possibility of retrieving shapes by matching sequences of boundary segments themselves, some attempt has to be made to represent these in a unique standard form. Objects with the same shape should have identical representations, however they were originally drawn. Given the domain of objects selected (2-D objects defined by boundaries represented as straight lines or circular arcs), this implies that representations of the object's outer boundary, all inner boundaries, and the relative positions of all inner and outer boundaries must all be in a form invariant to translation, rotation and scaling. Furthermore, there is a need to define a standard ordering for each shape element if one wishes to limit the complexity of the matching process. This requires that rules must be defined both to select a start segment and traversal direction for each shape boundary, both outer and inner, and also to order all inner shape boundaries.

3.2 Choice of shape representation

3.2.1 Outer boundary representation

As discussed in section 2.7 above, the most appropriate representation for shape boundaries was considered to be a set of line segments, each defined by its length and curvature. If these segments make up an ordered list in such a way that following the list corresponds to traversing the boundary in a given direction until the starting-point is reached, a very compact representation of the boundary can result, as angular changes in direction can be included as parameters of the preceding line segment (Fig 3.1). This form of boundary representation can easily be rendered unique provided length and curvature parameters are normalized, a standard direction of boundary traversal (clockwise or anticlockwise) is chosen, and a procedure exists for choosing a unique starting-point for boundary traversal.

3.2.2 Initial representation chosen

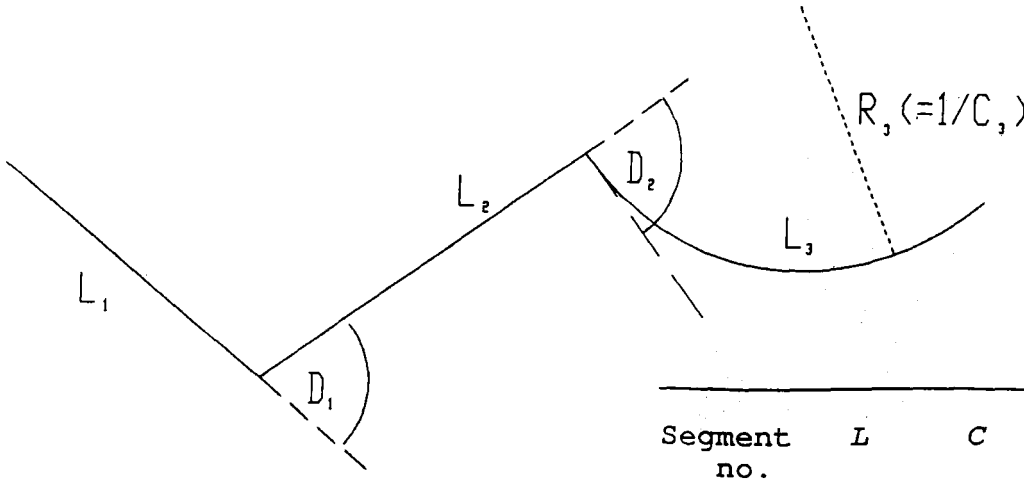
Initially, each boundary was to be represented by a list of 3- tuples

$$\{ L , C , D \}$$

each representing a single line segment, where L is the segment length, C its curvature (reciprocal of radius for circular arc segments, zero for straight-line segments), and D the discontinuity angle between the current segment and the next (Fig 3.1). The ordering chosen for outer boundary segments corresponds to anticlockwise boundary traversal, consistent with programming conventions for numerically-controlled machine tools. Segment representations could readily be transformed into a form invariant to scaling, rotation and translation by normalization using a standard length L_0, yielding the form:

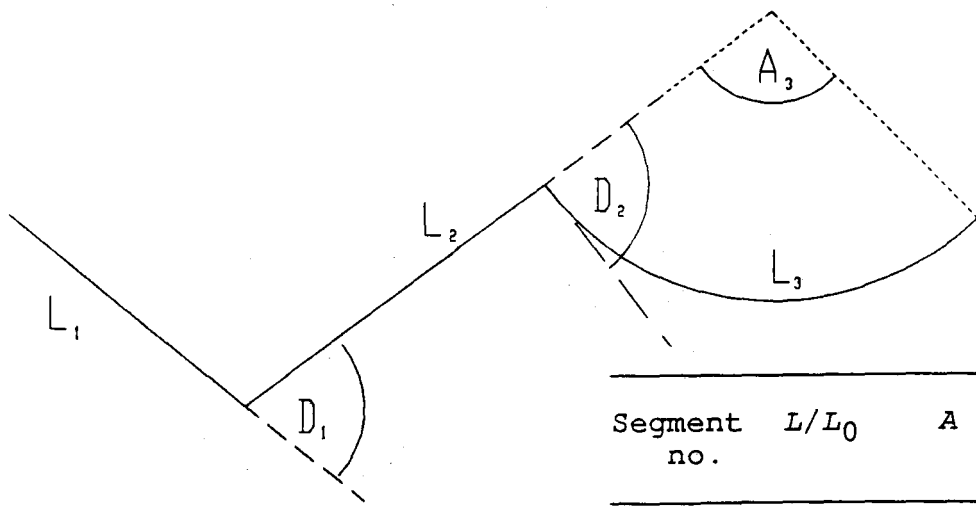
$$\{ L/L_0 , C*L , D \}$$

L_0 can conveniently be taken to be the boundary perimeter, so that each segment is represented by three parameters: its length (as a fraction of the entire boundary), its arcangle A, equal to the product of its curvature C and its length L, and D, the discontinuity angle with the next segment (Fig 3.2).



Segment no.	L	C	D
1	4.53	0	1.12
2	4.81	0	-1.38
3	3.90	0.27	...

Fig 3.1 Three line segments, each represented by its length L, *curvature* C *(zero for straight lines, positive for anticlockwise arcs, negative for clockwise arcs), and discontinuity angle* D *(again, positive for anticlockwise changes in direction, negative for clockwise).*



Segment no.	L/L_0	A	D
1	0.15	0	1.12
2	0.16	0	-1.38
3	0.13	1.61	...

Fig 3.2 The same three line segments, represented by normalized length L/L_0, *arc angle* A (= C/L) *and discontinuity angle* D.

Although the line segments conceptually form a closed ring, they have to be stored and processed as a linear sequence. To generate a canonical representation of the boundary, a unique start segment must therefore be defined, otherwise as many representations are possible as there are segments (Fig 3.3). To define a unique start point purely in terms of boundary shape, it is necessary to select a start line on the basis of invariant parameters such as those set out above. The most straightforward way to do this is to select as start line (a) the line segment with greatest length; if more than one line fulfils this criterion, select (b) the candidate with the greatest curvature; if this does not resolve the matter, select (c) the candidate with the largest angle of discontinuity between it and the next segment. If more than one candidate still remains, apply criteria (a) - (c) in turn to the line segment following each candidate, and if necessary to subsequent line segments, until either (i) a unique start line emerges, or (ii) the entire boundary has been traversed. In the latter case, the boundary is completely symmetrical, and a start line can be selected from the remaining candidates at random. See Fig 3.4 for examples of this process.

It is of course perfectly possible to choose any combination of extreme values for the parameters defining line segments in order to specify canonical form - for example, one might have chosen as start point the line with smallest curvature, smallest discontinuity angle, and smallest length, in that order. There appears to be no clear-cut reason for preferring any one combination of parameters to any other. The choice of starting- point has in any case no structural significance.

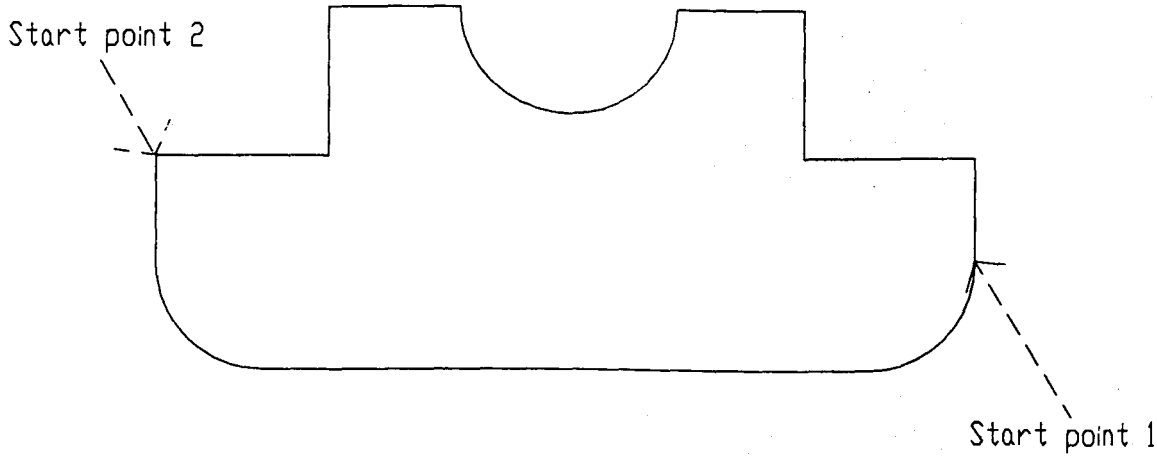
3.2.3 Final representation chosen

The above representation meets the first four criteria presented in section 2.5.2 above, but fails on the fifth. The method of boundary start line selection described in the previous section always yields the same result with identical shapes. However, small changes such as the addition of a notch in one side of an object can lead to a totally different start line being chosen (Fig 3.5). Hence quite large degrees of similarity between shapes could go unrecognized because their representations would be markedly different.

While this problem can never be completely overcome (see section 3.3.5 below), it can be alleviated in a large number of cases if object representations are preprocessed to remove local features, revealing their underlying shape (Fig 3.6). As observed in section 3.2.4, this approach has a number of additional useful aspects. The method involved is similar to that of Kyprianou (1980), though here the process is easier because it involves only two dimensions, and only two basic classes of local shape feature, midline features, sequences of short line segments interrupting an otherwise continuous long edge (Fig 3.7), and corner features, line sequences contained as it were in the jaws of a corner (Fig 3.8). The process can be regarded as the repeated application of the shape rewriting rules illustrated graphically in Figs 3.9 and 3.10. Definitions of line sequences making up midline and corner features are not lost, but absorbed into the definitions of the "skeleton" lines enclosing them (see section 3.3.3 below for a detailed discussion of the process of generating such descriptions).

Such shape feature definitions are essentially recursive, since a line enclosing a shape feature at one level can itself form part of a shape feature at a higher level. Repeated application of the rewriting rules to a complex shape will thus lead to a progressive simplification as local features are removed one by one, eventually leaving an irreducible "skeleton" shape (Fig 3.6). As discussed below, it is possible to regulate this process to allow a shape to be described hierarchically, each level of description including an extra level of local features (Fig 3.11). The boundary can be traversed at any of these levels; its representation then becomes a sequence of 4-tuples:

$$\{ L/L_0 , A , D , E \}$$



Segment	L/L_0	A	D
1	0.045	0	1.57
2	0.072	0	-1.57
3	0.063	0	1.57
4	0.054	0	1.57
5	0.141	-3.14	1.57
6	0.054	0	1.57
7	0.063	0	-1.57
8	0.072	0	1.57
9	0.045	0	0
10	0.070	1.57	0
11	0.251	0	0
12	0.070	1.57	0

Segment	L/L_0	A	D
1	0.045	0	0
2	0.070	1.57	0
3	0.251	0	0
4	0.070	1.57	0
5	0.045	0	.57
6	0.072	0	-1.57
7	0.063	0	1.57
8	0.054	0	1.57
9	0.141	-3.14	1.57
10	0.054	0	1.57
11	0.063	0	-1.57
12	0.072	0	1.57

Fig 3.3 Two alternative representations of a simple shape, using different starting points (boundary traversal anticlockwise in both cases).

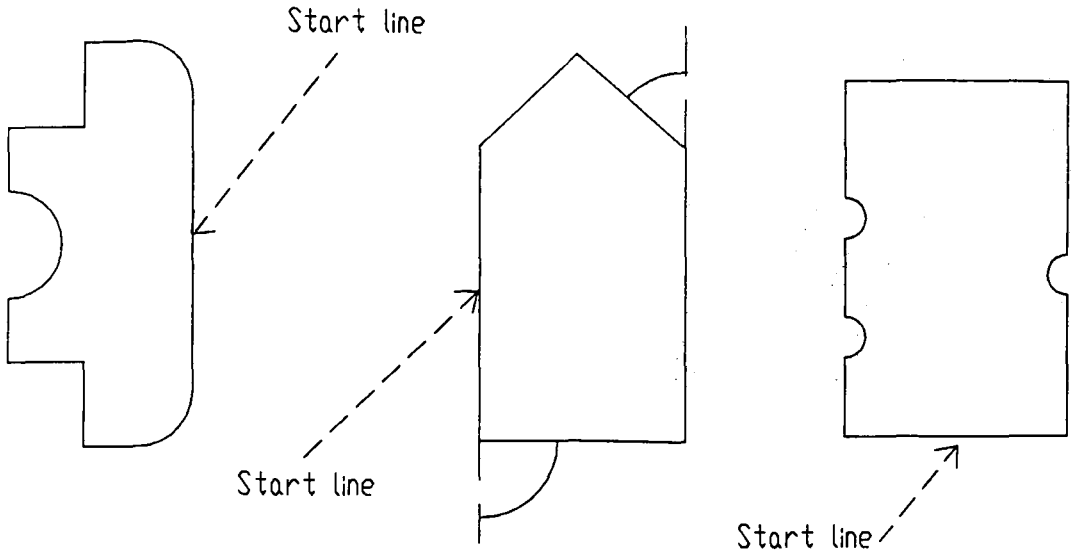


Fig 3.4 Three examples of start segment selection, showing selection of (a) the longest segment, (b) the segment with the greater discontinuity angle, and (c) the segment with the longer following segment.

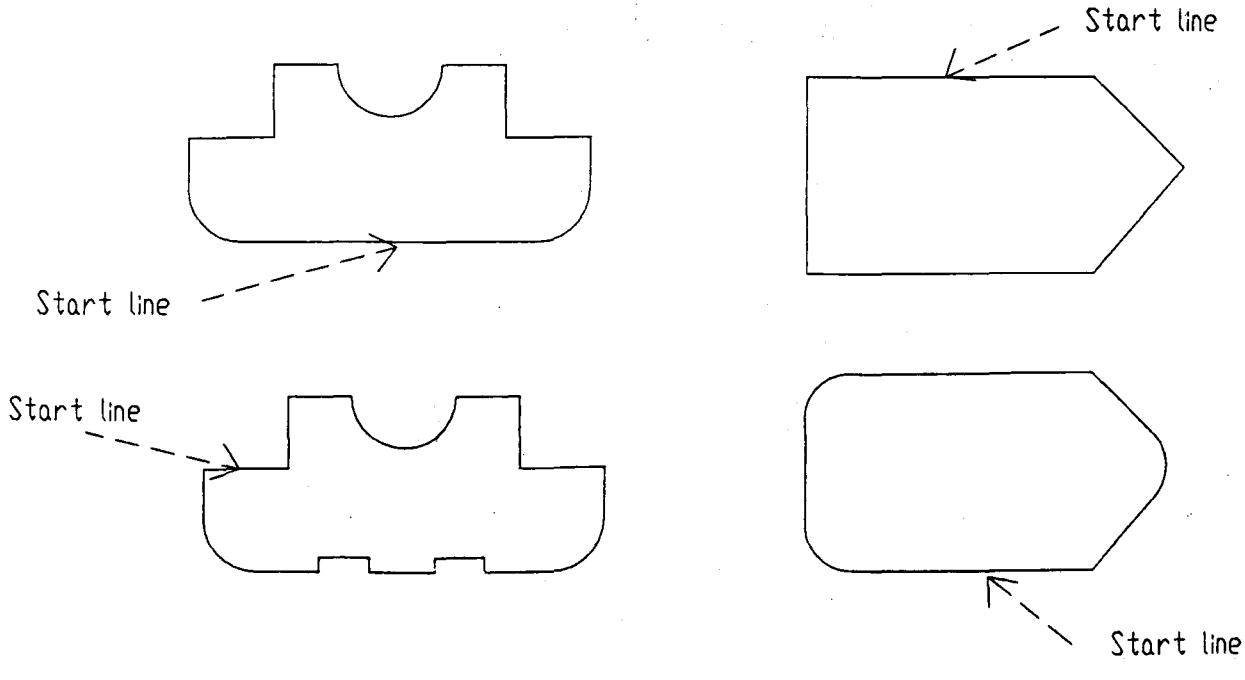


Fig 3.5 Lack of robustness of start segment selection procedure. Small changes in structure can alter start point, giving completely different representation.

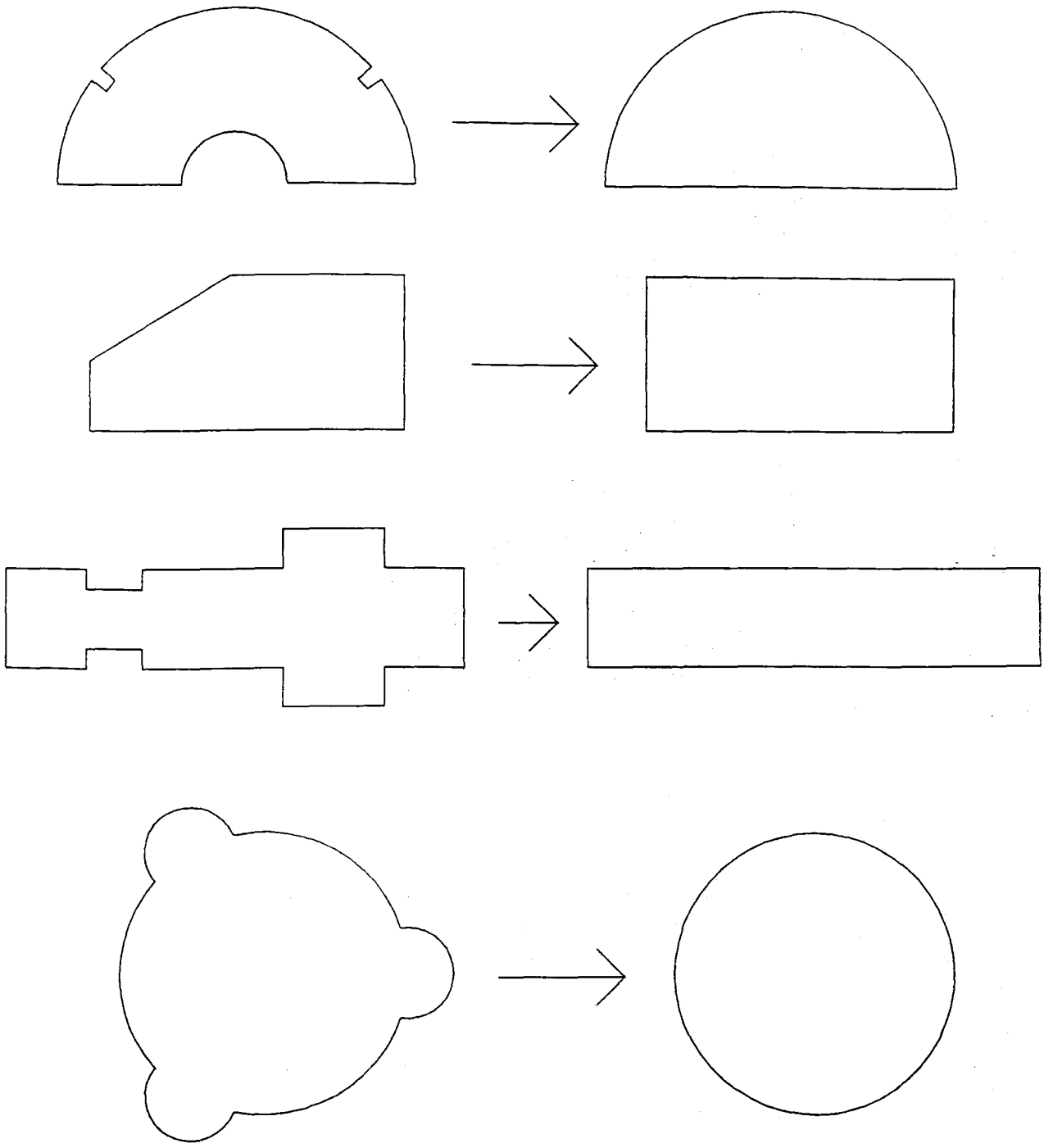


Fig 3.6 Underlying object shapes uncovered by removing local shape features.

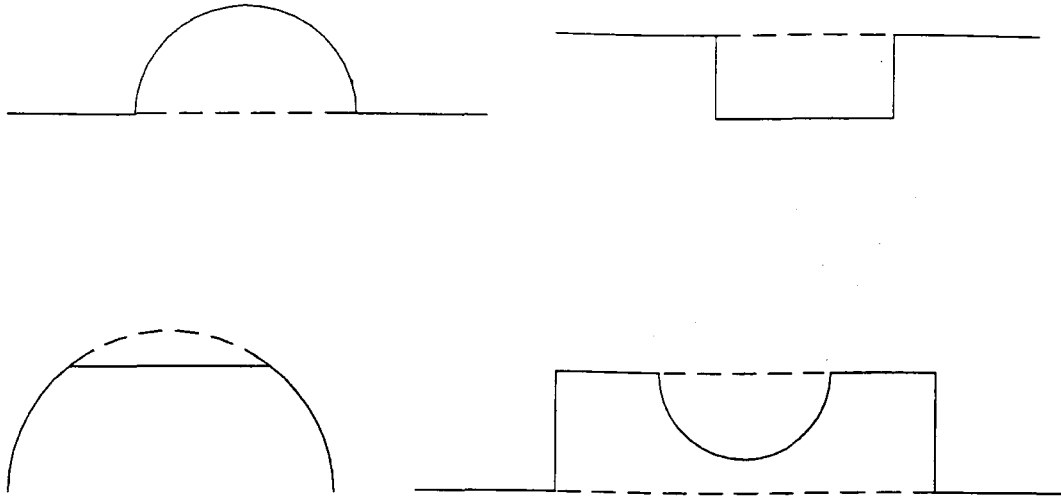


Fig 3.7 Examples of midline shape features.

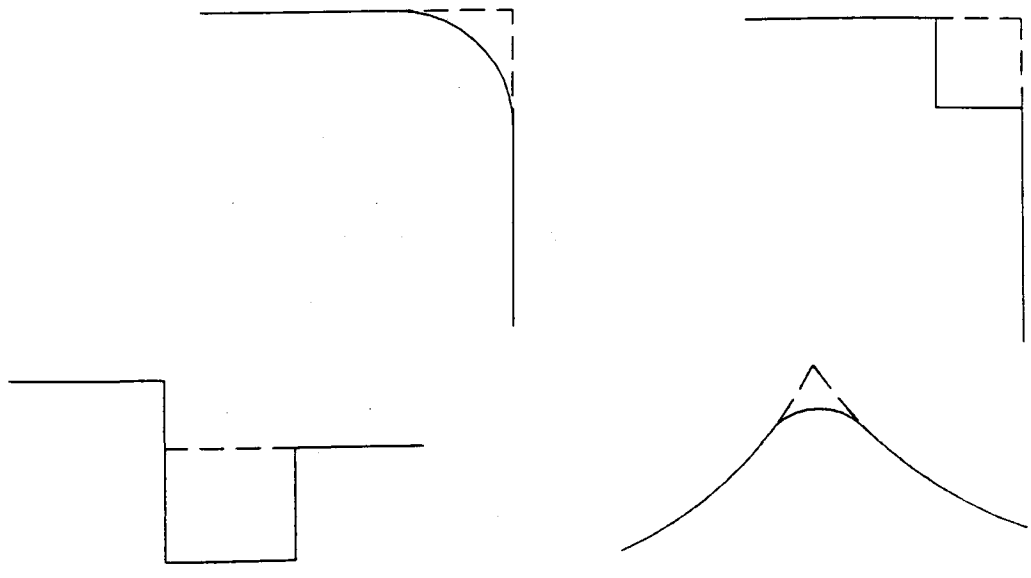


Fig 3.8 Examples of corner shape features.

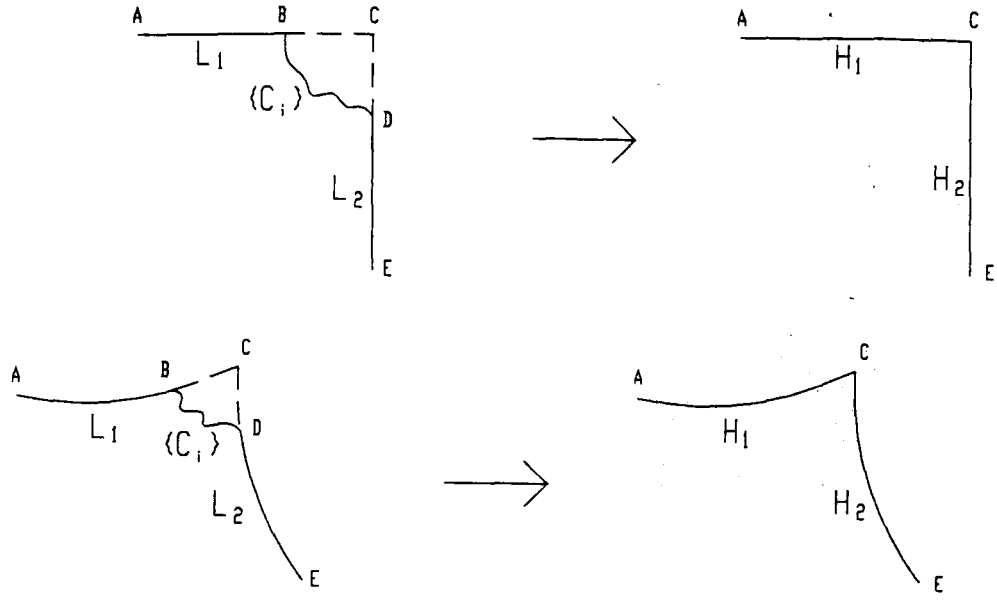


Fig 3.9 Corner feature rewriting rules. Assuming boundary traversal to be in the direction L_1, C_1, ... , L_2, new lines H_1 and H_2 inherit properties as follows: H_1 - length: distance AC measured along H_1; curvature: curvature of L_1; discontinuity angle: angle between tangents to H_1 and H_2 at C. H_2 - length: distance CE measured along H_2; curvature: curvature of L_2; discontinuity angle: discontinuity angle of L_2.

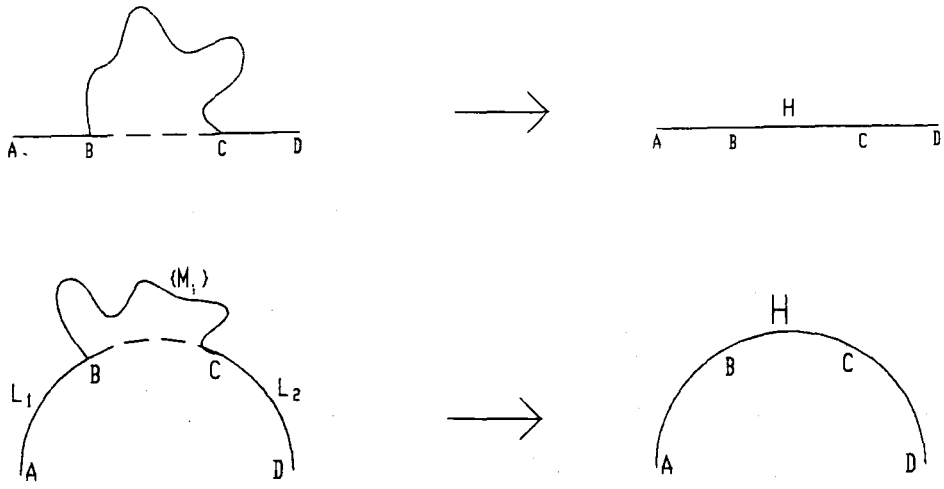
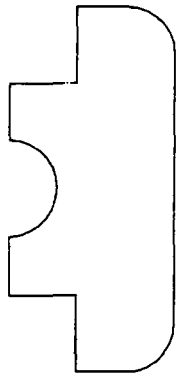


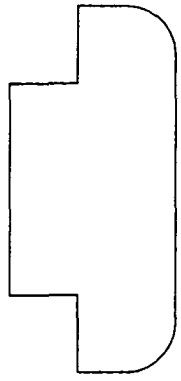
Fig 3.10 Midline feature rewriting rules. Again assuming boundary traversal to be in the direction L_1, M_1, ... , L_2, new line H inherits properties as follows - length: distance AD measured along H; curvature: curvature of L_1 (which must be equal to the curvature of L_2 within a very fine tolerance); discontinuity angle: discontinuity angle of L_2.

Lowest level:



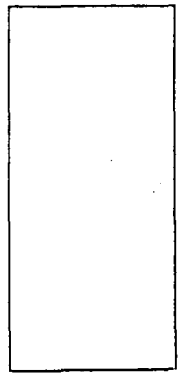
Segment	L/L_0	A	D	E
1	0.292	0	0	2
2	0.082	1.57	0	2
3	0.052	0	1.57	2
4	0.083	0	-1.57	2
5	0.073	0	1.57	2
6	0.063	0	1.57	3
7	0.164	-3.14	1.57	3
8	0.063	0	1.57	3
9	0.073	0	-1.57	2
10	0.083	0	1.57	2
11	0.052	0	0	2
12	0.082	1.57	0	2

Intermediate level:



Segment	L/L_0	A	D	E
1	0.292	0	0	2
2	0.082	1.57	0	2
3	0.052	0	1.57	2
4	0.083	0	-1.57	2
5	0.073	0	1.57	2
6	0.229	0	1.57	2
7	0.073	0	-1.57	2
8	0.083	0	1.57	2
9	0.052	0	0	2
10	0.082	1.57	0	2

Top level:



Segment	L/L_0	A	D	E
1	0.396	1.57	0	1
2	0.104	1.57	0	1
3	0.396	1.57	0	1
4	0.104	1.57	0	1

Fig 3.11 Representation of the shape from Fig. 3.3 at three levels of detail

where E represents the level at which boundary traversal takes place. L_0 here could be the boundary length at the relevant level, though it is normally simpler to take it as the top-level boundary length in each case.

In practice, this form of representation is wasteful of space. Since many line segments remain unchanged by the rewriting process and therefore form part of the structure at more than one level. The information about such lines would need to be duplicated at each level if the above form of representation were used direct. The modified form of representation shown in Fig 3.12 was therefore used, with each segment represented as a 5-tuple:

$$\{ L/L_0 , A , D , E_t , E_b \}$$

where E_t represents the level of the current line counting downwards from the top, and E_b represents its level counting upwards from the bottom. Again, L_0 represents the top-level boundary length. It is then easy to construct an algorithm to traverse the structure at any desired level V, for example by successively taking all line segments for which $E_t = V$, or $E_t < V$ and $E_b = 1$.

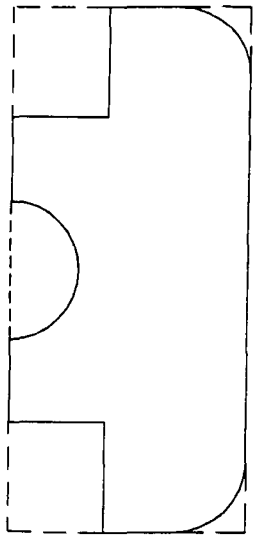
Such a hierarchical representation can be cast into canonical form by choosing a unique start line from the top-level line segments, using the procedure described in the previous section, with one addition. If traversal of the boundary at top level fails to identify a unique start line, the boundary is traversed at successively lower levels (Fig 3.13), until either a unique start line emerges, or the entire boundary has been traversed at the lowest level. As before, this indicates that the entire shape is symmetrical, and a start line can be chosen at random from the remaining candidate lines - unless there is an unsymmetrical configuration of inner boundaries. One further modification can be made to improve robustness - the use of a measure involving chord length rather than arc length when comparing candidate start lines. This makes no difference to the vast majority of shapes, but yields a more consistent start line in the cases shown in Fig 3.14.

Given two identical shapes of different size and orientation, it can be shown that this modified procedure (like the original) always yields the same start point. While no proof can be offered that such a hierarchical representation is always more robust to small changes of shape than the 3-tuple representation described above, the examples shown in Fig 3.15 give a clear indication of the potential advantages of this representation, which clearly meets criterion 5 better than its predecessor. However, the examples illustrated in Fig 3.16 show clearly the limitations of the method, and why it is inherently unable to generate robust representations in all circumstances.

3.2.4 The concept of the boundary level

A further important advantage of the type of hierarchical representation described above is the additional flexibility of shape matching it can provide, by allowing shape boundaries to be traversed at different levels. Each level of boundary traversal (from now on referred to as a *boundary level*) can be treated as a legitimate view of its parent boundary, and used to represent that boundary anywhere in the shape matching process. One might expect the ability to match query and stored drawing boundaries at *any* level to be a powerful aid to retrieval performance, as it allows inherent similarities to be recognized even where one object has a much more complex outline than another (as in Fig 3.17). The boundary level is thus regarded within the system as an important entity in its own right.

It can further be hypothesized that feature extraction should be based on separate processing of each boundary level, rather than the boundary as a whole, yielding a series of feature sets, each characterizing the boundary at a different level of detail. Identifying



Segment	L/L_0	A	D	E_b	E_t
1	0.396	0	1.57	2	1
2	0.292	0	0	1	2
3	0.082	1.57	0	1	2
4	0.104	0	1.57	2	1
5	0.052	0	1.57	1	2
6	0.396	0	1.57	2	1
7	0.083	0	-1.57	1	2
8	0.073	0	1.57	1	2
9	0.229	0	1.57	2	2
10	0.063	0	1.57	1	3
11	0.164	-3.14	-1.57	1	3
12	0.063	0	1.57	1	3
13	0.083	0	-1.57	1	2
14	0.396	0	1.57	2	1
15	0.104	0	1.57	2	1
16	0.052	0	0	1	2
17	0.082	1.57	0	1	2

Fig 3.12 Representation of all drawing levels in single table.

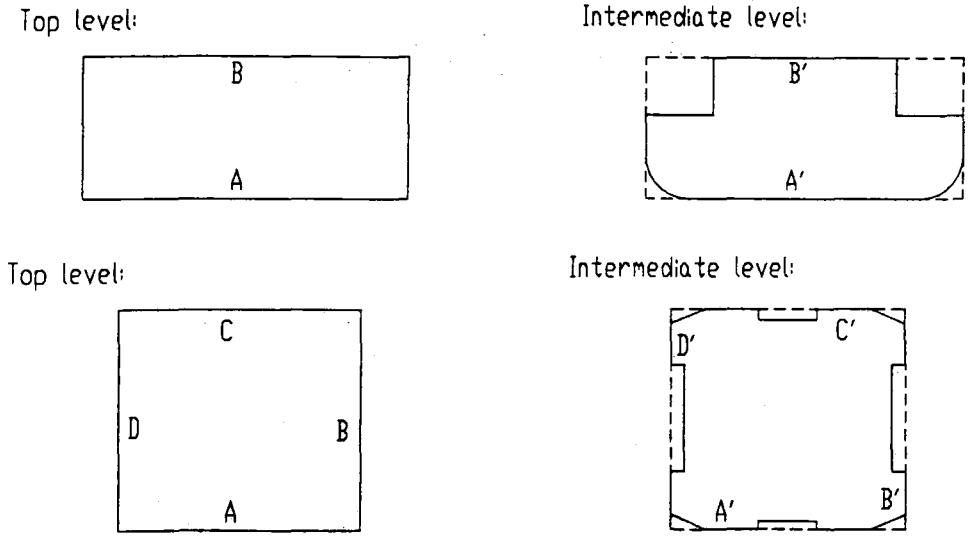


Fig 3.13 Two examples where boundary traversal is needed at more than one level to resolve otherwise identical starting points. In the top example, line A is favoured as starting-point over B because its lower-level analogue A' is longer than B'. In the bottom example, A and C remain as possible starting points after second-level traversal since A' and C' are longer than B' or D'. Further traversal at lower levels, and possibly involvement of inner boundaries, would be necessary to resolve between A and C.

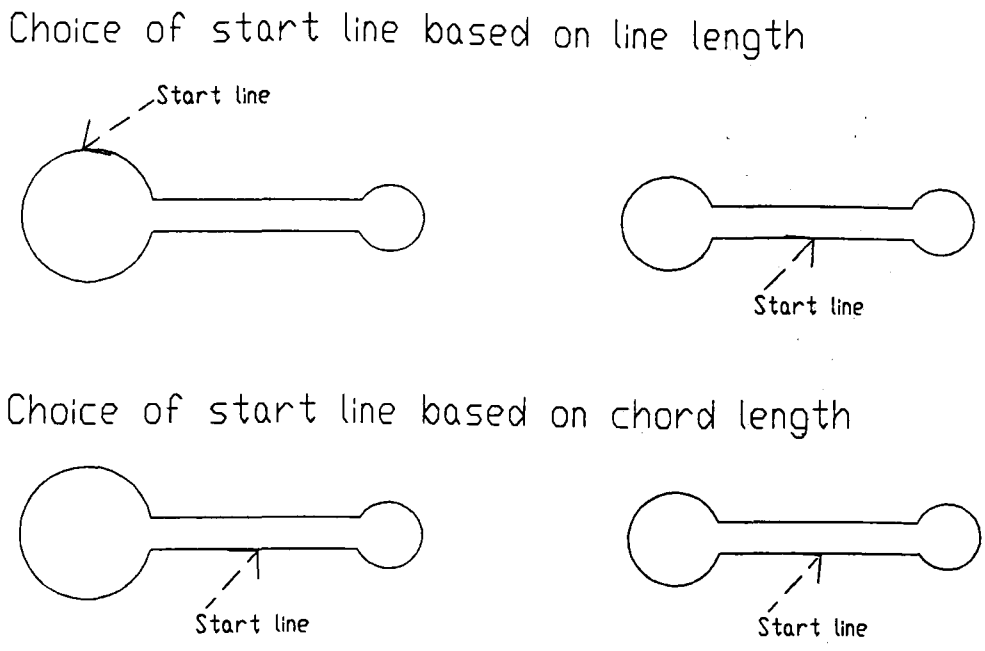


Fig 3.14 An example where use of chord length (the straight-line distance between start and end points) rather than segment length as prime discriminator between possible start lines makes choice of start line and start direction less sensitive to small changes in object shape.

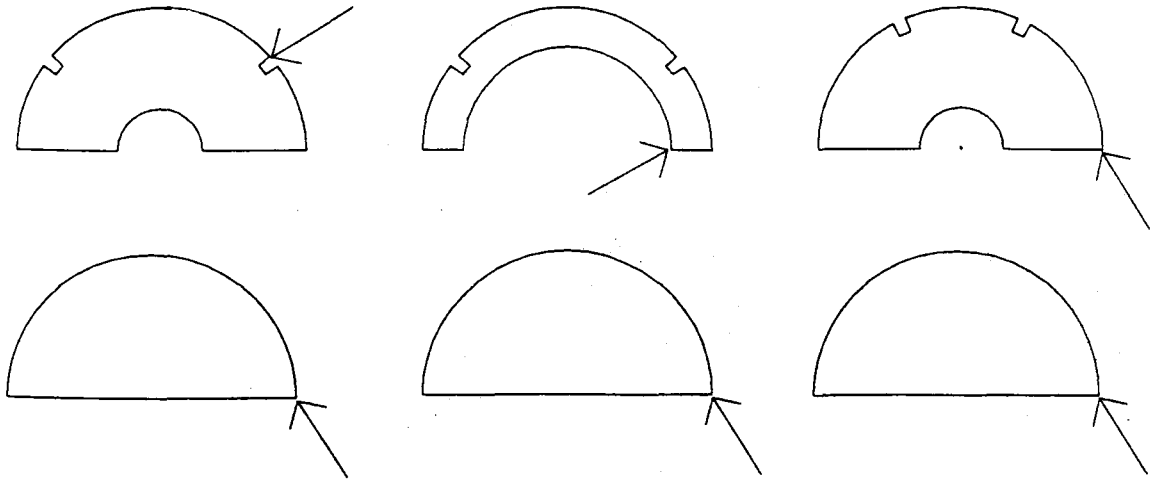


Fig 3.15 Examples of improved robustness in choice of canonical start line by use of multi-level shape description. The three low-level descriptions above have their longest lines in different relative positions. Using the higher-level descriptions below, the major semicircle becomes the start line in all cases, forcing the same choice of start point on all three shapes.

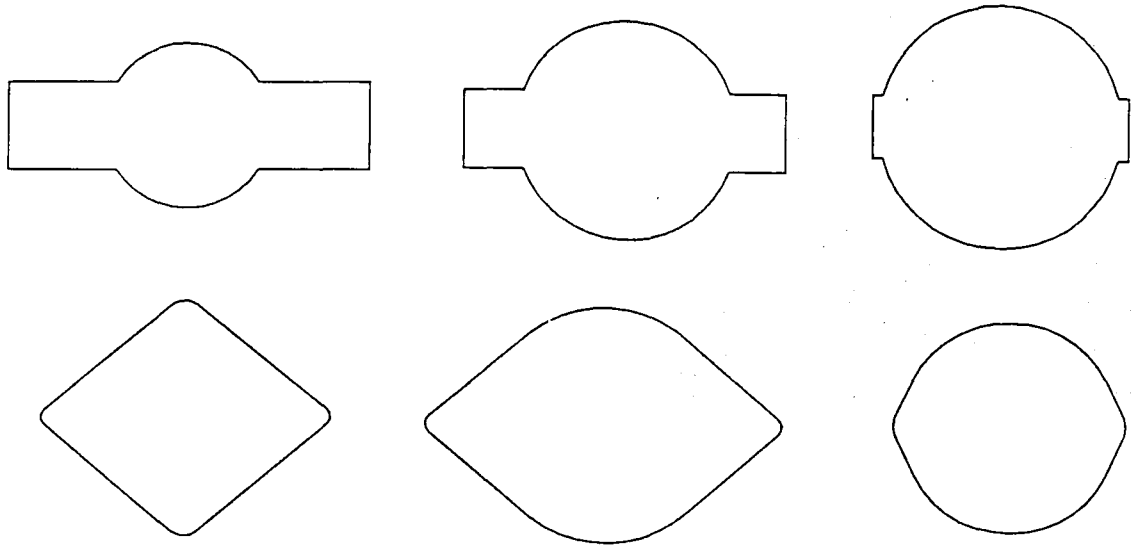


Fig 3.16 Examples of shapes where completely robust unique representations cannot be generated. While most observers would agree that the left-hand shape in both series is basically straight-edged, the position gradually changes as the relative length of the curved segments is increased, until the essentially circular right-hand shape is reached. At some point, therefore, a small change in segment length will inevitably result in a major change in shape representation.

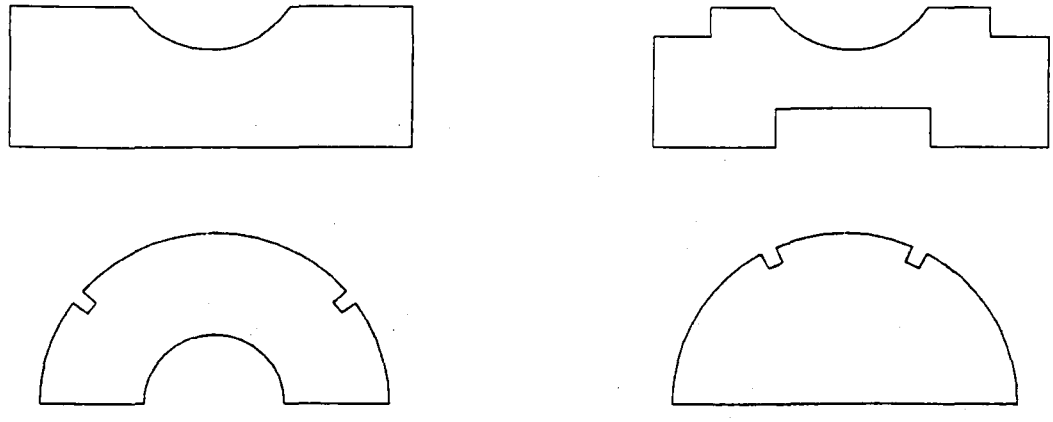


Fig 3.17 Examples of pairs of shapes where clear elements of similarity are present, but where direct comparisons of segment length, curvature or discontinuity angle would be unlikely to discover those similarities. The first pair of shapes, for example, shares only one common segment, the second pair none at all.

all shapes with specific features in common (such as the ($+90^o$ -90^o -90^o $+90^o$) discontinuity angle sequence characterizing rectangular depressions) is relatively easy with shapes such as that shown in Fig 3.17 if feature sets for intermediate levels are extracted and stored separately. Faced only with discontinuity angle sequences from the most detailed level of shape description, it would be much more difficult to recognize this type of feature. Hence the majority of extracted retrieval features (see chapter 4) in the prototype version of SAFARI are in fact associated with a specific boundary level, rather than the boundary in general.

3.2.5 Inner boundaries

Since inner boundaries are defined in an identical fashion to outer boundaries, an identical method was used for representing them, with two exceptions. Firstly, the segments making up an inner boundary were traversed in clockwise order; secondly, the length L_0 used to normalize segment length was chosen to be the outer boundary perimeter rather than the inner boundary's own perimeter. These decisions were taken to differentiate inner and outer boundaries. The first is consistent with the principle of NC machine tool operation (metal on the left of the cutting tool); the second permits the database to represent the relative size of inner and outer boundaries - essential for unambiguous representation of complete objects with inner boundaries.

As discussed later, the choice of start point in otherwise symmetrical inner boundaries is determined by the relative positioning of inner and outer boundaries.

3.2.6 Relative positions of inner and outer boundaries

To complete an unambiguous description of 2-D objects, it is necessary to specify in an invariant form:

(a) the start point and start direction of each inner boundary;

(b) the ordering of internal boundaries.

Start point and start direction. In view of the need to identify symmetry at a later stage of the shape-matching process, it was in fact decided first to calculate and store the position of each boundary centroid. The position of each inner boundary centroid is then defined in terms of the vector joining it to the outer boundary centroid; in turn, the start point of each inner boundary (the start point of the start line selected by the procedure described above) is defined by the vector joining it to its centroid (Fig 3.18). The lengths of these vectors are rendered invariant to translation, rotation and scaling by dividing them by the outer boundary perimeter L_0; their angles by relating them to a standard direction characteristic of the shape itself, such as the shape's major axis, or the start direction of the outer boundary start line. Similarly, inner boundary start directions are rendered invariant by relating them to the outer boundary start direction.

Ordering of inner boundaries. The final step in generating a standard ordering of all shape elements is clearly the specification of a standard inner-boundary ordering - though it should be noted that the problems of defining a robust ordering in all cases are as intractable here as in the case of outer boundary segments (section 3.2.3). This implies that an ordering of inner boundaries must be defined purely in terms of intrinsic shape parameters. It is valid to order inner boundaries purely in terms of parameters such as number of segments, relative distance from outer boundary centroid, relative length of perimeter and so on. Such an ordering is not however very useful because of the high degree of symmetry in many engineering shapes. The most appropriate ordering of inner

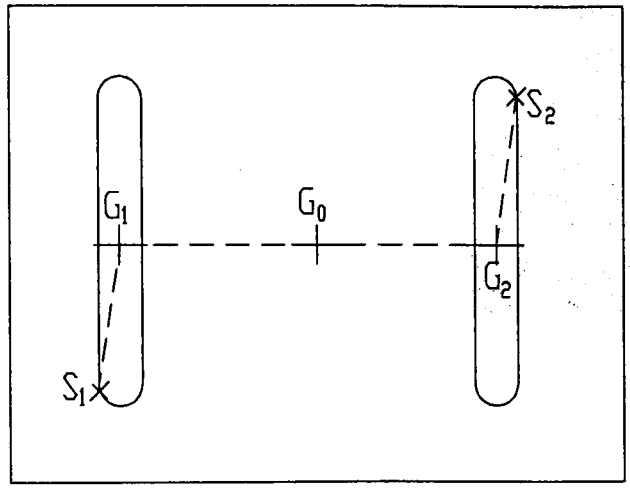


Fig 3.18 Invariant representation of inner boundaries in terms of centroid vectors $\boldsymbol{G_0G_1}$ *and* $\boldsymbol{G_0G_2}$*, linking inner and outer boundary centroids, and start point vectors* $\boldsymbol{G_1S_1}$ *and* $\boldsymbol{G_2S_2}$*, linking inner boundary centroids and start points.*

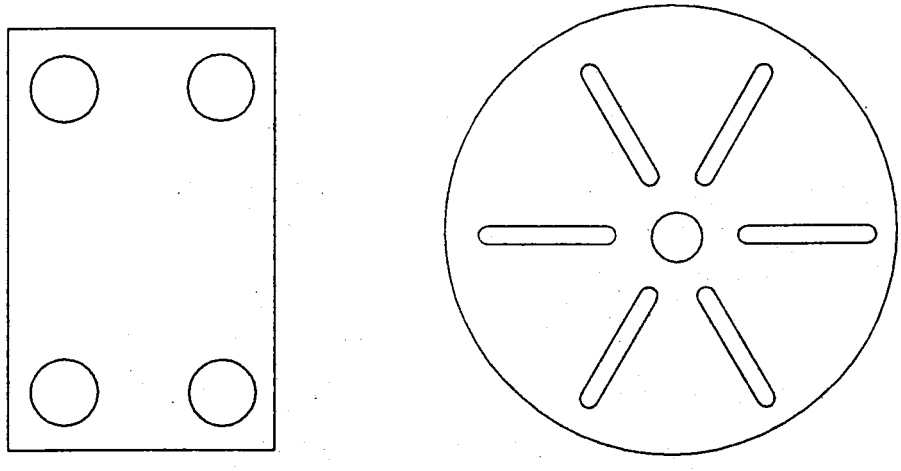


Fig 3.19 Highly symmetrical objects where the most logical inner boundary ordering is angular, as all have centroids virtually the same distance from the outer boundary centroid. The central inner boundary in the right-hand object would come at the end of the ordering sequence.

boundaries in shapes such as those shown in Fig. 3.19 is clearly by angle. All inner boundaries are of essentially equal size, shape and distance from the outer boundary centroid; any attempt to order them by the minute differences that might exist in these parameters would be totally counter-productive, as small changes in position - well within engineering tolerances - could lead to radical differences in ordering.

This leads to an apparent paradox. One can easily specify an ordering direction (clockwise or anticlockwise) for boundaries which are otherwise identical, but how should the start boundary be specified? The problem can in theory be solved in the same way as for inner boundary positions, by choosing a standard direction (such as the shape's major axis, or start direction of the outer boundary start line), and ordering inner boundaries on the basis of the relative angle between this and the vector joining each inner boundary centroid to the outer boundary centroid. (Fig 3.20). Unfortunately, this procedure does not work for objects whose outer boundary is a complete circle, as no outer boundary direction can be defined. Nor does it work for the class of otherwise symmetrical shapes where the relative positioning of inner boundaries creates a clear distinction between otherwise equivalent start lines (Fig 3.21). For these shapes, the standard direction defining the relative ordering of inner boundaries cannot be determined without reference to the positions of those inner boundaries.

There are two possible ways out of this paradox. Firstly, one could define a measure of inner boundary configuration which is independent of the ordering of individual boundaries, such as the sum S of (`length x angle`) for all vectors joining inner and outer boundary centroids. This could be used to distinguish directly between alternative outer boundary start lines, by determining S independently for each candidate start line, and then choosing the start line giving the greatest value for this measure. Shapes whose outer boundary is a complete circle have to be treated as a special case.

Alternatively, one could select each inner boundary in turn as candidate start boundary. This would then allow the remaining boundaries to be ordered by distance and angle with respect to the candidate inner boundary direction. Alternative inner boundary orderings could be compared, and a unique ordering selected, by successively comparing relative centroid distance, centroid angle, perimeter, start angle, etc of each boundary in the list until a difference emerges (Fig 3.22). If two or more orderings prove identical, the inner boundary configuration is symmetrical, and more than one candidate inner boundary direction has to be retained at this stage.

If no outer boundary start direction can be defined, this inner boundary standard direction becomes the standard direction for the entire shape. (If the inner boundary configuration is symmetrical, an arbitrary candidate direction can be selected). Otherwise, inner and outer boundaries are then oriented by recording the angle between outer boundary and inner boundary standard directions. If there is symmetry in either the outer boundary or inner boundary configuration of the object, each possible pair of alternative inner and outer boundary standard directions has to be compared to select a definitive representation (Fig 3.23).

3.3 Converting shapes to invariant form

3.3.1 Overview

To be of any real use, a shape retrieval system needs to be able to accept input from a wide variety of CAD systems. For reasons discussed in section 1.3, the most reliable way to ensure this is to base input to the system on a standard inter change format such as IGES. The problem of generating invariant shape descriptions of the kind described above can then be resolved into the following stages:

(a) extracting geometric information from IGES-format files;

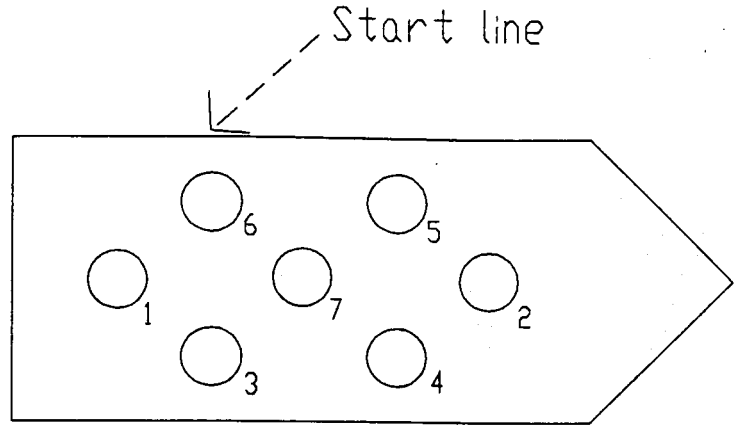


Fig 3.20 For unsymmetrical shapes, a logical start direction for inner boundary ordering is provided by the outer boundary start direction. Here, inner boundary 1 has its centroid vector most nearly parallel to the start line; subsequent boundaries are then ordered as shown.

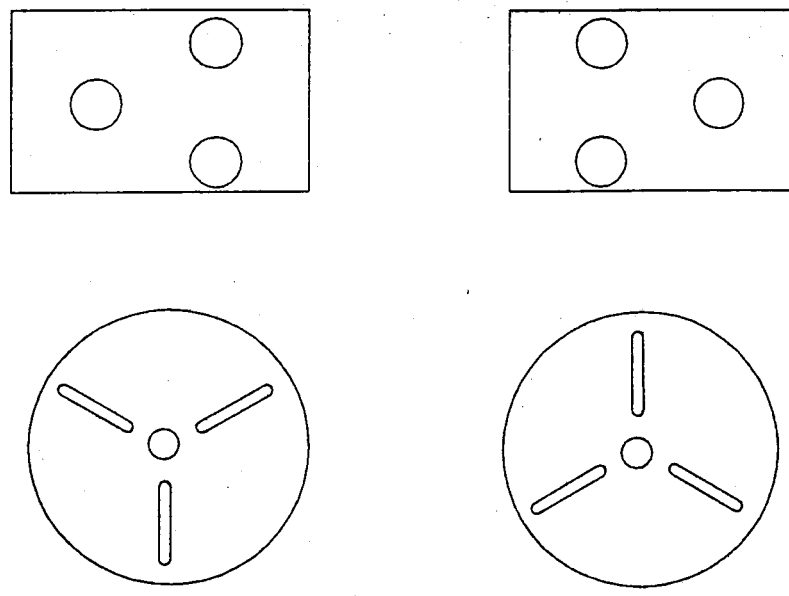


Fig 3.21 For shapes with symmetrical outer boundaries, no obvious start direction for ordering inner boundaries exists. Some alternative means is needed to select this start direction and hence the first inner boundary in sequence.

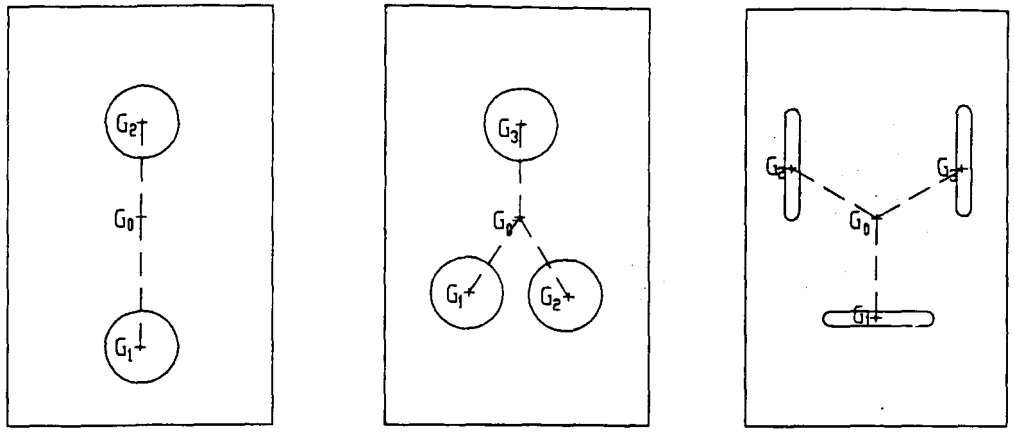


Fig 3.22 Resolution of inner boundary ordering problem by selecting each inner boundary in turn as candidate head of inner boundary list, and comparing alternative orderings of inner boundaries in terms of relative centroid distances and angles, and, if necessary, boundary size and shape. In case (a), inner boundary 1 takes precedence as distance G_0G_1 is greater than G_0G_2; in case (b), boundary 1 takes precedence because the angle $G_1G_0G_2$ is less than any of the other angles between centroid vectors; in case (c), all centroid distances and angles are equal, but boundary 1's start angle is greater relative to its centroid vector than the other boundaries.

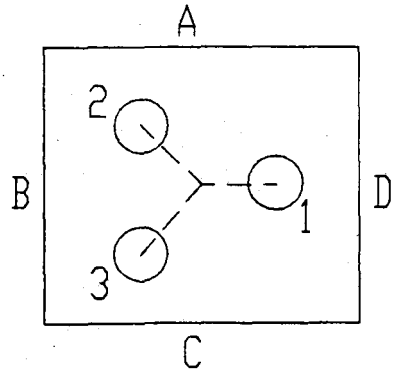


Fig 3.23 Comparison of candidate outer boundary start lines and inner boundary standard directions to yield canonical shape representation. The three possible inner boundary directions and four possible outer boundary directions are compared to find the pair giving the smallest difference in angles (in this case inner boundary 1 and outer boundary start line C).

(b) joining lines characterized in step (a) into connected boundaries;

(c) generating a hierarchical description of each shape boundary using the rewriting rules described above;

(d) converting these representations into invariant form.

Each of these steps was implemented as a separate program, as illustrated in Fig 3.24 and described below. All programs were written in VAX PASCAL, running under VMS on Newcastle Polytechnic's VAX 8700. Note that a further step:

(e) generating feature descriptions and loading complete shape descriptions into the shape database;

is described in detail in chapter 4.

3.3.2 Extraction of geometric information from IGES file

Program IGESTRAN takes a standard IGES file as input, identifies all drawing lines of suitable type, and stores their end-points and other defining parameters in an intermediate *line file* (file extension .LIN) for use by subsequent programs. It ignores dimension lines, text, and most non-geometric data such as associativity or property definitions.

The program reads the five sections of the IGES file in sequence:

Start section - ignored.

Global section - drawing filename, name of CAD system generating the drawing, drawing scale and resolution are extracted from input parameters for transfer to the output line file.

Directory section - each pair of directory entity (DE) records defining a drawing object is scanned. If the object is denoted "geometric" and "visible" on the input record, and is of acceptable type, a temporary record is created in main storage. The following IGES entity types are currently accepted - 100 (circular arc), 102 (composite data), 106 (copious data), 110 (straight line), 124 (transformation matrix), and - for input from the DOGS system only - one subclass of 406 (property). All other entity types are ignored.

Parameter section - each entity defined by a DE record pair will have one or more corresponding parameter entity (PE) records in this section. All PE records are read in sequence; where they match with the temporary records for accepted entities created above, appropriate line records are created, specifying both line type and defining coordinates.

Terminator section - Counts of accepted and rejected drawing entities are displayed, and "expected" and "recorded" record counts compared. If the input file is valid, appropriate header and line records are written to the line file.

3.3.3 Joining boundary lines

Program LINEJOIN takes as input the line file created by IGESTRAN, rejecting any lines shorter than a specified threshold and merging any overlapping lines, and groups all contiguous lines into boundaries. Sequences of line segments forming closed boundaries are written to a *boundary file* (extension .BND).

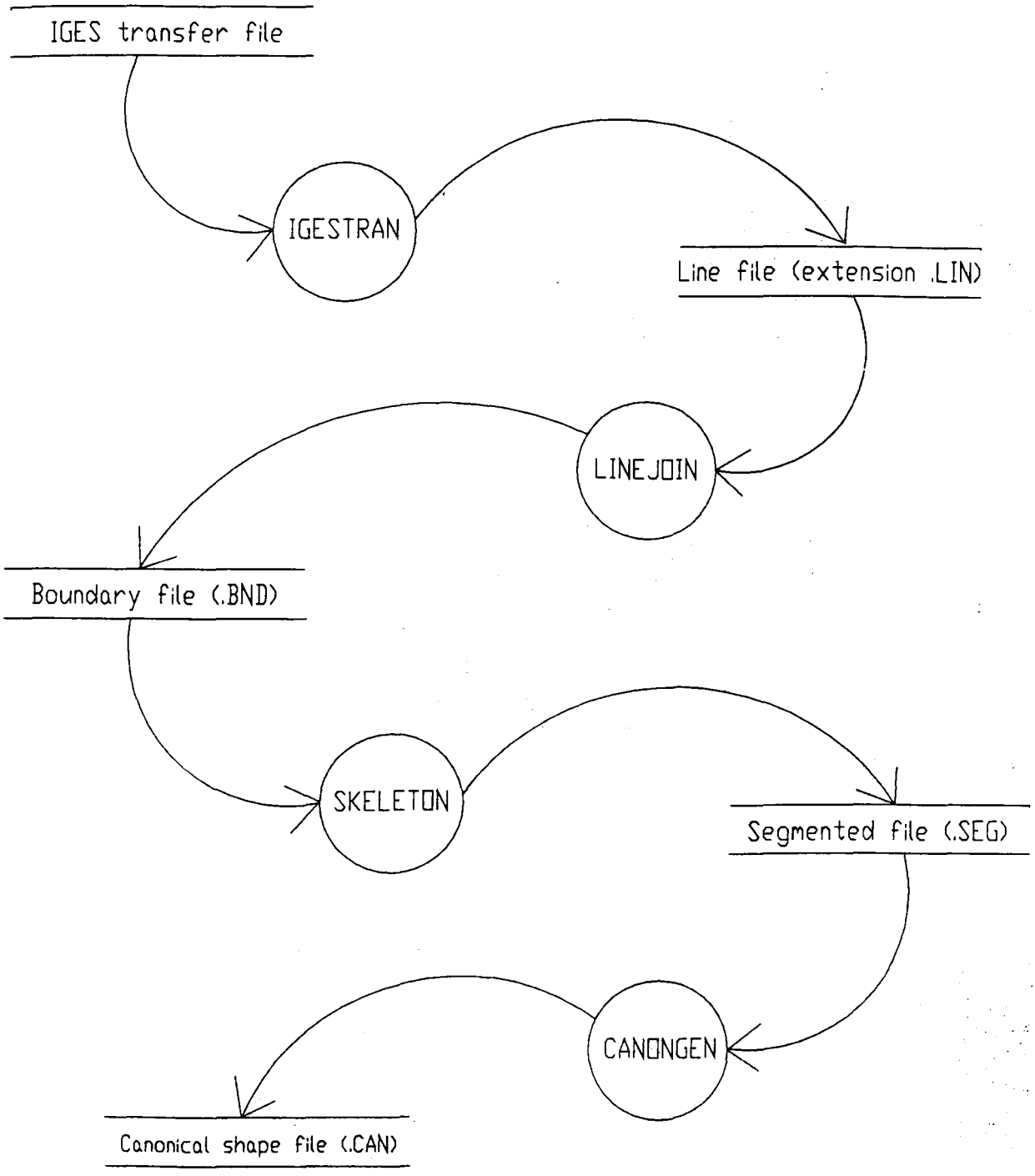


Fig 3.24 Data flow diagram showing the four stages in converting IGES format input drawings into invariant shape representations.

The program first reads in the entire line file, building up a two-way linked list of qualifying line records. A line record qualifies for inclusion if (a) its length is greater than a pre-set threshold tolerance (currently taken as 1000 x `drawing resolution`), and (b) it does not touch or overlap any existing lines following paths defined by the same equation. Lines below threshold length (which are usually remnants of deleted drawing features) are simply rejected. Contiguous or overlapping lines following identical paths (common in drawings constructed by copying or mirroring, as discussed in section 2.5.1) are merged, as illustrated in Fig 3.25. For all accepted lines, maximum and minimum *x*- and *y*-coordinate values are calculated.

When all valid lines have been read in, the program attempts to group as many as possible into complete boundaries, on the basis of the following algorithm:

```
Find the ungrouped line record with the lowest minimum X-coordinate
value (resolving ties using the lower minimum Y-coordinate). Denote
this the boundary start line;

While a start line can be found do

   Begin

   Create a new boundary header record, detach the start line
   record from the "ungrouped" list and link it to the boundary
   header;

   Record start and finish coordinates of boundary, and success at
   finding matching line;

   While the boundary list is incomplete (i.e. start and finish
   coordinates unequal) and matching line successfully found do

      Begin

      Search the list of ungrouped line records for lines whose
      start or finish coordinates match those of the ends of the
      growing boundary;

      If a matching line can be found then

         Detach this line from the ungrouped list and link it to the
         appropriate end of the growing boundary list

      else

      Record failure to find a matching line

      End;

   Find a new boundary start line as before

   End
```

To allow for errors in drawing, end-coordinate matching criteria are progressively relaxed if no exact match is found; if an approximate match is found, line and boundary end-points are adjusted accordingly.

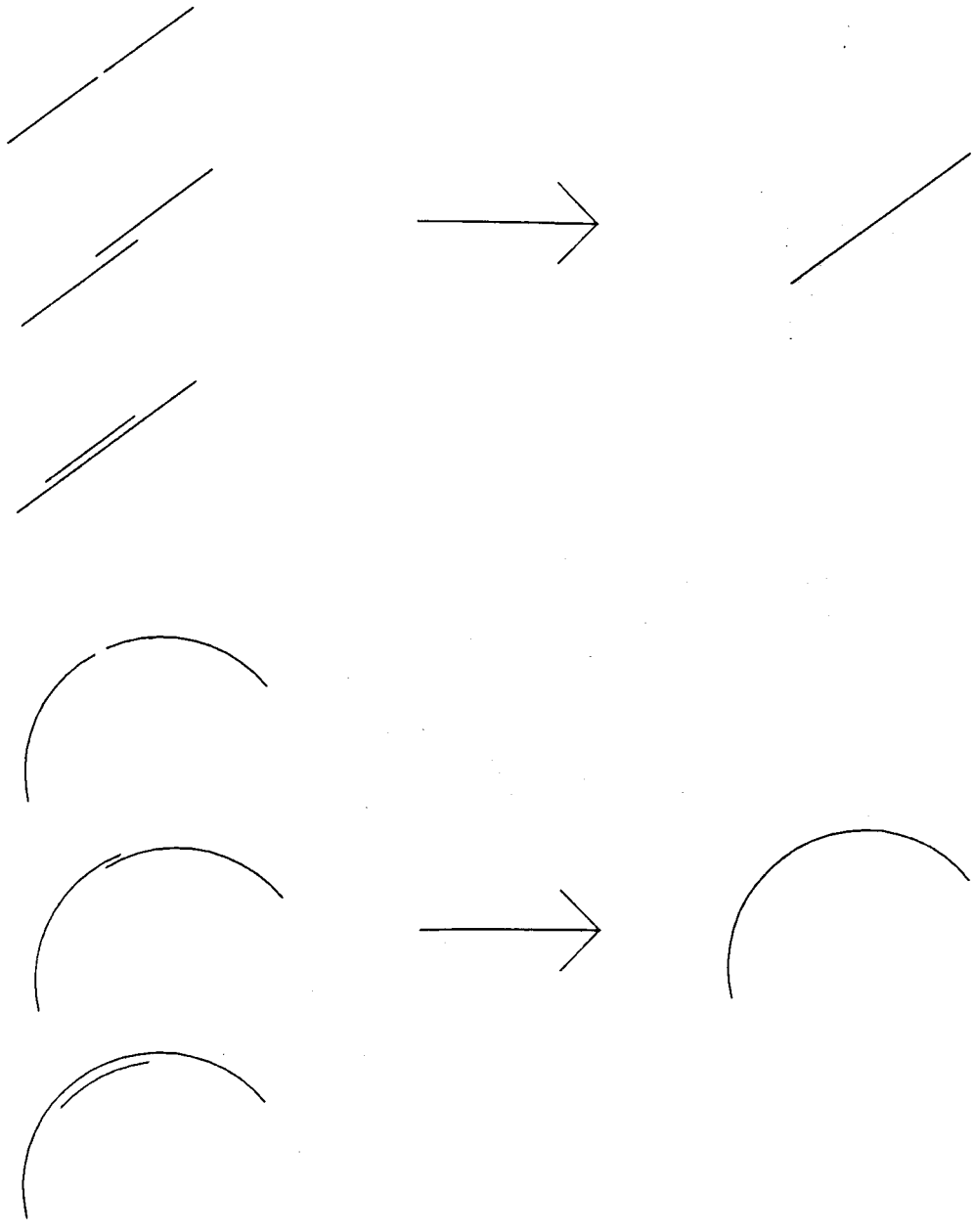


Fig 3.25 Merging of contiguous or overlapping lines from the original drawing to generate single line segments - each represented by a single record. Line segments were considered for merger only if both were straight lines with identical angle and distance from the origin, or both were circular arcs with identical radius and arc centre coordinates - within specified tolerances.

When all available lines have been grouped into boundaries, all complete boundaries are written to the boundary file, in the order in which they were created. With the given domain of shapes, this means that the outer boundary is always stored first, followed by inner boundaries in order of increasing minimum x-, then y-coordinates. Within each boundary, the header record is always written first, followed by the line records for the starting segment (the segment with minimum x-, then y-coordinates), then all remaining segments in order of boundary traversal in the appropriate direction (anticlockwise for outer boundary, clockwise for inner boundaries). The primary defining parameters for each line segment (length, curvature and discontinuity angle) are calculated and stored at this stage.

3.3.4 Uncovering underlying shape

Program SKELETON takes as input the boundary file created by LINEJOIN, applying the rewriting rules described in Section 3.2.3 to reduce lines defining corner and midline features to subordinate status, generating a hierarchical representation of each boundary which can be traversed at different levels. This representation is stored in a *segment file* (extension .SEG).

The program operates on each boundary in turn. Firstly, it builds up a two-way linked list of all boundary segments in order, and then scans the boundary to ensure that no line segments remain fragmented. It is important that this step precedes shape hierarchy generation, because preconditions for recognizing shape features all involve comparing the relative lengths of lines forming and enclosing potential shape features. The boundary is scanned once; if any neighbouring line segments are found to have discontinuity angles and normalized differences in curvature less than a specified threshold, a new parent record is created for the merged line, and linked into the boundary segment list (Fig. 3.26). Records for individual line fragments are retained as child records to the new parent. This ensures that the representation remains information-preserving.

The program then builds up a hierarchical description of the boundary shape. It creates new *header* records, as shown in Figs. 3.27 and 3.28, to represent the extended lines generated by the shape rewriting rules illustrated in Figs 3.9 and 3.10, and links them to existing records with *parent* and *child* pointers. All drawing lines are examined in turn, starting with the shortest, according to the following algorithm (NB - the term *shortest* here has the obvious meaning only if one line is shorter than all others by more than a specified tolerance. If two lines of essentially equal length are found, the shorter is defined by examining each candidate's successor lines (found by following the chain of *next* pointers from each line) until a difference in length is found, or the entire boundary has been traversed):

```
Mark all boundary segments 'unexamined', with level no = 1. Find
the shortest unexamined line (if any) and mark it 'examined';

While a new 'shortest' line can be found do

  Begin

  Attempt to process shortest line as (part of) a potential corner
  feature;

  If no shape feature discovered then

    Attempt to process shortest line as (part of) a potential
    midline feature;
```

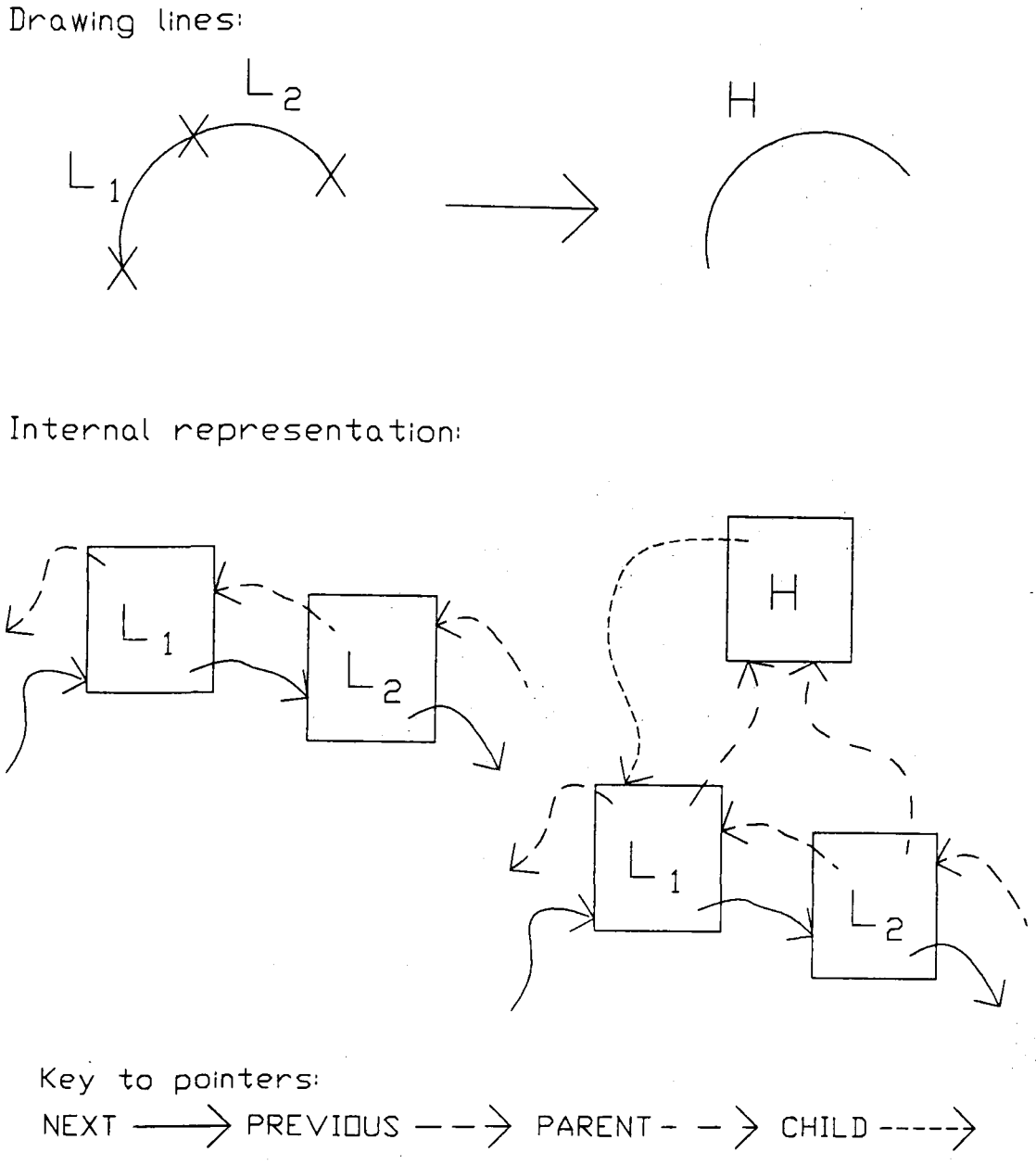


Fig 3.26 Replacement of contiguous lines of similar curvature with single "parent" record, a symmetric circular arc inheriting start and finish coordinates and start direction from "child" segments (which then determine parent segment's length, curvature and discontinuity angle).

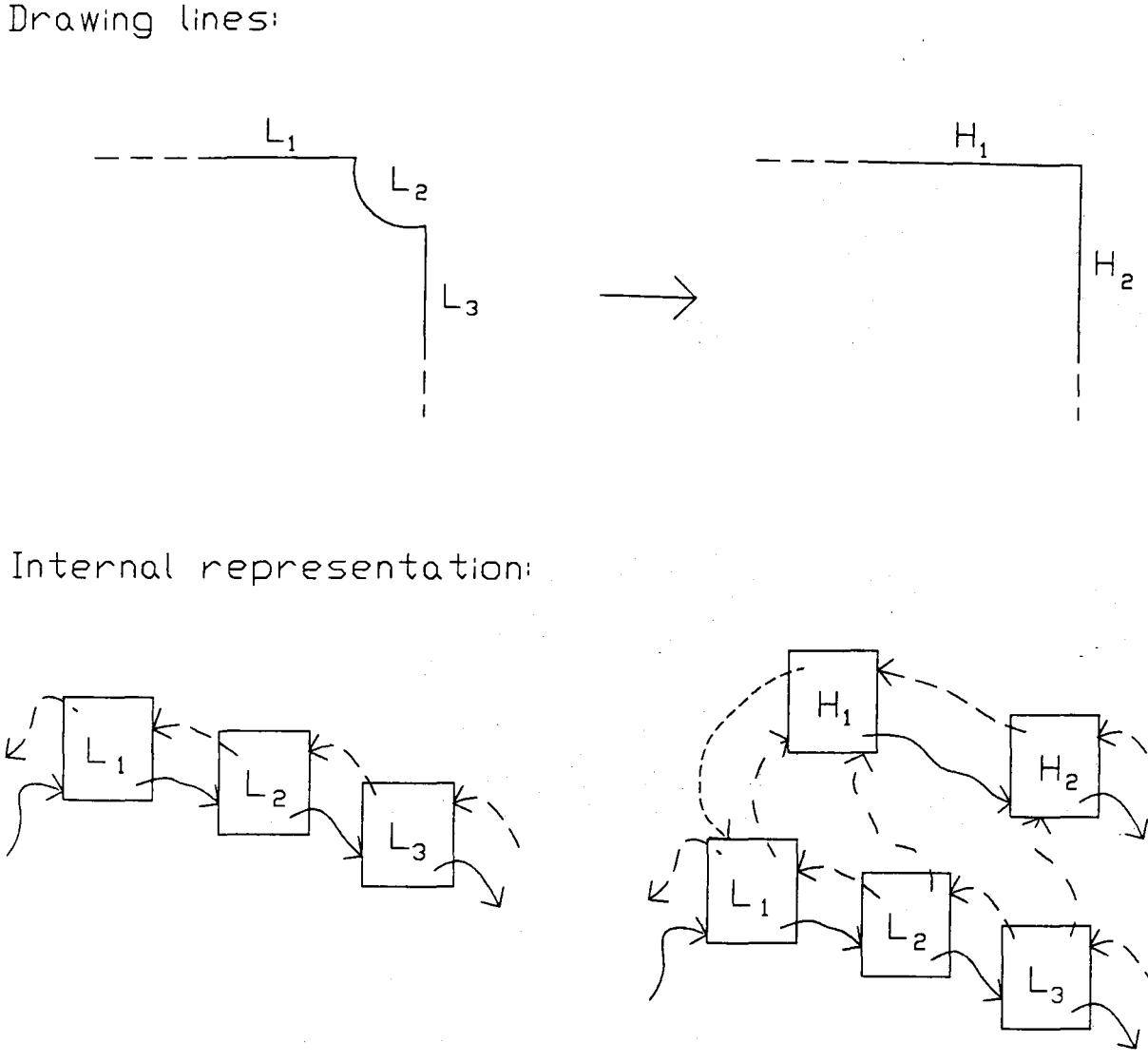


Fig 3.27 Creation of additional "parent" records to implement corner shape rewriting rules illustrated in Fig. 3.9. Parent records H_1 and H_2 are linked into the Next *and* Previous *pointer chains in place of L_1 and L_3.*

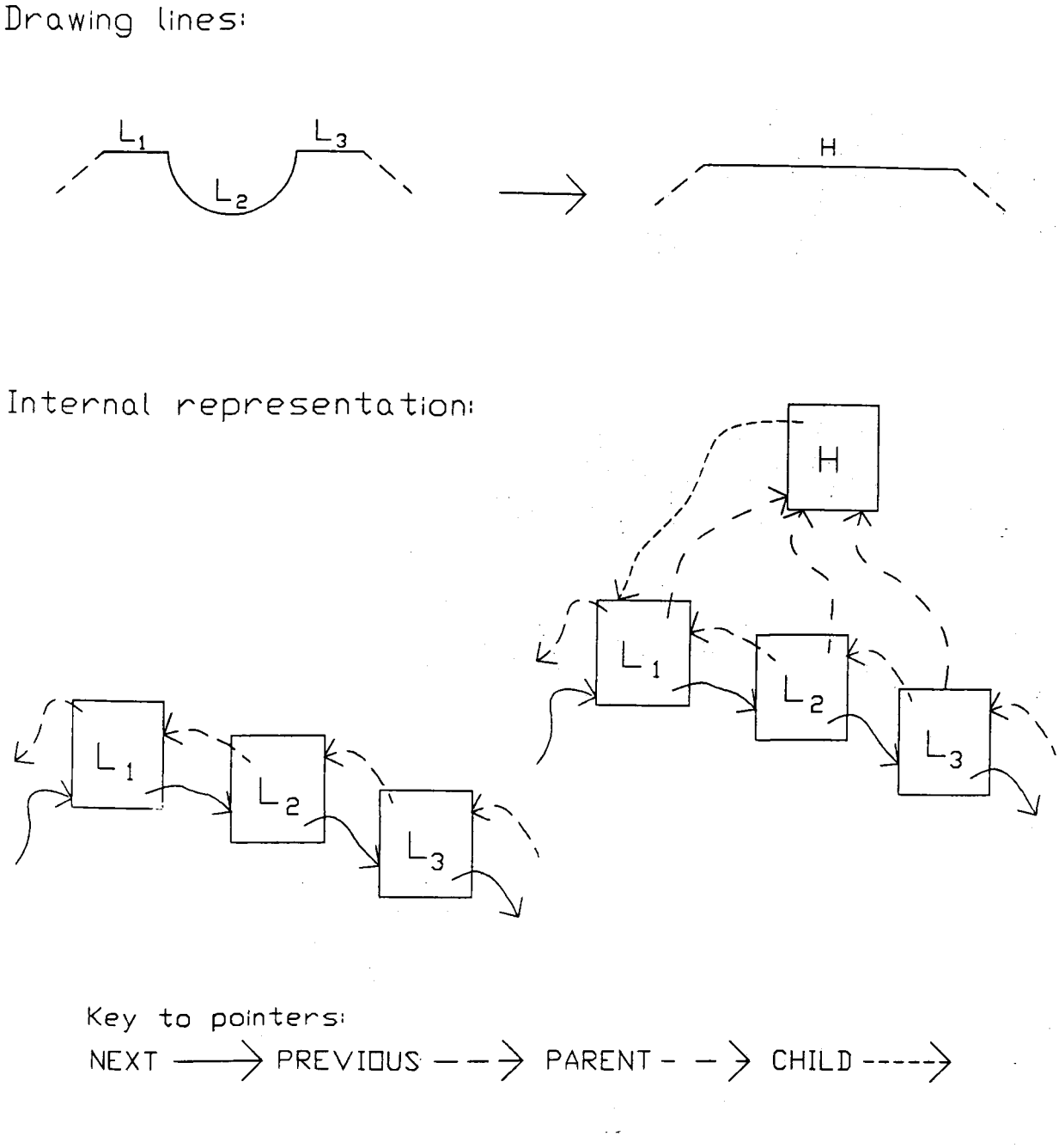


Fig 3.28 Creation of additional "parent" record to implement midline shape rewriting rules illustrated in Fig 3.10. Again, parent record H is linked into the Next *and* Previous *pointer chains in place of L_1 and L_3.*

```
    Find the next shortest unexamined line (if any) and mark it
    'examined'

End
```

An example of the type of hierarchy produced is shown in Fig 3.29. The process of building up the hierarchy is complicated by the need to ensure that the final structure produced is as far as possible independent of the order in which segments are processed. Even though a strict "shortest first" order is specified, similar but not identical shapes such as those shown in Fig. 3.30 need to yield similar-shaped hierarchies of header lines. The most effective way to achieve this was found to be to limit the creation of new records to cases where positional information would otherwise be lost.

Consider the case shown in Fig. 3.31. Because the header record created at one step encloses a corner or midline feature at the next, automatic creation of new header records every time a rewriting rule is invoked can lead to an unwieldy pyramid of header records. Worse still, the resulting structure is completely dependent on the order in which segments are processed.

If the creation of new header records is limited to cases where the coordinates of an intermediate point would otherwise disappear (Fig 3.32), a much more manageable structure results. First, such a structure is the minimum required to allow traversal of a shape boundary at any desired level. Since a new header record is created only where a point coordinate would otherwise be lost, one of its end-points must always contain unique information. Removal of *any* header record from the structure would therefore prevent traversal of the boundary at one or more levels. Secondly, such a structure is much less sensitive to the actual order in which line segments are processed - as shown in the example in Fig 3.33. An algorithm reliably yielding the minimum set of header records should generate this same set, whichever order of processing is chosen for the line segments.

The following algorithm was therefore devised to process corner features. It tests whether the shortest currently unexamined line, indicated by the pointer *Shortest*, enclosed by lines *First* and *Last*, is a potential corner feature. If so, it creates new header records *Head1* and *Head2*, which are added to the hierarchy if *First* and *Last* contain unique coordinate information, but which replace *First* or *Last* otherwise. *LenRatio* is a constant normally set equal to 1.

```
Set First to indicate the line preceding Shortest, Last the line
following it, and indicate no shape feature discovered;

While no shape feature discovered and number of lines separating
First and Last is less than 3 do

  Begin

  Repeat

    If First and Last would meet at a suitable angle when extended
    (e.g. between 30° and 150°), and appropriate qualifying
    conditions are met, e.g: Head1 must be longer than First *
    LenRatio, Head2 must be longer than Last * LenRatio, total
    length of line segments between First and Last must be less
    than LenRatio * (Head1 + Head2) then

      Indicate shape feature discovered;
```

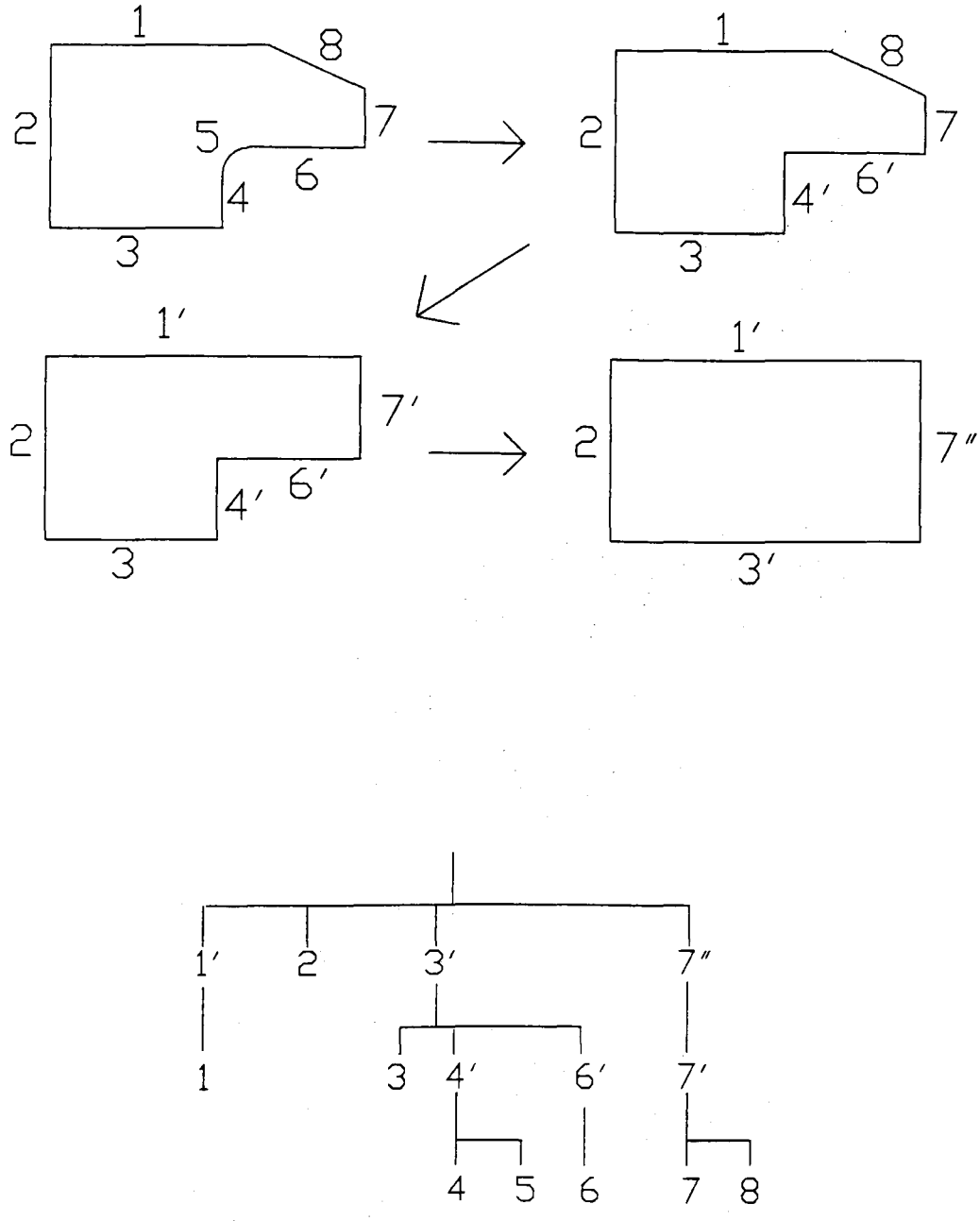


Fig 3.29 Hierarchy of line levels produced by repeated application of the shape rewriting rules described above. In theory, it should be possible to traverse such a structure at any given level - though in practice this is possible only where the application of the rules is strictly controlled. See Figs 3.31 and 3.33 below.

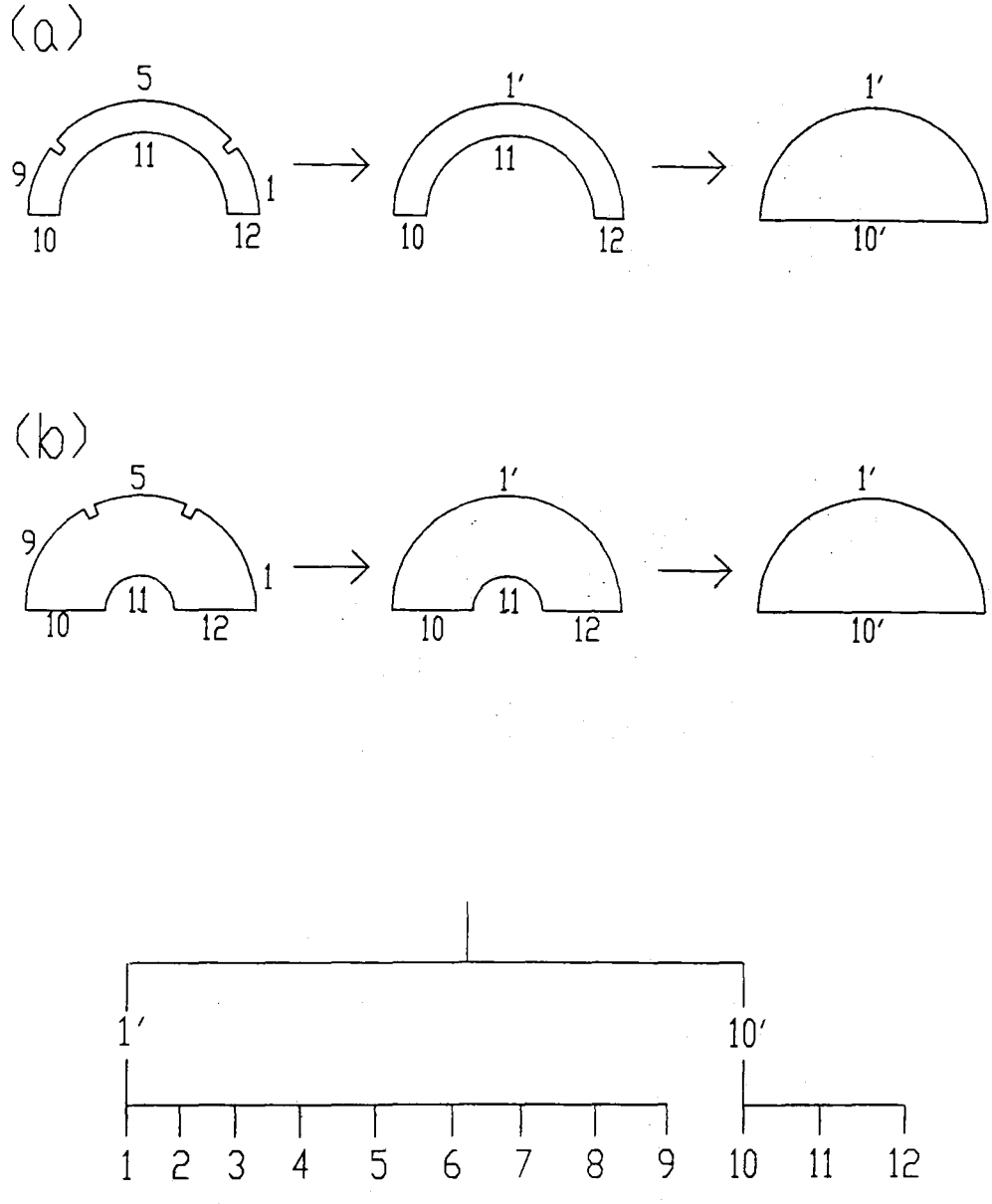
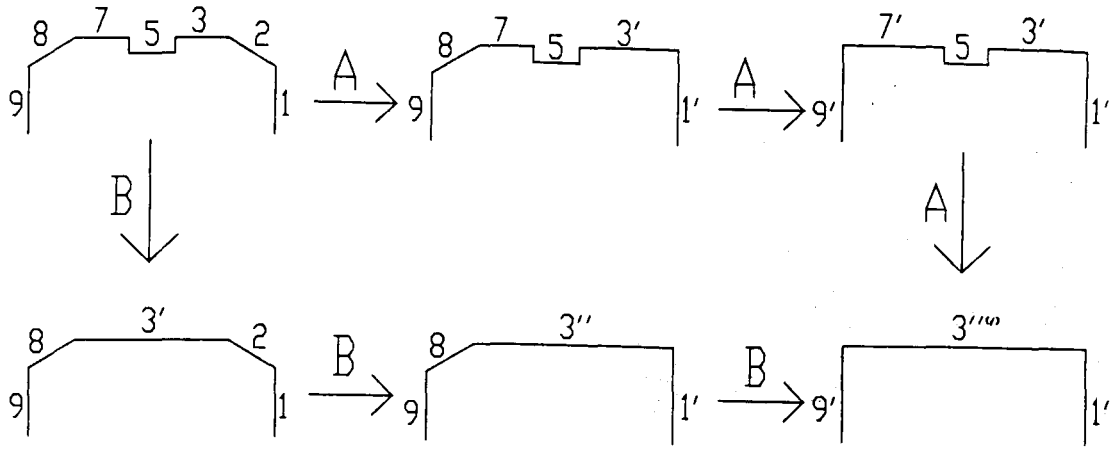


Fig 3.30 Need for similar shapes to generate similar hierarchies of intermediate level lines. In both cases, processing needs to yield two top-level lines (1' and 10'), irrespective of whether construction of the hierarchy begins between lines 1 & 5, 5 & 9, or 10 & 12.



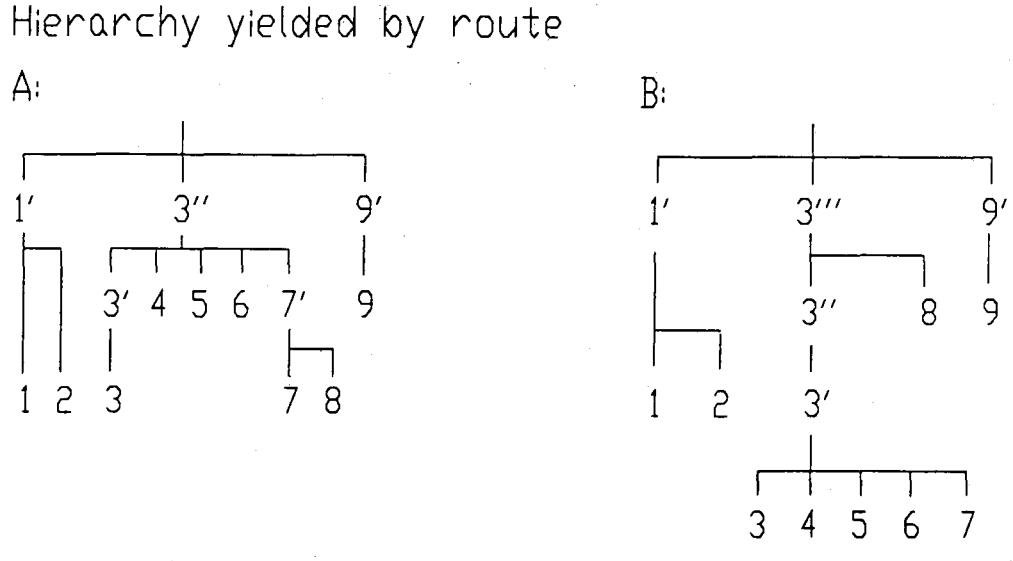


Fig 3.31 Unrestricted creation of header records when applying rewriting rules can lead to problems. Two alternative sequences of rewriting can yield markedly different hierarchies of inter mediate level lines, even though the top-level structure is the same in each case. In neither hierarchy is it obvious how traversal at intermediate line levels (see Figs 3.11 and 3.12) is supposed to be achieved.

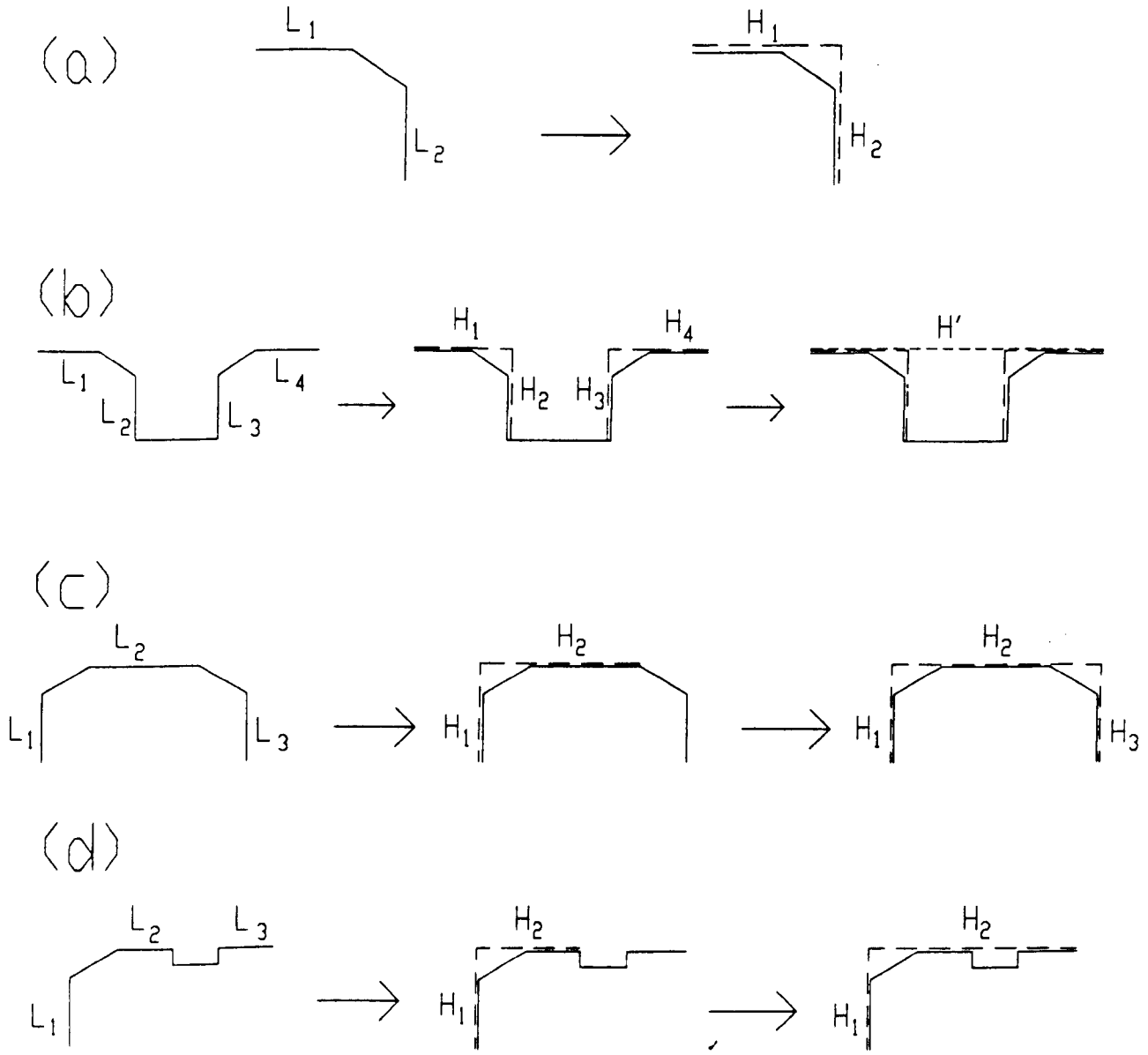
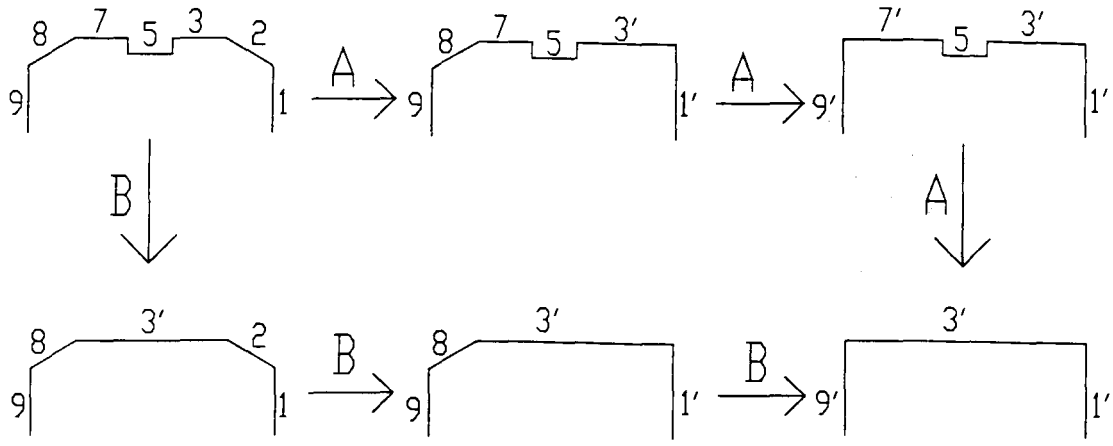


Fig 3.32 Illustration of rules for limiting the creation of new header records. In cases (a) and (b), new header records are created at each stage to prevent loss of information on the position of line segment end-points. In case (a), new header records H_1 and H_2 are created because extending L_1 and L_2 till they met would lose the original end coordinates of L_1 and the original start coordinates of L_2. A similar argument applies to both steps of case (b) - information on the position of line end-points is lost unless two levels of header record are created.

In cases (c) and (d), existing header records can be extended without loss of information. In both cases, header H_2 can safely be extended at step 2 without losing information on its original end coordinates, as these are identical with those of line L_2.



Hierarchy yielded by either route:

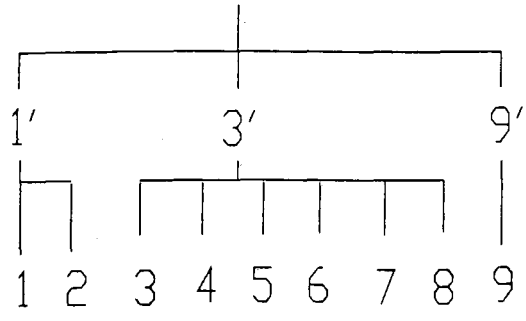


Fig 3.33 Limiting creation of header records as specified above yields a hierarchy which is both more compact and less sensitive to the order in which rules are applied. Here, sequences A and B both yield the same hierarchy, which can clearly be traversed at just two levels, 1'-3'-9' and 1-2-3-4-5-6-7-8-9.

```
        Create new line records Head1 and Head2, extensions of First
        and Last respectively, such that start coordinates of First
        coincide with those of Head1, finish coordinates of Last
        with those of Head2, and finish coordinates of Head1 with
        start coordinates of Head2;

        Mark Head1 as the parent of First and all intermediate line
        records, and Head2 as the parent of Last;

        If finish coordinates of First are identical to finish
        coordinates of its last child record then

          give Head1 the same level no as First and replace First
          by Head1 in the line segment hierarchy, marking Head1 as
          the parent of all First's child records

        else

          give Head1 a level number 1 higher than First, and retain
          both Head1 and First in the line segment hierarchy;

        If start coordinates of Last are identical to start
        coordinates of its first child record then

          give Head2 the same level no as Last and replace Last by
          Head2 in the line segment hierarchy, marking Head2 as the
          parent of all Last's child records

        else

          give Head2 a level number 1 higher than Last, and retain
          both Head2 and Last in the line segment hierarchy

      else

        Replace First and Last by their predecessors

    Until shape feature discovered or Last = Shortest;

    Replace Last with its successor

    End
```

Similarly, the following algorithm was devised to process midline features, testing whether the shortest currently unexamined line is a potential midline feature. If so, it creates a new header record *Head*, which is added to the hierarchy if *First* and *Last* both contain unique coordinate information, but which replaces *First* or *Last* (or both) otherwise:

```
Set First to indicate the line preceding Shortest, Last the line
following it, and indicate no shape feature discovered;

While no shape feature discovered and number of lines separating
First and Last is less than 5 do

  Begin
```

```
Repeat

   If First and Last can be described by the same equation, and
   the total length of all line segments between First and Last
   is less than LenRatio * (the length of Head) then

      Indicate shape feature discovered;

      Create new line record Head, such that start coordinates of
      First coincide with those of Head and finish coordinates of
      Last with those of Head;

      Mark Head as parent of First, Last and all intermediate line
      records;

      If start coordinates of Last are identical to start
      coordinates of its first child record then

         give Head the same level no as Last and replace Last by
         Head in the line segment hierarchy, marking Head as the
         parent of all Last's child records;

      If finish coordinates of First are identical to finish
      coordinates of its last child record then

         give Head the same level no as First and replace First by
         Head in the line segment hierarchy, marking Head as the
         parent of all First's child records;

      If neither of these conditions holds then

         give Head a level number 1 higher than First and retain
         Head, First and Last in the line segment hierarchy

   else

      Replace First and Last by their predecessors

Until shape feature discovered or Last = Shortest;

Replace Last with its successor

End
```

Some results of these algorithms are shown in Figs 3.34 and 3.35. Note that additional qualifying restrictions on corner features have been added to produce an intuitively more "natural" performance with a wide range of shapes, and to reduce dependence on order of processing. For example, corner features which would spoil later discovery of midline features are inhibited, and corner features bounded by curved lines or acute-angled straight lines are inhibited if they would spoil features bounded by right-angled straight lines.

Finally, the entire sequence of line records is written to file, using a recursive procedure which stores each line segment record at a given level, followed immediately by its child records, to whatever depth is required. The procedure records and stores the depth of

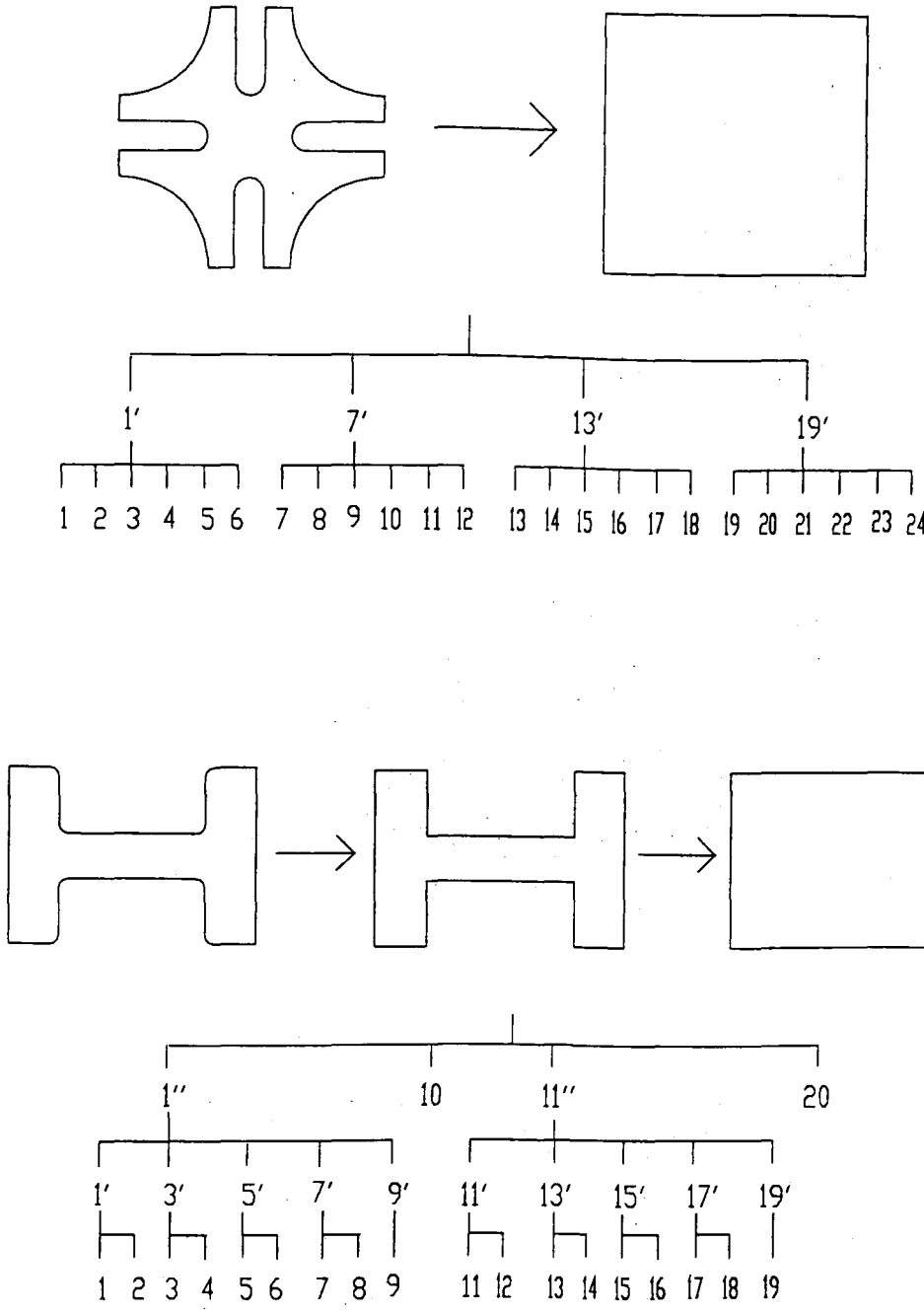


Fig 3.34 Multi-level representations of some test shapes, illustrating line segment hierarchies. The hierarchy can be traversed at any given level using the algorithm outlined in Section 3.2.3.

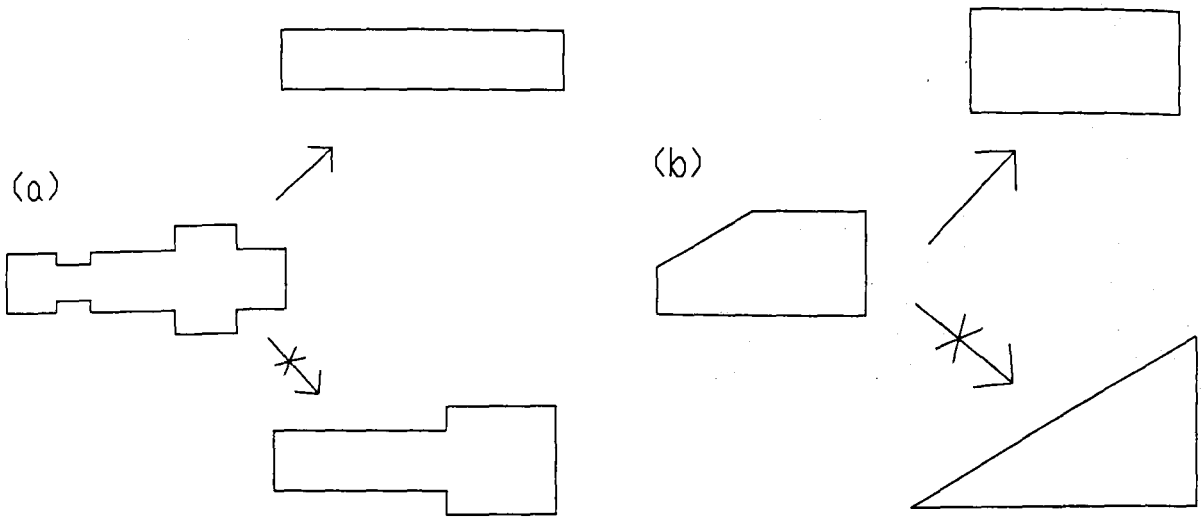


Fig 3.35 Cases illustrating need for additional restrictions on rewriting rules - (a) where line segment could ***either*** *form part of a corner feature* ***or*** *a midline feature, midline feature is preferred; (b) where diagonal segment between perpendicular line segments could yield* ***either*** *triangular* ***or*** *rectangular overall shape, rectangular is preferred.*

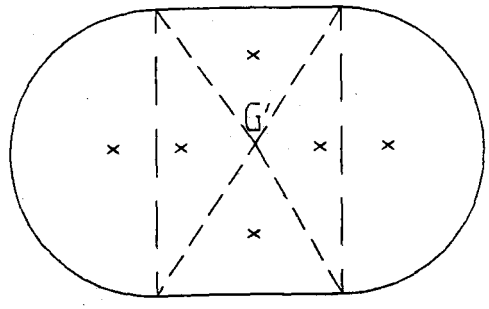


Fig 3.36 Triangulation of a boundary to calculate the position of its centroid. The centroid coordinates and area of each triangle or circular arc segment making up part of the shape enclosed by the boundary are calculated by appropriate formulae; moments are then taken to find coordinates of overall boundary centroid G_0. Point G' can be selected arbitrarily, as its position has no effect on the final result.

nesting, so that each stored line record contains two level numbers, as discussed in section 3.2.3; E_b, the (bottom up) level number assigned during construction of the feature hierarchy, and E_t, the (top-down) level number assigned at storage time. Together, these level numbers permit easy traversal of the shape boundary at any level.

3.3.5 Generating canonical representation

Program CANONGEN takes the segmented boundary representations from the segment file created by SKELETON, converts each boundary representation into canonical form, calculates the relative positions of inner and outer boundary centroids, sorts inner boundaries into a standard ordering, transforms the spatial relationship of inner boundaries into invariant form, and normalizes all lengths and directions in terms of outer boundary perimeter and major axis.

3.3.6 Outer boundary representation

The program first processes each boundary in sequence, reading the line segments making up each boundary into a two-way linked list, and validating each list by checking that it forms a closed boundary at whichever level it is traversed. The x- and y-coordinates of each boundary centroid are calculated as shown in Fig 3.36. The first step in converting each boundary into canonical form is then taken, by selecting a definitive start line, using the following algorithm:

```
Choose an arbitrary top-level line segment, denote it StartLine,
and mark it as a potential start segment. Find the next top-level
line in the list and denote it CurrentLine;

While a new CurrentLine can be found do

   Begin

   Set SL = StartLine, CL = CurrentLine, current level to 1, and
   action required to "unknown" {other legal values are "replace
   start line", which marks CurrentLine instead of StartLine as
   start segment, "add start line", which marks CurrentLine as well
   as StartLine as start segment, and "no action required"};

   While action required unknown do

      Begin

      Repeat

         Case of chord length of line CL
            > SL: replace start line
            < SL: no action required
            = SL:

            Case of length of line CL
               > SL: replace start line
               < SL: no action required
               = SL:

               Case of arc angle of line CL
```

```
                > SL: replace start line
                < SL: no action required
                = SL:

                Case of discontinuity angle following line CL
                  > SL: replace start line
                  < SL: no action required
                  = SL: replace CL and SL by their  successor lines
                  at current level;

        Until current action known or boundary completely traversed at
        current level;

        Increment current level;

        If no more levels to traverse and action required not yet
        known then

            add start line

        End;

    Replace CurrentLine by its successor

    End
```

This procedure always yields a unique start line unless the boundary possesses rotational symmetry - a boundary with *n*-fold symmetry will yield *n* potential start lines. If the shape as a whole contains unsymmetric features, the position and orientation of other boundaries is used to choose between candidate start lines, as outlined below. If not, choosing any of the candidate start lines will yield an identical representation of the boundary; the program therefore selects the first candidate start line it encounters.

3.3.7 Inner boundary representation

When all individual boundaries have been cast into canonical form (individual inner boundaries are processed in exactly the same way as the outer boundary), the program attempts to define an inner boundary standard direction, and canonical ordering of inner boundaries. The length and direction of each inner boundary's centroid vector (Fig. 3.18) are first calculated and stored. The inner boundaries are then sorted into a provisional order, by decreasing centroid vector length.

Drawings with zero or one inner boundaries are treated as special cases. Otherwise, the program then attempts to create one or more (if symmetric) canonical boundary orderings, as follows:

```
Choose the first boundary on the provisional boundary list, and
define UnitDir as the direction of this boundary's centroid vector;

Create a new temporary listing of the remaining inner boundaries,
following the provisional ordering except that a group of
boundaries whose centroid distances differ by less than a given
tolerance are instead sorted by (CentroidAngle - UnitDir). Adjust
inner boundary start lines/start directions where necessary to
minimize angle between UnitDir and start direction;
```

```
Denote this listing the 'current candidate list';

Attempt to find another candidate boundary to head the inner
boundary list (i.e. another boundary with centroid distance within
the given tolerance);

While another candidate boundary can be found do

   Begin

   Create a temporary listing headed by the new candidate boundary
   as specified above;

   Compare each boundary in the new listing with the corresponding
   boundary in the current candidate list, using the following
   parameters in turn until a difference emerges or all parameters
   have been compared: centroid angle, centroid distance,
   perimeter, no of segments, max no of levels, start angle, then
   chord length, arc length, arc angle and discontinuity angle of
   each line segment at each level;

   Case of the value of the discriminating parameter in the new
   list relative to that in the current list (with the exception of
   centroid angle, where actions are reversed):

      Greater: Replace the current list with the new list

      Less: Discard the new list

      Equal: Retain both old and new lists;

   Attempt to find another candidate boundary to head inner
   boundary list

   End
```

The procedure terminates yielding one or more (equivalent) orderings of inner boundary records. In the first of the three examples shown in Fig. 3.22, it creates one list headed by boundary 1, adds boundary 2 to that list, then terminates. The final inner boundary direction is thus that of the vector G_0G_1. In the second example, three candidate lists would be generated (1-2-3, 2-3-1, and 3-1-2); list 2-3-1 would then be compared boundary by boundary with list 1-2-3, and the second list rejected because boundary 3 has a larger centroid angle (relative to the list header, boundary 2) than boundary 2 (relative to its list header, boundary 1). Similar considerations cause rejection of the third list.

Once definitive ordering(s) of inner boundaries have been generated, the final resolution of inner and outer boundary directions becomes possible. Again, a number of special cases need to be treated - where no inner boundary is present, or there is a single inner boundary with zero centroid distance, inner boundary standard direction is set equal to the start direction of the outer boundary start line. Conversely, if no outer boundary start direction can be defined (because the outer boundary is a complete circle), outer boundary start direction is set equal to inner boundary standard direction. If both inner and outer boundaries are unsymmetric, the outer boundary direction is uniquely defined by the start direction of its start line, and inner boundary direction by the centroid vector of the first inner boundary. Otherwise, each candidate (outer boundary startline)/(inner

boundary list header) pair has to be compared to find the combination giving the smallest difference between inner and outer boundary directions. In the example from Fig. 3.23, combinations A1, A2, A3, B1, B2, B3, etc are successively tested to find the one where directions are most nearly parallel.

Once final inner and outer boundary directions are established, the chosen inner boundary ordering is made permanent, and inner boundary start angles/start lines adjusted where necessary to reflect the final direction chosen. All lengths and angles are normalized with reference to outer boundary perimeter and standard direction respectively.

3.4 Efficiency considerations

All the algorithms described here appear to be polynomial-bounded. IGESTRAN performs a sequential read of all DE records from the input IGES file, creating temporary records in main storage, then reads all PE records in sequence, matching them with the corresponding temporary records and creating new combined records. Provided DE and PE records are sorted in the same way in the input file (as they should be), this is a purely linear process, i.e. $O(n)$, where n is the number of records in the input IGES file.

The most processor-intensive routine within LINEJOIN is probably that of growing new boundaries. Each time a new segment is added to a boundary, a search has to be made through all unattached segments to identify the correct line to add to the growing boundary. In the worst case, this would involve $n(n-1)/2$ comparisons, where n is the total number of line segment records input - an $O(n^2)$ process.

SKELETON builds up its shape feature hierarchy by successively examining each boundary segment in order of increasing length. The process of comparing line pairs surrounding the current shortest segment, and generating one or more header records where appropriate, is lengthy, but independent of the number of boundary segments n except for very small values of n, since an upper bound is set on the number of candidate header pairs examined in each case. Each time a new header record is created, these comparisons have to be repeated at the next higher level. The total number of comparisons performed for any given boundary is thus a function not only of the number of segments it contains, but also of its shape feature content, in terms of the number of header records generated.

In the "best" case, where no shape features are detected, analysis is simple - precisely n comparisons are made. In the "worst" case, every segment in the original boundary becomes subsidiary to a higher-level segment, every segment at this level becomes subsidiary to a yet higher level, and so on. Since in this case each level can never contain more than half the number of segments of the level below, the total number of segments - and hence comparisons - cannot be greater than $2n$. In intermediate cases, the situation is harder to determine, though it would appear reasonable to regard the algorithm as $O(n)$, where n is the total number of line segment records input. (One could argue that since the boundary is first searched to find the shortest segment, it is strictly an $O(n^2)$ process. However, the amount of processing required to find the shortest line is trivial compared with that required for examining candidate header segments for the size of drawing handled here).

Finally, CANONGEN identifies unique boundary start lines by comparing segment sequences starting from each pair of candidate start lines in turn. In the worst case, this could involve comparing n segment pairs in each of N candidate sequences, where n is the total number of segments making up all levels, and N the number of top-level segments. This is essentially an $O(n^2)$ process. Generating a standard ordering of inner boundaries involves setting up and comparing a number of provisional orderings, each starting with a different candidate start boundary. In the worst case (n inner boundaries,

all the same distance from the outer boundary centroid), this would involve setting up and comparing n candidate ordered boundary lists, each generated using a simple exchange sort (an $O(n^2)$ algorithm). The overall process here is thus $O(n^3)$.

Typical cpu usage figures for each of these programs as implemented on a VAX 8700 are presented in table 3.4.1 below. To obtain these figures, each program was run with representative "simple", "average" and "complex" drawings, identified by ranking input files by the number of relevant input components, and selecting input drawings with percentile ranks of 10, 50 and 90 respectively. The figures, produced by accessing the processor's real-time clock from within the program, should be treated with some caution, since they represent total cpu usage for the current process, including all transfer of data between main storage and backing store, whether requested by the application program or caused by the operating system paging out blocks of memory to allocate to another process. This latter effect was minimised by running all jobs in batch mode at times when the machine was known to be lightly loaded, but could not be totally eliminated.

Table 3.4.1 - cpu usage for translation programs

Program name	Drawing comp-lexity	No of input records*	CPU usage (s)	
			processing+	total
IGESTRAN	Simple	74	0.15	0.25
	Average	135	0.28	0.38
	Complex	331	0.66	0.80
LINEJOIN	Simple	11	0.01	0.08
	Average	25	0.04	0.15
	Complex	64	0.10	0.20
SKELETON	Simple	8	0.02	0.12
	Average	17	0.04	0.13
	Complex	44	0.09	0.18
CANONGEN	Simple	11	0.02	0.10
	Average	24	0.03	0.13
	Complex	52	0.04	0.15

**parameter has slightly different meaning for each program; for IGESTRAN, it represents no. of fixed-length 80-character records in IGES transfer file; for remaining programs, it represents no. of line segment records in intermediate .LIN, .BND, or .SEG transfer files*

+excluding cpu time for opening and closing data files

Program IGESTRAN was on average the greatest user of cpu time, largely because of the sheer volume of data forming the average IGES-format transfer file. CPU usage figures for all programs appear consistent with predicted overall complexities.

3.5 Concluding remarks

This chapter has described in detail the shape representation methods adopted for the prototype version of SAFARI, and the processing required to generate such representations from drawings input in standard IGES format. The method can be distinguished from those adopted by other authors in two main respects: (a) the use of the concept of the boundary level, allowing each boundary to be viewed at differing levels of

detail, (b) the specification of a standard ordering for shape components, to generate a unique representation for each shape.

There are however a number of limitations on the extent to which such system objectives can be achieved in practice. The most important of these are as follows:

(a) Drawings must be expressed in IGES format, using only the limited domain of geometric entity types specified in section 3.3.2. This is not an inherent limitation of the method and could readily be overcome by enhancements to the initial translation module IGESTRAN.

(b) Some limited drawing inaccuracy can be tolerated - the boundary segment joining module LINEJOIN can recognize lines as contiguous if their end-points lie within a specified tolerance, and the hierarchy builder SKELETON similarly allows line segments to be recognized as collinear or concyclic within specified tolerances. The default tolerances used (0.01 drawing units with LINEJOIN, 0.01% of the outer boundary perimeter with SKELETON) are more than ample to cope with any possible inaccuracies in a professionally-produced drawing. However, they cannot be relaxed indefinitely without producing spurious effects, and the method may fail on drawings produced by untrained draughtsmen.

(c) The hierarchy builder SKELETON fails to yield satisfactory results with some types of highly-recursive shape such as that shown in Fig 3.37, where some parts of the boundary generate four or more levels of description, and others only one or two. A valid hierarchy is built, but cannot be traversed at all levels by the simple algorithm outlined in section 3.2.3.

(d) As discussed in sections 2.3 and 3.2 above, the canonicalization module CANONGEN cannot generate a unique shape representation which is completely robust to an arbitrary small change in shape parameters in all cases.

While such limitations may restrict the system's scope, it is not considered that they invalidate its overall approach in any way. The ultimate test of the usefulness of these shape representation techniques is the retrieval effectiveness of the entire SAFARI system, which can be empirically measured - a subject developed in more detail in chapter 8.

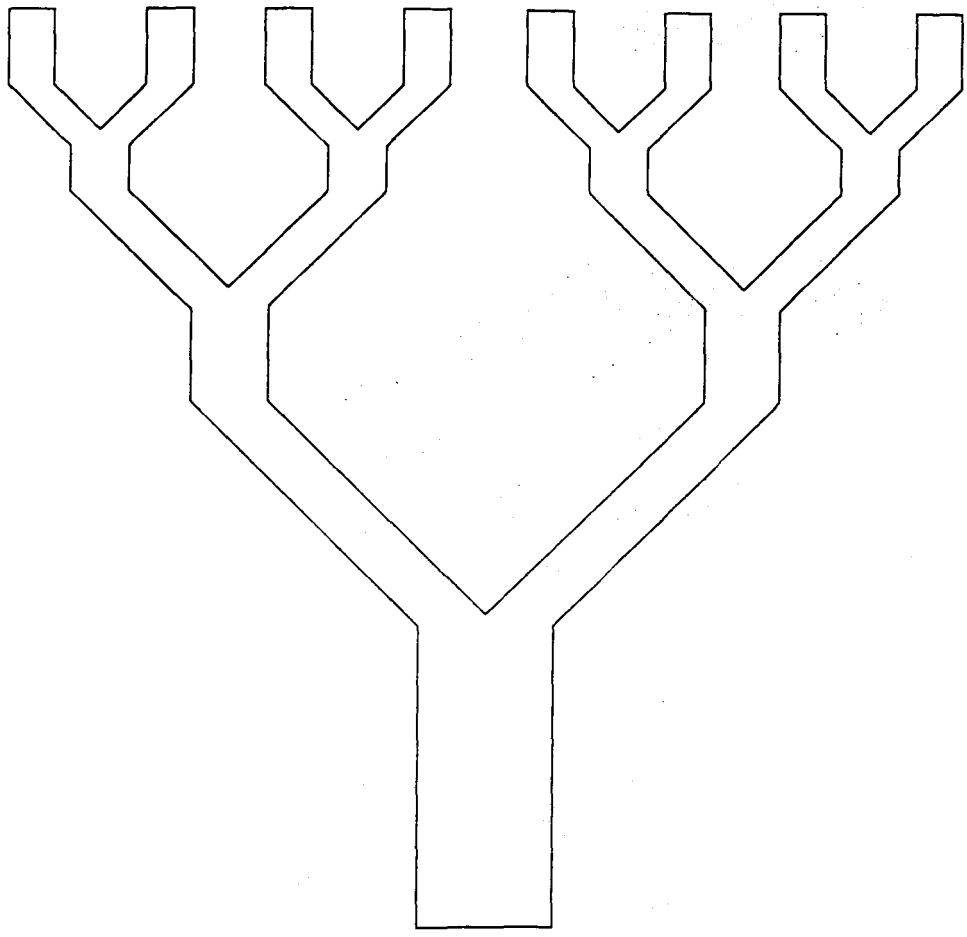


Fig 3.37 An example of the type of highly-recursive shape where the hierarchy generated by SKELETON cannot reliably be traversed with the standard algorithm used by later program modules.

CHAPTER 4. RETRIEVAL FEATURES

4.1 Introduction

To provide an engineering database with the capability of retrieving objects on the basis of shape, it is clearly necessary to identify a set of shape features whose presence or absence in stored drawings can be used as a basis for retrieval. Selection of such a feature set is one of the most difficult tasks in the design of any retrieval system, and is doubly difficult here because of the lack of any body of past queries to provide possible clues. The overall aim of the process can easily be formulated - to define a set of features which allows optimum system performance, in terms of ability to provide relevant answers to all possible user queries to any given collection of stored data. This aim is unfortunately of limited usefulness, as it requires a detailed analysis both of the structure of every collection of data to be housed, and of the queries to be put to the database. Since this is impossible to establish in advance, one has in practice to be content with the lesser aim of identifying features of use in answering a finite set of queries put to a finite set of stored data. This does however raise important questions about the degree to which it is possible to generalize from the results of a study such as this - see Chapter 8, below.

In one sense, the procedures outlined in chapter 3 have already yielded a useful set of retrieval features - the boundary segments themselves. Each of these represents part of the total structure in a form invariant to translation or rotation, and invariant to scaling if a suitable reference length (such as boundary perimeter) is available. Some retrieval capability is possible even if no further feature extraction is performed: when objects are represented in canonical form as an ordered list of segments, the use of suitable matching algorithms allows the user to search for objects which exactly match query objects in whole or in part. As the following section's discussion will show, however, users' retrieval needs are likely to be much wider than this. There needs, for example, to be a facility to recognize a slot whatever its dimensions, and also to identify a part as rotational or non-rotational. Simple segment matching cannot provide this. Some feature extraction method needs to be devised which can extract a range of shape features of use in retrieval from 2-D (and preferably 3-D) object representations, reliably and reasonably efficiently. This problem is unlikely to be solved within the compass of a single project, as it is essentially iterative. Only when a workable shape retrieval system has been developed it is possible to build up and analyze a representative body of queries from which future shape features can be derived.

4.2 Possible sources of shape features

4.2.1 Manual parts classification codes

Parts classification codes such as the Opitz code are at present the only practical means of providing shape retrieval for engineering drawings, and as such deserve close study as indicators of the types of shape feature that are considered important by design engineers and process planners. The Opitz code, for example, characterizes parts principally by overall shape, classing them either as rotational (derived from a basic cylindrical shape) or non-rotational (derived from a cuboid). It then subdivides rotational parts into three groups on the basis of length/diameter ratios, and non-rotational parts on the basis of length/width/depth ratios, with additional classes for shapes that are neither cylindrical nor cuboidal in origin. Whatever the overall shape, the code also indicates the presence of detailed shape features, including major internal machined features such as principal bores, internal grooves or screwthreads, and external machined features such as external grooves and slots.

One problem from the point of view of automatic shape analysis is that the Opitz code clearly distinguishes between functional and auxiliary machined features - the former

being an essential determinant of the component's fitness for its intended purpose. While an experienced engineer would have little difficulty in distinguishing between the two types of feature, he would in practice be using (implied) information about a part's function as well as its shape to assign its classification code. Such information would not be readily available to an automatic shape analysis program, suggesting that the task of emulating the manual classification process could prove difficult.

While alternative coding systems such as Brisch or MICLASS differ in some important respects from Opitz, their selection of shape features for classification is remarkably similar. The implication for the present study would thus seem to be that it is important both to characterize an object's overall shape (rotational or non-rotational, plus ratios of major dimensions), and to specify the nature and shape of machined features such as grooves, slots or auxiliary holes. Because of the difficulty of distinguishing between functional and non-functional shape features, there would seem to be little advantage in trying to incorporate any of the more detailed aspects of such classification systems.

4.2.2 Feature names

A study of the names that engineers use to identify machined features could in principle identify additional retrieval features. As discussed in Section 1.5 above, language can be a useful pointer to the way engineers think about design and manufacture. Although few such terms have rigid definitions, and (as observed above) there is little control over synonyms, such terms might provide an additional indication of features considered important for retrieval. Unfortunately, glossaries of such terms are not readily available - a problem already encountered by Patel (1985), who devised his own list of terms (shaft, flange, bush, slot, groove, etc), most of which could be associated with particular manufacturing processes. This approach was therefore not considered sufficiently fruitful to be worth pursuing further in the present project.

4.2.3 Automatic pattern recognition

Industrial machine vision systems, which aim to recognize components on conveyor belts or in storage bins, have many parallels with the present project. They need to be able to identify the type and possibly orientation of all components present in a given digitized 2-D image, even where some of the components are partially occluded by other objects. This is normally achieved by extracting suitable features from each object detected in the image, and comparing extracted feature values with reference values for each component, a process very similar to shape retrieval. Such systems are therefore a potentially valuable source of possible features.

Chin and Dyer (1986), reviewing the whole field of pattern recognition for robot vision, distinguish three types of feature:

1. **Global** features, characteristic of the shape as a whole, such as area, perimeter or moments of inertia. These can typically be used to generate a feature vector which can readily be matched with reference values for known objects to generate overall similarity measures - or to classify the object using statistical pattern recognition techniques (Duda and Hart, 1973). Their advantages include simplicity (both feature extraction and matching are rapid and straightforward processes in most cases) and invariance to scaling, translation and rotation. Their disadvantages are susceptibility to noise (not a problem in the present context) and inability to handle partial structures or occluded images.

2. **Local** (or **structural**) features, characteristic of individual parts of the shape (normally its boundary), such as individual lines, arcs or corners. These are normally used to form an ordered list (such as a sequence of alternating line segments and corners) which can then be matched with reference objects - typically using syntactic pattern recognition techniques (Fu, 1974). Chin and Dyer claim that both local feature

extraction and matching are computationally more expensive than global feature matching, since these normally involve some kind of parsing, a process that inevitably requires a certain amount of back-tracking. They also point out that not all local features are invariant to translation, rotation, and scaling, thus limiting their usefulness still more. They are however usable with partially occluded images.

3. **Relational** features, characterizing the relative size, position or orientation of groups of local features. These features are normally used to generate graphs relating key local features of the image to each other; again, these graphs can be matched with comparable graphs for reference objects. Relational features are thus the most computationally expensive of all to use, though again they can be used with partially occluded images.

These different kinds of feature can be combined. For example, Yachida and Tsuji (1977) describe a hierarchical system based on initial matching on global features (area, area/perimeter^2 ratio), followed by similarity matching between object and reference boundaries (expressed as polar coordinate values) and searches for circles, lines, and small holes at specified locations. Stockman et al (1982) used boundary edge elements (*real* edges) and vectors linking the centres of internal holes (*abstract* edges) for similarity matching. Umetani and Taguchi (1982) defined a wide variety of what they described as *global*, *local* and *concavity* properties (though their definitions of these differ considerably from those of Chin and Dyer) in experiments on random shape discrimination. Their global properties included various vertex angle, symmetry, complexity, and compactness measures; local properties comprised straightness and sharpness; and concavity properties included the number, length, depth and size of concave features. Some of their feature definitions are unfortunately not very clear.

These studies are obviously of value as sources of specific feature types. Many, if not all, of these features could be directly useful in shape retrieval. The major difficulty could in fact lie in choosing between the wide variety of available features. Possibly of greater value is the concept of tripartite division of features into global, local and relational, with each type of feature useful in a different context. One can immediately see how global features could be of most use in matching complete shape queries, and local features for partial shapes, with relational features of possible use in matching inner boundary patterns.

4.2.4 Human visual perception

Another possible source of retrieval features could be studies of the psychology of vision, though this concentrates on how humans see pictures or images, rather than on the objects themselves. The insights of the Gestalt school of psychology suggest that groups of drawing elements can be identified on the basis of their proximity, similarity of size and shape, continuity of line and *closure* - formation of complete or nearly complete geometric patterns (Zakia, 1975). Studies of how architects visualize drawings (Akin, 1978) have also suggested that, when asked to memorize drawings for later recall, they identified *chunks* of lines associated through geometry (adjacent or parallel lines often formed chunks), or similarity of function (load-bearing walls, external doors). The experiments of Fischler and Bolles (1986) on the way human subjects partitioned line drawings for later recognition or reconstruction appear to confirm that the process is highly subjective, and not based purely on the geometry of the drawing. Recognition of a drawing appears to depend on cues based on a combination of local shape features (individual sides and angles) and global features such as symmetry, repeated groups, and parallel segments. This again implies that recognition and extraction of features of use in retrieval may require situational knowledge not present in the drawing itself, reinforcing the message from section 4.2.1 above.

4.2.5 Information theory

A further factor in the choice of shape features comes from information theory. From Shannon's theory, the information conveyed by a message symbol *i* is

$$- p_i \log_2(p_i)$$

where p_i is the probability of occurrence of the symbol *i* in the message (all symbols are assumed to have independent probability distributions). This peaks at $p_i = 0.5$, implying that retrieval features have maximal information content if they are all independent and occur in 50% of the file - an ideal recognized for many years by designers of retrieval systems. This principle has been used to guide the selection of indexing fragments in both chemical (Adamson et al, 1973) and text (Lynch, 1977) retrieval systems. In both cases, naturally-occurring fragments (atoms or text words) were shown to have highly skewed frequency distributions, and the studies aimed to improve retrieval performance by finding sets of indexing fragments with much flatter frequency distributions. The problem with applying this approach to the present project is that representative collections of data are needed to calculate meaningful frequency distributions. Such 'standard' collections of drawings simply do not exist; the few surveys that have been carried out on shape feature distribution (e.g. Pratt, 1984) have been very selective. As implied in section 4.1 above, there are considerable difficulties in generalizing from results obtained using artificial test collections.

4.3 Criteria for feature selection

It is now possible to propose a reasonable list of criteria which a set of features for shape retrieval should meet, as follows:

1. The features should cover as wide a range of types as possible. In particular, the feature set should include examples of all three types distinguished by Chin and Dyer.

The range of queries to be handled by a system cannot easily be predicted in advance. It is, however, reasonable to assume that queries will include both complete and partial shape matching, and that they may well specify relative positions of key shape features such as inner boundaries. There is of course a risk of degrading performance if too large a feature set is specified.

2. Extracted features should be invariant to translation, rotation and scaling - and independent of choice of boundary start segment.

This should be obvious from the preceding discussion.

3. Features should be reasonably easy and economical to extract, and tolerably compact to store.

While machine efficiency is not a major criterion in a shape retrieval system, and feature extraction would normally be performed only once for each new shape added to the database, space considerations could be relevant in a large database. Rapid feature extraction from query shapes might be an important factor in ensuring user acceptability - it is doubtful whether any user would be prepared to wait for more than a minute or so while a query shape was being processed. Rapid feature extraction would become crucial for any feature which was generated at run time rather than stored in the database. There is of course an implied constraint that all features must be directly derivable from the representation chosen for each stored shape.

4. If a system is directed at a specific domain of shapes (such as engineering drawings), the feature set should take this into consideration.

For example, the features to be stored in a database of engineering parts should include each part's overall shape class and an indication of machined features.

4.4 Features chosen for prototype system

4.4.1 Introduction

Three main categories of features were identified, though Chin and Dyer's categorization was adapted slightly to emphasize the distinction between inner and outer shape boundaries:

1. **Global** boundary features (corresponding almost exactly to Chin and Dyer's *global* features, but including both Umetani and Taguchi's *global* and *concavity* features), reflecting the overall shape of a given boundary;

2. **Local** boundary features (including Chin and Dyer's *local* and some *relational* features), computed either from individual boundary segments or short sequences of contiguous boundary segments, reflecting specific features within a given boundary;

3. **Positional** features (mostly a subset of Chin and Dyer's *relational* features), specifically representing the number, type and pattern of inner boundaries within a given shape.

As discussed in section 3.2.4 above, it can be hypothesized that the chances of matching query and stored shapes are likely to be greatest if features are calculated and stored separately for each level of traversal of each boundary - in effect treating each boundary level as a separate boundary in its own right. This effectively allows the system to select the closest-matching views of both query and stored boundaries for similarity estimation. This approach was thus adopted for SAFARI; although a few parameters (such as boundary class or length/width ratio) were considered to be characteristic of the boundary as a whole, however it was viewed, the vast majority (such as mean segment length or arc angle variance) were associated with a specific boundary level.

4.4.2 Global boundary features chosen

The features below were therefore computed and stored for each level of each boundary. The rationale for selecting these features comes largely from the work of Umetani and Taguchi, and to a lesser extent the other authors cited above. They represent parameters analogous to those used by Umetani and Taguchi, modified to take account of the fact that the shapes used in the present study are made up of circular arcs as well as straight lines. The restricted measures of symmetry were necessary because curved objects can have an infinite degree of symmetry.

1. Mean segment length $ML = \Sigma (L_i)/n$

   where L_i is the (normalized) length of boundary segment i, and n the number of segments in the current boundary level.

2. Segment length variance $LV = \Sigma (L_i\text{-}ML)^2/n$

3. Mean segment arc angle $MA = \Sigma (A_i)/n$

   where A_i is the arc angle of boundary segment i.

4. Segment arc angle variance $AV = \Sigma (A_i\text{-}MA)^2/n$

5. Mean discontinuity angle between segments $MD = \Sigma (D_i)/n$

where D_i is the discontinuity angle between segments i and i+1 ($i < n$), and between segments i and 1 ($i = n$).

6. Discontinuity angle variance $DV = \Sigma (D_i\text{-}MD)^2/n$

7. Concavity index $CI = (\Sigma (A_i : A_i < 0)/(2\pi) + \Sigma (L_i : A_i < 0)/ \Sigma (L_i)) / 2$

a measure of the overall concavity of the shape, obtained by (separately) summing the arcangle and length of each concave arc, normalizing each sum by dividing by the total arcangle and perimeter of the shape boundary, and then taking the mean.

8. Degree of rotational symmetry *RS*

a lower bound for the number of axes of rotational symmetry, calculated as indicated in section 4.5.3 below. The method of calculation allows no more than one attempt at calculating symmetry per segment; hence $RS <= n$.

9. Degree of planar symmetry *PS*

a lower bound for the number of planes of axial (mirror-image) symmetry, calculated in a similar way to *RS*.

In addition, the following features, considered characteristic of the boundary as an entity rather than of any individual level, were computed and stored once for each boundary:

10. Boundary arc/line ratio $AL = \Sigma (L_i : A_i <> 0)/ \Sigma (L_i)$

the ratio of curved segment length to total boundary length, computed for the boundary at the highest possible level of traversal to minimize the effect of minor shape features such as chamfers and fillets.

11. Boundary length/width ratio *LW*

calculated as indicated in section 4.5.2 below, again using boundary traversal at the highest possible level to minimize the effect of minor shape features.

12. Boundary perimeter^2/area ratio $PA = \Sigma (L_i)/$(Boundary area)

calculated using boundary traversal at the lowest possible level, to preserve the value of this parameter as a measure of a shape's overall thickness.

13. No of rotational axes *NR*

calculated as the number of distinct centres of convex arcs in the top level of the boundary.

14. Boundary shape class *SC*

calculated as indicated below (section 4.5.2); can take the values *Rectangular*, *Other right-angled*, *Other straight-edged*, *Irregular*, *Circular* or *Multi-curved*. Examples of boundaries from each of these classes are shown in Fig 4.1.

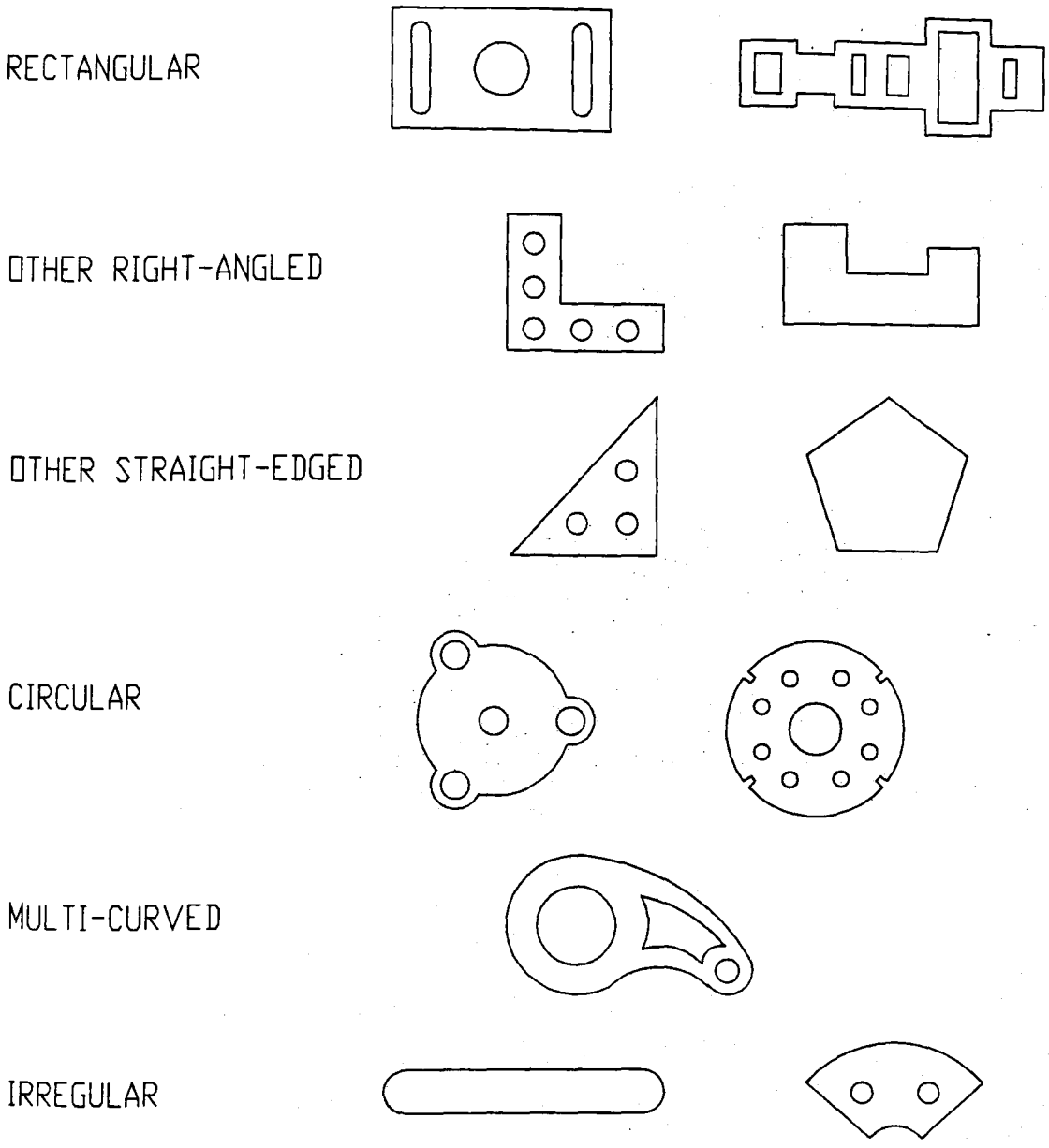


Fig 4.1 Grouping of shapes into families on the basis of top-level outer boundary segment traversal. Classification is performed firstly on the basis of straight-line to circular arc length, and subsequently on the presence of angles other than right angles for "straight-line" shapes and the number of centres of rotation for curved shapes. The resulting classification mirrors that of many manual parts classification schemes.

4.4.3 Local boundary features

Although individual segment parameters were not used directly for feature matching, they are listed here for the sake of completeness. They can be regarded as specifically local features, even though segment length needs to be normalized by dividing by a global feature, total boundary length, to render it invariant. It should be noted that the global and local features listed above and later in this section have all been derived from these four parameters.

1. Segment length L_i

   the length of segment *i*, normalized by dividing by the outer boundary perimeter.

2. Segment arc angle A_i

   the arc angle subtended by segment *i*.

3. Discontinuity angle between segments D_i

   the angle between segments *i* and *i*+1 ($i < n$), and between segments *i* and 1 ($i = n$).

4. Segment parent feature P_i

   the type of shape feature (if any) to which segment *i* belongs, computed as outlined below; can take the values *Protrusion*, *Depression*, *Corner* or *Absent*.

Two kinds of derived local feature were extracted for each level of each boundary - the first making use of data from individual boundary segments, the second from sequences of connected segments. A range of feature types of differing complexity was deliberately chosen to investigate whether increasing fragment complexity could be associated in any way with increased retrieval performance. Analyses of the distribution of each of these feature types in the test database suggested that the more complex feature types did exhibit a more even distribution than the simple feature types - though it would be dangerous to conclude too much from this (see section 4.2.5 above). In each case, the feature distribution is characterized by storing counts of the frequency of occurrence of each feature value present. The advantage of generating shape features of this kind lies in their flexibility. Unlike the local features described by Chin and Dyer, they do not have to be processed by parsing algorithms. They can be used to generate feature vectors for use in similarity matching in the same way as global features - or used as "index terms" in an inverted file to provide rapid retrieval of all shapes containing a given feature. They still retain the advantage of being usable with either complete or incomplete boundaries, though segment length distribution again has somewhat limited validity in the latter case. These local features are listed below.

5. Segment length distribution *LD*

   a vector indicating the number of boundary segments (if any) for which relative length $L_r = \log_2 (L_i / ML)$ falls into each of the following ranges:

$L_r < -3.0$	$-0.9 <= L_r < -0.7$	$0.7 <= L_r < 0.9$
$-3.0 <= L_r < -2.5$	$-0.7 <= Lr < -0.5$	$0.9 <= L_r < 1.1$
$-2.5 <= L_r < -2.0$	$-0.5 <= Lr < -0.3$	$1.1 <= L_r < 1.3$
$-2.0 <= L_r < -1.5$	$-0.3 <= L_r < -0.1$	$1.3 <= L_r < 1.5$
$-1.5 <= L_r < -1.3$	$-0.1 <= L_r < +0.1$	$1.5 <= L_r < 2.0$
$-1.3 <= L_r < -1.1$	$+0.1 <= L_r < +0.3$	$2.0 <= L_r < 2.5$
$-1.1 <= L_r < -0.9$	$+0.3 <= L_r < +0.5$	$2.5 <= L_r$
	$+0.5 <= L_r < +0.7$	

The length ranges were chosen to give a reasonably large number of categories, and also as even a distribution of segment lengths between categories as possible.

6. Arc angle distribution *AD*

a vector indicating the number of boundary segments (if any) for which arc angle A_i falls into each of the following categories:

$$\begin{array}{rcl} & A_i < -\pi & \\ & A_i = -\pi \pm 0.01 & \\ -\pi < & A_i < -\pi/2 & \\ & A_i = -\pi/2 \pm 0.01 & \\ -\pi/2 < & A_i < 0 & \end{array} \qquad \begin{array}{rcl} & A_i = 0 \pm 0.01 & \\ 0 < & A_i < \pi/2 & \\ & A_i = \pi/2 \pm 0.01 & \\ \pi/2 < & A_i < \pi & \\ & A_i = \pi \pm 0.01 & \end{array}$$

$$\pi < A_i$$

The angle ranges were chosen to reflect the fact that most machined parts have edges that are straight lines (zero arc angle), or form complete circles (arc angle 2π), semicircles (arc angle π) or quadrants (arc angle $\pi/2$).

7. Discontinuity angle distribution *DD*

a vector indicating the number of boundary segments (if any) for which discontinuity angle D_i falls into each of the following categories:

$$\begin{array}{rcl} & D_i < -\pi/2 & \\ & D_i = -\pi/2 \pm 0.01 & \\ -\pi/2 < & D_i < 0 & \end{array} \qquad \begin{array}{rcl} & D_i = 0 \pm 0.01 & \\ 0 < & D_i < \pi/2 & \\ & D_i = \pi/2 \pm 0.01 & \end{array}$$

$$\pi/2 < D_i$$

The angle ranges were chosen to reflect the fact that the overwhelming majority of machined parts have edges that meet at right angles or zero.

8. Parent feature distribution *FD*

a vector indicating the number of instances (if any) of each type of parent feature (*Protrusion*, *Depression* or *Corner*) in the given boundary level. This type of feature (relatively simple to extract as a by-product of the process of building up a hierarchical description of the shape) was included to give some indication of the presence of machined features.

9. Segment length/arcangle distribution *SL*

a frequency count of fragments representing both segment length and arc angle type; each fragment represents one of the segment length ranges enumerated under (5) above, and one of the arc angle categories enumerated under (6) above. The entire set of such fragments can be regarded as a two-dimensional feature vector, and could in theory categorize each line segment much more accurately than the use of separate length and arc angle distribution counts.

10. Arc angle triplet *AT*

a frequency count of fragments based on the properties of each line segment and its preceding and succeeding vertices; each fragment represents the categories (as defined under (6) and (7) above) into which the arc angle A_i, the preceding discontinuity angle D_{i-1} and the succeeding discontinuity angle D_i respectively fall. The complete fragment set can be regarded as a three-dimensional feature vector. The potential advantage of this and the following type of feature is the

ability to characterize a shape using the relationship between one segment or vertex and the next - which might be important in the light of the studies discussed in section 4.2.4. This feature could also have relevance as an indicator of an object's angular shape, of potential importance in searching for classes of object which match on angles but not linear dimensions. Examples of these triplet features are shown in Fig 4.2.

11. Discontinuity angle triplet *DT*

a frequency count based on the properties of each vertex and its preceding and succeeding line segments; each fragment represents the categories into which the discontinuity angle D_i, the line classes of the preceding and succeeding line segment pair L_i and L_{i+1}, and the length ratio $R_1 = \log_2(L_i/L_{i+1})$ respectively fall. In this case, the discontinuity angle categories are as defined in (7) above, the line class categories are

LL	(both segments straight lines)
LA	(segment *i* a straight line, segment *i*+1 a circular arc)
AL	(segment *i* a circular arc, segment *i*+1 a straight line)
AA	(both segments circular arcs)

and the length ratio categories are

$R_1 < -3.5$	$-1.5 <= R_1 < -0.5$	$1.5 <= R_1 < 2.5$
$-3.5 <= R_1 < -2.5$	$-0.5 <= R_1 < +0.5$	$2.5 <= R_1 < 3.5$
$-2.5 <= R_1 < -1.5$	$+0.5 <= R_1 < +1.5$	$3.5 <= R_1$

Again, the complete fragment set can be regarded as a three-dimensional feature vector. This feature, emphasizing the relative length and type of adjacent segments, was intended to complement the angle-based feature described in (10) above. Examples of these triplet features are shown in Fig 4.2.

12. Parent feature composition *PF*

a frequency count of fragments based on the properties of each type of parent feature and its composition; each fragment represents the overall feature type (as defined under (8) above), together with the numbers of straight-line and circular arc segments making up the feature, thus providing a measure of the feature's complexity. Again, the complete fragment set can be regarded as a three-dimensional feature vector. Examples of these features are shown in Fig 4.2.

4.4.4 Inner boundary position features

The final kind of feature aimed to provide an indication of the type, number, and relative position of inner boundaries. Enumeration and classification of these boundaries can yield useful features, but the question of how to recognize and represent patterns of holes must also be addressed. Ample evidence exists, both from informal conversations and from examination of the structure of parts classification systems, that this is an important area.

The problem was tackled by grouping all inner boundaries into specified families on the basis of shape, size and proximity - criteria suggested by Zakia (1975) as having deep roots in the way humans perceive objects. The spatial distribution of each family of inner boundaries could then be examined for the presence of one or more specified regular patterns, which could then be used as features characterizing the entire set of inner boundaries. The feature set used was as follows:

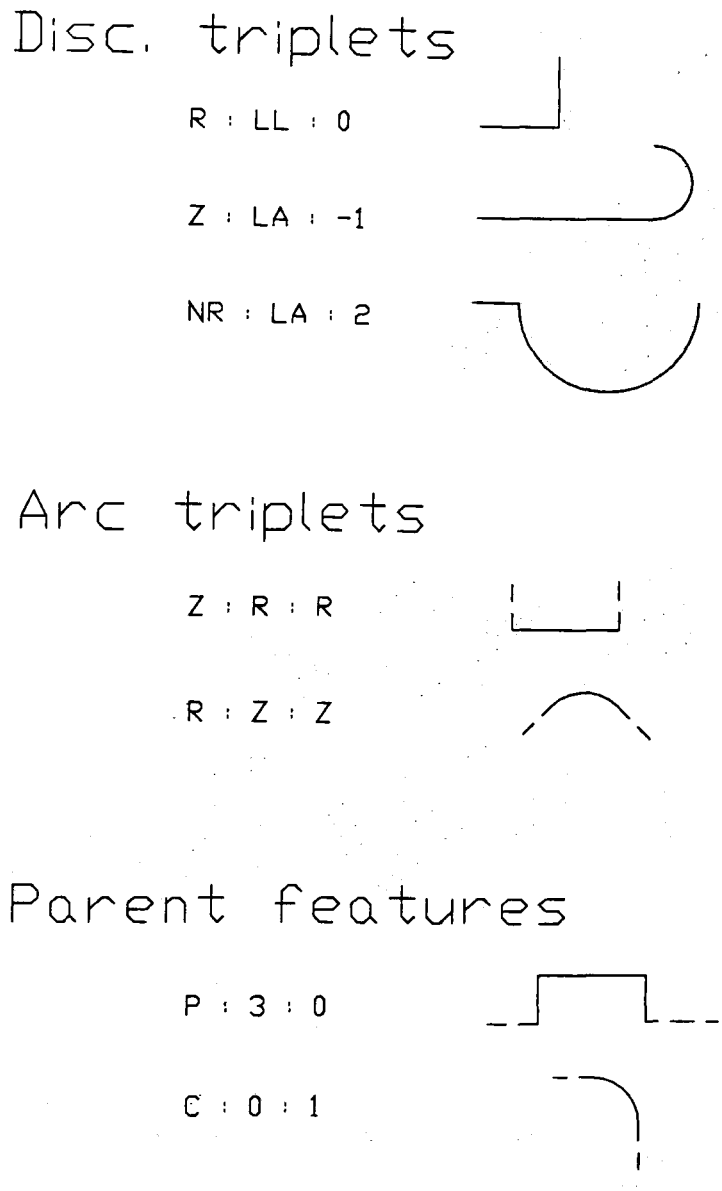


Fig 4.2 Illustrations of some local boundary features. Discontinuity angle triplet *features indicate the environment of each boundary angle, showing successively the angle type, the types of line enclosing the angle, and the relative lengths of these two lines.* Arc angle triplet *features illustrate the angular environment of each line (arc angle class plus discontinuity angle with each adjoining line).* Parent feature composition *fragments indicate the presence of local shape features (protrusions, depressions, corner features, etc), together with their composition in terms of numbers of straight lines and circular arcs.*

1. Number of inner boundaries *NB*

   a simple count of inner boundaries in the drawing

2. No of distinct inner boundary families *BF*

   a count of distinct inner boundary families identified

3. No of curved inner boundaries *CB*

   number of inner boundaries falling into shape classes *Circular* or *Multi-curved*

4. No of straight inner boundaries *SB*

   number of inner boundaries falling into shape classes *Rectangular*, *Other right-angled* or *Other straight-edged*

5. No of irregular inner boundaries *IB*

   number of inner boundaries in shape class *Irregular*

6. Boundary family characteristics *BC*

   a frequency count of each boundary family, characterized by shape class, number of segments in top level of boundary, and mean length-width ratio and perimeter for all boundaries included in class.

7. Boundary pattern features *BP*

   a frequency count of each type of pattern feature identified by examining relative positions of inner boundary centroids both within each inner boundary family and within the drawing as a whole. The rationale for feature selection was the provision of a wide variety of pattern types, guided both by studies of human perception and examination both of parts classification schemes and actual examples of machined parts. The feature types recognized by the prototype system include the following patterns based on line continuity (see examples in Fig 4.3):

   Collinear boundaries - maximum number of inner boundary centroids lying on a straight line
   Concyclic boundaries - maximum number of inner boundary centroids lying on a circular arc

   the following patterns based on *closure* (formation of regular or other recognizable polygonal shapes - the set of shapes included most common triangular and quadrilateral combinations recognizable as regular shapes, plus a limited selection of many-sided polygons):

   Equilateral triangles - number of distinct equilateral triangles formed by inner boundary centroids
   Right-angled triangles - as above, for right-angled triangles
   Isosceles triangles - etc.
   Squares
   Rectangles
   Rhombi
   Parallelograms
   Trapezoids

Regular pentagons
Regular hexagons
Regular octagons
Other regular polygons

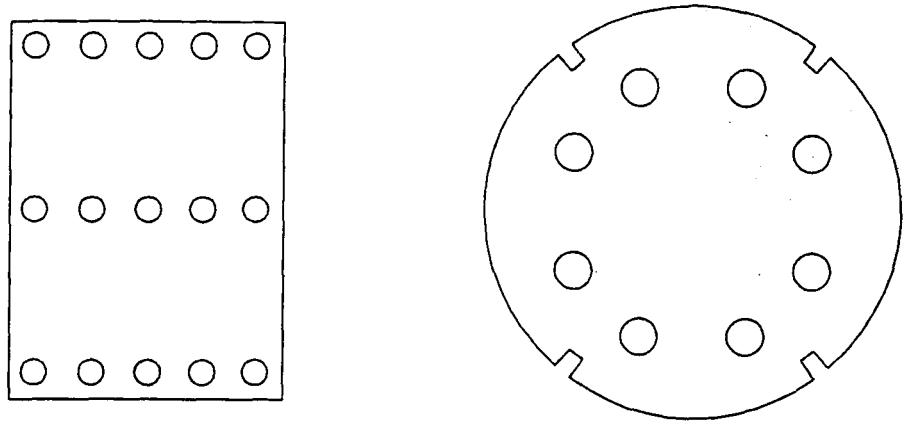
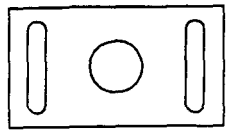


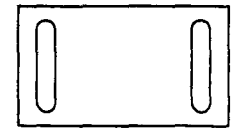
Fig 4.3 Examples of inner boundary pattern features based on continuity. In the first shape, three groups of boundaries can be recognized as forming a pattern through collinearity; in the second, a single group of inner boundaries can be recognized because they all lie on the same circular arc. Note that boundaries also need to be similar in size and shape in order to belong to the same family.

and the following patterns (see examples in Figs 4.4 and 4.5) based on symmetry (arrangement of inner boundaries in patterns around the overall shape centroid, an important consideration in the machining of rotational parts in particular):

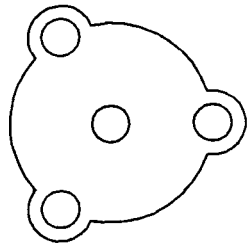
Boundary at centroid - number of inner boundaries (0 or 1) with centroid coincident with outer boundary centroid
Boundaries round centroid - maximum number of inner boundaries on arc centred on outer boundary centroid
I-stars - pairs of inner boundaries equidistant from and collinear with outer boundary centroid
V-stars - pairs of inner boundaries equidistant from outer boundary centroid and making an angle of 120^{o}
L-stars - pairs of inner boundaries equidistant from outer boundary centroid and making an angle of 90^{o}
T-stars - sets of three inner boundaries equidistant from outer boundary centroid, making one angle of 180^{o} and two of 90^{o}
E-stars - sets of three inner boundaries equidistant from outer boundary centroid, making three angles of 120^{o}),
Y-stars - sets of three inner boundaries equidistant from outer boundary centroid, making one angle less than 180^{o} (but not equal to 120^{o} within the specified tolerance), and two equal angles greater than 90^{o}



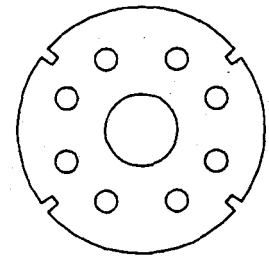
Boundary at centroid



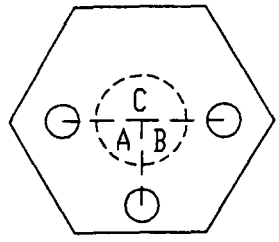
No boundary at centroid



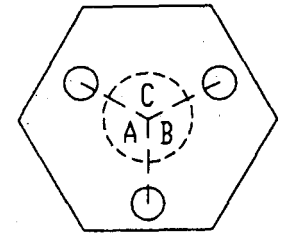
Boundaries round centroid = 3



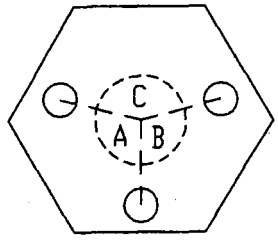
Boundaries round centroid = 8



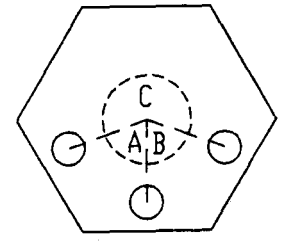
T-star (A=B=90°; C=180°)



E-star (A=B=C=120°)

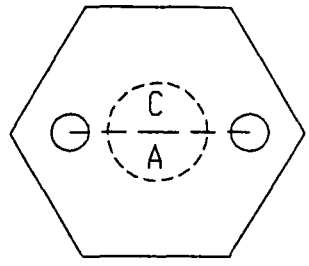


Y-star (A=B>90°; C<180°)

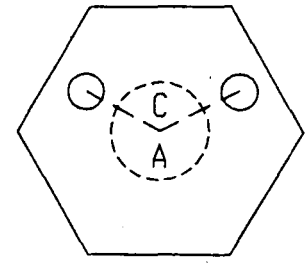


M-star (A=B<90°; C>180°)

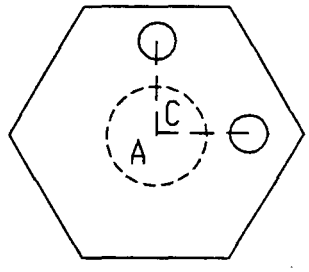
Fig 4.4 Examples of inner boundary pattern features based on symmetry about the outer boundary centroid. Boundary at centroid *indicates the presence of an inner boundary centred on the outer boundary centroid;* boundaries round centroid *indicates the maximum number of boundaries in any circular arc centred on the outer boundary centroid;* T-, E-, Y-, *and* M-stars *indicate the presence of three concyclic inner boundaries arranged around the outer boundary centroid in the specific configurations illustrated.*



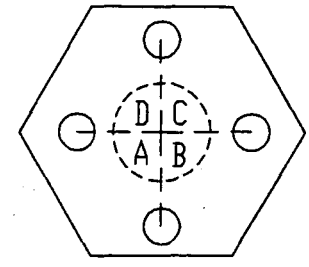
I-star (A=C=180°)



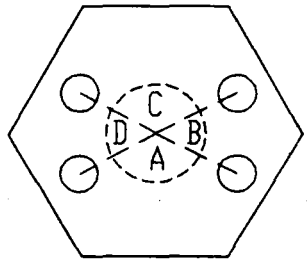
V-star (A>180°, C<180°)



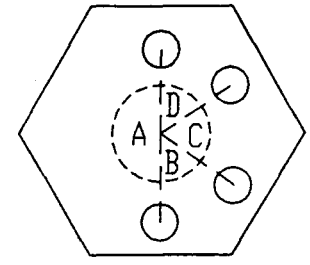
L-star (A=270°, C=90°)



R-star (A=B=C=D=90°)



X-star(A=C; B=D)



K-star (A=180°; B=D)

Fig 4.5 Further examples of inner boundary pattern features based on symmetry about the outer boundary centroid. I-, V-, *and* L-stars *indicate the presence of two concyclic inner boundaries arranged around the outer boundary centroid in the specific configurations illustrated;* R-, X-, *and* K-stars *indicate similar combinations of four concyclic inner boundaries.*

M-stars - sets of three inner boundaries equidistant from outer boundary centroid, making one angle greater than 180^{o} and two equal angles less than 90^{o}

R-stars - sets of four inner boundaries equidistant from outer boundary centroid, making four angles of 90^{o}

X-stars - sets of four inner boundaries equidistant from outer boundary centroid, making two opposite equal angles greater than 90^{o} and two equal angles less than 90^{o}

K-stars - sets of four inner boundaries equidistant from outer boundary centroid, making one angle of 180^{o} and two opposite equal angles less than 90^{o}; the magnitude of the fourth angle is immaterial.

4.5 Method of feature extraction

4.5.1 Introduction

The bulk of feature extraction is performed by program DATALOAD, which takes as input the canonical shape description generated by program CANONGEN (chapter 3) - though the basic local boundary features L_i, A_i, D_i, and P_i are generated at an earlier stage, by program SKELETON. (In the current prototype of SAFARI, some global boundary features, such as *LW* and *PA*, are generated by programs SKELETON or CANONGEN for ease of computation).

The program reads in the canonical shape description one boundary at a time, traversing each boundary at progressively lower levels, and extracting relevant feature values. At present this requires several passes through each set of boundary segments, though there is no inherent reason why all global and local features for a given level could not be extracted in a single pass. Once all boundaries have been processed, the shape's inner boundaries (if any) are grouped into families on the basis of similarity of size, shape and position, and inner boundaries searched for the presence of the position features referred to above.

More detailed descriptions of the derivation of each type of feature are presented below.

4.5.2 Global boundary features

1. Boundary arc/line ratio *AL*, no of rotational axes *NR* and boundary shape class *SC* are calculated in a single top-level traversal of the boundary (presently within program SKELETON), during which counts are accumulated of the number of both positive (R_p) and negative (R_n) right-angled vertices, the total length of all curved segments, and the number of distinct axes of rotation (arc centres not coinciding within a specified tolerance). Total boundary length and *n*, the number of boundary segments, are already avail able. *AL* and *NR* are readily computed when boundary traversal is complete, and *SC* is then assigned a value as follows:

```
If AL < 0.1 then

 {shape is predominantly straight-edged, so classify on basis of
  number of positive and negative right angles}

  If Rp = n then
     SC = Rectangular
  else if (Rp+Rn) = n then
```

```
        SC = Other right-angled
    else
        SC = Other straight-edged

else if AL > 0.9 then

  {shape is predominantly curved, so classify on basis of number of
   axes of rotation}

   If NR = 1 then
        SC = Circular
   else
        SC = Multi-curved
else

    SC = Irregular;
```

2. Boundary length/width ratio *LW* is computed in a single traversal of the top boundary level. All segment coordinates are transformed so that the boundary centroid lies at the origin, and the shape's major axis becomes the x-axis. As the boundary is traversed, tallies are kept of maximum and minimum x and y values. *LW* is then given by Max ($(X_{max} - X_{min})/(Y_{max} - Y_{min})$, $(Y_{max} - Y_{min})/(X_{max} - X_{min})$).

3. Boundary perimeter^2/area ratio *PA* is computed (presently within CANONGEN) in a single traversal of the bottom boundary level, using triangulation to calculate its area as outlined in section 3.3.6.

4.5.3 Global level features

1. Mean segment length *ML*, segment length variance *LV*, mean segment arc angle *MA*, segment arc angle variance *AV*, mean discontinuity angle between segments *MD*, and discontinuity angle variance *DV* are all calculated for each boundary level in a single pass which accumulates totals of each of these parameters and their squares, and then uses conventional formulae to calculate means and variances. Concavity index *CI* is calculated in a similar fashion.

2. Degree of rotational symmetry *RS* and degree of planar symmetry *PS* are calculated together, separately from the other level features. This cannot be achieved in a single pass through each boundary level. Indeed, the construction of efficient algorithms for determining shape symmetry is still a research issue in its own right (e.g. Leou and Tsai, 1987). As noted above, some limitation on the symmetry calculation process is necessary to restrict answers to a finite number. The algorithm used generates approximate symmetry measures by comparing cumulative distance and angle traversed at the end of each line segment when boundary traversal is initiated from two vertices i and j (where i is the canonical boundary start point, and j is successively set to every (other) vertex in the current boundary level), and incrementing symmetry counts if cumulative distance and angle from these two starting-points remain identical (within specified tolerances) the whole way round the boundary. Rotational symmetry is calculated by traversing the boundary from the two starting-points in the same direction, planar (mirror-image) symmetry by traversing in opposite directions. Repeating the process in each direction for all j yields the two symmetry measures *RS* and *PS* for that level. While these measures are in no sense rigourous, they have the benefit of simplicity and consistency, and can be computed reasonably efficiently - an important design criterion. They do in practice yield intuitively sensible results if appropriate tolerances are set.

4.5.4 Local boundary features

1. Segment length distribution *LD*, arc angle distribution *AD*, discontinuity angle distribution *DD*, parent feature distribution *FD*, segment length/arcangle distribution *SL*, arc angle triplet *AT*, discontinuity angle triplet *DT*, and parent feature composition *PF* are again all computed in a single pass through each boundary level. The single-element parameters can all be calculated by incrementing the appropriate element of an accumulator array; the more complex parameters require comparisons with previous values stored in temporary records.

4.5.5 Inner boundary position features

1. Number of inner boundaries *NB*, no of curved inner boundaries *CB*, no of straight inner boundaries *SB*, and no of irregular inner boundaries *IB* can readily be derived from data stored in each boundary header record.

2. No of distinct inner boundary families *BF* and boundary family characteristics *BC* cannot be computed until inner boundaries have been grouped into families. In the interests of economy, a full clustering approach was rejected in favour of a (single-pass) grouping of boundaries on the basis of class, size (perimeter) and shape (length/width ratio), followed by a check on the cohesiveness of any family containing more than two boundaries. This is done by rearranging the boundaries within each family into nearest-neighbour order, starting with the two closest (not a computationally expensive task since very few families contained more than 4 or 5 boundaries), and splitting any families with an inter-boundary distance of more than twice the minimum distance. The process is illustrated in Fig 4.6.

3. Boundary pattern features *BP* are computed by successively applying a series of feature recognizers, firstly to the entire set of inner boundaries, then to each individual inner boundary family in turn.

   Where five or more inner boundaries are present (a rare event), the entire boundary set is first examined to identify whether their centroids form a regular polygon, by invoking procedure *TestForPolygons*. This first sorts all boundaries into order on the basis of the angle of the vector joining their centroids to that of the (arbitrary) first boundary in each family, to ensure that the polygon formed by linking each of the *n* boundaries present to its neighbour contains no intersecting sides (see Fig 4.7). It then tests this polygon for the presence of identical sides and angles. setting the count of *Regular pentagons*, *Regular hexagons*, *Regular octagons* or *Other regular polygons* to 1 if appropriate.

   Where four or more inner boundaries are present, procedure *TestForQuadrilaterals* is invoked to detect and count patterns of inner boundaries in the shape of squares, rectangles, rhombi, parallelograms, or trapezoids. This procedure systematically searches every possible unique combination of four boundaries to establish whether their centroids form a square (all four sides equal, all four angles equal), a rhombus (all four sides equal), a rectangle (all four angles equal), a parallelogram (opposite angles equal), or a trapezoid (adjacent angles equal), incrementing counts of *Squares*, *Rectangles*, *Rhombi*, *Parallelograms* and *Trapezoids* as appropriate. A given quadrilateral is counted only under one heading - e.g four boundaries forming a square are counted just as a square, not as a rhombus or rectangle as well.

   Procedure *TestForConcylicity* is also invoked if four or more inner boundaries are present, to identify the maximum number of boundary centroids lying on any given circular arc. Each unique combination of three boundaries is examined, and the

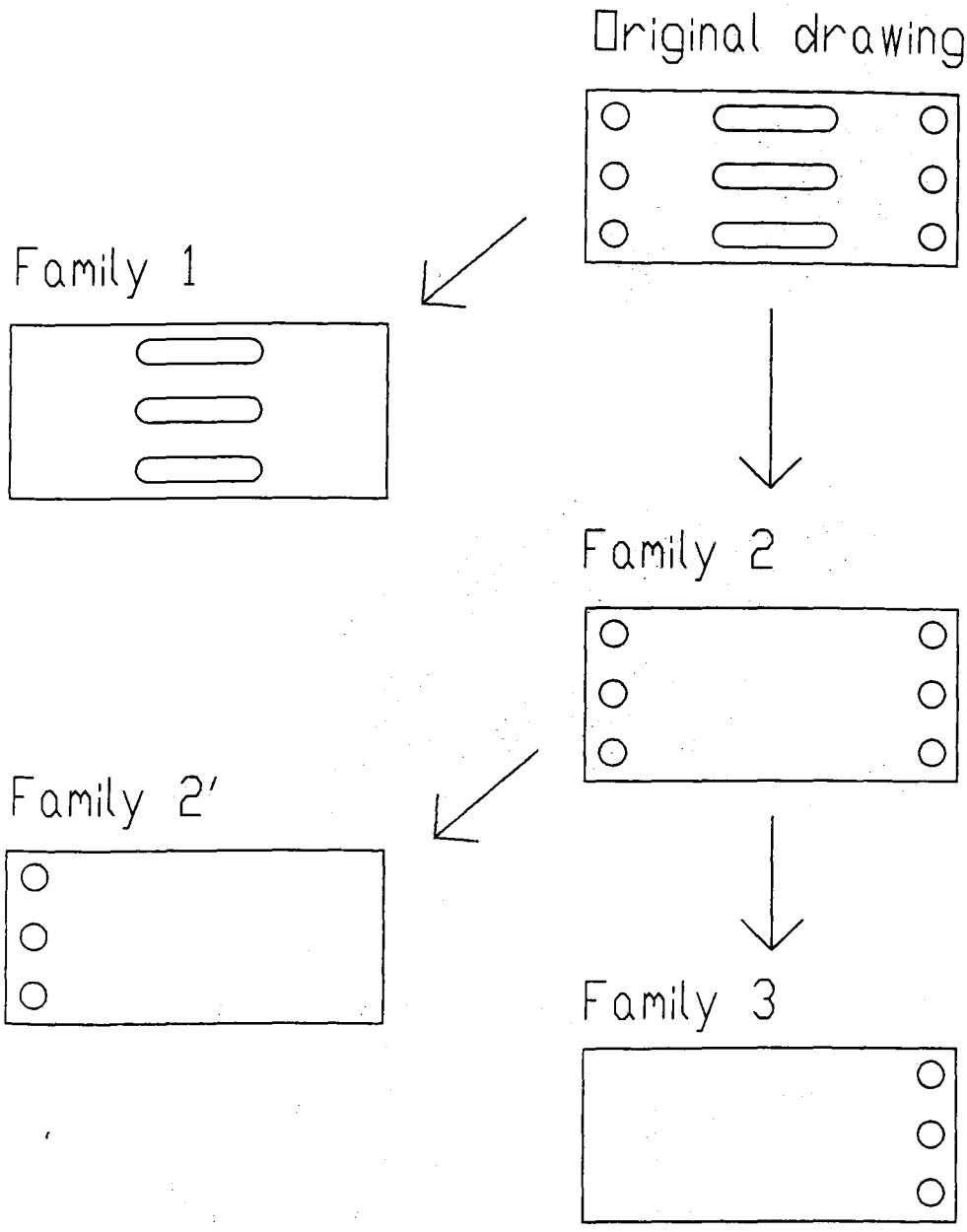


Fig 4.6 The process of generating inner boundary families. In the illustration shown, two separate families are generated on the basis of similarity of size and shape; family (2) is then split into two on the basis of proximity.

position of the centre of the circle (if any) passing though the three boundary centroids established. All remaining boundaries are then examined to establish whether their centroid lies on this circle. If so, a temporary count of concyclic boundaries is incremented, and an attempt made to find further boundaries lying on the same circle. When all boundaries have been examined, the maximum value of this count is stored as *Concyclic boundaries*, provided it exceeds three.

Where three or more inner boundaries are present, procedure *TestForTriangles* is invoked to detect and count patterns of inner boundaries in the shape of equilateral, right-angled or isosceles triangles. This procedure systematically searches every possible unique combination of three boundaries to establish whether their centroids form an equilateral triangle (all three angles equal), an isosceles triangle (any two angles equal), or a right-angled triangle (one angle equal to $\pi/2$), incrementing counts of *Equilateral triangles*, *Right-angled triangles*, and *Isosceles triangles* as appropriate. A given triangle cannot be counted as both equilateral and isosceles, but can be counted as both right-angled and isosceles.

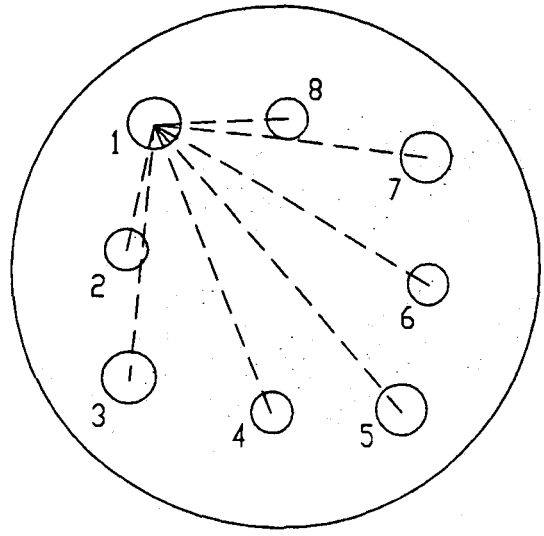


Fig 4.7 Sorting inner boundaries by the direction of the vector linking their centroids with that of the first boundary in the family, in order to prevent the sides of the resultant polygon from intersecting. The procedure produces a valid ordering for all non-degenerate n*-sided polygons, though it yields a consistent and intuitively sensible ordering only for convex polygons. Since the polygon detector rejects all non-convex polygons, this limitation does not affect the validity of the polygon detection process.*

Procedure *TestForCollinearity* is also invoked here to identify the maximum number of boundary centroids lying on any given straight line. Each unique pair of boundaries is examined, and the direction of the line passing though their centroids established. All remaining boundaries are then examined to establish whether their centroid lies on this line. If so, a temporary count of collinear boundaries is incremented, and an attempt made to find further boundaries lying on the same line. When all boundaries

have been examined, the maximum value of this count is stored as *Collinear boundaries*, provided it exceeds two.

Finally, procedure *TestForCentroidFeatures* is invoked provided two or more inner boundaries are present. This categorizes all inner boundaries within a given family according to their distance from the outer boundary centroid - effectively grouping them into concentric rings (Fig 4.8). Each ring containing two or more boundaries is then examined for the presence of *star* features (Figs 4.4 and 4.5). If four or more boundaries are found within a given ring, each possible combination is searched for the presence of *R-stars*, *X-stars*, or *K-stars*, as defined in section 4.4.4 above. If

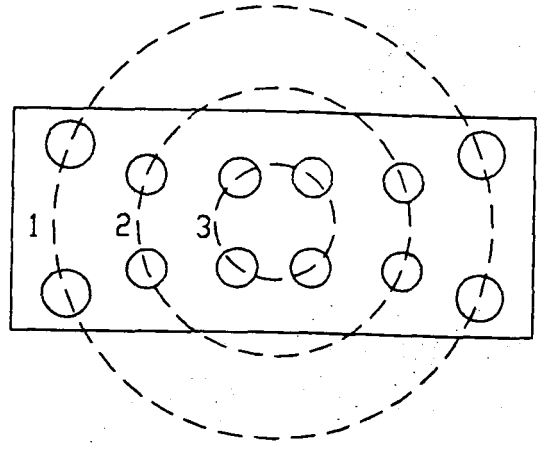


Fig 4.8 Grouping inner boundaries into concentric rings centred on the outer boundary centroid, as a prelude to generating the symmetry features shown in Figs 4.4 and 4.5.

found, counts for these features are incremented. If none of the above features are detected, each combination of three boundaries within a ring is examined for the presence of *E-stars*, *Y-stars*, or *M-stars*, and counts incremented as appropriate. Failing this, each combination of two boundaries is examined for the presence of *I-stars*, *V-stars*, and *L-stars*. Finally, counts are stored of the maximum number of inner boundaries in any ring (if greater than 1), and the number of inner boundaries (0 or 1) coincident with the outer boundary centroid.

4.6 Efficiency considerations

As indicated in section 4.3, the computational efficiency of feature extraction is a significant though not overriding consideration. Extraction of most of the global and local features described above can in fact be achieved in a single pass though the segments forming one boundary level, hence qualifying as an $O(n)$ process, n being the number of segments in a given boundary level. The exception is the symmetry calculation, which requires complete traversal of the current boundary level starting from each vertex in turn, and is hence an $O(n^2)$ process in the worst case.

The only processes giving cause for concern are the inner boundary pattern detectors described in section 4.5.5, though even this is polynomial-bounded. The quadrilateral detector examines every possible combination of four inner boundaries in turn, requiring $n(n\text{-}1)(n\text{-}2)(n\text{-}3)$ comparisons, and hence an overall complexity of $O(n^4)$. Similarly, the triangle and concyclicity detectors examine every possible combination of three inner boundaries, requiring $n(n\text{-}1)(n\text{-}2)$ comparisons and hence a complexity of $O(n^3)$. The polygon detector is less expensive, as it makes only one attempt to match inner boundary patterns to a regular polygonal shape. The situation is in practice manageable because inner boundary pattern detection is carried out only once for each complete family of inner boundaries (and hence seldom more than twice for any given drawing) - as opposed to global and local boundary feature extraction, which have to be performed once for each level of each individual boundary. It is very rare for an inner boundary family to contain more than 5 or 6 individual boundaries, thus setting a manageable upper limit to computation times for most drawings. However, it must be recognized that computation times for complex shapes with 20 or more inner boundaries could become excessive.

In practice, the vast bulk of cpu usage for program DATALOAD was concerned with database access, with feature extraction proving a relatively economical process. Table 4.6.1 illustrates some representative results, obtained on Newcastle Polytechnic's VAX 8700 in the same way as those presented in Section 3.4 above.

Table 4.6.1 - cpu usage for program DATALOAD

Drawing complexity		CPU usage (s)			
No of boundaries	No of line segments	local/global feature generation	inner boundary feature generation	data base loading	total
1	15	0.01	*	0.42	0.60
1	37	0.04	*	0.56	0.79
1	73	0.10	*	0.70	0.98
4	12	0.02	*	0.45	0.59
4	21	0.03	*	0.52	0.85
4	56	0.06	*	0.81	1.34
7	18	0.03	0.02	0.60	0.84
10	24	0.05	0.18	0.54	1.13
15	76	0.12	0.81	1.12	2.77

**cpu usage too small to measure*

CHAPTER 5. DATABASE DESIGN

5.1 Database requirements for CAD systems

As noted in section 1.4 above, the value of database support for engineering applications is increasingly being realized - as are some of the essential differences between commercial and engineering applications. In their review of database requirements for CAD, Staley and Anderson (1986) list a number of these differences:

Business applications	Engineering applications
Few record types: many instances	Many record types: many instances
Simple relationships between data items	Complex relationships between data items
Static database structures	Dynamic database structures
Many short transactions	Fewer but much longer transactions
Transactions generally simple, involving few records	Transactions may be complex, and involve large numbers of records

which they then use to derive a comprehensive list of criteria that any CAD database system should meet. Although a list of this kind inevitably presents an over-simplified view of the situation, it does usefully bring out the fact that DBMS developed for commercial applications are not necessarily applicable to engineering data. Their views are echoed by many other authors, e.g. Kemper and Wallrath (1987a), who survey the adequacy of existing database models as vehicles to support geometric modelling, and suggest possible alternatives.

Staley and Anderson's most important design criteria included:

- ability to support multiple engineering applications
- support for dynamic schema modification and extension
- support for embedding semantic information in the database
- ability to handle lengthy transactions in a multi-user environment
- support for multiple versions of a design.

Not all of these are relevant to the present project. As discussed in section 2.2.1, there is a fundamental difference (which too few database designers seem to appreciate) between active and completed drawings. Support for schema modification and control of lengthy update transactions are clearly of crucial importance for the former, but virtually irrelevant for the latter. It could be argued that the reverse is true for the ability to embed semantic information in a database - though devotees of feature-based design (section 1.6 above) would dispute this.

Some authors (e.g. Howard and Rehak, 1986) would go further than this, and argue that conventional databases alone are inadequate to support the design process. Since the process of creating a new design generally involves the exercise of judgement rather than simply following instructions, the designer needs to draw not just on stored facts, but on a variety of situational knowledge requiring inferences to be drawn from those facts. This is the kind of support offered by *knowledge-based* or *expert systems* (Hayes-Roth et al, 1983), which apply inferential knowledge stored in the form of *rules* or *frames* to provide solutions to specific problems within their domain of expertise. Howard and Rehak's KADBASE system shows how knowledge bases and databases can be integrated in an intelligent CAD system, and points the way to some potentially exciting research. Again,

however, it must be stressed that their system is aimed at managing the process of creating or modifying an active drawing. While the long-term implications of such systems for drawing archive management (like those of feature-based design) could be considerable, their immediate relevance to the SAFARI project is limited.

5.2 Adequacy of existing database models

5.2.1 General observations

The most widespread DBMS in current use (the vast majority of which is for commercial applications) are based either on the CODASYL (CODASYL, 1971) or relational (Codd, 1970) model of data - though relational DBMS are increasing in popularity to such an extent that they seem likely to replace virtually all existing CODASYL DBMS applications before the end of the decade. Both types of system are firmly based on the concept of *data independence* - the ability to define logical data structures independently of the way they are physically implemented, allowing the process of modelling the natural structure of the data for a particular application to be separated from that of choosing the most efficient storage structures and access paths. The essential differences between the two models lie in the way they represent data at the logical level, and in the tools they offer for data manipulation.

5.2.2 The CODASYL model

If, following Chen (1976), one models data in terms of *entities* (concrete or abstract objects), *attributes* (characteristics or properties of entities), and *relationships* (links showing an association between two or more entities), then a CODASYL database will represent each entity as a database record of appropriate type, and each of its attributes as a field within the appropriate record. Relationships between entities are represented as CODASYL sets (which, it should be noted, do not conform to the mathematical definition of a set, as member records within a set must have a defined sequence), each of which links one *owner* record to zero or more *member* records of a different type - for example, a personnel database might contain a set linking together all employees (member records) belonging to a given department (owner record). A given record may participate as either owner or member in any number of different set types, allowing quite complex relationship networks to be built up.

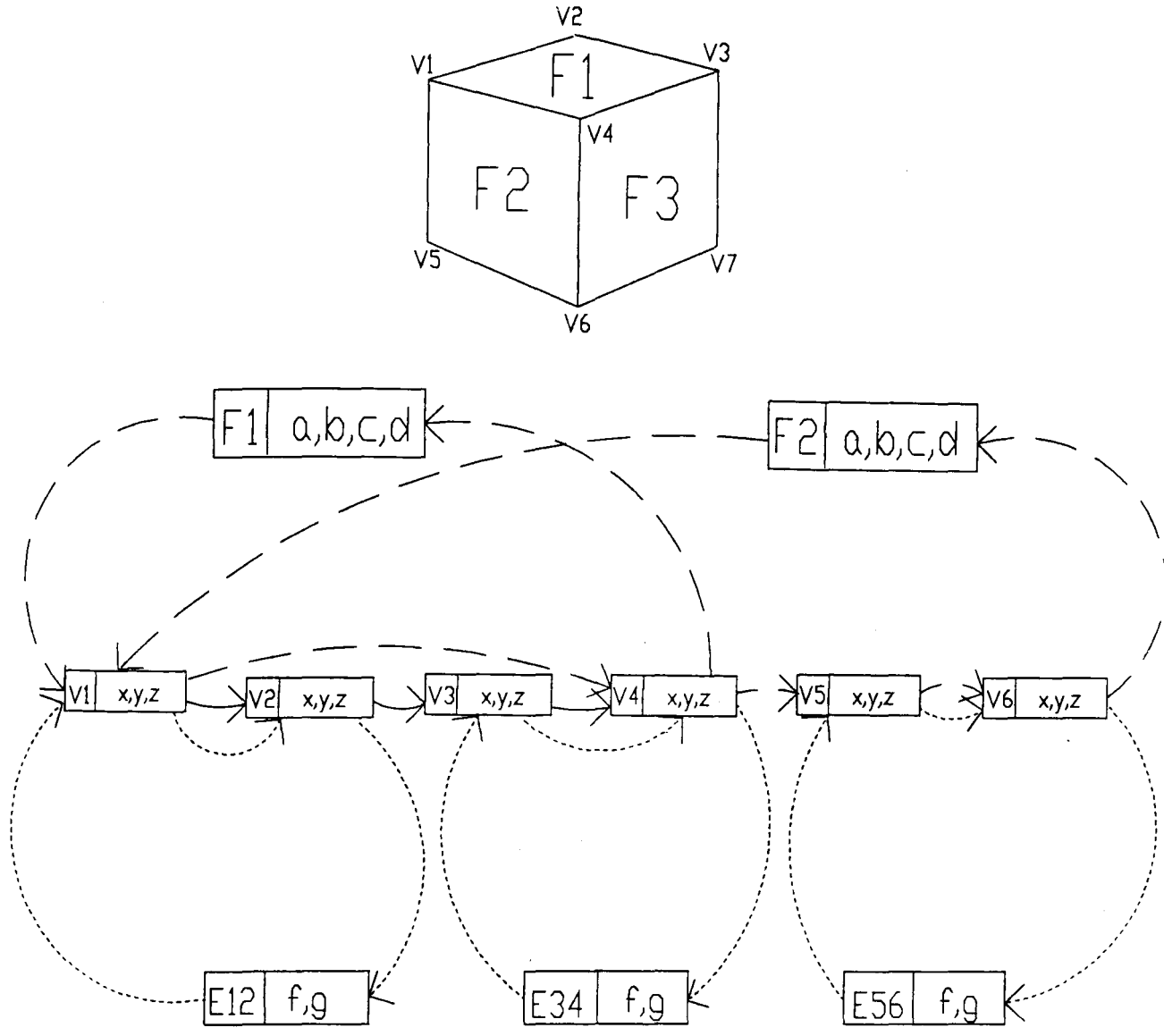
An example of part of a hypothetical CODASYL schema for a geometric modelling database is shown in Fig 5.1, and a sample instance in Fig 5.2. Here, the basic entities are considered to be the faces, edges, and vertices of each shape, and the geometric coordinates or parameters of defining equations their attributes. These therefore comprise the three record types shown. Topological information such as face-vertex and edge-vertex connections can usefully be regarded as relationships, and thus represented as CODASYL sets. Note that the need to use two different constructs (records and sets), often cited as a disadvantage of the CODASYL system, is actually quite useful here, as it emphasizes the difference between geometry and topology.

5.2.3 The relational model

By contrast, a relational database supports only one construct: the *relation* or table. Each table (which *can* be regarded as a set in mathematical terms) represents a complete set of data on all entities of a given type. Each row (or *tuple*) of the table represents a single instance of an entity, and each column a single attribute. Note that each attribute must be atomic; repeating groups of values are not allowed. Any such attribute must be removed to a new relation - part of the data analysis technique known as *normalization* (Codd, 1972).

```
SCHEMA NAME GEOMODEL

{Definition of FACE record, specifying its identifying
 FACE_NO, and the coefficients a, b, c and d of its
 defining equation}

      RECORD NAME FACE
           ITEM FACE_NO          TYPE INTEGER
           ITEM A_COEFF          TYPE FLOATING
           ITEM B_COEFF          TYPE FLOATING
           ITEM C_COEFF          TYPE FLOATING
           ITEM D_COEFF          TYPE FLOATING

{Definition of EDGE record, specifying its identifying
 EDGE_NO, and its defining coefficients f and g}

      RECORD NAME EDGE
           ITEM EDGE_NO          TYPE INTEGER
           ITEM F_COEFF          TYPE FLOATING
           ITEM G_COEFF          TYPE FLOATING

{Definition of VERTEX record, specifying its identifying
 VERTEX_NO, and its x, y and z coordinates}

     RECORD NAME VERTEX
           ITEM VERTEX_NO        TYPE INTEGER
           ITEM X_COORD          TYPE FLOATING
           ITEM Y_COORD          TYPE FLOATING
           ITEM Z_COORD          TYPE FLOATING

{Definition of FACE_VERTEX set, indicating the sequence of
 vertices bounding each face}

     SET NAME FACE_VERTEX
          OWNER FACE
          MEMBER VERTEX
                INSERTION AUTOMATIC
                RETENTION MANDATORY
                     ORDER NEXT

{Definition of EDGE_VERTEX set, indicating the vertices
 bounding each face}

     SET NAME EDGE_VERTEX
          OWNER EDGE
          MEMBER VERTEX
                INSERTION AUTOMATIC
                RETENTION MANDATORY
                     ORDER NEXT
```

Fig 5.1 CODASYL schema defining part of geometric modelling database. The geometry of faces, edges, and vertices is described by appropriate records; topology is represented by CODASYL sets showing face-vertex and edge-vertex links.



Key: F1, F2, . . . : faces
E12, E34, . . : edges
V1, V2, . . . : vertices
a, b, c, d : coefficients of equations of planar surfaces
f, g : coefficients of equations of linear edges
x, y, z : coordinates of vertices
— —— — : links for face-vertex set
---------- : links for edge-vertex set

Fig 5.2 Representation of part of the structure of a cube by the CODASYL database defined in Fig 5.1

Each row has to be uniquely identified by one or more *primary key* attributes. Relationships between entities are represented by *foreign keys*, additional attributes whose values can be matched with primary key values in other tables. Thus an EMPLOYEES relation in a personnel database would contain a foreign key attribute EMP-DEPT to indicate the department to which each employee belonged. The greater flexibility of this approach (connections do not have to be established until run time) is undoubtedly a major factor in the popularity of relational systems - though it can bring problems of data integrity.

The relational schema corresponding to the CODASYL example above is shown in Fig 5.3, and a sample instance in Fig 5.4. Here, one set of relations is used to define the geometry of faces, edges, and vertices, and a second set defines the object's topology in terms of face-vertex and edge-vertex connections.

Another major difference between CODASYL and relational systems lies in their query languages. CODASYL's founding fathers expected the prime route of database access to be from application programs written in third-generation languages such as COBOL or FORTRAN, and devised a set of data manipulation commands to be used as extensions to these languages. These imply that data should be processed one record at a time, and that it is up to the application programmer to specify the correct access path to the desired record. Codd, on the other hand, recognized the power of the set theory approach, and argued for data manipulation operators which operated on a complete data set, simply specifying what data were required and leaving it to the DBMS to devise appropriate access paths. The availability of (relatively) user-friendly query languages such as SQL - a direct development of his approach - has probably been the biggest single factor in ensuring the widespread adoption of relational systems.

Both models of data have their critics, both in the commercial and engineering field. **Rigidity** is perhaps the worst problem, and one which can be levelled with more justice at CODASYL than relational DBMS. CODASYL schemas have to be precompiled before any database creation can take place, and all but the most minor changes in database structure involve unloading the database, recompiling schemas and then reloading the database in its new format. Even in commercial database applications this can be a major problem; in the kind of engineering environment envisaged by Staley and Anderson it would be totally unacceptable. Most relational DBMS are more flexible than this, allowing relations to be added, modified or deleted on-line without affecting the operation of other parts of the database. Even here, though, the degree of flexibility offered falls far short of the ideal.

Data **fragmentation** is a problem inherent in the relational approach, since it is a side-effect of the normalization process. In the commercial field it is often an advantage to separate out data items that are sometimes but not always linked, such as parts and suppliers. It is of no obvious advantage to separate out items which *are* always linked (since the existence of the latter is dependent on the former, like invoice headers and invoice detail lines), though the penalty of having to store these two data types separately is not usually too great. It becomes a serious *dis*advantage when items that form a natural hierarchy, like the faces, edges and vertices of each component in a large engineering assembly, have to be separated out and each stored in a different relation, only to be reassembled (often at inordinate computational expense) whenever the assembly needs to be manipulated. The CODASYL approach is at less of a disadvantage here; repeating groups can be accommodated if required, and even where subordinate items are hived off into separate records, the CODASYL set structure keeps all relationships explicit.

Other problems with the relational approach include the inherent lack of ordering of rows in a relational table, and the limited range of data types supported. Many types of engineering data (such as the sequence of edges bounding a face) are inherently ordered, and such ordering has to be explicitly indicated in a relational database as an extra

```
CREATE SCHEMA "GEOMODEL" ;

{Definition of FACE table, specifying FACE_NO and defining coefficients
 as before. NOT NULL & UNIQUE are entity integrity constraints,
 ensuring that FACE_NO is a unique identifier}

      CREATE TABLE FACE
            (FACE_NO        INTEGER     NOT NULL UNIQUE,
             A_COEFF        FLOAT,
             B_COEFF        FLOAT,
             C_COEFF        FLOAT,
             D_COEFF        FLOAT)

{Corresponding definition of EDGE table}

      CREATE TABLE EDGE
            (EDGE_NO        INTEGER     NOT NULL UNIQUE,
             F_COEFF        FLOAT,
             G_COEFF        FLOAT)

{and VERTEX table}

      CREATE TABLE VERTEX
            (VERTEX_NO      INTEGER     NOT NULL UNIQUE,
             X_COORD        FLOAT,
             Y_COORD        FLOAT,
             Z_COORD        FLOAT)

{Link between face and its corresponding vertices now indicated by
 another table. Note that if vertex ordering needs to be specified, it
 must be indicated explicitly within each row of the table. The CHECK
 clauses are referential integrity constraints, to ensure that the face
 and vertex numbers specified in this table match with those in the
 defining FACE and VERTEX tables}

      CREATE TABLE FACE_VERTEX
            (SEQUENCE_NO    INTEGER     NOT NULL UNIQUE,
             FACE_NO        INTEGER     NOT NULL,
                  CHECK(FACE_NO IN (SELECT FACE_NO FROM FACE)),
             VERTEX_NO      INTEGER     NOT NULL
                  CHECK(VERTEX_NO IN (SELECT VERTEX_NO FROM VERTEX)))

{Similarly, edge-vertex links now indicated by a table}

      CREATE TABLE EDGE_VERTEX
            (SEQUENCE_NO    INTEGER     NOT NULL UNIQUE,
             EDGE_NO        INTEGER     NOT NULL,
                  CHECK(EDGE_NO IN (SELECT EDGE_NO FROM EDGE)),
             VERTEX_NO      INTEGER     NOT NULL
                  CHECK(VERTEX_NO IN (SELECT VERTEX_NO FROM VERTEX)));
```

Fig 5.3 A relational schema defining the same geometric modelling database as in Fig 5.1. Here, both geometry (face and edge equations, and vertex coordinates) and topology (face-vertex and edge-vertex links) are represented by relations holding foreign key values.

Face relation

Face no	Surface equation coefficients			
F1	a1	b1	c1	d1
F2	a2	b2	c2	d2
.	.	.	.	.
.	.	.	.	.

Edge relation

Edge no	Equation coefficients	
E12	f1	g1
E34	f2	g2
.	.	.
.	.	.

Vertex relation

Vertex no	Point coordinates		
V1	x1	y1	z1
V2	x2	y2	z2
.	.	.	.
.	.	.	.

Face-vertex relation

Sequence no	Face no	Vertex no
1	F1	V1
2	F1	V2
3	F1	V3
4	F1	V4
5	F2	V1
.	.	.
.	.	.

Edge-vertex relation

Sequence no	Edge no	Vertex no
1	E12	V1
2	E12	V2
3	E23	V2
4	E23	V3
5	E34	V3
.	.	.
.	.	.

Fig 5.4 Representation of the cube in Fig 5.2 in relational form.

attribute value - with the (erroneous) implication that it is an inherent property of the line element itself. Similarly, engineering data often form natural arrays - and arrays, being repeated groups, are anathema to a relational database. Again, CODASYL databases suffer fewer problems here, as they can (indeed must) specify record ordering within each set, and do generally have the facility to handle at least one-dimensional arrays.

5.3 Alternative database models

The problem with the relational database approach can be simply stated. The validity of the approach rests on the (normally implied) assumption that each row of a relation can stand alone as representing a meaningful entity in its own right (e.g. a part, or supplier, or employee). The further we depart from this situation, the less viable the approach becomes. Hence in the extreme case where one relation holds definitions of all straight lines from all components of all drawings in the database, another holds all circular arc definitions, and so on, we have a situation where the relational approach is totally inappropriate.

How can this situation be resolved? Two basic approaches are currently receiving attention. The first is to modify the relational approach to attempt to overcome the problems outlined above. The second is to attempt to develop new database models from scratch. (The third possible approach, trying to modify the CODASYL model to overcome its limitations, does not appear to have been considered). Rather confusingly, workers in both areas claim to be developing what they describe as **object-oriented** databases.

The object-oriented approach was first developed as a programming paradigm, its best-known implementation being the Smalltalk language (Goldberg and Robson, 1983). Essentially, it regards all computations as involving a set of *objects* whose identity persists over time even though their *state* (or value) may change. Objects may be simple (containing a single scalar variable) or arbitrarily complex. However, all objects must conform to two constraints: objects can communicate with each other only via *messages* exchanged though a public interface; and an object's state may be changed only by a *method* (or procedure) specified for that class of object. Object classes typically form hierarchies, with subclasses inheriting properties, methods or both from the classes above them. The advantages claimed for the approach are that object *encapsulation* (predefining both properties and methods, and permitting data exchange only via external messages) provides a useful discipline, and hides irrelevant implementation details from the user. Inheritance of properties from classes higher up the hierarchy can also be valuable.

The idea of representing each real-world entity by just one object is clearly an attractive one, particularly for engineering applications. It is therefore not surprising that there has been an enormous upsurge of interest in applying the object-oriented approach to the database field (e.g. Dittrich and Dayal, 1986). Many prototype object-oriented database systems have been described in the literature, and a few (such as Vbase and GemStone) have reached the marketplace. Unfortunately there is no clear-cut definition of what is or is not an object-oriented DBMS; there is no body of underlying theory as with relational database, and no defining standards committee as with CODASYL.

So far at least, the greatest progress in applying the object-oriented approach to engineering databases seems to have come from extending the relational model - in particular the work on NF^2 (non-first normal form) databases reported by Dadam et al (1986) and Kemper and Wallrath (1987b) as part of the AIM (advanced information management) project, a joint venture between IBM and the University of Karlsruhe. These workers have extended the conventional relational model by relaxing the condition that all attribute types should be atomic. Instead, attribute types may include arrays, ordered lists - and other relations, allowing the construction of nested relations of the

Face-edge-vertex relation

Face no	Surface equation coefficients				Edges						
					Edge no	Equation coefficients		Vertices			
								Vertex no	Point coordinates		
F1	a1	b1	c1	d1	E12	f1	g1	V1	x1	y1	z1
								V2	x2	y2	z2
					E23	f2	g2	V2	x2	y2	z2
								V3	x3	y3	z3
					E34	f3	g3	V3	x3	y3	z3
								V4	x4	y4	z4
					.	.	.	.	.	.	.
								.	.	.	.
F2	a2	b2	c2	d2	E15	f5	g5	V1	x1	y1	z1
								V5	x5	y5	z5
					E56	f6	g6	V5	x5	y5	z5
								V6	x6	y6	z6
					.	.	.	.	.	.	.
								.	.	.	.

Fig 5.5 Example of nested relation representing a geometric structure. Note the repetition of data at the lowest level, an inevitable consequence of the hierarchy imposed by such nesting.

type shown in Fig 5.5. The advantage of this approach is that arbitrarily complex relations can be built up, exactly mirroring the structure of hierarchically-structured database objects, however complex - hence overcoming the fragmentation problem. Since arrays and ordered lists are also allowable data types, it is possible to model the natural structure of the underlying data without undue difficulty. The authors propose a query language based on extended SQL, permitting data description and manipulation. They also address the vexed question of performance, ignored by most workers in the object-oriented field at present, showing how such complex objects might be indexed, how hierarchical queries might be optimized, and how clustering might improve performance. The major drawback of this approach - a problem shared by all object-oriented approaches to date - is that objects whose natural structure is a network, not a hierarchy, are still poorly modelled. An example of this can be seen in Fig 5.5, where vertex descriptions have to be repeated for each edge using them. However, the NF^2 approach is clearly a promising one.

5.4 File organization for information retrieval

One type of information system has remained outside the mainstream of database development - the bibliographic information retrieval system, which aims to locate documents relevant to a given enquiry. Such retrieval systems, which allow users to search text files of journal articles or abstracts for the presence of suitable keywords, are now a well-established tool of the reference librarian's trade. Although individual systems differ to some extent, the majority accept queries interactively from a terminal in the form of complete or truncated keywords, and identify any documents containing those keywords for display or further searching. Few, if any, have made use of database management software, for the following reasons:

- many such systems pre-date the widespread use of DBMS;
- the type of data stored (mainly free text) is not inherently suitable for storing in the short fixed-length fields for which the majority of DBMS were designed;
- there is little need for data independence;
- only a small range of transaction types needs to be supported, characterized by a high (but predictable) rate of retrieval transactions, a low rate of insertions, and the virtual absence of updates;
- the need to provide short response times in a busy multi-user environment.

The most common form of organization used (which can be regarded to a limited extent as logical rather than physical, since it can be implemented in more than one way) is the **inverted file**. Here, the documents themselves are stored as ordinary text files, but an additional index or directory file is set up, with an entry for every key word appearing in any document on file, listing the address of each document containing that key word. When the system is queried, the inverted file is searched to yield the list of documents containing the specified keyword. If desired, the documents on this list can be fetched and displayed one by one. Alternatively, the search can be refined by selecting more keywords and combining the document lists associated with each keyword according to the rules of Boolean algebra. Given suitable access paths to the inverted file, and methods for combining document lists, the technique can prove highly efficient for retrieval; hence its widespread use. It is much less efficient when new documents are added, as the inverted file has to be separately updated for every keyword in every new document. Many variants on the inverted file concept have been proposed (see van Rijsbergen (1979), chapter 4), though few have been adopted in operational systems.

A radically different method of data organization is the **clustering** technique described by Salton and co-workers, and implemented in their experimental SMART system (Salton et al, 1971). This can certainly be regarded as a logical rather than physical technique, as it can be implemented in a variety of ways, including physical contiguity, pointers, or indexes. A similarity measure is calculated for every pair of documents in the

collection, on the basis of the number of keywords they share. Using the technique of cluster analysis (Everitt, 1980), the entire collection is then grouped into clusters of similar documents. Each cluster can then be represented in the retrieval system by its centroid in k-dimensional document space, where k is the total number of distinct keywords present in the document collection (effectively a single dummy document representing the entire cluster). An incoming query is first matched against each cluster centroid to find the most similar cluster(s), and then against individual documents in the cluster. Judging by the SMART experience, the technique works well for relatively small static document collections, and could in principle be extended to larger collections by including more than two levels in the clustering hierarchy. However, the difficulty of adapting the technique to dynamically-growing collections (when should a new document be added to an existing cluster, and when should it trigger off the creation of a new cluster which could necessitate substantial reorganization of the entire database?) has prevented any widespread adoption of the technique.

5.5 Database requirements of present project

The database requirements of a retrieval system for completed drawings are fortunately less stringent than those of a comprehensive drawing management system which has to support the creation, modification and storage of active drawings. For the present project, flexibility and ease of schema alteration were minor considerations - the structure of completed drawings is by definition stable. The availability of a high-level query language like SQL was of no importance at all, since its expressive power is far too limited to be of effective use in graphical query formulation (see chapter 7 below). Similarly, facilities for integrity and concurrency control were not of major importance. The main criteria were therefore considered to be:

1. Ability to model the natural structure of the data without distortion. This is an obvious but crucial test for any DBMS.

2. Ability to provide access to each stored data item via *any* of its attributes, not just via designated keys. Data access paths for experimental systems are impossible to predict with certainty.

3. Availability of reliable programming language interfaces. Much of the programming for the prototype system has inevitably to be done at a relatively low level; different parts of the system might require development in different languages.

4. Ability to deliver reasonable performance. Shape matching is known to be a computationally expensive process, and any prototype system would probably have to be developed on a busy multi-access machine where economy of resource utilization was essential.

A final constraint was that any database software used would have to be already available on existing hardware, since no funds were available for hardware or software purchase. This effectively narrowed the choice to Rdb/VMS, a relational system, or VAX/DBMS, a CODASYL system. The third option, not to use general-purpose database software at all, but to create the set of interlinked files required to implement one of the types of specialist file organization discused in section 5.4, was rejected early on because of the volume of essentially unproductive work involved in writing file handling routines. (However, an investigation of the applicability of clustering shapes on the basis of features such as those described in chapter 4 remains an interesting possibility for the future).

1. Both DBMS were capable of modelling all the data structures required for the prototype database; however, the CODASYL database was marginally more suitable

for modelling the drawing-boundary-line segment hierarchy, and coped much more easily with array-type extracted features and inherently ordered line segments.

2. Both DBMS provided a variety of access paths, allowing any record to be retrieved via any of its attributes.

3. Both DBMS had adequate interfaces to conventional programming languages such as COBOL, FORTRAN and PASCAL.

4. Preliminary investigations suggested that the CODASYL DBMS would give better performance than the relational, for two reasons - both inherent in the underlying data model.

   Firstly, as discussed in section 5.2, the relational DBMS would fragment the natural drawing-boundary-line segment hierarchy, storing all drawing records in one relation, all boundary records in another, and all line segment records in a third. The reconstruction of the complete group of records making up a single drawing would thus be a significant retrieval task in itself. Rdb/VMS does allow a limited amount of physical record clustering, allowing (for example) all segment records belonging to a given boundary to be housed on the same physical database page as the boundary record, thus reducing the computational overhead to some extent. Unfortunately this mechanism cannot cope with more than one level of hierarchy. The CODASYL database VAX/DBMS, by contrast, allows the database designer to specify two or more levels of physical record clustering via set membership, effectively allowing the entire group of records to be read into main storage in a single operation.

   Secondly, line segments are invariably processed in a fixed order, which can be readily implemented in the CODASYL DBMS by set ordering. Traversing a drawing boundary thus becomes a simple matter of following a pointer chain in main storage. By contrast, the relational DBMS has to retrieve the complete set of boundary segments and then sort them into order every time it requires access.

Perhaps surprisingly, then, the CODASYL database management system VAX/DBMS was chosen in favour of its relational rival as the development vehicle for the prototype database. It should be noted that this does not imply that it is an ideal platform for a shape database system. In the long run, DBMS based on the NF^2 model would probably prove superior - though the possibility of adapting the CODASYL model to overcome the limitations discussed above should not be ruled out.

5.6 Implementation of the prototype database

5.6.1 Logical data modelling

Data modelling for this application is a relatively straightforward process. Three geometric entities can readily be identified, forming a natural hierarchy - the drawing itself, its constituent boundaries, and their constituent segments. To these must be added two entities derived from the feature extraction process described in chapters 3 and 4; the *boundary level*, representing a specific route of traversal of the line segment tree making up a boundary, and the *family*, a group of associated inner boundaries. It could be argued that these derived items (particularly boundary level) are merely views of other types of data rather than entities in their own right. However, the large number of feature values (mean and variance of segment length, arc angle, etc) which are clearly associated with the boundary level suggests strongly that this should be classed as an entity in its own right. Most relationships between entities are obvious except for the position of the segment entity, which could logically be linked either to the boundary or to the boundary level entity. As discussed below, this is effectively an implementation decision. The

preliminary entity-relationship diagram for the prototype database is thus as shown in Fig 5.6.

Assignment of attributes to the appropriate entity type is also straightforward in most cases:

- drawing: identifying information about the drawing itself, plus some derived inner boundary features;
- boundary: identification and position information, together with some derived shape features;
- boundary level: the majority of derived shape features;
- segment: length, arc & discontinuity angles, and parent features;
- family: type and number of constituent boundaries, plus derived inner boundary position features.

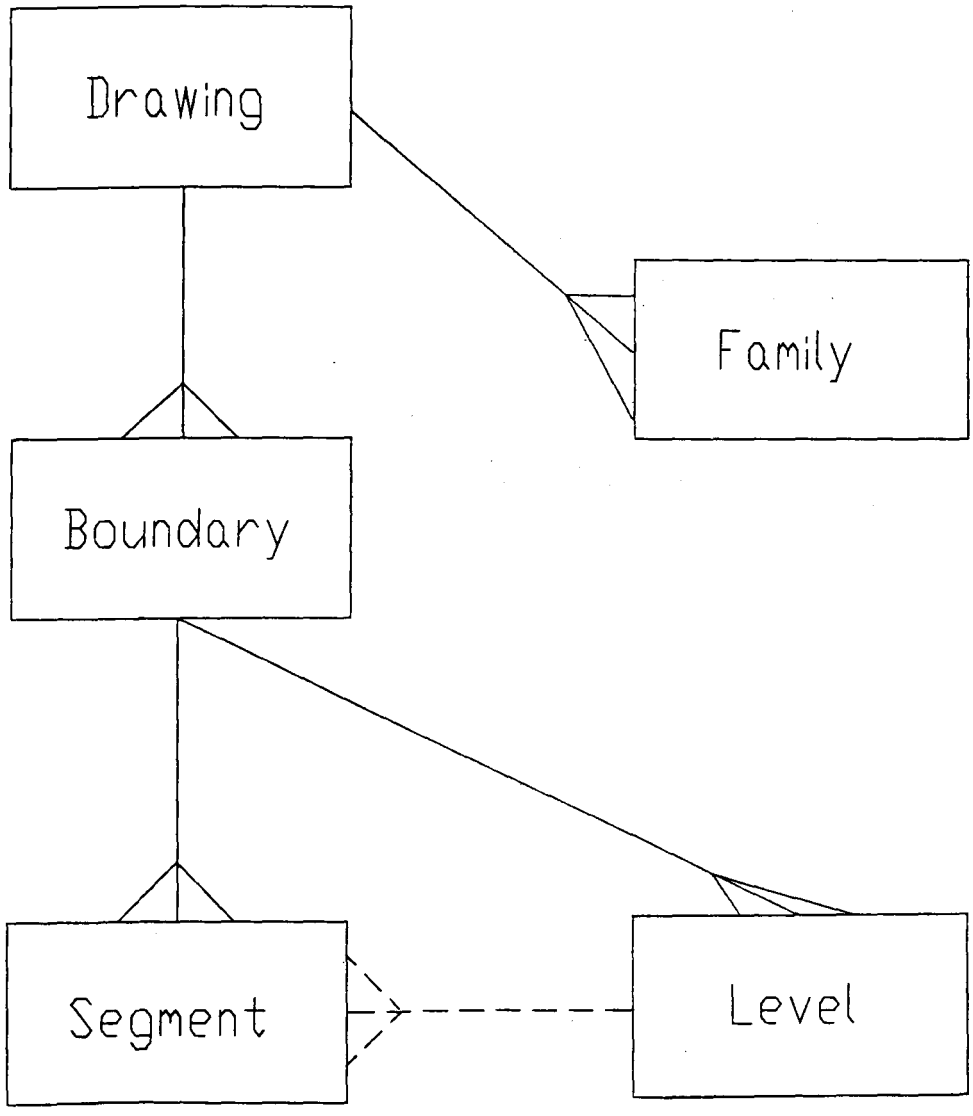


Fig 5.6 Preliminary entity-relationship diagram for prototype database.

The decision to assign properties such as arc/line ratio, length/width ratio and shape class to the boundary rather than the boundary level entity was taken because these were taken as overall indicators of boundary shape (and in fact used mainly as preliminary indicators that a boundary shape looked promising) rather than detailed descriptions of shape characteristics. Hence there was little point in computing and storing them separately for each boundary level.

5.6.2 Physical implementation

Mapping these structures on to a VAX/DBMS conceptual schema is again a straightforward task, except for the extracted features (arc and discontinuity angle triplets, segment length/arc angle distribution, and parent feature composition) which form multi-dimensional arrays. While it would have been possible to hold these as repeating groups within boundary level records, the arrays would have been very sparsely populated (for example, the discontinuity angle triplet feature can take 252 possible values, only 4 or 5 of which are present in most boundaries). It was therefore decided that further normalization was justified in the interests of economy of storage (particularly as this also has implications for speed of processing); additional entities were thus created both for these and the inner boundary position features, for which the same arguments applied.

For similar reasons, it was decided to link all line segments directly to their parent boundary record rather than via boundary level records. This means that each segment needs to be stored once only. All segments making up a given boundary are linked into a single CODASYL set; this can then readily be traversed at any given level using the procedure outlined in section 3.2.3. The alternative, separately storing the sequence of segments making up each boundary level, was rejected because this would have meant storing many of the segments several times over, once for each level in which they participated. This would have virtually doubled the size of the database.

One final addition was caused by the desire to allow a search to be restricted to a single shape class if required. This could most easily be achieved by creating a *shape class* record for each of the classes identified in section 4.4.2, and using this as set owner to link all drawing records within its shape class. The final entity-relationship diagram for the database is thus as shown in Fig 5.7.

The physical layout of a database on disk is often a crucial factor in determining its performance in use. Unfortunately, a great deal of information is needed about database usage before reliable decisions can be made about record placement or access method specification. In the absence of such information, the decision was taken to use system defaults for record placing and access methods for the prototype database, with two exceptions:

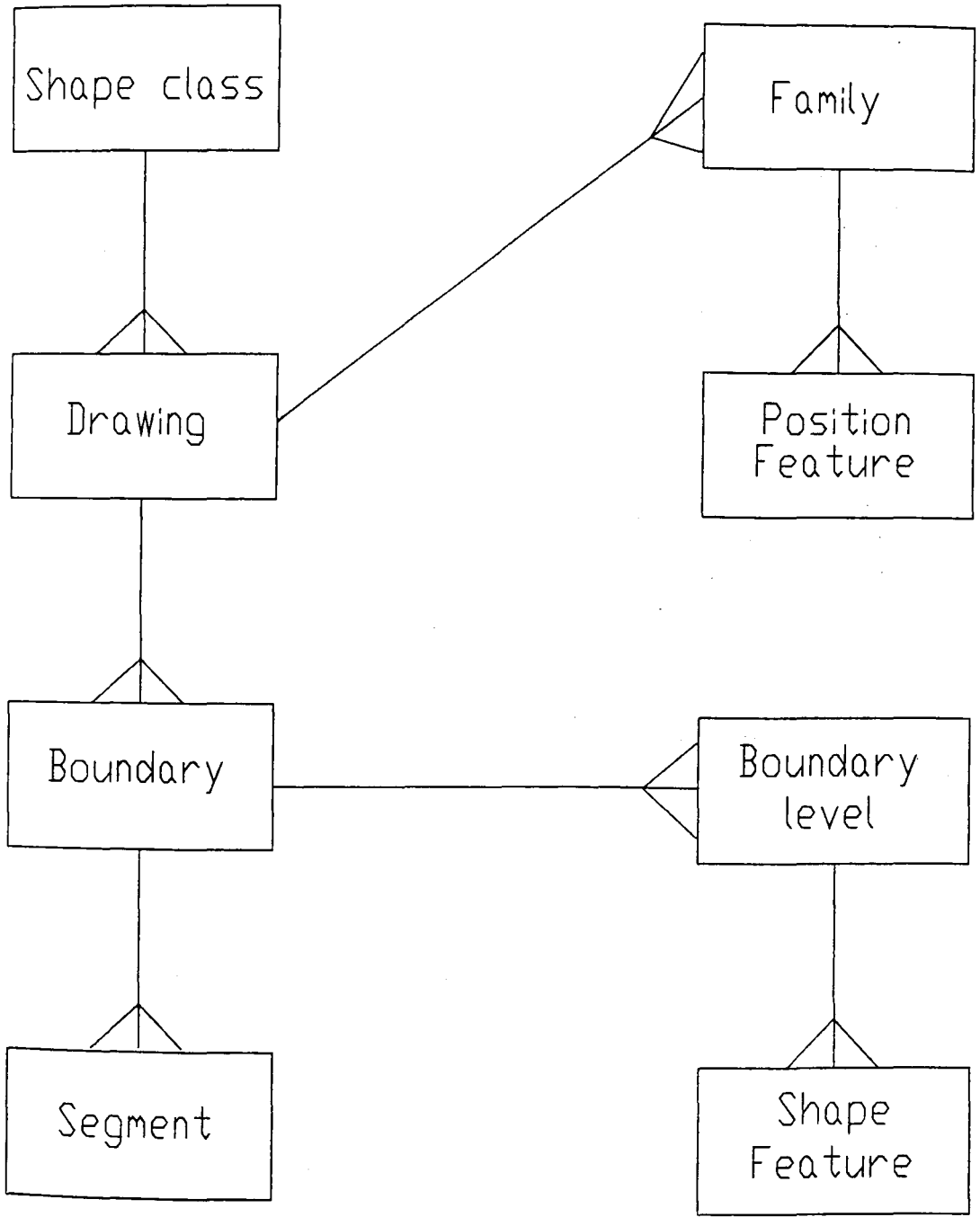


Fig 5.7 Final entity-relationship diagram for prototype database

1. Two separate database areas have been defined, one to hold geometric information (drawing, boundary and segment records), and one to hold derived feature information (boundary level, family and feature records). Drawing, boundary and segment records are then clustered as a two-level hierarchy, on the basis that most segment matching requires sequential segment traversal for each drawing boundary in turn. This can be achieved with a single disc read provided all such clustered records can be contained on a single database page. For similar reasons, boundary level and associated boundary feature records are clustered, as are family and associated drawing feature records.

2. In order to maximize the chances of accommodating a complete drawing hierarchy on one database page, the database page size has been increased from its default of 1 Kb to 4 Kb. Given record sizes of 84 bytes for drawing records, 44 for boundary records, and 28 for segment records, this allows drawings of up to (say) 6 boundaries and 120 line segments to be accommodated on a single page. Over 90% of the drawings in the test database fall into this category.

CHAPTER 6. RETRIEVAL CAPABILITIES

6.1 Introduction

For a database to be of use in locating existing designs possessing desired shape characteristics, a variety of retrieval capabilities is almost certainly needed. While a design engineer may be interested in the overall shape of a part, a production engineer wishing to identify a part family with group technology in mind may wish to identify all parts requiring a particular drilling pattern, or all parts which can be turned on a particular machine. In a field where information needs are so ill-defined, flexibility is clearly an important requirement of any retrieval system.

Many useful parallels can be drawn between a shape retrieval system and bibliographic information retrieval systems of the type discussed in section 5.4. Such systems, though in widespread use, do in fact have a number of significant limitations, largely springing from the fact that both the writer of an article and its would-be reader deal in abstract concepts, while the retrieval system can handle nothing but character strings representing words or perhaps classification codes. The mapping of one on to the other is a chancy process, and one that can in some ways be compared to the process of feature extraction in a shape retrieval system. The text strings used to characterize a document, whether taken from a *controlled vocabulary* (where index terms must be chosen from a restricted list) or free text, are effectively the features by which a document is characterized. The process of information retrieval is thus one of translating an information need into a set of desired features, followed by a search of a suitable collection to identify documents possessing those features. Despite obvious differences, shape retrieval is an analogous process, and both types of system share a number of common problems, particularly the difficulty of specifying a user's information needs, and of determining whether an item is actually relevant to a given query. This theme will be taken up again in chapter 8.

6.2 Types of Shape Retrieval

In order to develop any worthwhile retrieval system, one needs to formulate some hypotheses about the types of retrieval the system needs to support. It can be useful in this context to categorize these different types of retrieval, and then to identify the means by which they might be provided. Tamura and Yokoya (1984) have already attempted this task in the context of image databases in general, distinguishing three different levels of retrieval:

Level 1: retrieval by an identifier
Level 2: retrieval by a combination of plural keys
Level 3: similarity retrieval by a given sample.

For shape databases, one can usefully carry this process further, and define six different types of retrieval (though it is not necessarily helpful to order them into levels as Tamura and Yokoya have done):

Type A, retrieval by an externally-assigned identifier (Fig 6.1), corresponds to Tamura's Level 1. The problems involved in retrieval at this level are trivial.

Type B, finding an exact match of an existing structure (Fig 6.2), has no direct counterpart in Tamura's scheme. It could however be important for registration purposes (answering the question "have we made this part before?"). It is not a difficult problem to solve if objects can all be represented exactly, and in the same canonical form. The query structure is then simply reduced to the same form of representation as objects in the database, and string- or graph-matching algorithms used to check whether representations are identical.

Query input

Results displayed

```
Display part no  236
```

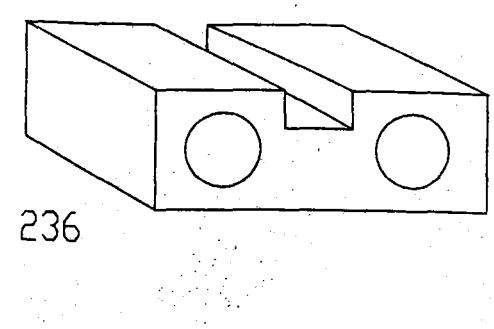
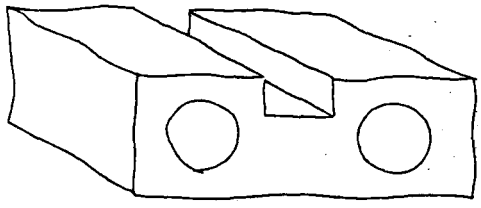


Fig 6.1 Type A retrieval (by externally-assigned key)

```
Display part identical to
```



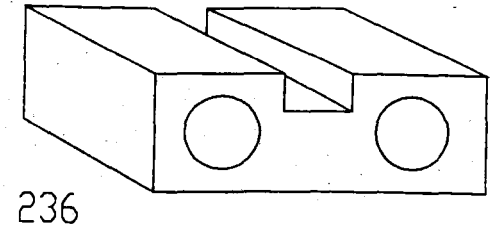
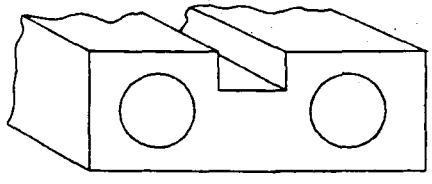


Fig 6.2 Type B retrieval (identity matching, with query input as a sketch)

```
Display part containing
```



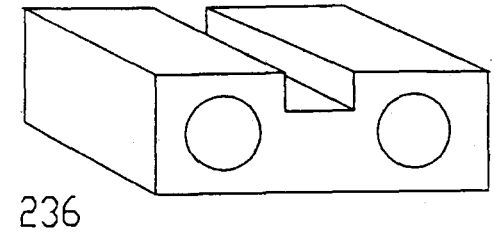


Fig 6.3 Type C retrieval (finding part containing given partial structure)

In practice, this type of retrieval is unlikely to prove useful unless the query is expressed as a rough sketch, the system then displaying a finished drawing in answer to the query. Otherwise the user is effectively required to design an object in its entirety in order to test whether it is on file. Where this is the case, or where objects have to be represented as a range of tolerance values, the process effectively turns into the more complex one of similarity matching, discussed under Type F below.

Type C, identification of all objects which partly match a query structure (Fig 6.3), again has no direct counterpart in Tamura's scheme. It is probably of more practical use than Type B retrieval, but harder to achieve. It requires successive matching of some representation of the query shape against all potentially-matching segments of each stored shape. This is effectively a parsing process, involving successive matching of stored shape elements with each element of the query shape using appropriate similarity estimation, string- or graph-matching algorithms, and inevitably entails a great deal of back-tracking. Representing objects in canonical form is of limited help here, as the required sequence of shape elements could be present anywhere in the object's stored representation. There is still advantage in ensuring that every individual shape element has an invariant representation, as this can greatly simplify the operation of the matching algorithms.

Achieving adequate performance for this type of retrieval with a database of any size is also a problem. Some means has to be found to screen out obviously unsuitable shapes, limiting the computationally-expensive process of string- or graph-matching to a small subset of the whole database. This can best be achieved by analysing all object representations on file for the presence of a suitable set of easily-matched shape features (such as number of right-angled vertices or acute-angled arc segments, as described in chapter 4 above), identifying those present in the query fragment, *screening out* those stored shapes lacking these features, and searching only the remainder. If the database can be indexed on these features, very rapid retrieval can be achieved.

Type D, retrieval by Boolean combinations of features (Fig 6.4), corresponds to Tamura's Level 2. The user specifies a set of desired shape features (by using keywords, or choosing from a text or icon-based menu), and the system retrieves all objects meeting the specified criteria. This type of retrieval can readily be provided if feature extraction has taken place as discussed under Type C above - in fact, it corresponds to the screenout phase of Type C retrieval.

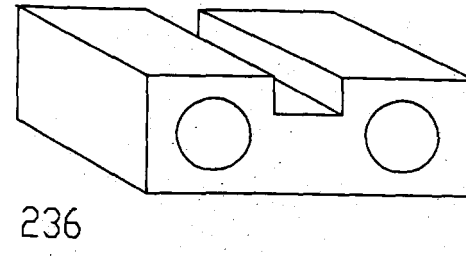
If a predetermined set of features has been defined (such as number and composition of protrusions and depressions), feature extraction can be performed once for all when each object is added to the database, and indexes created on these feature values, thus permitting rapid retrieval. However, one disadvantage of a fixed set of features like this is that (like controlled vocabulary in a text retrieval system) it may have insufficient precision to define some kinds of query, particularly where an unusual type of structure is being sought. The freedom to define additional retrieval features, either when new types of object are added to the database, or when an unusual query is encountered, would clearly be desirable. Run-time feature generation would be an extremely difficult facility to implement. As discussed in section 5.2 above, most existing database models find it hard to cope with the run-time creation of new data types. While this problem could be circumvented by defining standard primitives from which features of arbitrary complexity could be built, it is hard to see how a system could be given sufficient intelligence either to recognize when a new feature was needed, and then to define and extract a suitable feature from query and stored shapes - or to guide an end-user through the same task.

Query input

Results displayed

Display part with features:

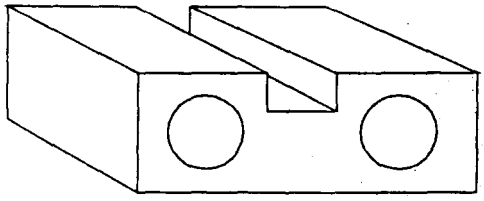
rectangular block
2 parallel holes
1 groove



236

Fig 6.4 Type D retrieval (by Boolean combination of features)

Display code for part identical to

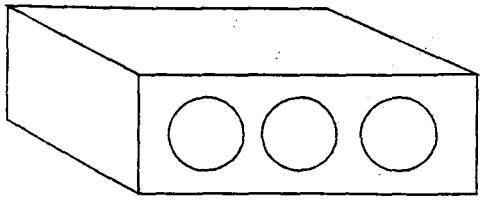


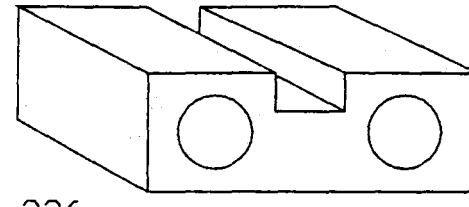
Part no = 236

Opitz code = 80503

Fig 6.5 Type E retrieval (identifying part family)

Display part similar to





236

Fig 6.6 Type F retrieval (by similarity estimation)

Type E, identifying the part family to which a query structure belongs (Fig 6.5), is in some ways related to Type F, general similarity retrieval, though it aims to generate a class code as output, not a set of drawing identifiers. The task here is similar to classical pattern recognition; given an unknown object (the query structure), which class does it fall into? The difficulty here arises mainly from the fact that most pattern recognition tasks involve assignment to one of a fairly small number of possible classes (reported studies of aircraft and chromosome recognition, for example, have involved assigning "unknown" objects to one of no more than about five categories). Classifying a workpiece according to the Opitz code is a far more difficult process; the code runs to more than 10 000 possible categories, and (see section 4.2.1) part of the classification task involves the use of judgement about the function as well as the shape of individual shape features. A hierarchical pattern classifier could in theory be employed, using one set of criteria to assess whether the part was rotational or non-rotational, then another set to establish the presence or absence of appropriate internal machined features, and so on, but the size and complexity of the task - and the number of examples needed for the "training set" of examples needed to establish the ground rules - render it an unattractive option.

The results from published studies of this kind have been varied. Moayer and Fu (1976) attempted to classify human fingerprints in this way, with distinctly mediocre results. Kyprianou (1980) achieved much more intuitively satisfactory results in classifying rotational parts into subclasses on the basis of length/diameter ratios, tapering and the presence of machined features, though his system generated its own shape classes using the technique of cluster analysis, rather than trying to fit a pre-existing code such as Brisch or Opitz.

Type F, similarity retrieval (Fig 6.6), corresponds to Tamura's Level 3. This form of retrieval - locating in the database the objects most similar to a query object - is both potentially the most useful and the most difficult to specify in detail. Engineers asked to define what they mean by "most similar" find it very difficult to explain to anyone else just what they mean by this term. It may be that they can define similarity only after the event, when they have seen some search results displayed. In general terms, similarity retrieval involves selection of suitable shape features, definition of similarity measures, and choice of suitably efficient algorithms for matching the query object with those in the database. The mechanics of similarity estimation are discussed below, in section 6.3.

Though these types of retrieval are distinct, the techniques for their implementation overlap to a large extent, implying that a multi-purpose system capable of providing all these types of retrieval (with the possible exception of Type E) could readily be developed. Whatever type of retrieval is provided, any practical engineering database would need to combine geometric and non-geometric data (materials, cost, weight, etc.). This should pose no special problems if geometric features are extracted as described above and entered in a database alongside non-geometric features; both types of feature could then be combined freely to formulate queries. The main design challenge here centres around the development of a suitable user interface to handle multi-media query input and results display.

6.3 Mechanisms for shape retrieval

6.3.1 General observations

The parallels that have already been drawn between shape retrieval and both bibliographic retrieval and pattern recognition can be usefully extended to the process of query matching. As with feature extraction, the literature in both areas describes an

enormous variety of matching techniques, again presenting a bewildering choice of possible models. The discussion below is necessarily selective, and covers only a fraction of potentially relevant papers on the subject. The classification of methods is highly subjective, and complicated by the fact that few authors restrict their systems to a single type of method (a point worth noting in itself).

6.3.2 Boolean searching

This technique is well established in the bibliographic field. Effectively, it involves a search for the presence or absence of specified combinations of attributes (or attribute values). Objects will be retrieved only if they possess the exact combination of attributes specified. Objects are normally retrieved into temporary sets which can be printed out on- or off-line, or combined by the searcher to refine the search if the initial output looks unpromising. The matching process is very simple, since most systems are based on inverted files which effectively provide an index to every word in the database. A search for a given combination of terms is thus generally implemented as a comparison of the item lists for each term specified, a process that can be made very efficient.

This search paradigm is obviously relevant to Type D retrieval as defined above, and could readily be implemented if the types of features likely to be of most use in retrieval could be identified and presented to the user in text or iconic form. As discussed below, this is not an impossible objective. Its relevance to other kinds of retrieval is more problematic. When estimating the similarity between query and stored shapes, one needs to be able to take a combination of feature values into consideration. It is unlikely that the presence or absence of a single feature will of itself determine a shape's suitability, except as a way of screening out obviously unsuitable shapes as discussed under Type C. Despite its widespread use in other forms of information retrieval, then, this search method is likely to have limited applicability to shape retrieval systems.

6.3.3 Similarity matching

6.3.3.1 General principles

The general principle of similarity matching is very simple, given a set of objects and a set of properties characterizing them. A measure of the similarity between any two objects *i* and *j* (where *i* and *j* can be two stored items or a stored item and a query) can be computed by some comparison operation performed on their property vectors $\mathbf{P}_i$ and $\mathbf{P}_j$. This may be a simple count of the number of properties the objects have in common, some function computed directly from differences in corresponding feature values, or a complex sequence of operations on properties that are themselves vectors. The end-product is in general a measure of *similarity* (ranging from 0 where objects have no features in common to 1 for identical objects) or *distance* (where a value of 0 indicates that two objects are identical, and there is in general no upper limit) between objects, which can be used either to rank objects in order of similarity or to reject objects with similarity below a specified cut-off value. (Note that for a measure D to qualify as a true distance measure, several conditions need to hold, including the commutativity relationship $D_{ij} = D_{ji}$ and the triangular inequality $D_{ij} <= D_{ik} + D_{jk}$). Similarity and distance measures of this kind have been used both in pattern recognition and bibliographic information retrieval systems.

6.3.3.2 Simple feature matching

One of the earliest examples of similarity retrieval in the bibliographic field was the experimental SMART system (Salton, 1971), designed as a vehicle to test alternative document indexing approaches. Unlike most commercial information retrieval systems, it aims to rank all documents in a test collection for relevance to an incoming query in

order that as accurate an idea as possible can be gained of the system's retrieval effectiveness. The measure normally used to assess similarity between a document and a query is the *cosine correlation*, a measure of the correlation between the occurrence of index terms in a query q and a document d:

$$C(d,q) = \frac{\sum_{t=1}^{T} (D_t * Q_t)}{\left(\sum_{t=1}^{T} (D_t^2) * \sum_{t=1}^{T} (Q_t^2) \right)^{1/2}}$$

where D_t is the occurrence of term t in the document, and Q_t its occurrence in the query, and T is the total number of terms. This is a true similarity measure, ranging from 0 if document and query have no terms in common to 1 if they share an identical set of terms.

Similarity estimation is an essential first step in cluster analysis (see section 6.3.4), and the practitioners of this art have studied the question of similarity and distance measures in some detail (Everitt, 1980). Similarity measures suggested for *binary* data (features which are either present or absent) are the simple matching coefficient $(a+d) / (a+b+c+d)$ or Jaccard's coefficient $(a+d) / (a+b+c)$, where a represents the number of features which the objects in question have in common, b and c the numbers of features present in one object but not the other, and d the number of features absent from both objects. The choice between coefficients depends largely on the significance to be placed on negative matches. For *quantitative* data, a possible similarity measure is Gower's coefficient

$$S_{ij} = \frac{\sum (1 - \mathrm{Abs}(X_{ik} - X_{jk}) / R_k}{\sum (W_{ijk})}$$

where X_{ik} is the value of feature k in object i, R_k is the range of values taken by feature k, and W_{ijk} is the weight reflecting the validity of the comparison between i and j over feature k (1 if valid, 0 if not). However, R_k is very sensitive to extreme values of k, and a more robust comparison can often be obtained by calculating a distance measure between i and j such as the *Euclidean distance*

$$D_{ij} = \left(\sum (X_{ik} - X_{jk})^2 \right)^{1/2}$$

or the *absolute metric*

$$D_{ij} = \sum \left(\mathrm{Abs} \ (X_{ik} - X_{jk}) \right).$$

In either case, it is normal to transform all raw feature values to zero mean and unit standard deviation, to give each feature equal weight.

Many workers in the field of pattern recognition have used techniques of this kind, though normally as part of a more complex system such as that described by Yachida and Tsuji (1977), which used such techniques as the first stage of a hierarchical parts recognition system combining the use of global and local shape features, or the context-driven template matcher of Ben-Bassat and Zaidenberg (1984), which selected appropriate sets of features for matching on the basis of previous comparisons.

Closer to the shape retrieval field, Lee (1980) has suggested similarity measures for classifying triangles as equilateral, isosceles, right-angled or other, and quadrilaterals into squares, parallelograms, and so on. He also presents a series of similarity measures for classifying human chromosomes, but these are so specific that they are virtually impossible to apply to general similarity estimation.

6.3.3.3 Matching using transformations

To some extent, it can be argued that all similarity estimation involves some element of mathematical transformation. However, a number of specific types of transformation have been repeatedly used by different groups of workers. Most of these, such as the Hough (Hough, 1962), Fourier (Zahn and Roskies, 1972) and state-space (Mokhtarian and Mackworth, 1986) transformations, are of prime use in recognizing shape features from noisy digitized images, and hence of little use in the present context. However, the use of a much simpler transformation, using local segment parameters to generate plots of θ, the cumulative curvature, against s, cumulative arc length, appears to have much more relevance. θ-s plots have been used by Perkins (1978) and Turney et al (1985), among others, as a more robust alternative to segment-by-segment matching for both complete and partial boundary matching. The use of this technique overcomes many of the problems of defining an edge discussed in section 2.5.1.

6.3.3.4 Stochastic methods

The prime technique discussed under this heading is statistical pattern recognition (Chen, 1973). The basic aim of the method is object recognition via classification - though it should be noted that (unlike the clustering techniques discussed below) the method needs to be used in conjunction with a pre-existing classification scheme. A large initial feature set is selected, and the values of these features analysed within a *training set* of items whose classification is known, with the aim of identifying a small set of features with a strong statistical association with a particular class, or more specifically with the power to discriminate between classes. Unknown items subsequently encountered can then be classified by analysing for the same features, and then predicting probable class membership using discriminant analysis or similar methods. Statistical methods have been used successfully in a number of areas, including OCR (Greanias et al, 1963), radiodiagnosis (Kruger et al, 1972) and classification of blood cells (Mui et al, 1977). Their usefulness in shape retrieval seems limited, however - classification alone is not capable of providing the range of retrieval capabilities discussed above.

Other stochastic techniques have been described in the literature, such as the shape matcher of Bhanu and Faugeras (1984), which attempts to match unknown and reference shapes by assigning vertex labels so as to minimize the probability of local mismatches. This work has limited relevance to the present project, with one possible exception - matching of inner boundary patterns.

6.3.4 Clustering

The aim of cluster analysis, as discussed in section 5.4, is to generate a completely objective classification of the objects under investigation, based purely on the properties

of those objects. A similarity measure is computed for each pair of items in the collection, using measures of the type described in section 6.3.3.2. An item-item similarity matrix is then generated, and items aggregated into clusters on the basis of their similarity to each other. As the threshold similarity for joining a cluster is relaxed, new items join existing clusters, which may in turn merge with each other, thus generating a hierarchical classification or *dendrogram*. The technique was first applied to the classification of biological specimens (Sokal and Sneath, 1963), but has since found adherents both in the information retrieval field (e.g. Salton, 1971), and the pattern recognition field (Duda and Hart, 1973).

While clustering alone cannot provide the full range of retrieval capabilities required by a shape retrieval system, it could have an important part to play in shape classification or similarity retrieval (types E and F as defined in section 6.2 above), especially when combined with a suitable query interface (see section 7.2.4 below). Rather than performing a sequential search of the entire database, or comparing lists of drawing identifiers in an inverted file, a system based on clustering could identify the cluster most similar to a given query, and allow users to browse through individual drawings within that cluster, selecting those they considered most useful. Given that most people seem to be able to judge shape similarity in a fraction of the time it takes to read an item of text, this could prove the method of choice for small to medium-sized systems - though the problems of coping with additions to the shape database (see section 5.4) cannot be ignored.

6.3.5 Syntactic pattern recognition

The fundamental axiom of syntactic pattern recognition (Fu, 1982) is that all images are made up of primitives whose relationship with each other can be described by a formal language, or *shape grammar*. Suitable parsing algorithms can thus be devised either to extract desired features from image descriptions or to recognize patterns of image elements common to both unknown and reference structures. A wide variety of applications has been reported, including the characterization of bubble-chamber photographs in particle physics (Shaw, 1970); the automatic classification of human chromosomes into median, submedian or acrocentric on the basis of their digitized out lines (Lee and Fu, 1972); and the automatic recognition of aircraft type from silhouettes (You and Fu, 1979). This last paper is of particular interest for its use of *attributed grammars*, which augment normal rewriting rules with additional semantic rules, making it possible to pass on attribute values from drawing primitives (or terminal symbols) such as individual boundary vectors to compound structural features (non-terminal symbols) such as tail assemblies. Some use has also been made of higher-order shape grammars such as *web*, *plex*, *graph* and *tree* grammars (Fu, 1982; Lin and Fu, 1984) in areas such as fingerprint classification (Moayer and Fu, 1976), though the results obtained hardly seem to justify the increase in complexity involved.

The use of syntactic pattern recognition techniques in engineering has already been discussed (Section 1.6). Kyprianou's (1980) use of a pattern grammar was clearly useful in allowing recognition of local shape features such as slots, bosses and pockets. While later workers such as Choi et al (1984) and Henderson and Anderson (1984) did not specifically devise shape grammars, they clearly made use of syntactic concepts in defining and recognizing local shape features for machining. Where syntactic pattern recognition techniques appear to have been less successful is in recognizing global shape properties such as symmetry, parallel edges, or repeated features, which are likely to be important in feature-based retrieval. It is not clear that any major advantage stems from slavish adherence to the syntactic (or any other single) method. Most recent advances in the field seem to have come from the application of grammatical concepts where appropriate, within the context of more general methods.

6.4 Design criteria for a prototype shape retrieval system

A number of conclusions can be drawn from the preceding discussion. Firstly, while a prototype system could in theory attempt to offer all types of retrieval capability except perhaps Type E, the single most useful type of capability to offer would be Type F, similarity matching. This would automatically include Type B, identity matching (though not necessarily its sketch input capability). If the ability to match incomplete as well as complete query shapes was provided, this would effectively also allow Type C, partial shape matching. Evaluation of system performance in response to Type C queries would in turn provide the basis for development of Type D (Boolean feature combination) capabilities.

This implies strongly that the prototype system needs to use one or more of the similarity-matching techniques outlined in section 6.3.3. There appears to be little justification for building Boolean search facilities into the system at present, and no advantage in using techniques of statistical or syntactic pattern recognition - the former lacking sufficient discriminating power, and the latter likely to prove computationally very expensive without any guarantee of good results. As observed above, clustering could prove a useful way of improving system performance in the future, but should be considered as an enhancement of other similarity-matching techniques rather than as a substitute. It also has the problem that objectively evaluating the results of a clustering technique is fraught with difficulties.

The choice between different similarity-matching techniques, and indeed between the different feature types discussed in chapter 4, is more problematical, since no clear-cut guidelines emerge from a study of the literature. It was thus decided that the way forward would be to create a prototype system capable of supporting a wide range of feature types and similarity-matching paradigms, effectively acting as a test-bed in much the same way as the SMART system had done for text retrieval twenty years before.

The design criteria for such a system thus include:

1. The system must be able to support Type F retrieval as defined above (this automatically includes Type B), and preferably Type C as well.

2. The system should be able to accept query specifications in the form of complete or incomplete shapes drawn from the same domain as those stored in the database.

3. The system should be capable of matching queries at a range of different levels (e.g. matching outer boundary shape only, outer boundary shape plus inner boundary positions, or outer and inner boundary shapes).

4. The system needs to be able to support a variety of shape-matching paradigms, using as wide a range as possible of the feature types characterized in chapter 4, and capable of extension to handle other feature types where necessary.

5. Results using different shape-matching paradigms must be comparable, thus allowing valid conclusions to be drawn on their comparative effectiveness. If possible, the system should rank retrieved shapes in order of similarity in the same way as the SMART system, thus allowing the SMART retrieval efficiency measures (see Chapter 8) to be used if desired.

6. While computational efficiency is not a major consideration, matching algorithms which are known to be inefficient should be avoided where possible.

Criteria for interface design are discussed separately, in Chapter 7.

6.5 Capabilities of the prototype system

The above criteria were reflected in the design of RETRIEVE, the program which performs matching between query and stored shapes. This program accepts queries in the form of files specifying complete or incomplete query shapes, together with a statement of the run-time search options required. It then matches the query shape successively with all (or, if required, a subset of) stored drawings, computing a distance measure between the query and each stored drawing using the specified matching technique, and builds up a list of retrieved drawings in ascending order of distance from the query shape. This list can be printed when matching is complete, or used to generate a graphical display of retrieved shapes.

Queries are submitted to the program as files for two reasons - firstly (as discussed in chapter 7), the process of graphical query formulation, and subsequent translation of the query shape into the same format as the shapes stored in the database, is a complex one, involving several separate program modules. The most effective way of interfacing between query formulation and matching programs is to use intermediate files. Secondly, some reusable query representation is essential if the same query is to be re-run several times using different search parameters, to compare the effectiveness of alternative matching procedures.

As discussed above, the prototype version of the system is intended as a test-bed for comparing the retrieval effectiveness of alternative feature sets and matching techniques. The range of alternatives offered at present is meant to be illustrative rather than exhaustive. It is expected that this range could usefully be extended in the future. Three basic types of search are offered at present, though many variations of each are permitted. These comprise global feature matching, local feature matching, and what will be referred to as *segment matching*, effectively a form of θ-s matching as discussed in section 6.3.3.3. Each can be used for matching of outer boundaries only, or for matching inner boundary positions or shapes as well.

Global feature matching is, as one might imagine, based on the global features defined in section 4.4.2, plus (if appropriate) inner boundary position features 1 - 5 as defined in section 4.4.4. It relies on computing a distance measure (a true distance measure, as it is based on Euclidean distances between feature values) between query and stored shape boundaries based on normalized differences between global feature values from both boundary and boundary level records. All query and stored shape boundary levels are examined, in the sequence specified below (section 6.6.3).

Local feature matching uses a combination of the local features defined in section 4.4.3, plus inner boundary features from section 4.4.4 if appropriate. Two alternative means of calculating difference measures (almost certainly not true distance measures in this case) are used. The first, referred to below as *local matching*, is basically analogous to global matching, aiming to assess overall similarity between query and stored shapes. It computes a difference measure on the basis of differences in frequency of each feature present in either query or stored shape, and would therefore tend to exclude shapes containing large numbers of features not present in the query. The second, *existence matching*, works on the principle that a shape containing specified features should be retrieved however many additional features it contains. In this case, a tally is kept of the number of query features present in the stored shape, and used to compute a difference measure as specified below. As with global matching, all query and stored shape boundary levels are examined in turn whichever method is specified.

Segment matching aims to compute difference measures between query and stored shapes by measuring the differences between their θ-s plots at comparable boundary levels. As observed above, this means of searching should be less sensitive to changes in boundary segment composition. It also provides a test of the hypothesis that there is value in trying to cast shapes into canonical form; if the hypothesis is correct, a valid

difference measure can be obtained by comparing query and stored shapes just once at each level, starting from their canonical starting point. (The standard method of comparing θ-s plots involves repeating the comparison process using each possible boundary start point in turn, a computationally expensive process). Again, a number of run-time options is offered.

Finally, **combined matching** may be specified. This uses one of the feature-based methods (global, local, or existence matching) as a preliminary screening search (the user can specify a maximum number of shapes, a maximum difference threshold, or both), followed by segment matching on the subset of shapes retrieved by the preliminary search.

Some limitations on the types of matching available have to be made where incomplete query shapes are specified, since many parameter values (particularly global shape features) are undefined for incomplete shapes. Global matching is meaningless in this case and therefore prohibited. Both local and existence matching are permitted, though the feature set they use has to be slightly curtailed. Segment matching using a canonical start point is also meaningless, and therefore not offered by the present version of the system. (Repeated local θ-s comparisons would however be an effective, if computationally expensive, way of searching for partial shape matches, and would therefore be a useful facility to offer in any future version of the system, particularly if combined with a preliminary screening step based on local or existence matching).

6.6 Detailed program operation

6.6.1 Initialization and query input

The program begins by reading in the sequence of run-time parameters which will govern the operation of the current session. These include a specification of the search paradigm required (global, local, existence or segment, separately or combined), the level of searching required (outer boundary shapes only, outer boundary shapes plus inner boundary positions, outer boundary shapes plus inner boundary class, all boundary shapes), the type of output required, the maximum difference cutoff and maximum number of drawings to be retrieved, whether the search should be limited to a subset of the database, as well as parameters specific to the match type or search level chosen.

The program is then ready to process queries according to the parameters specified. As indicated above, each query has to be submitted as a file generated as outlined in chapter 7, below. The program accepts a query file name and attempts to open and read in the contents of this file, building up a representation of the query in main storage (using a series of linked lists) which exactly mirrors the structure of stored shapes in the database (Fig 6.7). A number of validity checks are performed at this stage, including a check that all query boundaries have the expected number of segments, that traversal at all levels involves a total angle of 2π, and that specified match type and query structure are compatible. If shape matching is to be limited to a given shape class, the appropriate *ShapeClass* record in the database is located to identify the correct *DrawingClass* set to search.

6.6.2 Shape matching - general

The required set of drawings (which may include the entire database) is now fetched one by one from the database and matched using the appropriate technique (see below). The overall process for any single match type is identical, with one exception; with feature (but not segment) matching, it is possible to specify a *pre-screening* step, rejecting out of hand any drawing for which outer boundary *PA*, *LW* or *AL* ratios diverge from comparable query values by more than a specified amount. The main purpose of this is to reduce search times; it seems to have little effect on retrieval performance in most cases.

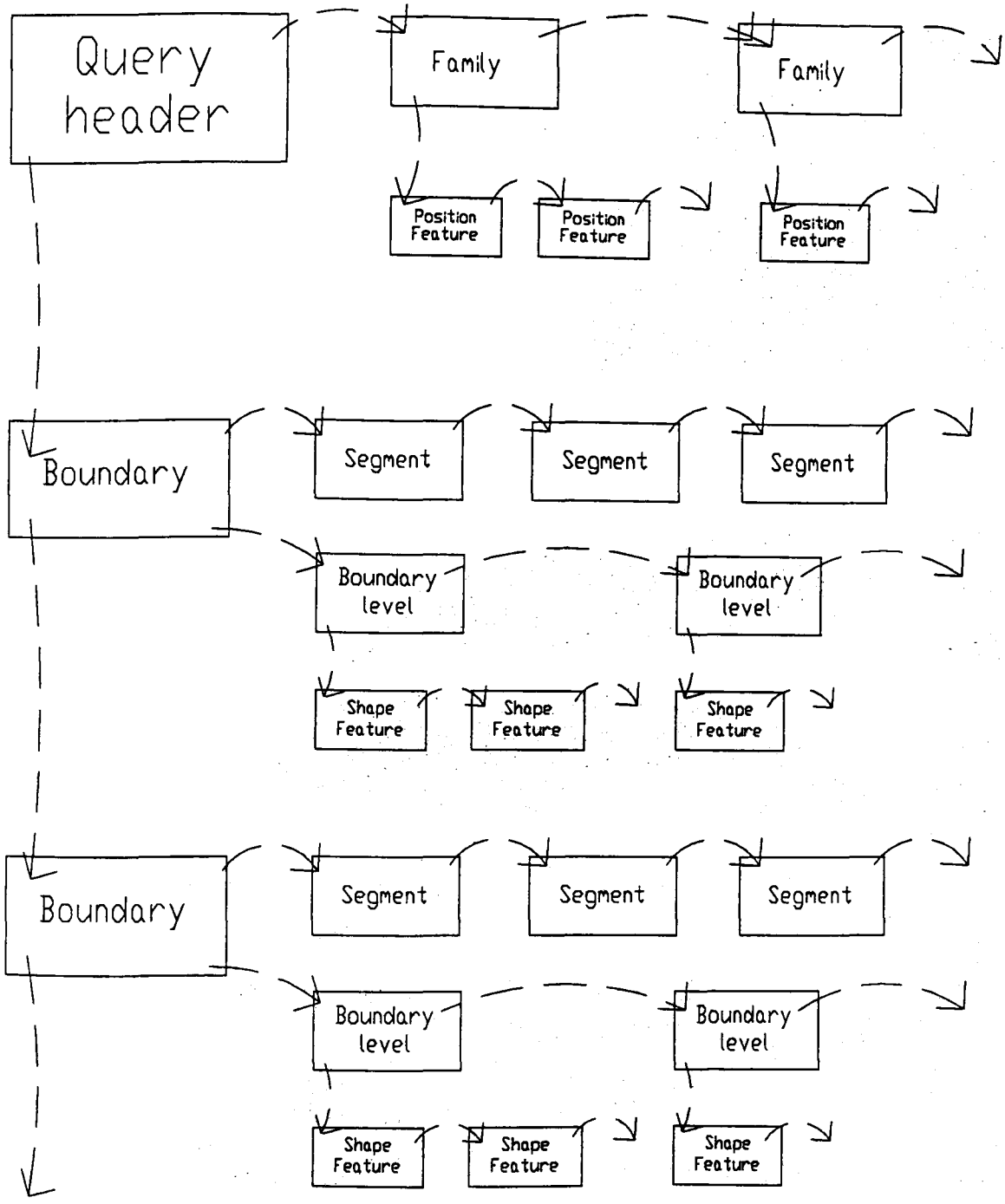


Fig 6.7 Illustration of linked list structure representing query in main storage.

Whatever matching process is specified, the first step is to match the outer boundary of the current drawing with the outer boundary of the query, thus yielding a difference measure between the query and the current drawing. This process is not quite as straightforward as it seems, as each boundary can be defined at more than one level, and (as indicated in section 4.4), many shape features are inherent properties of a specific boundary level, not the boundary as a whole. This raises the question of which level from the query boundary to match with which level of the drawing boundary - particularly when q, the number of levels in the query boundary, and d, the number in the drawing boundary, will often be different.

Since each successive boundary level presents a view of the boundary at a progressively greater level of detail, it is logical to match corresponding boundary levels from query and drawing where $q = d$. Indeed it is necessary if a query shape is to yield a difference measure of zero when matched with itself. Where $q <> d$, it is superficially tempting to treat both in a symmetric way, and (for example) to match the first (or last) $\min(q,d)$ levels. This is however unsound for a number of reasons. In the first place, a query specification and the drawing it retrieves are not symmetric. The former is open-ended, a statement of those features the user considers desirable, and a specification which may be met in a number of ways, possibly unforeseen by the user. The latter is a fixed object which may or may not be relevant to an enquiry. If a query boundary has three levels, it is fundamental to the search process that all three levels are matched against every comparable drawing boundary in the database, however many levels this has. Otherwise difference measures obtained with one drawing are not comparable with those obtained with another, and hence no similarity ranking is possible.

(In passing, it may be remarked that this fundamental asymmetry between drawing and query inevitably has the consequence that the overall difference measure M between query and drawing cannot qualify as a strict distance measure, as the measure M_{dq} computed by reversing the roles of query and drawing is not in general identical to the measure M_{qd} computed as detailed below, since it may involve comparisons between different numbers of levels. This does not appear to detract from its use fulness in ranking drawings in order of similarity).

The solution chosen was hence to ensure that exactly q comparisons took place between each query and drawing boundary. Where $q < d$, some drawing levels will be ignored. Where $q > d$, all drawing levels will be used, some more than once. In either case, the program attempts to identify the q likely closest matches between query and drawing boundary levels, and derives its distance measure from these, using the algorithm set out below. The process is illustrated graphically in Fig 6.8.

If matching is limited to outer boundaries, the process ends once a difference measure has been calculated. Assuming the query-drawing difference measure is less than the specified cutoff value, the drawing's reference number, original file name and difference measure are added to the list of drawings already retrieved. This is maintained in order of increasing difference value; if the list size is exceeded by adding a new drawing, the last drawing on the list is automatically removed.

If inner-boundary searching is specified, the precise sequence of events depends on the exact match type and search level specified. However, the overall process is again the same - one or more further difference measures is calculated by matching inner-boundary features, weighted appropriately, and added to the outer-boundary difference measure to give an overall measure of query-drawing difference. Processing then continues as specified in the paragraph above.

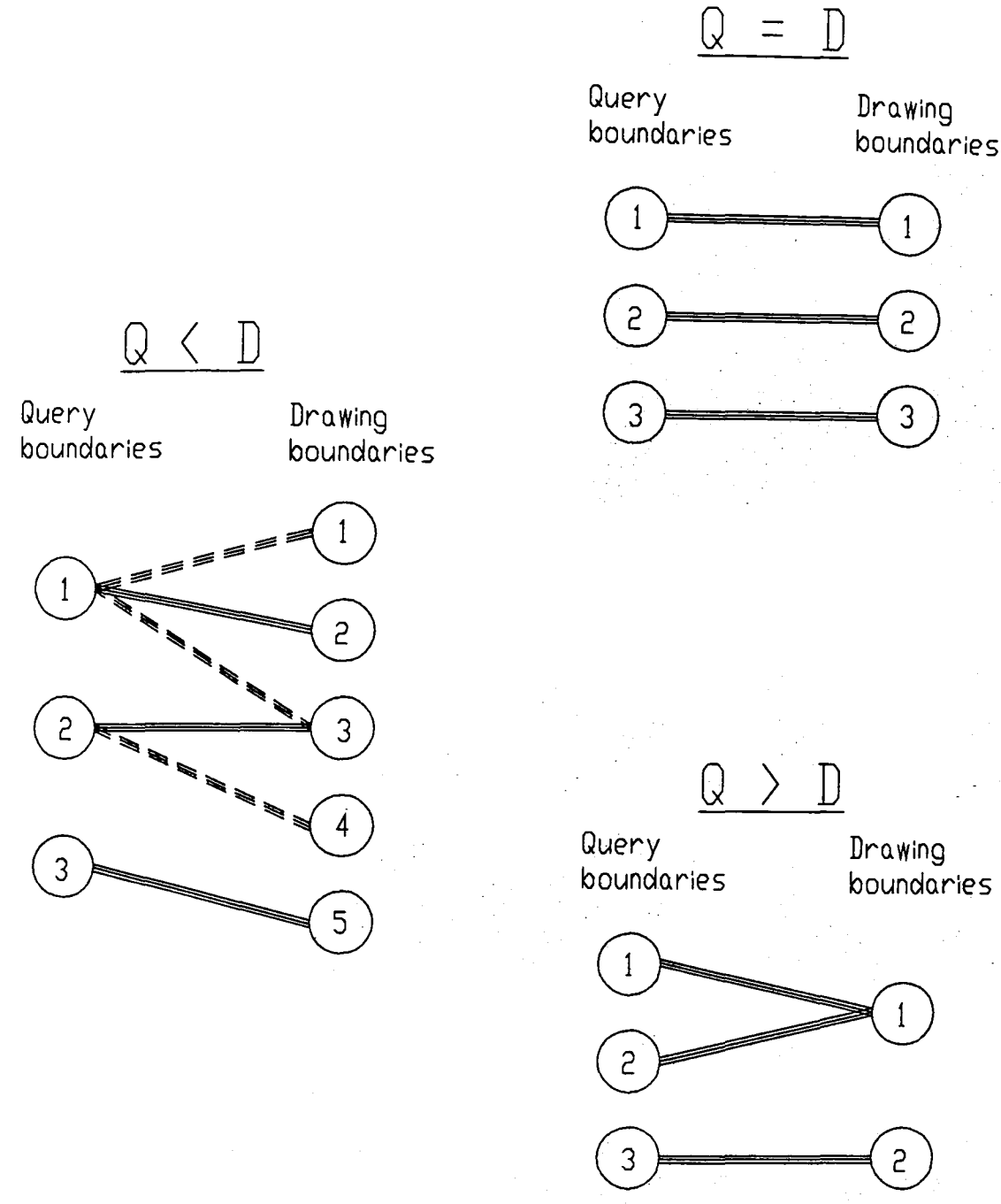


Fig 6.8 Matching between query and drawing inner boundaries. Exactly Q *comparisons (where* Q *is the number of query inner boundaries) are recorded whatever the value of* D *(the number of drawing boundaries).*

If combined feature and segment matching is specified, a two-stage process takes place. Firstly, a feature search is carried out on the set of drawings to be searched, and retrieved drawing identifiers stored in an intermediate list. Each drawing in the intermediate list is then fetched again in turn from the database, and subjected to segment matching. A combined difference measure is then computed, and if this is less than the specified cutoff value, the drawing identifier is added to the final list of retrieved drawings as before.

With feature matching at the inner boundary position level, the required difference measure can be calculated from inner boundary position feature records, without reference to the inner boundary records themselves. With all other match and level types, however, individual inner-boundary representations from query and drawing must be directly compared. This raises a problem analogous to that of matching boundaries with different numbers of levels - which query boundaries should be matched with which drawing boundaries when query and drawing will have different numbers of inner boundaries Q and D? It turns out that the solution is also analogous. To ensure comparability between comparisons with different drawings, it is again important to ensure that exactly Q comparisons take place between each query and drawing, ignoring some drawing inner boundaries where $Q < D$, and using some more than once where $Q > D$. Two additional questions remain - is a basically linear traversal of each inner boundary list, using a variant of the boundary level traversal algorithm described above, acceptable in this case? How should the special case of a drawing with no inner boundaries (for which there is no analogue at the drawing level, since each boundary has by definition to have at least one level) be treated?

The short answer to the first question is "yes". In theory, one could match all Q query inner boundaries with appropriate combinations of Q drawing inner boundaries, selecting the combination giving the best fit after exhaustive trial-and error matching - or, more elegantly, using a label assignment algorithm such as that used by Bhanu and Faugeras (1984). However, a stated design objective of the present system is the avoidance wherever possible of algorithms that require backtracking. It also seems sensible to exploit the fact that inner boundaries in both query and stored shapes are ordered firstly by distance from outer boundary centroid, and secondarily according to the angle between their centroid vector and the shape's prime direction (see section 3.2.3 above). Thus wherever a single-pass comparison of inner boundary lists is possible, it should be attempted. Hence the level-traversal algorithm was adapted for following and comparing inner boundary lists.

The second question is harder to answer. Q inner boundary comparisons are clearly needed to maintain comparability even when a drawing has no inner boundaries. (Note that the reverse case - a query with no inner boundaries - is no problem, because $Q = 0$ and hence no comparisons take place with any drawing inner boundaries, another example of the fundamental asymmetry between query and drawing). The only obvious solution, somewhat lacking in elegance, is to compute a series of null similarity values, based on the absolute values of the relevant parameters in the query, and then to use these instead of difference measures where necessary. For example, if computing a measure of inner boundary position difference between a query with four boundaries and a drawing with none, the program will simply use the appropriately normalized sum of the four query boundary centroid distances instead of summing actual distances between query and drawing boundary centroids.

6.6.3 Feature matching in detail

This process starts with an initialization step that includes an optional pre-screening stage, as indicated in section 6.6.2. This computes an initial difference measure between query and drawing

$$M_{qd} = 2.5*\mathrm{Abs}(AL_q - AL_d) + \mathrm{Abs}(\log_2(PA_q/PA_d)) + 0.5*\mathrm{Abs}(\log_2(LWq/LW_d))$$

where *AL* (as defined in section 4.4.2) provides a measure of overall curvature and hence shape class, *PA* an estimate of the shape's low-level thickness and to some extent complexity, and *LW* a measure of overall aspect ratio. The measure is purely empirical but seems to be successful at screening out about half to two-thirds of the entire drawing file (more for some queries) if a cutoff threshold is set at a level of between about 2 and 3. Assigning the higher weighting to the *AL* ratio appears significantly to improve screenout of obviously un suitable drawings, again suggesting the importance of shape class in retrieval.

6.6.3.1 Boundary feature similarity matching

If the drawing is not rejected by this screenout process, procedure *MatchFeatureSimilarity* is then called to compute a difference measure between query and drawing outer boundaries, using the following algorithm (presented here in slightly simplified form), which generates both minimum and cumulative level difference measures for all query boundary levels, as discussed in the previous section:

```
Fetch Boundary and first BoundaryLevel records for both query and
stored drawing;

Compute B, the difference measure derived from feature values in
Boundary records, using appropriate match type, and set C, the
cumulative level difference measure, to zero;

While unprocessed query BoundaryLevel records remain do

  Begin

  Compute M, the minimum difference measure from feature values in
  current BoundaryLevel records using appropriate match type, set
  temporary difference T = M, and indicate current pair of
  BoundaryLevel records as current minimum;

  While unequal numbers of unprocessed query and drawing
  BoundaryLevel records remain do

    Begin

    If more BoundaryLevel records remain in query than in drawing
    then

      fetch next BoundaryLevel record from query

    else

      fetch next BoundaryLevel record from drawing;

    Compute new difference measure L from feature values in
    BoundaryLevel records using appropriate match type;

    If L < M then

      Begin
```

```
        Set M = L, and indicate current pair of BoundaryLevel records
        as current minimum;

        If more BoundaryLevel records remain in query than in drawing
        then

          set intermediate difference I = T

        End;

      Set T = T + M

      End;

    Set C = C + M + I;

    Fetch next BoundaryLevel record after current minimum from query;

    If more BoundaryLevel records remaining in drawing then

      fetch next BoundaryLevel record after current minimum from
      drawing

    End;

  Compute final difference measure M, calculated as M_tot = B + C,
  M_avg = B + C/q, or M_min = B + min(M), according to run-time option
  selected.
```

If query and drawing boundaries have equal numbers of levels, this reduces to a single pass through both, allowing total, average or minimum level difference measures to be calculated as indicated above. If there are fewer query levels than draw ing levels, the inner loop allows each query level to be matched against all "spare" drawing levels, and the closest-matching pair selected, without incrementing the cumulative difference measure. If there are more query levels than drawing levels, the inner loop matches "spare" query levels against each drawing level in the same way - but this time increments the cumulative difference measure, so that each query level is effectively matched as required. The detailed matching processes differ depending on whether global, local or existence matching is specified, as detailed below.

6.6.3.2 Global feature matching

If global matching is used, the query-drawing difference measure B calculated from boundary records is a weighted distance measure computed from values of the features *AL*, *PA*, *LW*, and *NR* using the absolute metric

$$D_{ij} = \frac{\sum \text{Abs}(X_{ik} - X_{jk})}{\text{S.D.}(X_k)} .$$

Similarly, the difference measure L calculated for each level is the weighted sum of D_{ij} for the features *ML*, *LV*, *MA*, *AV*, *MD*, *DV*, *RS*, *PS* and *CI*. Weighting of each measure is currently performed simply by dividing the total by the number of contributing parameters, except again for *AL*, where early experiments suggested that increasing the weight of this parameter could improve retrieval performance.

6.6.3.3 Local feature matching

With local matching, the boundary record difference measure B can either be set to zero or calculated in the same way as for global matching, depending on the run-time option chosen. No specifically local features are stored in boundary records, so that strictly speaking, no computation can be performed at this stage. However, the option to use global measures as a possible performance enhancement has been provided for comparative testing purposes. Note that where query boundaries are incomplete, this measure is always set to zero.

The boundary level difference measure L is calculated from a combination of the distribution parameters *LD*, *AD*, *DD* and *FD*, stored in boundary level records, and the more complex features *SL*, *AT*, *DT* and *PF*, stored in boundary feature records. Again, the precise combination of features used can be specified at run time. While both are processed in the same logical fashion, there are inevitably differences in the way this is implemented. With each parameter selected, a difference measure is calculated by computing the difference and sum of each pair of corresponding values, and dividing one by the other:

$$L_{qd} = \frac{\sum_{i=1}^{N} \mathrm{Abs}(D_i - Q_i)}{\sum_{i=1}^{N} \mathrm{Abs}(D_i + Q_i)}$$

where N is the total number of possible values for the feature in question. For the distribution features, this can readily be computed by successively comparing counts in each array element. For the more complex features, lists of query and drawing feature records are compared, and counts of feature value sum and difference accumulated as follows (Fig 6.9):

- where an unmatched feature record is found in either list, its value is added both to the cumulative sum and cumulative difference;

- where a matching record is found, the sum of feature values from both records is added to the cumulative sum, and the absolute difference in feature values added to the cumulative difference.

All values are weighted by dividing by the total number of features selected, in order to maintain as much comparability as possible between measures.

6.6.3.4 Existence matching

Existence matching uses the same feature set as local matching, but adopts rather different matching principles. Whereas local matching attempts to measure similarity by considering all features present in either query or drawing (in other words giving weight to negative and positive feature matches), existence matching considers only those present in the query. The presence or absence of drawing features not specified in the

(a)

Feature range	Frequency in query record	Frequency in drawing record	Sum	Difference
0.1 - 0.3	2	1	3	1
0.3 - 0.5	3	1	4	2
0.5 - 0.7	4	5	9	1
0.7 - 0.9	5	2	7	3
0.9 - 1.1	1	5	6	4
1.1 - 1.3	0	1	1	1
1.3 - 1.5	0	0	0	0
	Total:		30	12

$$L_{qd} = 12 / 30 = 0.4$$

(b)

Query features		Drawing features		Sum	Difference
Type	Frequency	Type	Frequency		
		D324	2	2	2
D334	2			2	2
D357	1	D357	2	3	1
		D552	2	2	2
D553	3			3	3
D567	2	D567	2	4	0
	Total:			16	10

$$L_{qd} = 10 / 16 = 0.625$$

Fig 6.9 Illustration of similarity calculations (a) for simple features from boundary level records (showing part of a feature array), (b) from more complex features (showing sample boundary feature records).

query is immaterial. Furthermore, the actual value of specified features is immaterial provided they reach a given threshold. This obviously implies a slightly different approach to feature comparison. Rather than computing a difference measure directly from differences in feature values, feature values from drawing records are classed as *present* or *absent* depending on whether they exceed a specified threshold (or, in a few cases, fall into a specified range). A difference measure can then be computed as

$$\frac{\text{(no of possible features - no of features present)}}{\text{(no of possible features)}}.$$

As with local matching, the boundary record difference measure B can be set to zero or calculated from global parameters as required. The same parameters are used as for global or local matching, but the difference measure is computed on the basis of the number of features whose values differ from those in the query by less than a specified amount. Again, this measure has to be set to zero where query boundaries are incomplete.

The boundary level difference measure L can be calculated from the same combinations of features as local feature matching, using a process identical except for the formula used to calculate the difference measure:

$$L_{qd} = \frac{\underset{i=1}{\overset{N}{\text{Count}}}\left(D_i : (D_i > Q_i * T) \text{ and } (Q_i > 0) \right)}{\underset{i=1}{\overset{N}{\text{Count}}}\left(Q_i : Q_i > 0 \right)}$$

where T is a threshold whose value can be specified at run time. A comparative illustration of the operation of existence and local matching is shown in Fig 6.10.

6.6.3.5 Penumbral matching

It can be argued that local feature (or existence) matching as specified above suffers from a fundamental flaw; it assumes that all parameters in the feature arrays are independent of each other. This is manifestly not the case, since most are (arbitrary) divisions of continuous variables. The feature matching method above assumes, for example, that a query specifying the presence of two line segments in the length range 0.7-0.9 can be satisfied only by a drawing with two line segments in exactly the same length range. In practice, such a query could well be satisfied by a drawing with line segments in the length ranges 0.5-0.7 or 0.9-1.1, since the dividing lines are necessarily arbitrary and could well have separated line segments with infinitesimal differences in length.

Whether this effect actually degrades retrieval performance in practice is a subject for further investigation. It is clearly more of a problem for some parameters than others. The parent feature distribution *PD*, which can take only four discrete values, is obviously not susceptible. The angle distribution features *AD* and *DD* are probably at risk only to a minor extent because of the deliberate bias of angle ranges towards right angles. But the length distribution features *LD*, *SL* and to some extent *DT* could well be seriously affected.

Several possible ways of overcoming this problem could be considered. The matching process could be extended to include neighbouring feature ranges - though this would probably increase the number of false positive matches generated. The database could be enhanced, for example by defining an extended length distribution feature specifying two overlapping length distribution ranges. A line of relative length 0.8537 could thus be

(a)

Feature range	Frequency in query record	Frequency in drawing record	Feature relevant?	Feature qualifies?
0.1 - 0.3	2	1	Y	N
0.3 - 0.5	3	1	Y	N
0.5 - 0.7	4	5	Y	Y
0.7 - 0.9	5	2	Y	N
0.9 - 1.1	1	5	Y	Y
1.1 - 1.3	0	1	N	N
1.3 - 1.5	0	0	N	N
Total counts:			5	2

$$L_{qd} = (5 - 2) / 5 = 0.6$$

(b)

Query features		Drawing features		Feature relevant?	Feature qualifies?
Type	Frequency	Type	Frequency		
		D324	2	N	N
D334	2			Y	N
D357	1	D357	2	Y	Y
		D552	2	N	N
D553	3			Y	N
D567	2	D567	2	Y	Y
Total counts:				4	2

$$L_{qd} = (4 - 2) / 4 = 0.5$$

Fig 6.10 Illustration of similarity calculations for the same set of features as in Fig 6.9, using existence matching with a threshold T = *1.*

counted both as a 0.7-0.9 length line and as a 0.8-1.0 length line. Searching could then match query length distributions in, say, the 0.7-0.9 range with drawing lines in either the 0.6-0.8 or 0.8-1.0 ranges.

The strategy adopted for the prototype system was to provide two possible matching modes - the conventional mode described above and *penumbral matching*. This is a form of relaxation method, in which each query or drawing feature value V'_i used in the comparisons detailed above is computed from the weighted average of its stored value V_i and that of its neighbours, as follows:

$$V'_i = 0.5*V_i + 0.25*V_{i+1} + 0.25*V_{i-1} \quad (1 < i < N)$$
$$V'_i = 0.75*V_i + 0.25*V_{i+1} \quad (i = 1)$$
$$V'_i = 0.75*V_i + 0.25*V_{i-1} \quad (i = N)$$

Analogous definitions are used for the more complex two- and three-dimensional features *SL*, *AT*, *DT* and *PF*. An example of penumbral matching is shown in Fig 6.11. While penumbral matching of the simpler distribution parameters is computationally no more expensive than simple matching, its use with more complex features is at present computationally unattractive because it requires the run-time generation of additional feature records. This difficulty could readily be overcome in a future version of the system if required.

6.6.3.6 Inner boundary position matching

If inner boundary position matching has been specified, the program then computes a difference measure on the basis of differences in the number and relative position of inner boundary centroids. This level of matching compares drawing and position feature records rather than the inner boundary records themselves. As with boundary feature similarity matching, the approach taken depends on whether global, local or existence matching has been specified. As far as possible, the approach adopted for a particular type of matching is continued in this section - global matching calculates difference measures on the basis of summary statistics such as inner boundary counts, local matching compares values of specific features, and existence matching looks for the presence of specific features. The matching process described below does not make exhaustive use of all feature types (in particular it makes very little use of data in boundary family records), but could readily be extended if a higher level of discrimination were required.

If global matching is specified, matching is limited to drawing records; a distance measure is computed between query and drawing on the basis of parameters *NB*, *CB*, *SB*, *IB* and *BF* (as defined in section 4.4.4), using the same metric as in section 6.6.3.2. If local matching is used, drawing records are optionally matched in the same way as for global matching. Position feature records *BP* (as defined in section 4.4.4, and illustrated in figs 4.3-4.5) are then matched as outlined in section 6.6.3.3. If existence matching is used, drawing records are optionally matched on the same parameters as for global matching, but using the techniques outlined in section 6.6.3.4.

Whichever method of matching is selected, the difference measure computed is then multiplied by a position weighting factor (supplied at run time) and added to the outer boundary difference measure as a combined indicator of shape difference.

6.6.3.7 Inner boundary shape feature matching

If the highest level of shape matching is specified, searching of inner boundary shapes, the program then attempts to compute a measure of inner boundary shape difference. For the reasons discussed in section 6.6.2, this involves finding the closest possible match for each query boundary, then summing the total difference measure over all query

(a)

Frequency range	Transformed frequency in query record	Transformed frequency in drawing record	Sum	Difference
0.1 - 0.3	2.25	1	3.25	1.25
0.3 - 0.5	3	2	5	1
0.5 - 0.7	4	3.25	7.25	0.75
0.7 - 0.9	3.75	3.5	7.25	0.25
0.9 - 1.1	1.75	3.25	5	1.5
1.1 - 1.3	0.25	1.75	2	1.5
1.3 - 1.5	0	0.25	0.25	0.25
	Total:		30	6.5

L_{qd} = 6.5 / 30 = 0.2167

(b)

Query features		Drawing features		Sum	Difference
Type	Frequency	Type	Frequency		
		D314	0.33	0.33	0.33
		D323	0.33	0.33	0.33
D324	0.33	D324	0.67	1.00	0.33
		D325	0.33	0.33	0.33
D333	0.33			0.33	0.33
D334	0.67	D334	0.33	1.00	0.33
D335	0.33			0.33	0.33
D344	0.33			0.33	0.33
D347	0.17	D347	0.33	0.50	0.17
D356	0.17	D356	0.33	0.50	0.17
D357	0.33	D357	0.67	1.00	0.33
D358	0.17	D358	0.33	0.50	0.17
D367	0.17	D367	0.33	0.50	0.17
		D542	0.33	0.33	0.33
D543	0.5			0.50	0.50
		D551	0.33	0.33	0.33
D552	0.5	D552	0.67	1.17	0.17
D553	1.0	D553	0.33	1.33	0.67
D554	0.5			0.50	0.50
D557	0.33	D557	0.33	0.67	0.00
		D562	0.33	0.33	0.33
D563	0.5			0.50	0.50
D566	0.33	D566	0.33	0.67	0.00
D567	0.67	D567	0.67	1.33	0.00
D568	0.33	D568	0.33	0.67	0.00
D577	0.33	D577	0.33	0.67	0.00
		Total:		16.00	7.00

L_{qd} = 7 / 16 = 0.4125

Fig 6.11 Effects of specifying penumbral matching on the two matching cases shown above (assuming both sets of features are complete). Note that additional feature records need to be created in the second case, where feature D_{xyz} *needs to generate penumbral neighbours in two dimensions, namely* $D_{x(y+1)z}$, $D_{x(y-1)z}$, $D_{xy(z+1)}$, *and* $D_{xy(z-1)}$.

boundaries to generate the required measure, which is again weighted and added to the combined indicator described in the previous section. Procedure *MatchFeatureSimilarity* (section 6.6.3) is used to compute a difference measure between each pair of boundaries.

The algorithm for selecting the closest-matching pair of boundaries for matching is virtually identical to the algorithm for selecting the closest-matching pair of boundary levels, described in detail in section 6.6.3.1, and is therefore not described again. The only major difference, as indicated in section 6.6.2, is the need to deal with the special case of a drawing with no inner boundaries.

6.6.4 Segment matching

6.6.4.1. Principles of segment matching

Segment matching as defined here provides a completely different way of matching query and stored shapes, and one that can usefully complement feature matching. It is in fact essential if Type B retrieval (identity matching) is to be provided. All a search of extracted global or local features can do is to confirm that query and stored shapes share a number of common aspects. A difference measure of zero does not prove that query and drawing shapes are identical. As discussed in section 6.3.3.3, the principle of expressing closed boundary shapes as plots of θ, the cumulative angle traversed, against s, the cumulative arc length, is well established. θ here is a single-valued function of s, which increases from 0 to 2π as the boundary is traversed completely in an anti-clockwise direction. (Some authors prefer a slightly different transformation, subtracting $2\pi s/S$ (where S is the total boundary length) from θ, to give a horizontal plot which effectively shows the extent to which the shape differs from a circle). Circular arcs in the original boundary are transformed into straight lines in the θ-s plot, with gradient indicating their curvature. Straight lines in the original boundary become lines with zero gradient in the plot, and vertices become discontinuities or vertical lines (Fig 6.12).

Such a plot can easily be used to generate a measure of the difference between two shapes, by summing the area between their θ-s plots (Fig 6.13) - or indeed by integrating any measure derived from differences in θ between query and stored shapes for a given s. As discussed in section 6.5, the process is usually a computationally expensive one because it has to be repeated many times over for each pair of boundaries being compared, using a different starting point each time, to find the closest-matching relative orientation of the two shapes. The present program uses what is effectively a heuristic approach by comparing each pair of query and drawing boundaries just once (at each level selected), using the canonical start point for each boundary identified in earlier processing. This approach has the advantage of greatly increased computational efficiency, requiring only a single pass through each set of line segments for each level - though it runs a risk of missing similar shapes where the canonicalization process has assigned different start points.

Calculation of a difference measure between two boundaries is best achieved, first by normalizing both boundary perimeters to the same length, then by subdividing both boundaries into an equal number of corresponding subsegments, each terminated when a discontinuity is encountered in either boundary (Fig 6.9). A distance measure can then readily be computed for each pair of corresponding subsegments, by integrating either the absolute magnitude or the square of the angular difference D between cumulative query angle θ_q and drawing angle θ_d over the length L of the subsegment. If the absolute difference is used, this gives a difference measure for each subsegment of:

$$M_a = 0.5 * L * (\mathrm{Abs}(D_s) - \mathrm{Abs}(D_f))$$

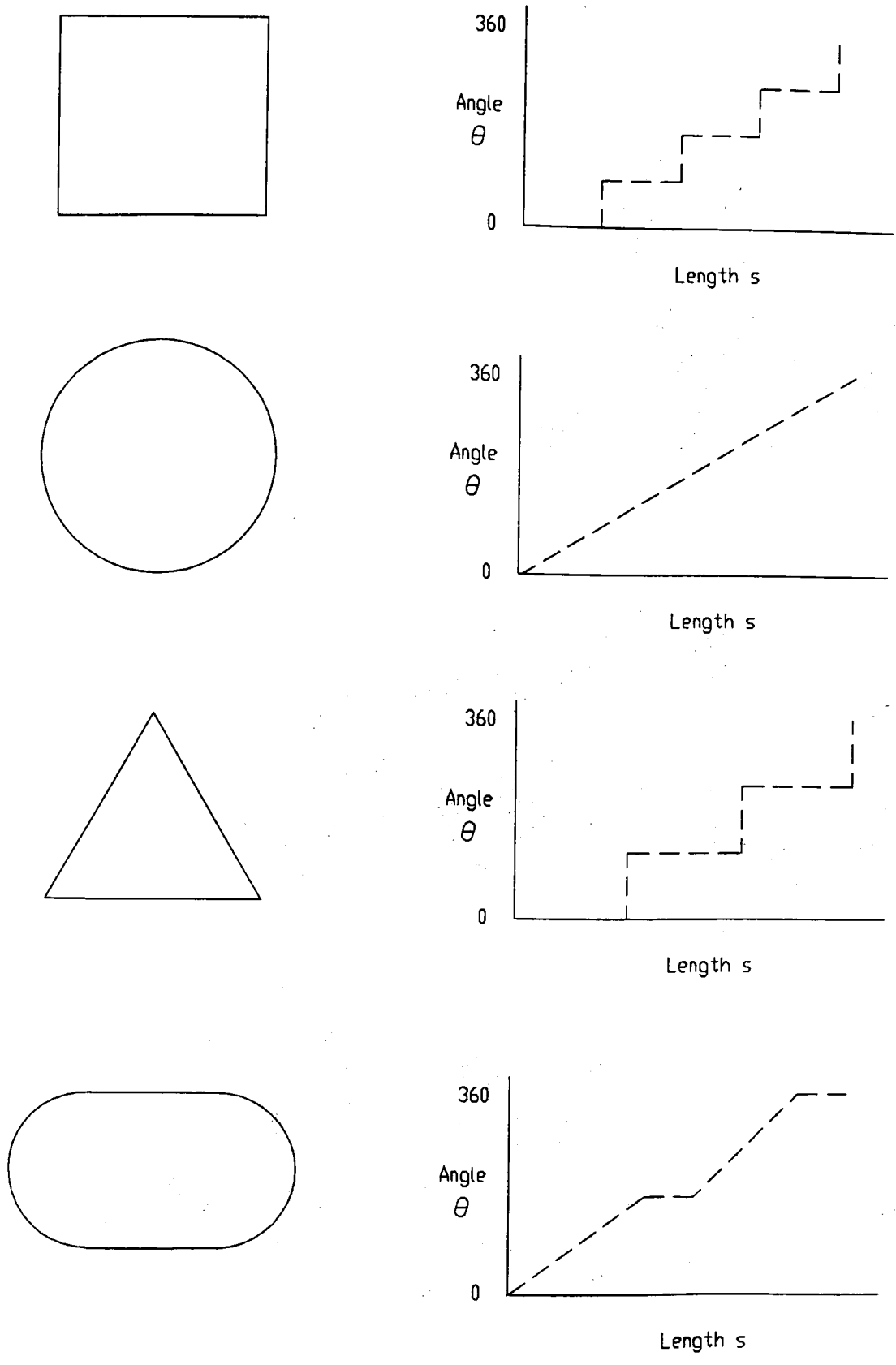


Fig 6.12 Some common shapes expressed as θ-s plots.

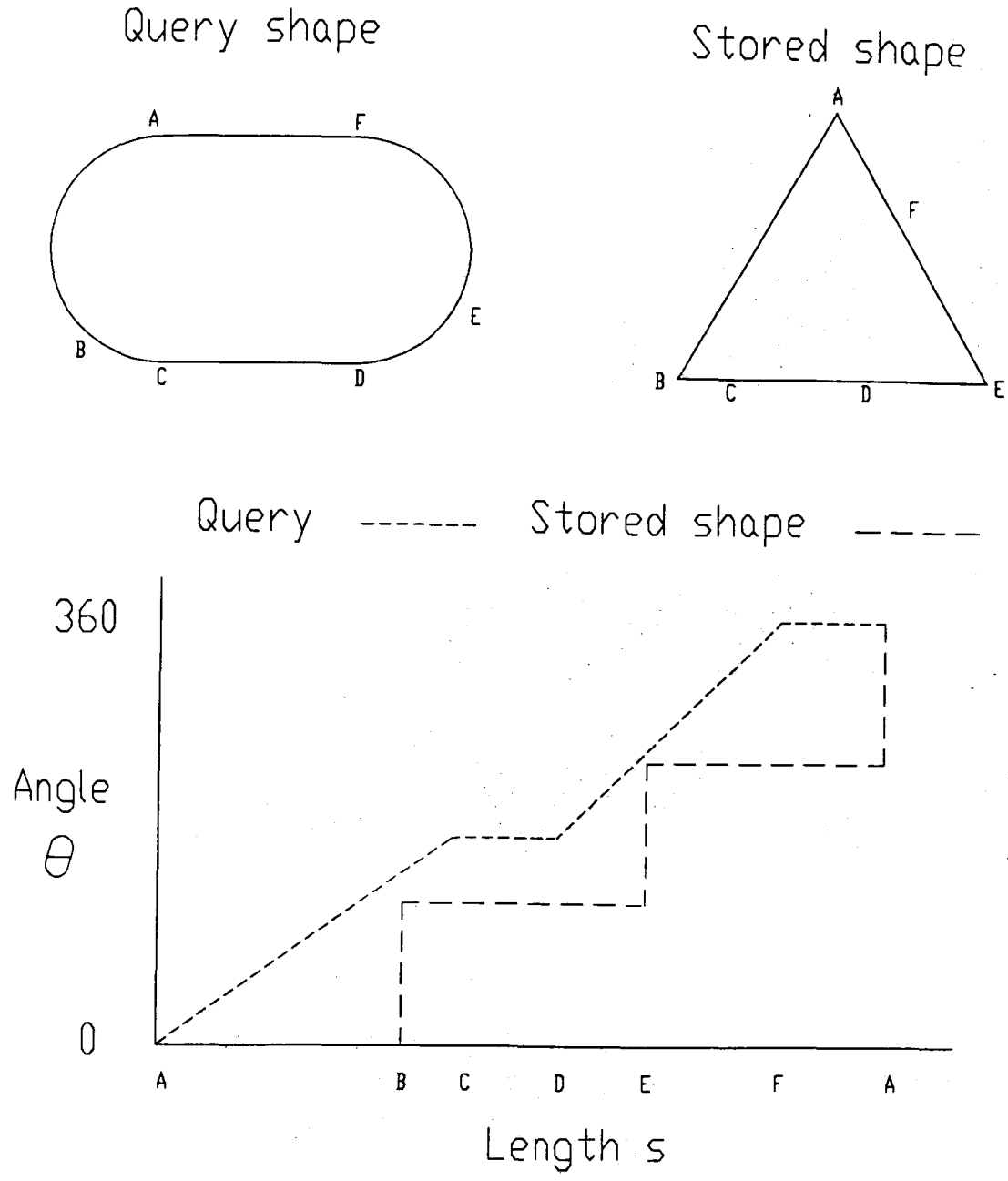


Fig 6.13 Similarity matching between query and stored shapes. Both shapes are divided into subsegments as shown in the top illustration. The lower illustration shows cumulated angle plotted against cumulated length for both shapes; the angular difference between query and stored structures is summed over the entire boundary to yield a global distance measure D *indicating the difference between the two shapes.*

where D_s is the difference between cumulative query and drawing angle at the start of the subsegment, and D_f the difference at the finish. If the square of the angular difference is used, the corresponding difference measure for each subsegment becomes:

$$M_s = L * (D_s * D_f + (D_s - D_f)^2 / 3)$$

where D_s and D_f are start and finish angular differences, as before. Whichever measure is used, it can then be summed over the entire set of subsegments making up one boundary level and used to give an overall measure of difference between the two boundaries.

6.6.4.2 The segment matching process

Segment matching of query and drawing boundaries begins with selection of the appropriate traversal level for each boundary. A range of options is provided, under which a pair of boundaries may be matched at

- top level only
- bottom level only
- *combined* matching, top-level followed by bottom-level matching,

or

- *all level* matching, successive matching of all query levels in a similar way to that outlined in section 6.6.3.1.

In the two latter cases, difference measures from different levels are added, normally weighting lower-level difference measures by an *attenuation factor* specified at run time.

For each query-drawing pair selected for matching, the following algorithm is invoked. It traverses query and drawing boundaries, repeatedly identifying corresponding subsegments and calculating θ and s values in each, and hence deriving a difference measure which is then cumulated over all subsegments. It computes the second measure M_s defined above - preliminary experiments suggested that this measure may be slightly less sensitive to very small differences in shape than M_a, though in most cases both measures give identical similarity rankings. The square root of M_s is in fact used in subsequent calculations.

Let L represent current normalized subsegment length,
S_q and S_d the (normalized) length of the current query and drawing line segments,
A_q and A_d the current query and drawing line segment arc angles,
D_q and D_d the current query and drawing line segment discontinuity angles,
L_q and L_d lengths traversed along current query and drawing line segments,
T_q and T_d cumulative angles traversed around query and drawing boundaries including the current subsegment,
P_q and P_d cumulative angles traversed up to the beginning of the current subsegment,
D_s and D_f the differences between cumulative query and drawing angle at the start and finish of each subsegment,
and M the overall difference measure to be computed. Then:

Calculate normalization factors required to set each boundary perimeter to 100, and zeroize L_q, L_d, P_q, P_d and M;

```
Get first line segments of query and drawing boundaries at
appropriate level;

If Sq <= Lq then
  Add Dq to Pq, get next query segment, and zeroize Lq;
If Sd <= Ld then
  Add Dd to Pd, get next drawing segment, and zeroize Ld;

While more query or drawing segments remain do

  Begin

  Set L = min (Sq-Lq, Sd-Ld);
  Set Tq = Pq + L * Aq / Sq;
  Set Td = Pd + L * Ad / Sd;
  Set Ds = Pq - Pd, and Df = Tq - Td;

  Add L * (Ds * Df + (Ds - Df)2 / 3) to M;
  Set Pq = Tq, and Pd = Td;
  Add L to Lq and Ld;

  If Sq <= Lq then
    Add Dq to Pq, get next query segment, and zeroize Lq;
  If Sd <= Ld then
    Add Dd to Pd, get next drawing segment, and zeroize Ld

  End.
```

6.6.4.3 Inner boundary matching

The matching process above can obviously be applied to outer or inner boundaries. However, a complete identity match requires more than an indication that all boundary shapes are identical. It needs also to establish identity of position and size. Hence a complete inner boundary match involves a pairwise comparison of query and drawing boundaries (in the same sequence as that used for inner boundary feature matching, section 6.6.3.7), computing the following difference measures for each query-drawing boundary pair:

(a) Position difference, the Euclidean distance between query and drawing boundary centroid positions, calculated as

$$PD = (CD_q^2+CD_d^2-2*CD_q*CD_d*\cos(CA_q-CA_d))^{1/2}$$

where CD_q, CD_d, CA_q and CA_d are respectively centroid distances and centroid angles of query and drawing boundaries. In use, the measure is normalized by dividing by the average centroid distance for all inner boundaries in the database.

(b) Class and size difference, a measure similar to the initial screening measure described in section 6.6.3, except that as well as comparing *AL*, *LW* and *PA* ratios, it also computes measures of size and orientation similarity (unimportant at the outer boundary matching level, but important in distinguishing between shapes which are identical in every respect except relative inner boundary size or orientation). The additional terms in the measure are perimeter difference ($\log_2$ of the ratio of query and drawing boundary perimeters) and axis difference, the angle between major axes of query and drawing boundaries after allowing for possible rotational symmetry.

(c) Inner boundary shape difference, calculated as described in section 6.6.4.2 above.

The three difference measures for each query-drawing boundary pair, weighted according to parameters supplied at run time, are then added to generate a total inner boundary difference measure, which in turn is added to the outer boundary measure to give a final measure of difference between query and drawing shapes.

As with feature matching, the matching process need not involve all levels. Any of the following four levels of matching may be specified at run time:

- outer boundary shape only
- outer boundary shape plus inner boundary position (using measure (a) above)
- outer boundary shape, inner boundary position, class and size (using measures (a) and (b))
- complete matching of all boundaries (using all the above measures).

6.6.5 Accumulation and display of results

As indicated in section 6.6.2, the program successively matches the query file with all drawings from the specified target set within the database, building up a list of retrieved drawings in order of similarity, using the specified match type. At the end of the matching process, this is used to generate a display of retrieved drawings in an appropriate format, specified at run time. Further details, and examples of output, are described in the next chapter (section 7.7).

Once results have been displayed, the program is ready to accept the next query, using the same set of run-time parameters to specify the level and type of matching.

6.7 Mirror images

A decision was taken at an early stage to regard the mirror-image of an unsymmetrical object as a separate shape. Despite the fact that two-dimensional objects such as those represented in the test database can be converted to their mirror-images simply by turning them upside-down, it was decided to allow mirror-images to retain their separate identity, for two reasons:

- parts may have different finishes on upper and lower surfaces, which are therefore not freely interchangeable;

- in any future extension of the system to cover 3-D object representations (which cannot readily be converted to their mirror-images without passing into the fourth dimension), it will be important to be able to distinguish a face from its mirror-image.

In the meantime, however, the system needs to be able to make the connection between mirror-images where necessary. This is an easy task in some areas, less so in others:

- all global and most local features are completely symmetric, so that matching on these features, however performed, will always retrieve mirror-images. The only exceptions to this are the local triplet features *AT* and *DT*, which are inherently directional, and are thus unsuitable for use if mirror-images are to be retrieved.

- all inner feature boundary pattern features are inherently symmetric; again, matching of these features will always retrieve mirror-images.

- boundary segment matching as described in section 6.6.4 is obviously directional, and thus provides the most reliable means of distinguishing between mirror-images if this is required. A problem obviously arises when no distinction is to be drawn between mirror-images. This is overcome in the present system by the expedient of matching each drawing boundary twice with each query boundary, once traversing the query boundary in the same direction as the drawing boundary, once in the reverse direction. Even though this provides only for mirroring about an axis perpendicular to the canonical start line, it appears to provide an effective (if rather inefficient) solution in practice.

- inner boundary position, class or shape matching requires inner boundaries to be processed in a set order, starting with those furthest from the outer boundary centroid and working gradually inwards. Within a group of boundaries whose centroids all lie a fixed distance from the outer boundary centroid, boundaries are arranged anti-clockwise, in centroid angle order (section 3.2.3). The program therefore generates a second ordering for query inner boundaries, in order to be able to match drawing inner boundaries in mirror-image order: again, not a totally satisfactory solution, as it deals with only a single plane of mirroring.

The success or otherwise of these expedients can be judged from the results presented in chapter 8.

6.8 Efficiency considerations

Although computational efficiency was considered in section 6.4 to be a design criterion of minor importance, some discussion of the question is clearly warranted here. In many ways, the efficiency of a system such as this is unusually difficult to evaluate, as it falls into the category neither of data processing systems where processing requirements are so simple that one can assess performance purely by considering file access requirements, nor of complex mathematical or scientific software where I/O considerations can be ignored. Both central processing and disc access contribute significantly to overheads in a system such as this. To assess system efficiency in detail would be a significant task in its own right.

One can however make a number of general observations. Firstly, all algorithms used here are polynomial-bounded, setting a reasonable upper limit on the resources they consume. Most are in fact simple $O(n)$ or $O(n^2)$ processes. For example, feature matching of a single boundary requires one simple set of comparisons for each drawing level processed, provided global matching or local/existence matching using the simple features located in boundary level records are used. Disc access is almost certainly the rate-limiting step here; the matching process is hence of complexity $O(n)$, where n is the number of levels processed in searching the average boundary.

How does this relate to q and d, the number of levels in query and drawing boundaries? In the best case, no backtracking at all takes place, so the total number of levels processed is $\text{Max}(q,d)$. In the worst case, $\text{Abs}(q\text{-}d)$ abortive matches take place for each successful one, so the number processed is $qd - \text{Min}(q(q\text{-}1),d(d\text{-}1))$. In the average case, one could assume $\text{Abs}(q\text{-}d)/2$ abortive matches between the first pair of boundaries, a number that would effectively halve each time, giving an estimate of $\text{Max}(q,d)+\text{Abs}(q\text{-}d)$. The overall complexity is thus certainly at worst $O(qd)$, and in many cases will be effectively $O(d)$ for query shapes with relatively few levels, and $O(q)$ for more complex query shapes.

Local or existence matching using the more complex features housed in boundary feature records is a less efficient task, as the entire set of feature records for a single level of the drawing boundary has to be read in from disc and compared with corresponding query records. Whether the rate-limiting step is the reading in of feature records or their

comparison remains to be established. In either case, however, the processing load per level is effectively proportional to the average number of feature records per boundary level in the stored drawing. If penumbral matching is used, additional features have to be generated and inserted in an ordered feature list, an $O(n^2)$ process.

Inner boundary matching uses a version of the level-matching algorithm analysed above to select boundary pairs for matching. It is therefore reasonable to assume that processing times for matching Q query inner boundaries with D from the drawing will in the best case be proportional to $\text{Max}(Q,D)$, in the worst case QD - $\text{Min}(Q(Q\text{-}1),D(D\text{-}1))$, and in the average case, $\text{Max}(Q,D) + \text{Abs}(Q\text{-}D)$ - again at worst $O(QD)$.

Segment matching is obviously a more computationally-expensive process than feature matching, as a complete sequence of line segments has to be matched for each level, not just a single level record. The process is analogous to local/existence matching using boundary feature records, as discussed above. Again, it remains to be established whether the rate-limiting step is the reading in of segment records or their comparison, but in either case the processing load per level is effectively proportional to s, the average number of segment records per boundary level. The load is normally limited in practice by restricting matching to just top and bottom levels rather than comparing all levels, as for feature matching. Segment matching at the inner boundary level is the most expensive task of all, as the entire process has to be repeated at least $\max(Q,D)$ times, which could result in a worst-case complexity of $O(n^3)$ where n is the total number of segment records to be compared per drawing.

Some quantitative figures on cpu usage by program RETRIEVE are presented in section 8.4.6 below.

CHAPTER 7. INTERFACE DESIGN

7.1 General considerations

The design of a suitable user interface for input of queries and display of results is an area of crucial importance in determining user acceptability. The problems involved in providing users with the means to formulate queries of the six types distinguished in the previous chapter are far from trivial. The systems designer is placed in an awkward situation by the very nature of the subject. The accurate formulation of a graphical query is inevitably a tricky and time-consuming task, and one that may well require a certain modicum of drawing ability. Users have to work much harder to make their wishes known to the system than with a conventional DBMS or bibliographic retrieval system, and hence are likely to be less tolerant of poor system performance.

The question of who would use such a system also has to be addressed. Although "user-friendly" query languages such as SQL have been offered by commercial database management systems for many years, experience suggests that few engineers or managers actually use such facilities direct, except for very straightforward queries. Even fewer managers use bibliographic retrieval services direct. In both cases, an intermediary with a detailed knowledge of the database and the query language generally inputs the actual query, and forwards the results to the original enquirer. The development of graphical front-end interfaces (see section 7.2.2 below) may alter this situation - though if one remembers that in their time both SQL and COBOL were hailed as end-user languages that would make programmers redundant, a measure of cynicism is justified. The implications for graphical database design are probably that one should aim where possible for a system that can be used both by inexperienced and skilled operators, and leave the question of who should use such systems to be settled later.

Four main types of interface have so far been used by graphics database designers:

- Command languages (essentially text-based)
- Menu-driven interfaces (text or graphics-based)
- Example-based interfaces (text or graphics-based)
- Novel graphics systems.

Some of these appear to be inherently more suitable for some types of query than others. By analogy with bibliographic retrieval systems, one might for example expect a command language to prove most suitable for type D retrieval (Boolean feature combination), while an example-based interface would seem ideal for type F retrieval (similarity matching).

7.2 Some existing systems

7.2.1 Command language-based systems

This type of interface, where users type in a series of (textual) commands from a formally-defined *command language*, has been the standard method of access to both database management and information retrieval systems for many years. Some, like SQL, are in such widespread use that they have become international standards. It is therefore not surprising that most early image database systems relied almost exclusively on command languages. Database researchers were relatively familiar with their capabilities, particularly in formulating complex queries, and found a worthwhile challenge in extending them to cover new types of data. One of the earliest systems of this kind was GRAIN, already described in some detail in Chapter 1, above. The GRAIN language, based on relational algebra, provided conventional operators allowing retrieval of images by Boolean combinations of features, as well as additional commands to display retrieved

pictures as line drawings or raster images, and to invoke user-defined similarity measures. It also had a limited ability to handle queries on spatial relationships between items in pictures.

Similar picture query languages described in the literature have included SQUEL (Spatial QUEry Language), described by Herot (1980), and ISQL (Image Structured Query Language), described by Assmann et al (1984), based respectively on the conventional query languages QUEL and SQL. Perhaps more surprisingly, the more recent GRIM_DBMS described by Rabitti and Stanchev (1989) also relies on a text-based command language for query formulation.

7.2.2 Menu-based systems

Systems where users formulate queries by combining text descriptors or icons selected from screen displays, are attractive both for their simplicity for casual users, and for their ability to mix text and graphics in queries if required. Perhaps the most relevant example of a pure menu-based interface is the DLink feature-based design system described by Patel (1985). Here, the user chooses desired structural features from a list displayed on the screen, and combines them to generate a representation of the desired design object to be fed into a geometric modelling package. A query structure representing a complete or partial design could be built up in exactly the same way, by combining desired features (represented either as text or as icons) with specified orientation.

Pure menu-based systems tend to lack flexibility, and can prove tedious in operation for experienced users. A more recent development that is rapidly gaining in popularity is the WIMP environment (the acronym standing variously for Windows, Icons, Menus & Pointers or Windows, Icons, Mice & Pull-down menus, according to taste). This provides the user with a separate screen window for each task being undertaken, a mouse or similar pointing device to select appropriate actions (often represented on the screen by icons) or choices from menus which are "pulled down" when required. Environments of this type are increasingly being used to provide graphical front-ends to conventional databases (e.g. Burgess (1986), Wu (1987) and Kim et al (1988)), information retrieval systems (Frei and Jauslin, 1983), and multi-media databases (Frasson and Er-Radi (1986), Leong et al (1989)). While each system described in the literature differs in detail from its neighbours, most offer the user a graphics environment in which they can build up a query, examine the structure of the database, browse through a thesaurus of permitted index terms (in the case of the bibliographic system), or display the results of a search. The use of high-resolution bit-mapped screens allows full mixing of text and graphics, and some systems can handle surprisingly complex queries (Kim et al, 1988).

7.2.3 Example-based systems

Systems where the user provides a text or graphical example of the kind of output desired, and the system then searches for the best possible match(es), are again a potentially attractive proposition where graphical input of queries is required. The QPE language described in Chapter 1 obviously falls into this category. Like its text-based parent QBE, it allows the user to formulate queries by filling in a specimen of desired output on a table displayed on the screen. Unlike QBE, it can display answers in graphical form (as a sketch or raster image), and permit entry of spatial queries via a light pen, trackball, or joystick. A wide range of pictorial operators is also supported, allowing the computation and display of the centre point of a line or region, the line between the nearest points in two regions, or the union or intersection of any two lines or regions. However, the majority of commands remain essentially text-based.

Another example-based system briefly described in the literature is ARES (Ichikawa et al, 1980). While few details of the system are available, it apparently allows users to input a sample image which is then subjected to a feature extraction process involving application of error-correcting codes to the bit map of the image, and then matched with

existing images in the database. The form in which the query image is submitted to the system is not described.

7.2.4 Novel systems

One completely novel approach to pictorial data retrieval is that of Herot (1980). His Spatial Data Management System (SDMS), though conceived as a graphical query interface for a conventional relational database, is in many ways even better suited to pictorial databases. Entities in his prototype system were displayed to the user as icons (warships, displayed as silhouettes, were used as examples). A "world view" of the entire database was shown on one screen; every ship was displayed as a miniature icon, colour coded to indicate its state of readiness, and grouped to show present location. A second screen allowed the user to "zoom in", showing any desired part of the database (e.g. ships in the Indian ocean) in greater detail. As the user homed in on a particular ship, its icon was magnified to fill the entire screen, and additional data about the ship (tonnage, speed, present commander) automatically displayed. This kind of data organization proved particularly useful where users simply wished to browse through the database, or had difficulty in formulating their queries. It was supplemented by a more conventional query language which could be used to put specific queries to the system.

7.3 Interface design criteria

7.3.1 Query formulation

Criteria for selecting query languages are by no means clear-cut. In their review of query languages for pictorial databases, Chang and Fu (1981) argued that the most effective approach to query language design was to extend existing text-based languages, rather than to design new ones from scratch. While conventional query languages were unable to handle the full range of pictorial queries, those based on the relational database model could provide a substantial proportion of the required query facilities, and could readily be extended without distorting their essential structure. The use of an existing relational DBMS as the basis for an image database system simplified the design task, as well as providing useful database features such as applications independence, security and integrity.

This view now seems rather naive. Chang and Fu made no attempt to characterize the type of user they had in mind when comparing query languages - but it seems clear that the needs of end-users such as geographers or engineers were given scant consideration. Few end-users have ever been happy with conventional database query languages even for numerical and text applications - hence the increasing interest (see below) in graphical front-ends to conventional databases. It is completely unrealistic to expect engineers to formulate queries to a graphical database in a dialect of SQL!

A more thoughtful review of graphical interface requirements comes from Kim et al (1988), in the presentation of their graphics-based database query language PICASSO - which is explicitly designed with end-users in mind. They stress the ease with which inexperienced users can use graphics-based interfaces to build quite complex queries, and to explore the structure of a database if required. They suggest four design principles for graphical interfaces, as follows:

- The graphical interface should be able to provide information to the user about the structure of the underlying database;

- The user should be able to formulate queries incrementally, using the results from one step as the starting point for the next;

- There should be a facility allowing the user to browse the database freely;

- Graphical feedback should be provided to assist query formulation.

Later workers such as Leung et al (1989) have endorsed and extended these principles, adding two more:

- There should be consistency between different parts of a language that deal with the same type of information;

- The level of detail provided by the system at any given point should be user-selectable.

While some of these points are debatable (how much does an end-user *really* need to understand about the underlying structure of a database?), they do provide a useful context for evaluating potentially useful features for new systems.

7.3.2 Display of results

As proposed above, a graphics database system should be able to provide users with a choice of output formats. At the lowest level, a graphical query could be satisfied by display of drawing number or design file name, leaving it up to the user to examine the drawing or file to establish whether it meets the criteria specified. Most users, one feels, would expect a drawing of each retrieved object to be displayed, probably in order of presumed similarity to the query. Some users would undoubtedly require a copy of the original design file, on which to base a modified design.

The difficulty of precisely specifying search criteria in advance is also recognized above, as the need to provide a means of incremental query formulation, so that the user can if necessary apply a trial-and-error process of formulating a query, running it, examining the results, and modifying the query accordingly. This clearly implies the need for a system to be able to display results from intermediate search steps at any time. It also suggests that the technique of *relevance feedback* (Salton, 1971) may be of use. Here, the user first formulates some approximation to a query, the system displays the results, and the user selects the items which appear most relevant. The system then uses this feedback to modify the query, repeating the process until the user is satisfied, or no further improvement in retrieval performance can be achieved. In a situation where many user queries may well be ill-defined, such feedback would clearly be highly desirable. Its implementation as part of a similarity retrieval system is not fundamentally difficult - the system could simply use drawings selected by the user as queries in their own right.

7.3.3 Applicability of existing types of interface

Command languages (with suitable display facilities for retrieved drawings) can readily handle type A and D queries as defined above, for either 2-D or 3-D objects. They can be used for incremental query formulation, in browsing, or to show database structure if required. Their limitations stem from the fact that they are essentially text-based. Even when extended to allow specification of shape or position information, they are far too unwieldy for anyone but the most dedicated enthusiast. As far as any end-user is concerned, they are effectively unable to handle any type of query which requires graphical input.

Menu-based query formulation could be used for all types of retrieval, particularly those involving graphical input (types B, C, E, and F). If used intelligently in a windowing environment, it can prove sufficiently flexible to be acceptable to most users, both for specific query formulation or browsing. Incremental query formulation guided by display of intermediate search results can easily be provided. Menu-based methods may well prove the only feasible means of formulating 3-D queries.

Example-based query formulation is an attractive proposition for query types B, C or F if sample queries can be easily built up on the screen, or submitted in the form of sketches. The method is not really suitable for incremental query formulation, and would be difficult to adapt for 3-D queries, but is well suited to browsing. The main difficulty lies in ensuring that the example has been described to the system with sufficient accuracy - a particular problem with sketch input, where some degree of interpretation is necessary ("are those two sides supposed to be parallel?"). Several sketch interpretation systems have been described in the literature (e.g. Liardet et al (1978), Kato et al (1982)); these could be used to "clean up" freehand sketches of desired objects prior to feature analysis and database query as described above. In principle, a language like QPE could also be used for example-based retrieval. However, its textual bias means that it suffers from the same inherent problems as command languages, and is thus only really suitable for type D queries.

Novel methods of data organization and display such as Herot's could readily be applied to an engineering database, particularly if used in conjunction with some form of clustering or parts classification scheme, as suggested in section 6.3.4. Identifying parts similar to a given object would involve displaying a "world view" of icons representing typical members of each part family; "zooming in" to a selected part family would then display each of its individual members, allowing users to identify parts of interest to them. From the user's point of view, the advantages would be considerable - no query language to learn; no need to define the query object in any detail; no draughting ability required; no problems with 3-D objects (as long as desired features could be recognized from the 2-D drawing). It could well be the method of choice for handling type F (similarity retrieval) queries - though it would be of little use for formulating specific queries of types B, C or D.

7.4 Interface design for the prototype system

The discussion above has highlighted the need for a shape retrieval system to operate in a wide variety of query and display modes. No single type of interface is likely to be suitable for all types of user and all types of query. Any attempt at tailoring interface design to the needs of specific users at this stage would be doomed to failure - no reliable information is yet available on how searchers would actually use such a system.

With this and the design criteria discussed in Section 7.3 in mind, therefore, the eventual aim of the prototype system is to offer the following query interfaces:

1. An example-based interface, where a complete or incomplete query structure can be built up on the screen, and then submitted for feature extraction and shape matching as described in Chapter 6. It is expected that this would be the main method for specifying type B, C or F queries. Query formulation would use either keyboard input or a mouse or similar pointing device to specify line type, size and position - the former where accurate query specification was paramount, the latter where speed and ease of use was of more importance. A relatively straightforward extension to the system could allow users to select any retrieved drawing and find those drawings most similar to it, thus providing a measure of relevance feedback.

2. An extension to (1), in which input could be submitted to the system as a rough sketch, which could be digitized either by following its outline with a stylus and tablet, or placing the sketch in front of a digital camera, extracting lines from the resultant image, and "cleaning up" the sketch as outlined in section 7.3.3.

3. A menu-based interface, allowing users to select and combine query features (represented as text descriptors or icons), perform Boolean searches on the database, examine results and amend queries where required, much as envisaged by Frasson and Er-Radi (1986). This would be the prime method for formulating type D queries.

4. A browsing interface, based on Herot's ideas as described in the previous section. Drawings would be clustered according to similarity measurements computed as specified in the previous chapter, and icons representing each cluster displayed on the screen. Selection of an icon would then allow display of all drawings (or sub-classes) in the cluster; selection of an individual drawing would display full information about the object and its drawing file. Inexperienced or casual users would normally access the database in this mode.

Between them, these interfaces offer a comprehensive set of query specification tools, with the ability to handle a range of query types and interaction modes. However, they cannot all be integrated with the shape matching process as closely as one would like. The size and complexity of the shape file needed to represent an example-based query (see Fig 6.3, in the previous chapter), plus the need to perform canonicalization and feature extraction on the query shape before matching takes place, mean that formulation of example-based queries and shape matching must remain loosely-coupled processes. By contrast, menu-based and browsing interfaces do not require complex intermediate files to link with the shape-matching process, and can therefore be tightly coupled. The main implication of this is that incremental query formulation should be much easier for menu-based than example-based queries, and response times should be significantly shorter.

Implementation of the full range of options was considered too ambitious a task to be attempted within the present project, particularly as some options required special-purpose hardware which was not readily available. Option 1 was selected for implementation in the initial prototype, on the following grounds:

- it provides a means of formulating the potentially most useful types of query (similarity matching and partial shape matching);

- its retrieval performance is readily susceptible to evaluation, unlike, say, option 4, where there is no objective means of assessing how well the system has clustered or displayed the shapes in the database;

- it could be readily implemented without recourse to special-purpose hardware.

7.5 Implementation of the query formulation module

7.5.1 Overview

The overall purpose of the query formulation interface is to allow a user to build up an example query shape on the screen, which can then be subjected to feature extraction and shape matching as indicated in the previous chapter. This effectively requires that the user is provided with a graphics editor, though one with strictly defined capabilities. Valid queries obey similar syntax rules to valid drawings - one or more boundaries, each consisting of one or more line segments, each of which may be a straight line or circular arc. No boundary segment may touch or intersect any other segment. The only difference is that query boundaries do not need to be closed.

The query formulation program thus needs to be able to build up such query shapes on the screen, to allow the user to verify their correctness and modify them if necessary, and to store the results in a form that can then be subjected to feature extraction. The most straightforward way to achieve this is clearly to set a cursor at some (arbitrary) start point, and let the user specify length, type and direction of each line segment until the query boundary is complete. Each line segment is assumed to start where the previous segment finishes. Limiting the user's freedom of action in this way makes it much easier to ensure the validity of the final query. Incomplete boundaries can be handled in a

number of ways, but the most satisfactory is to allow the user to define dummy segments which can connect two normal boundary segments (Fig 7.1). This has the advantage that queries involving two stretches of boundary which are not physically connected, but whose relative position is still important, can readily be specified. It also greatly simplifies the process of feature extraction, as programs SKELETON, CANONGEN and DATALOAD (all of which rely on shape boundaries being closed) can be readily adapted to process queries with dummy segments.

Some of the more detailed design options were constrained by the available hardware and software - DEC VT240 terminals (providing what by today's standards are only medium-resolution graphics displays), and GKS (Graphical Kernel System) graphics. GKS, an international graphics standard defining a set of routines which can be called from applications programs to provide notionally device-independent graphics input and output, can in theory support virtually any type of interaction between user and machine. However, the low level and stereotyped nature of the functions it provides acts as a severe deterrent to the designer's imagination. For example, GKS provides no functions to draw a circle on the screen, or to accept numeric input from the keyboard, leaving this to the applications programmer. There is thus a very strong incentive to choose a design that is simple to program, rather than one that will prove easy to use.

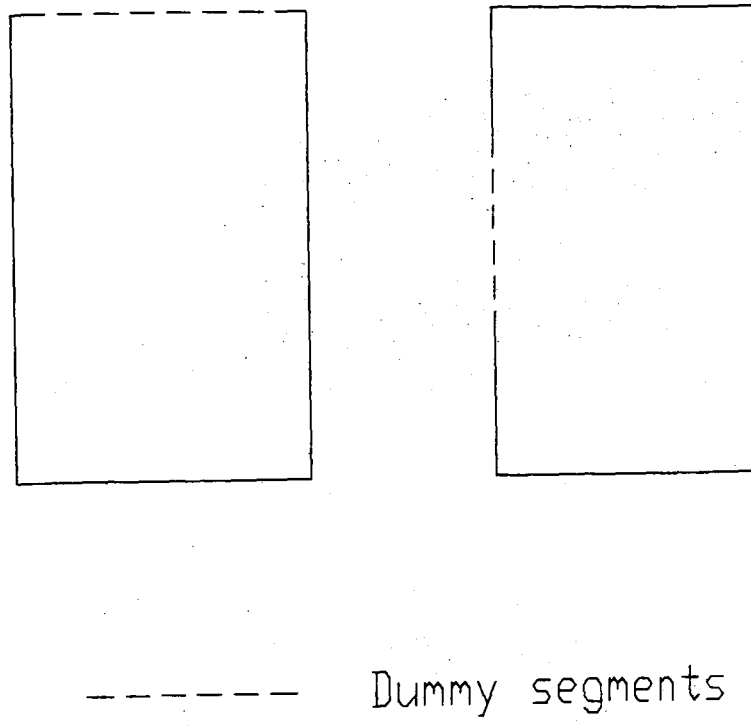


Fig 7.1 Use of dummy segments to indicate incomplete query shapes. In the first example, the three solid segments constitute the query, the dashed dummy segment notionally closing the boundary but playing no part in query matching. Hence this query should match with rectangular shapes of any length. In the second example, the dummy segments indicate that either long side may include protrusions or depressions - though the relative positions of the two halves of the query are important, so that it should match only shapes derived from rectangles of similar length/width ratios.

Some features of GKS did prove useful. It is relatively easy for the programmer to set up (text) menus from which the program user can choose with a suitable pointing device (mouse or arrow keys) - though the position of the menu on the screen has to be specified in device-dependent coordinates. Considerable use was made of such menus for guiding the flow of control: at each stage, a menu of legal commands was displayed on the right of the screen, thus preventing the user from selecting an inappropriate command, and reducing the amount of error-handling necessary.

Two main options were available for allowing the user to indicate the length and direction of each line segment: keyboard input, where the user was prompted to enter the relevant numeric values from the keyboard (as text strings, which were then converted to numeric values by the applications program), and *locator* input, where a graphics cursor was moved around the screen (by mouse or arrow keys) until it reached the desired position, when depression of a mouse button or the <enter> key made its current coordinates available to the program. Perhaps surprisingly, the former type of input proved much more successful in practice for query formulation, as the accuracy of locator input did not prove sufficient to ensure that supposed right angles really were 90^{o}, or that opposite sides of a supposed rectangle were equal in length within the tolerances required by the feature extraction programs. Keyboard input was thus used exclusively in evaluating the initial prototype.

7.5.2 Detailed program operation

On startup, program QUERYFORM displays an initial explanation screen, followed by an initial choice menu, inviting the user to start constructing a drawing boundary or to exit the program. Assuming the user chooses the *Start boundary* option, the system then creates a header record for that boundary, and prompts the user for the position and start direction of the first line (see Fig 7.2). Queries are built up in a screen window initially spanning an X-coordinate range of -120 to +120, and a Y-coordinate range of -100 to +100. If the user simply hits the <enter> key, the program chooses default values for boundary start position and direction. Note that although the feature extraction and shape matching programs make no use of Cartesian coordinates of line end-points or arc centres, an arbitrary coordinate system has to be used in the query formulation program, so that GKS can display the query shape on the screen. Most of this can be hidden from the user, but some fixed starting-point and direction for the whole process have to be specified.

The user is then invited to choose one of the following commands:

Add line: add a new straight-line segment to the current boundary
Add arc: add a new circular arc segment to the current boundary
Add dummy: add a new dummy segment to the current boundary
Quit boundary: abort construction of the current boundary.

In all but the last case, the system prompts the user for appropriate parameters to define the current line, creates an appropriate segment record to hold these parameters, and displays the resulting line on the screen - as a solid line for a straight-line or circular arc segment, as a dashed line for a dummy segment. If the current boundary is not closed (i.e. boundary start and end points are not coincident), the user is then prompted for the next command, now having a slightly wider choice:

Add line: add a new straight-line segment to the current boundary
Add arc: add a new circular arc segment to the current boundary
Add dummy: add a new dummy segment to the current boundary
Undo last: remove the last line from the current boundary
Rescale: rescale the current query display
Close boundary: add a segment to close the current boundary
Quit boundary: add a dummy segment to close the current boundary.

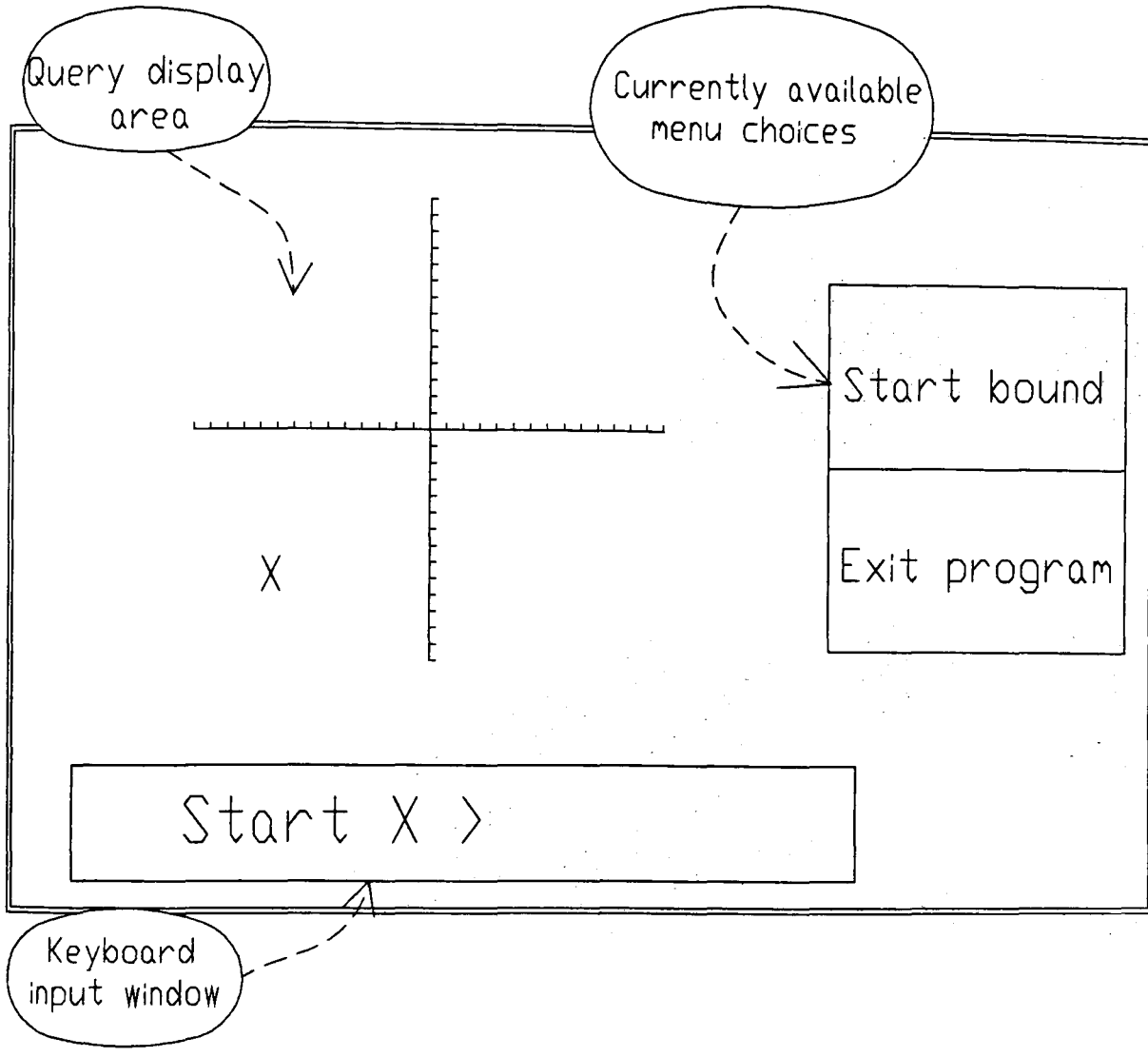


Fig 7.2 Query input screen at start of boundary creation process. Note use of three separate screen areas for query shape display, keyboard input and command menu. Coordinate axes are displayed only when boundary creation is first initiated; the cross in the query display area denotes the default start position for the current boundary.

The *Add line*, *Add arc* and *Add dummy* commands have the same effect as before, except that a new line or arc segment is validated by checking that it does not intersect any existing boundary segment. If it does, the segment is rejected, and the user invited to enter another command. *Undo last* deletes the last line from the current boundary, and may be used repeatedly; *Rescale* changes the scale of the query display, allowing the user to zoom in or out as required. *Close boundary* may be used as a quick way to terminate boundary construction, allowing the system to calculate the length and direction of the last segment; *Quit boundary* effectively allows construction of an incomplete boundary to be terminated, adding a dummy segment for compatibility purposes.

A typical input screen with a partly complete query shape is shown in Fig 7.3.

Once the boundary is complete, the user is invited either to enter another boundary or quit the program. There is no limit to the number of boundaries (or segments) that may be entered, though the operation of the GKS segment-drawing routines and the validation procedures which ensure that no line segments intersect make the entry of a complex shape a somewhat tedious process. If the *Quit program* option is chosen, and at least one valid boundary has been created, the program will write the "raw" query formulation to file, for input to the feature extraction stage.

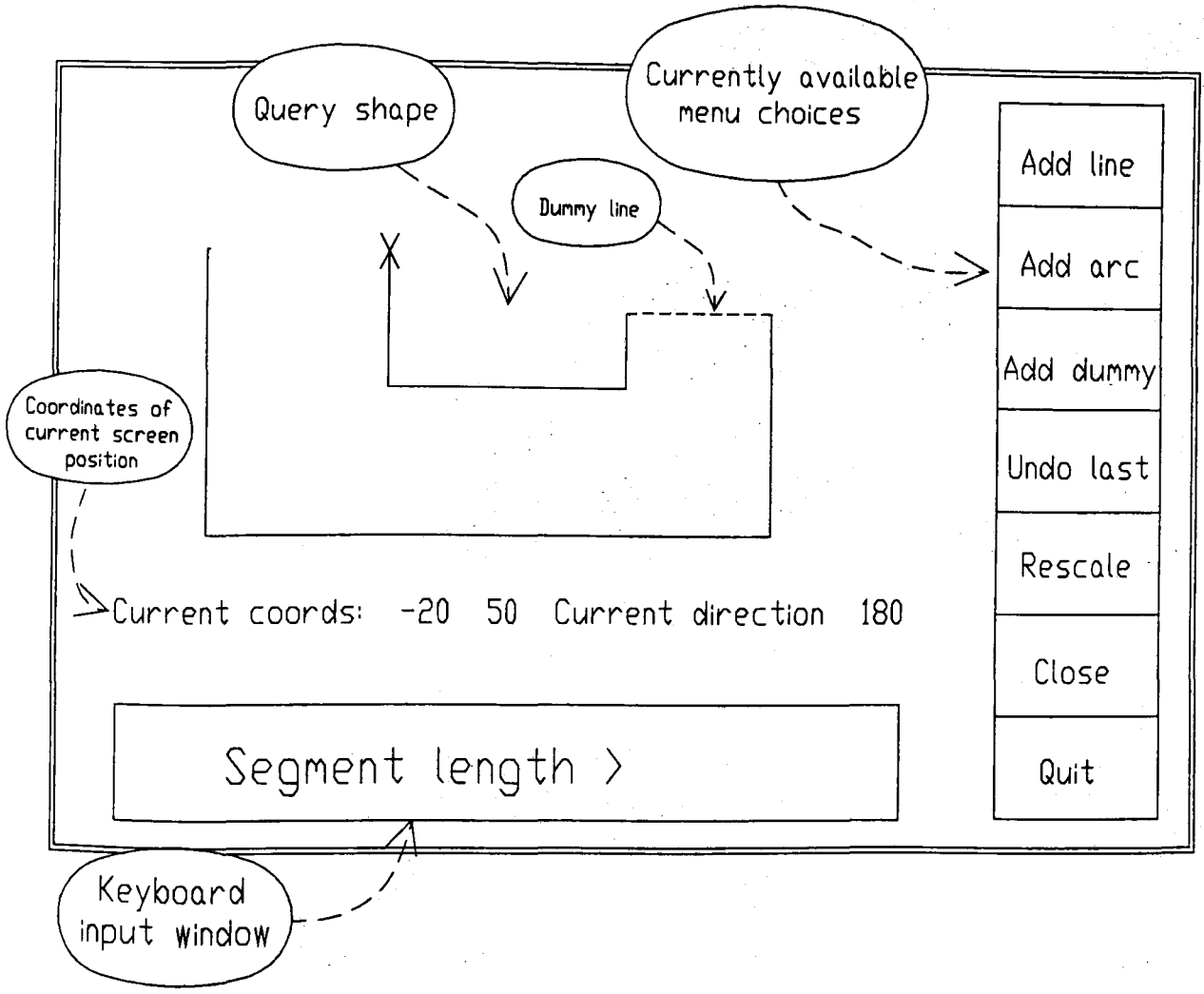


Fig 7.3 Query input screen showing partly-completed query shape. Note representation of dummy segment as dashed line.

7.6 Feature extraction

Before the "raw" query shape can be matched with shapes in the database, it must be converted to a similar form. This process involves three separate steps, each corresponding to one processing stage for stored shapes (see Chapter 3):

1. generation of a hierarchical boundary representation;
2. casting the representation into invariant form;
3. feature extraction and query file generation.

Building a hierarchical boundary representation is accomplished by program QUERYSKEL, derived from program SKELETON (described in section 3.3.3). Processing is identical, with one exception; dummy segments cannot themselves be extended to form higher-level segments in shape hierarchies, though they can be included as bottom-level intermediate segments within such hierarchies (Fig 7.4).

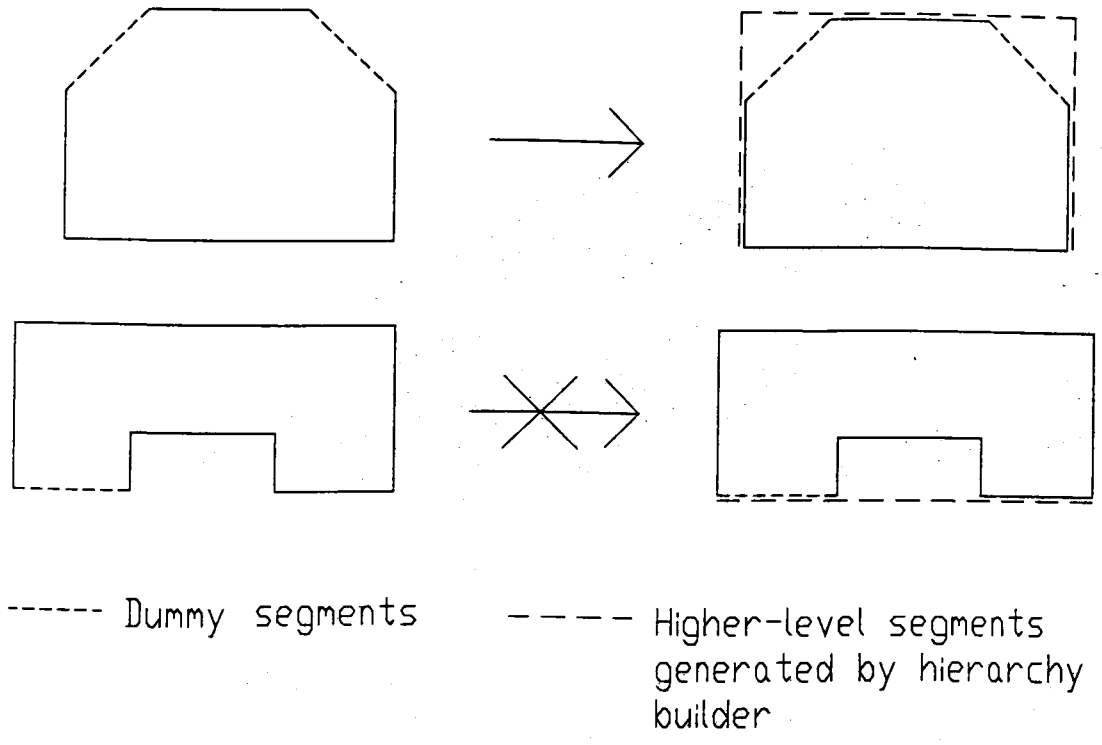


Fig 7.4 Rules for inclusion of dummy segments in shape hierarchies. Dummy segments can be included only as intermediate segments within hierarchies, as shown in the first example. They cannot themselves be used as the basis for higher-level segments as in the second example.

Program QUERYCAN casts query representations into canonical form using the same method as CANONGEN (section 3.3.4). Dummy segments have a length of zero. Hence when comparing alternative paths around the boundary, a path with a real segment at a given position will always be selected in preference to one with a dummy segment at that position.

Program QUERYFEAT extracts all relevant features of the type described in chapter 4, using procedures analogous to program DATALOAD (section 4.5), and stores the final query representation in a file suitable for input to program RETRIEVE (section 6.6). For queries without dummy segments, the feature extraction process is exactly as described in section 4.5; where dummy segments are present, some features cannot be defined, so that null values have to be set. As indicated in section 6.5, program RETRIEVE tests each query for the presence of dummy segments, and curtails the range of feature types matched where necessary.

The rules for determining feature validity in incomplete query boundaries are simple. Where dummy segments are present in a given boundary at a specified level, none of the global features defined in section 4.4.2 are assigned a value. Local boundary features (section 4.4.3) are computed where values can be calculated without reference to dummy segments (Fig 7.5). Inner boundary position features (section 4.4.4) are held to be dependent on relative positions of boundary centroids, not individual segments, and are therefore calculated. This is not strictly valid, since the presence of a dummy segment in the top level of a boundary implies that the boundary centroid position cannot be defined. However, changes in the shape of individual boundary segments seldom have a major effect on the boundary centroid position. It was therefore considered worthwhile to include such features for boundaries with dummy segments, at least in the initial prototype, since it was a reasonable hypothesis that their effect on retrieval performance would be beneficial.

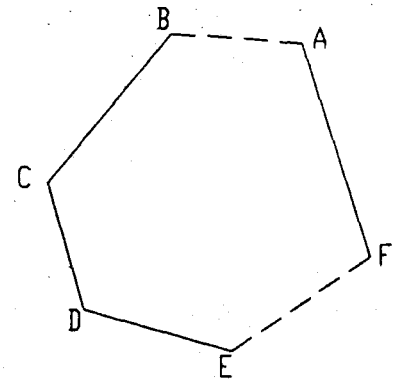


Fig 7.5 Local shape feature generation from incomplete shapes. Arc angle values are valid for line segments BC, CD, DE *and* FA; *discontinuity angle and triplet values for angles* C *and* D; *arc angle triplet values only for segment* CD.

7.7 Display of results

As indicated in section 6.6.5, program RETRIEVE allows the user to select one of a number of output formats, following the principles outlined in section 7.3.2:

(a) A list of drawing identifiers, file names, and difference measures, displayed on the screen in order of increasing difference from the query (i.e. descending similarity). The list header contains full details of all run-time parameters specified.

(b) The same output as (a), but directed to a print file (Fig 7.6).

(c) If a suitable graphics terminal is available (DEC VT240 or similar), retrieved shapes may be directly displayed. The query is first drawn on the screen, then retrieved shapes, in similarity order, together with identifiers and distance measures. Drawings are scaled so that up to four may appear on each screen.

(d) The same output as (c), but directed to a Tektronix plot file (which can also be displayed on a laser printer - see Fig 7.7).

(e) The same output as (c), but directed to a Postscript file for output to a laser printer.

No relevance feedback facility is available at present.

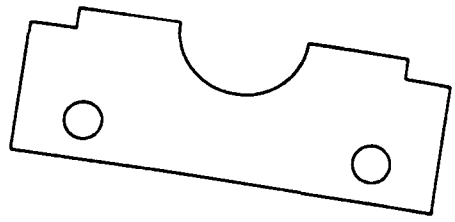
```
Match of query DRAW048        (draw048.QUE ) by RETRIEVE, version 4.0
Drawings searched:  1 - 188 retrieved: 30 limit: 60 cutoff: 250.00
Init cutoff:1.21 IB weighting - position: 0.20 class: 0.40 shape: 0.10
Feature Match IB positions only Mirror images All classes
Local feature matching taking minimum level and feature values
Features used -  Arc Triplets Parent Features SegLength Distribution
```

Drawing no	File name	Similarity Measure
54	ENG09	0.3006
147	ENG119	0.3030
164	ENG137	3.7345
34	MORECIRC	3.7451
111	ENG71	4.1110
76	ENG38	4.1121
102	ENG63	4.1986
127	ENG94	4.2023
74	ENG034	4.2150
79	ENG035	4.2448
83	ENG43	4.2536
140	ENG114	4.2732
70	ENG023	4.3171
101	ENG61	4.3283
96	ENG62	4.3283
24	CIRCLES	4.3471
75	ENG36	4.3500
154	ENG123	4.3596
80	ENG37	4.3692
149	ENG124	4.3777
65	ENG22	4.4044
69	ENG21	4.4273
139	ENG112	4.4432
171	ENG122	4.4478
175	ENG162	4.5259
35	MORECIRC35	4.7934
25	CIRCLE25	4.8013
137	ENG107	4.8266
39	TRIANGLE	4.9122
72	ENG27	4.9484

```
End of report - total CPU time used (ms)     1790
```

Fig 7.6 An example of printed output, showing ranking of retrieved drawings in similarity order.

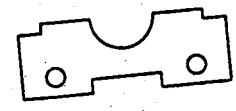
Query name DRAW089



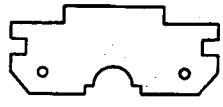
Drawings retrieved, in similarity order

Drawing no 108
D = 1

Drawing no 114
D = 13



Drawing no 115
D = 53



Drawing no 119
D = 53

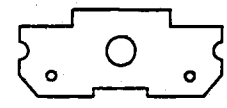


Fig 7.7 A graphical display of retrieved drawings, again ranked in similarity order. Note that the values of the difference measure D *shown here are multiplied by 10 and then rounded to the nearest integer.*

CHAPTER 8. TESTING AND EVALUATION

8.1 Scope of the evaluation process

The preceding chapters have described the SAFARI system, and the reasoning behind the decisions taken in its design, in some detail. Some measure of evaluation of the prototype system is clearly required in order to test the basic validity of its approach before further development can be undertaken. It is perhaps important at this stage to distinguish between program and system testing, which aims to ensure that the system's design objectives have been met, and system evaluation, which tries to establish whether these objectives are in fact worthwhile.

For conventional computer systems such as invoicing or stock control, the distinction between these two processes is clear-cut. Testing involves verifying that all possible types of input to the system lead to correct outputs or changes to data stores. For a large system, this is a decidedly non-trivial task, but can at least rely on the truth of one axiom - for every input, it is possible to define unequivocally what the system's response should be. In other words, there is always a set of "correct" values against which the system's output can be checked. Evaluation, on the other hand, is much more subjective, centring on aspects such as user acceptability or value for money. Often, it is not carried out at all, the fact that someone has been prepared to fund the systems development process in the first place being considered sufficient justification for its existence.

For systems which aim in one way or another to emulate human judgement, the picture is more complicated. At the lowest level, it is still essential to verify that each system component behaves in the way its designer intended. Program testing can be carried out in exactly the same way as for conventional software, by comparing output values with those expected by the designer. Overall system effectiveness is however a much more difficult concept to measure, as it may be impossible to define a "correct" output for a given input. For example, a medical expert system, faced with a given set of symptoms, may well come up with a diagnosis which one specialist would endorse and another would reject. Even if further investigation of the patient revealed that the expert system's diagnosis was in fact correct, one cannot rule out the possibility that it reached the right conclusion for the wrong reasons. Similarly, a bibliographic retrieval system may perform according to its designer's specification in that the references it generates in response to a given query always contain the keywords specified. Whether this set of references is actually useful to the enquirer (the real criterion of system effectiveness) is another matter.

One can thus distinguish at least three levels of evaluation for such a system - testing of program correctness, measurement of system effectiveness, and assessment of overall fitness for purpose. For prototype systems such as SAFARI, the second of these is probably of prime importance. This is not to downgrade the importance of comprehensive program testing, which is just as necessary here as it is for any other system. It is simply a recognition that any number of standard techniques are applicable (e.g. Myers, 1979).

Assessment of a system's overall fitness for purpose, while a key issue for operational systems or later prototypes, is simply not possible with early prototypes. Such an assessment requires a variety of factors to be evaluated, including effectiveness in providing the required output, acceptability of user interface, response times, and cost-benefit ratios. Most of these factors can be evaluated only when it has been shown that the system does in fact work - and in some cases only after the system has been in operation for a significant period of time (cost-benefit analysis, for example, requires a system to have been operational for long enough to have produced measurable effects on the behaviour of its users).

Choosing appropriate measures of system effectiveness thus appears to be a crucial issue. The evaluation process requires some objective success criteria to be set, together with some reliable way of measuring how well these criteria have been met. What criteria should be chosen, and how can they best be measured?

8.2 Evaluation techniques used with related systems

One obvious source of evaluation measures would seem to be the literature on related systems - which raises the question of what kind of system SAFARI really is. This question is discussed further (and hopefully resolved) in Chapter 9. Meanwhile, one might expect to look to four related types of system, all of which attempt in some way to mimic human judgement, for ideas on evaluation - bibliographic information retrieval systems, expert (knowledge-based) systems, image database systems of the type described in Chapter 1, and pattern recognition systems.

8.2.1 Information retrieval systems

Information retrieval systems (see chapter 5) have a long history, predating the computer by a considerable period. Since the mid 1950s, a substantial body of research into their effectiveness (summarised in Sparck Jones, 1981) has been built up. Six evaluation criteria, originally proposed for the Cranfield experiments on index language effectiveness (Cleverdon et al, 1966), appear to have stood the test of time:

1. The **recall** of the system, i.e. the proportion of relevant material in the system actually retrieved in response to a query;

2. Its **precision**, i.e. the proportion of retrieved material that is actually relevant;

3. Its **coverage**, i.e. the extent to which all relevant material is included in the system;

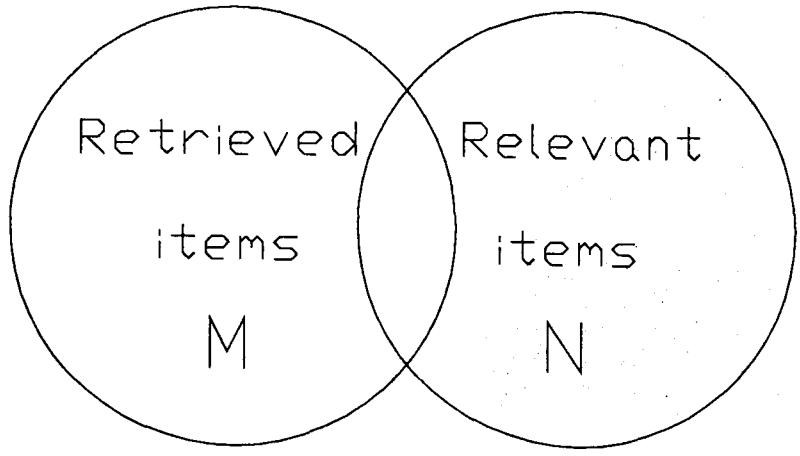
4. The **time lag** between receipt of a query and delivery of the system response;

5. The form of **presentation** of the output;

6. The **effort** required by the user to obtain search results.

The first two of these measures (illustrated as a Venn diagram in Fig 8.1) can be regarded as prime indicators of system effectiveness, i.e. ability to present users with useful material while screening them from irrelevant information. They have the advantage of being easily expressible as numeric quantities, thus allowing valid comparisons to be made between the performance of different systems. The remaining measures are harder to quantify, and hence less immediately appealing as performance indicators. The third measure is in any case primarily of interest to the manager of the data collection rather than to the system designer, while measures 4 - 6 can be regarded principally as indicators of user acceptability.

To assess precision and recall values, it is essential to have some means of judging the relevance of a retrieved item to the query. Such judgements are notoriously subjective. Only the person framing the original query can really tell whether a given document contains useful information. Even then, (s)he may well not judge the document's usefulness by the same criteria as those used to formulate the query. The difficulty of emulating human relevance judgments accounts for many of the problems of evaluating that performance. Failure to appreciate these difficulties invalidated a number of early



$$\text{Recall} = \frac{|M \cap N|}{|N|} \qquad \text{Precision} = \frac{|M \cap N|}{|M|}$$

Fig 8.1 Diagram illustrating the definition of precision and recall

studies. Retrieval experiments now normally use either the enquirer's own relevance judgements (accurate but possibly idiosyncratic), or that of a panel of independent experts (objective but possibly based on a misunderstanding of the query).

Despite this, precision and recall (or measures derived from these parameters) have become the *de facto* standard for evaluating retrieval systems, or comparing the relative effectiveness of different types of search strategy or indexing language. For any given system, they tend to show complementary behaviour as the retrieval cutoff point is varied. Any action which increases recall (such as broadening the scope of a query) will decrease precision, and *vice versa*. Hence one can characterize the behaviour of most retrieval systems by plotting precision-recall graphs of the type shown in Fig 8.2.

Graphs of this sort are a somewhat cumbersome way of recording effectiveness, particularly when comparing the relative performance of two systems. A single numerical measure of effectiveness was first proposed by Swets (1963), who postulated the following criteria for the ideal retrieval measure:

1. It would be a pure effectiveness measure, reflecting only the system's ability to distinguish between wanted and unwanted items;

2. It would be independent of retrieval cutoff point;

3. It would be expressible as a single number;

4. It would provide an absolute scale on which the performance of different systems could be compared.

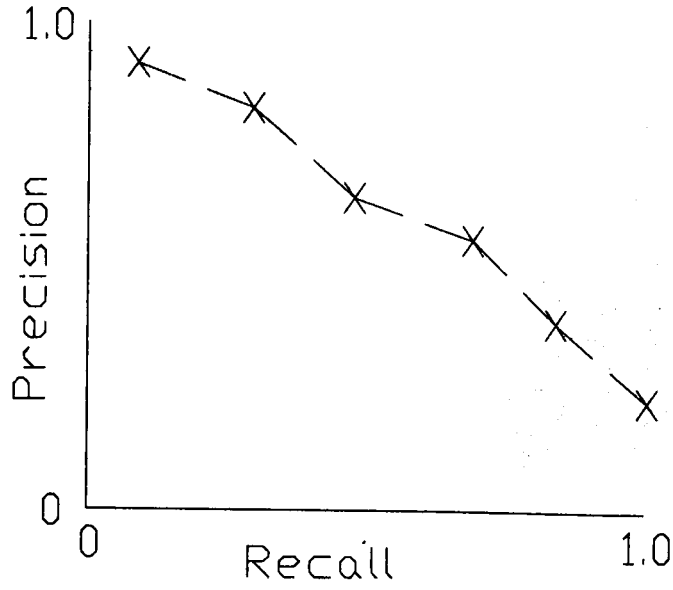


Fig 8.2 Typical precision-recall graph illustrating the inverse relationship between these two measures. As a search formulation is broadened, recall invariably rises and precision diminishes.

His own *E* measure, based on statistical decision theory, is potentially able to satisfy all of the above criteria. However, it relies on the assumption that the numbers of both relevant and non-relevant documents retrieved at a given level of some control variable (such as depth of indexing or number of search concepts combined) are normally distributed with respect to that variable. Since this is rarely the case in practice, his measure has limited validity and is seldom used.

For systems capable of ranking output in presumed order of relevance, two single-number measures of effectiveness, which meet the above criteria and make no assumptions about the underlying distribution of the data, have been proposed. Both were used in the evaluation of the SMART system (Salton, 1971), and both compare the actual ranking of retrieved documents with the best and worst possible cases. *Normalized recall* (so called because it is computed by considering recall values at each possible document ranking level) can be computed as:

$$R_n = 1 - \frac{\sum_{i=1}^{n} (R_i) - \sum_{i=1}^{n} (i)}{n(N - n)}$$

where R_i is the rank at which relevant document i is actually retrieved, n is the total number of relevant documents, and N the size of the whole document collection. In a similar fashion, *normalized precision* is defined as:

$$P_n = 1 - \frac{\sum_{i=1}^{n} (\log R_i) - \sum_{i=1}^{n} (\log i)}{\log \binom{N}{n}} .$$

It is important to note that either of these measures can stand alone as an indicator of retrieval performance. Despite their names, normalized precision and recall are not a complementary pair of performance indicators, nor do they bear an inverse relationship to each other. The only difference between them lies in the weighting assigned to different document ranks - the normalized precision measure giving more weighting to the rankings of the first few relevant documents retrieved, the normalized recall measure being more sensitive to the ranks of the last few relevant documents retrieved.

Several other composite measures of system effectiveness have been proposed (see van Rijsbergen (1979), chapter 7), though none would seem to have any compelling advantages over those outlined above.

8.2.2 Expert systems

Whatever performance measures are chosen, the evaluation of an information retrieval system is essentially a test of its relevance judgements. In the same way, the evaluation of an expert system is a test of its decision-making. In both cases, it is necessary to compare performance with that of human subjects faced with the same decisions. However, the expert systems community does not appear either to have made use of the formal performance measures described above, nor to have developed its own measures. (This may of course simply be a reflection of the relative youth of the discipline).

The need for careful and objective evaluation, and the problems of achieving this in practice, are, however, well recognized by expert systems researchers. The review by Gaschnig et al (1983) notes the importance of formal testing, though it illustrates the difficulty of comparing machine judgements with those of humans by comparing the results of two trials of the MYCIN medical diagnosis system in different hospitals. Despite identical experimental design, the performance of MYCIN appeared to be substantially better in one hospital than another, a difference attributed at least in part to differences in diagnostic style between the two hospitals.

This review contributes a number of generally applicable insights. It stresses the need for objective standards against which performance can be assessed, the need for careful experimental planning to avoid bias, the value of sensitivity analysis (testing the effects on performance of small changes in input parameters), and the value of eliminating as many variables as possible before measuring system performance. It also lists a number of potential pitfalls, including failure to clarify what is being evaluated, biasing the results by selecting too narrow a range of test cases, failing to select an appropriate standard, overgeneralizing from results obtained, and attempting detailed evaluation at too early a stage of development.

8.2.3 Image database systems

While some pictorial database systems, such as those described by Chang et al (1980) or Chock et al (1984), are designed to handle queries for which an unequivocal "right" answer can be defined (such as identifying all grid cells on a given map classified as "forest", or calculating the distance between two identified city centres), most have a wider scope. For example, ARES (Ichikawa et al, 1980), GRAIN (Chang et al, 1980) and REDI (Chang and Fu, 1980) all allow users to input queries involving similarity estimation, which require the system to make relevance judgements. However, none of these authors present any examples of output from their systems, let alone an evaluation of their performance. The only workers in this area who acknowledge the problem of achieving acceptable values of precision and recall appear to be Rabitti and Stanchev (1989) - and even they give no actual evaluation results.

8.2.4 Pattern recognition systems

Parallels between the SAFARI system and image recognition systems have already been drawn in Chapter 4. Such systems again aim to mimic human judgement, in this case in recognizing familiar shapes within a digitized image. Perhaps because most image recognition systems deal with relatively well-defined situations (such as whether a given object is present in a given scene or not), the question of evaluation is clearly considered of less importance than with information retrieval systems or expert systems. If a system has been designed to decide whether a given image is that of a hammer or a pair of pliers, there is not much difficulty in deciding whether its decision is correct or not. Even with more complex recognition tasks, such as identifying, counting and locating (say) steering knuckles in a whole heap of assorted motor parts, a "right" answer can still be defined without the need to involve a panel of independent observers.

Like most of the work on image databases referred to in the previous paragraph, studies such as those of Yachida (1977), Perkins (1978), Turney (1982), Bhanu (1984) and Stockman (1985) thus appear to have little to contribute in the area of evaluation. Indeed, one might legitimately criticize some of these studies for the small number of test cases they use to demonstrate the validity of their approach.

8.3 Evaluation approach chosen for SAFARI

8.3.1 General observations

The above studies highlight the need for an objective evaluation of the prototype SAFARI system, to determine the extent to which its underlying assumptions form a valid basis on which to build an operational system. Since the system is expected to be able to handle graphical queries in the same way that a bibliographic system handles text, performance measures based on precision and recall would seem to be highly appropriate. The fact that they can yield a numerical indication of performance is particularly helpful, as one can then readily judge whether any given modification to the system is likely to be beneficial or not.

Objectivity in the selection of test data, and in the assessment of the relevance of output shapes to the original query input, are clearly of crucial importance, though not easy to ensure. Also of major importance is the need to limit the scope of the initial evaluation to those parameters that can realistically be tested with an early prototype. An obvious example is the need to test system effectiveness (through precision, recall or related measures) before making any attempt to investigate the acceptability of the user

interface. Unless one knows that a system is capable of delivering reliable results, assessing user reactions to its interface is unlikely to prove worthwhile.

The prototype version of SAFARI aims primarily to retrieve drawings similar to a given query shape (Type F retrieval). It is thus reasonable to start by testing the system's performance in this area. Unless the system's judgments of shape similarity can be demonstrated to be reasonable, there is little point in extending its scope to other types of retrieval. The decision was thus taken to base initial evaluation purely on the system's ability to identify and rank those drawings most similar to a given query shape. This required (a) a reasonable-sized collection of test shapes both for the database and to act as queries, (b) an objective standard for judging the similarity of stored and query shapes, and (c) a reliable measure comparing system performance with the standard. The following sections describe these aspects in turn.

8.3.2 The test database

Ideally, one would base a test database for a system like SAFARI on a random sample of all possible shapes from its chosen domain, so that all possible types of shape and geometric feature stood an equal chance of being represented. Unfortunately, such a concept is impossible to translate into reality - any attempt at identifying shape or feature classes in advance immediately introduces an element of prejudice. Failing this, some collection of shapes assembled for some other purpose, such as drawings of spare parts stocked by an electrical goods firm, might suffice, though such a collection could well contain a preponderance of shapes of a particular type. The situation is not unlike that in the text retrieval world, where many authors (see e.g. Sparck Jones, 1981) have lamented the lack of comparability of retrieval experiments using different document collections, and called for standard collections that any experimenter can use.

The logistic difficulties of assembling such a collection of shapes for the evaluation of SAFARI proved to be considerable. Only a limited proportion of industrial parts falls into SAFARI's restricted domain of shapes. Hence the drawing collections of local engineering firms were of little use *as collections*, even though some of their individual parts could be used. For the initial test database, therefore, shapes were taken from a variety of sources - actual machined parts, illustrations from parts catalogues, and textbooks of engineering drawing (perhaps particularly suitable, as one might expect examples to be deliberately selected to cover as wide a range of drawing types as possible) providing the largest numbers of shapes.

The majority of shapes were drawn using the DOGS CAD system at Newcastle Polytechnic, though a minority were produced using the MEDUSA system, as a test of compatibility. In several cases, identical shapes were drawn at different orientations and sizes, or using different CAD systems, in order to check that SAFARI would in fact process their different representations in the same way. In all cases, drawings were made available to the SAFARI system as IGES-format transfer files, as described earlier. The test database eventually contained 187 such shapes.

The prime purpose of this test collection was obviously to act as a database of stored shapes against which queries could be put. It also provided the majority of test data for the shape analysis programs described in Chapters 3 and 4. Log files were generated by test versions of all programs, and a random selection of these were carefully examined to ensure that test shapes were in fact being processed as expected. A check was also made that the segment hierarchies generated by program SKELETON (chapter 3) appeared intuitively sensible, and program modifications made where (as in Fig 3.33) this appeared not to be the case.

A subset of this test collection was also used to provide the query shapes for the initial evaluation experiments. Although it can be argued that such "queries" are somewhat

artificial, the same can be said of virtually any query shape put to the system at its present stage of development. It can equally be argued that the only truly realistic way to test retrieval effectiveness is to use examples of actual queries put to an operational system - something that is just not possible in the present context, as no operational systems exist. In any case, assessment of the system's ability to rank stored shapes in order of similarity to any given stored shape *is* a realistic test of any system that aims to provide relevance feedback in the way described in section 7.3.2 above.

8.3.3 Standards for judging similarity

8.3.3.1 Selection of a suitable standard

As discussed above, the only completely realistic standard for judging retrieval performance - users' own judgement of the relevance of each stored item to their original queries - cannot be used at this stage of prototype development because no body of users or queries exists. Some compromise was therefore essential if any worthwhile yardstick was to be defined. Since initial testing was to be limited to type F (similarity matching) queries, the most sensible approach was considered to be to ask a group of potential users to identify those drawings from the test database with highest similarity to a series of query shapes. Assuming some reasonable degree of consensus between subjects, the results of such an experiment could then yield a reasonably objective indication of the drawings that the system ought to retrieve in response to each of the query shapes tested.

The only "potential users" available in sufficient numbers and willing to participate in such an experiment were mechanical engineering undergraduates at Newcastle Polytechnic. These students were considered acceptable subjects, as there was no reason to suspect that their judgements of shape similarity would differ materially from those of eventual users of any shape retrieval system. All students were skilled in engineering drawing, and had experience of using CAD systems. It therefore proved relatively easy to explain the purpose both of the system and of the experiment to them.

8.3.3.2 Experimental design

All 187 drawings in the test database were printed out using a high quality laser printer, and photocopied on to cards. Two separate sets of cards were produced, the first (printed on buff card) showing the complete drawing, the second (printed on yellow card to minimize the risk of confusion) showing outer boundaries only. (This was done so that the system's performance in outer and inner boundary matching could be judged separately if required). Since a number of shapes in the database had been drawn with identical outer boundaries, differing only in inner boundary pattern, this left each "outer boundary" pack with a significant number of duplicate shapes. With the agreement of three independent observers (the author's wife and children), 17 such duplicate shapes were identified and removed from each yellow pack.

16 of the 187 drawings from the buff-coloured "complete drawing" pack were then selected as "query" shapes by the same observers, without the author's participation. These 16 drawings were then removed from each pack, leaving 171 to act as "stored" shapes. An identical procedure was adopted for the yellow "outer boundary only" pack. Again, 16 drawings were selected as query shapes (6 identical to "complete drawing" queries, 10 different), and removed from each yellow pack, this time leaving 154 "stored" shapes. The two series of query shapes are illustrated in Figs 8.3 - 8.4 and 8.5 - 8.6.

Two experimental sessions were held, involving a total of 58 students. At each session, the purpose and background of the experiment were explained, and each student given a pack of "stored" shapes, a set of 8 "query" shapes of the same colour, and a form for recording results. Students were then asked to take each "query" shape in turn, and after

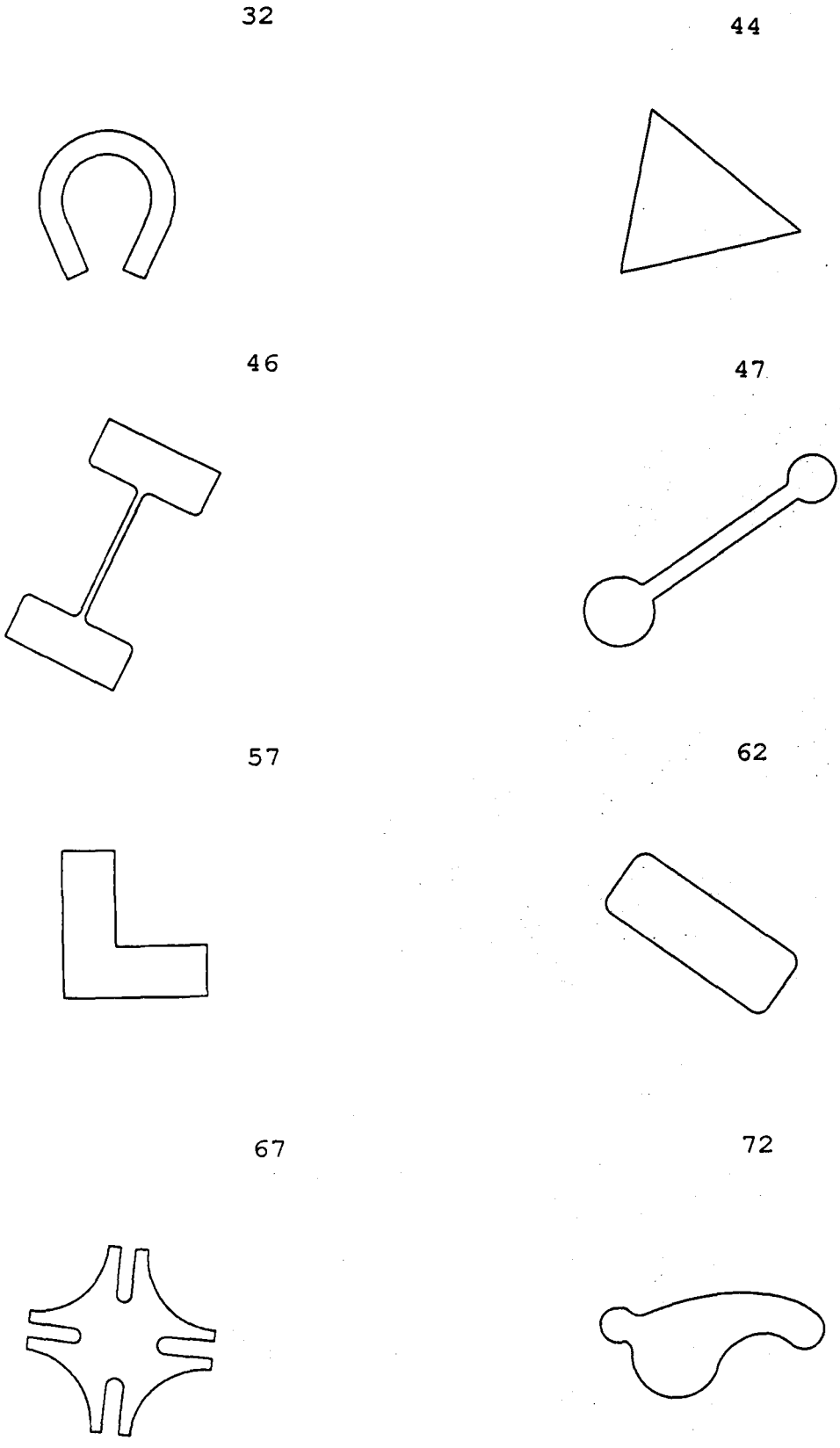


Fig 8.3 The first eight outer-boundary query shapes

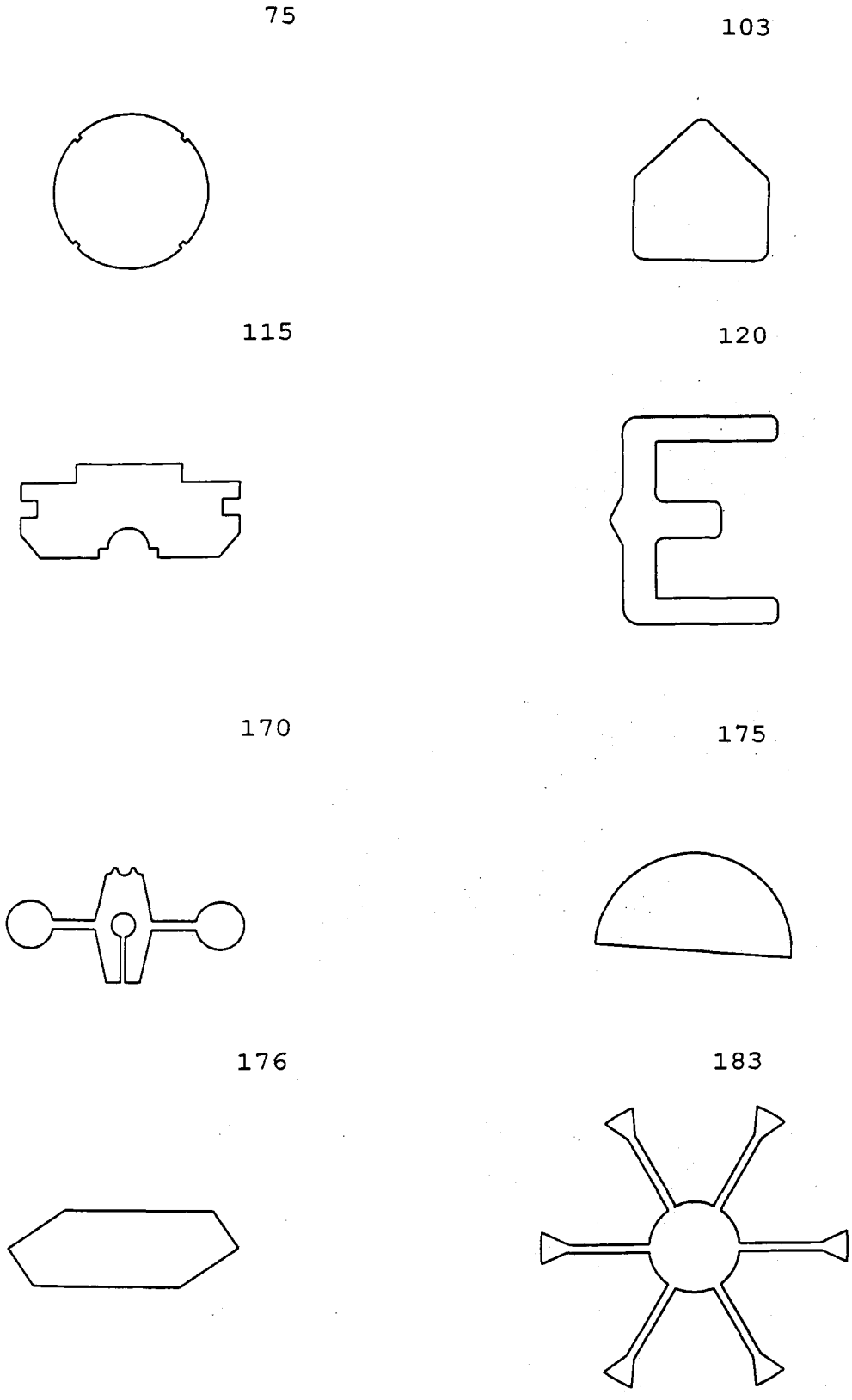


Fig 8.4 The remaining outer-boundary query shapes

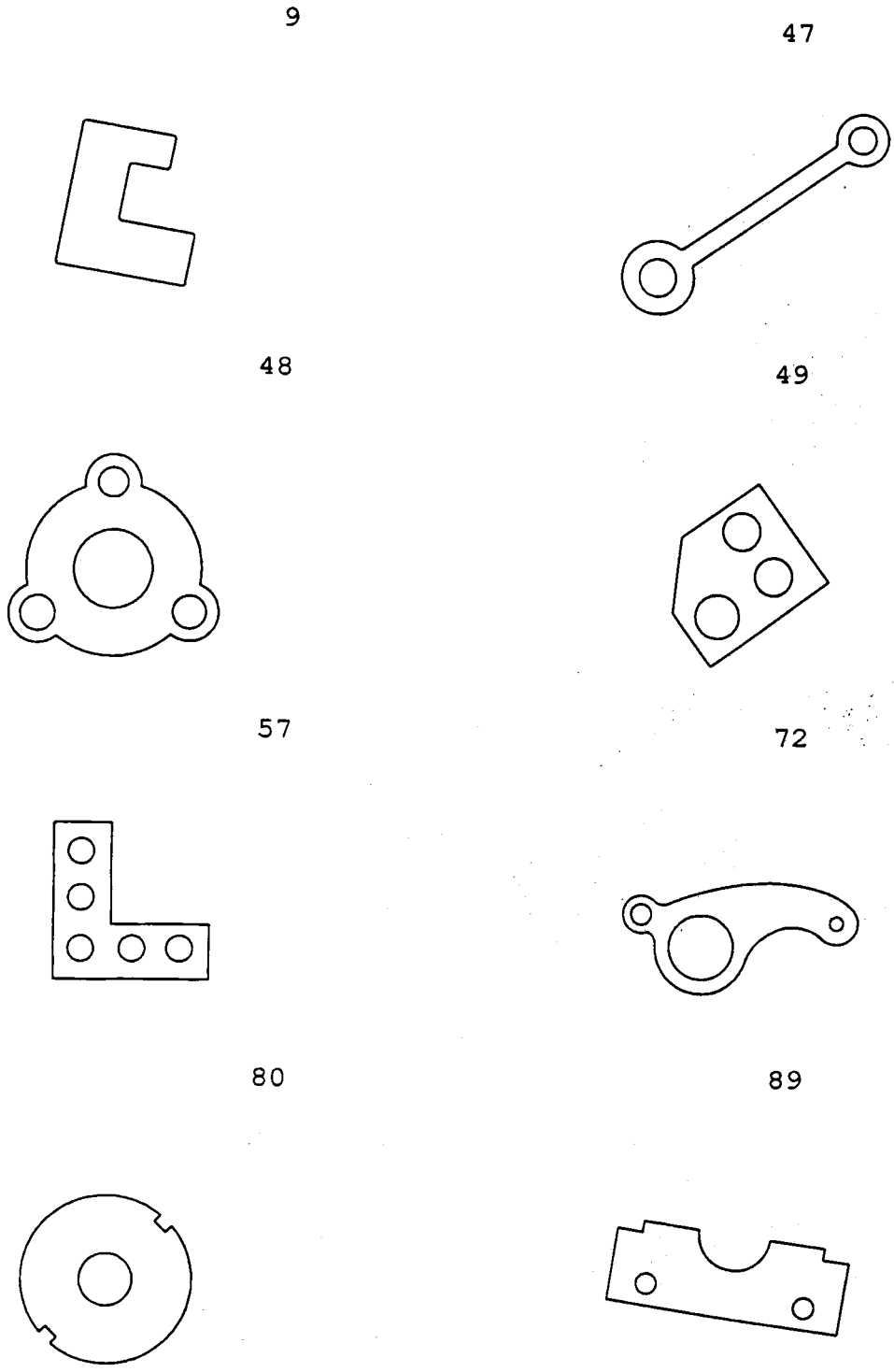


Fig 8.5 The first eight ***all-boundary*** *query shapes*

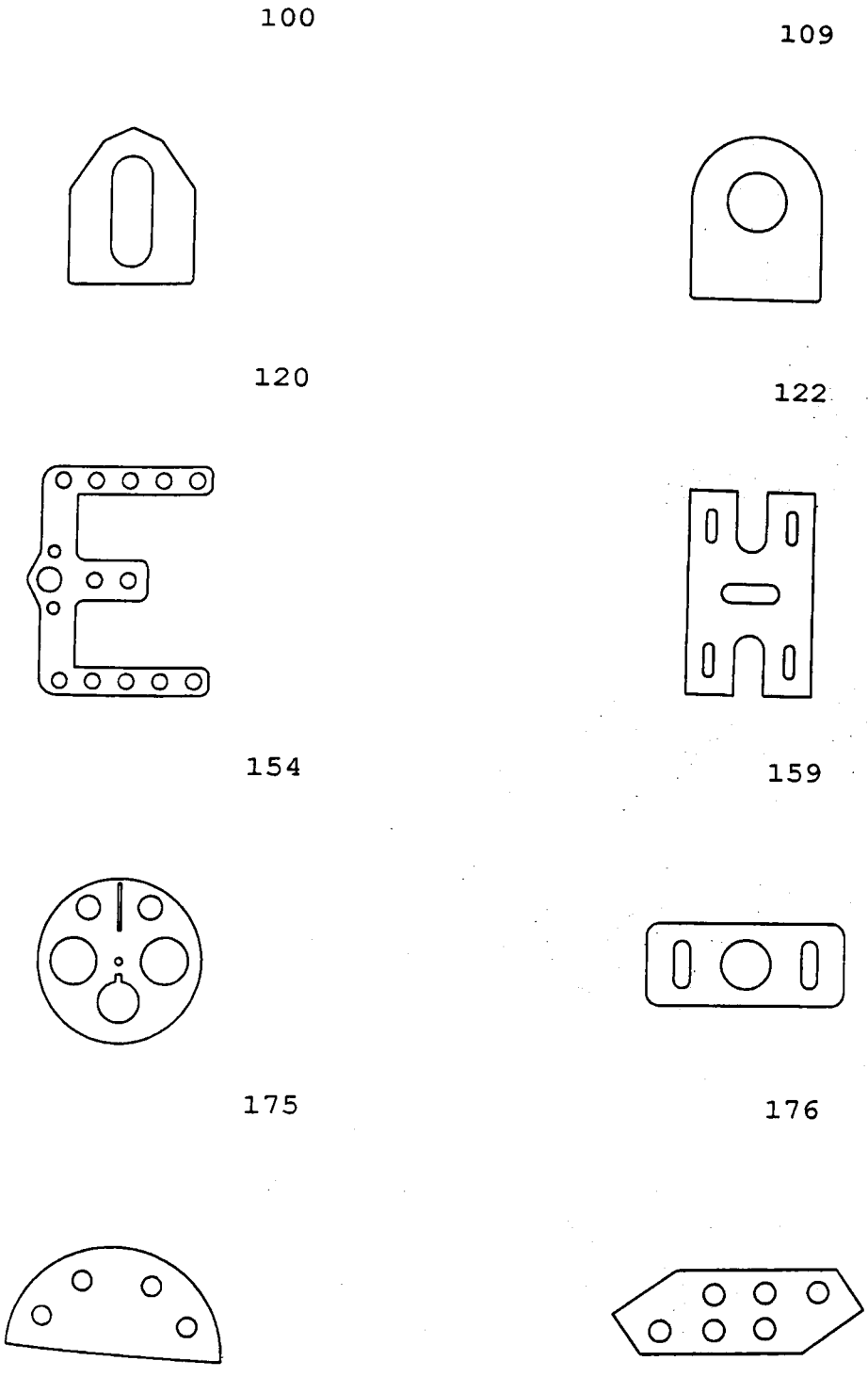


Fig 8.6 The remaining all-boundary query shapes

examining the pack of "stored" shapes, to write down on the form the drawing numbers of any stored shapes they considered identical to the query shape, plus up to five shapes they considered closely similar, in order of similarity to the query. Students all worked individually, and were allowed up to an hour to complete the task. Quite deliberately, they were given no guidance on how to judge similarity - though at the end of the session they were invited to comment briefly on the method they had adopted. Nor was any pressure put on students to complete all eight queries if they were running short of time or losing interest. At the completion of the experiment, forms were analysed and counts recorded of the number of subjects who had ranked each "retrieved" drawing in similarity positions 0 (identical), 1, 2, 3, 4 and 5.

8.3.3.3 Experimental results

Results for four of the test queries, reflecting varying degrees of consensus, are presented below in some detail. Full results for all queries are tabulated in Appendix A.

Query 170 (outer boundary only) produced the most consistent results of any of the queries. It was one of a series of drawings used to illustrate the effects of changing dimensional parameters in otherwise identical shapes, so the degree of consensus here was perhaps not surprising. Table 8.3.1 indicates the number of students ranking each of the retrieved drawings at the specified position (e.g. 1 of the 16 students considered that drawing 167 matched the query exactly; 12 of the 16 students ranked drawing 166 third in similarity to the query). The drawings themselves are illustrated in Fig 8.7.

Table 8.3.1. Query no: 170, no of subjects: 16

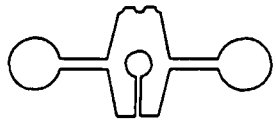
Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5
167	1	13	1	1	0	0
169	0	0	13	2	1	0
166	0	3	1	12	0	0
168	0	0	1	1	5	7
172	0	0	0	0	9	6

Query 89 (all boundaries) again elicited a high degree of consensus with the highest-ranking drawings (11 of the 13 subjects judging query 89 considered shape 108 to be identical to the query, and 12 that shape 114 was the most similar of those remaining). Although there was a good measure of agreement over which remaining shapes should be included in the similarity listing (apart from drawing 150, selected by only one student, and which seemed, at least to the author, a distinctly eccentric choice) there was little consensus over their actual ranking. Students' choices are listed in table 8.3.2, and illustrated in Fig 8.8.

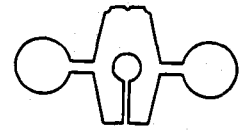
Table 8.3.2. Query no: 89, no of subjects: 13

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5
108	11	1	0	0	0	0
114	0	12	1	0	0	0
115	0	0	4	5	1	0
119	0	0	3	1	4	0
90	0	0	2	3	3	1
150	0	0	0	0	0	1

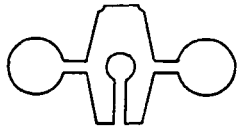
Drawing no 167



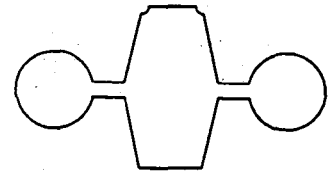
Drawing no 169



Drawing no 166



Drawing no 168

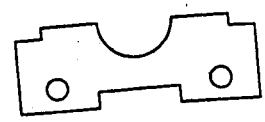


Drawing no 172

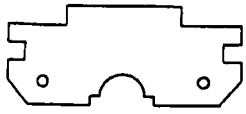
Fig 8.7 Students' judgements of the five shapes most closely resembling query shape 170, based only on outer boundary shape

Drawing no 108

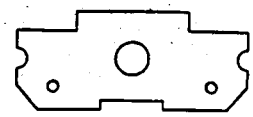
Drawing no 114



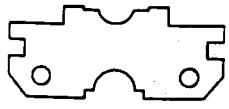
Drawing no 115



Drawing no 119



Drawing no 90



Drawing no 150

Fig 8.8 Students' judgements of the shapes most similar to query shape 89, based on the complete shape (outer and inner boundaries)

Two examples of queries where less consensus was apparent are also presented here. **Query 175** (all boundaries) has a longer 'tail' than the two previous examples, and two schools of thought could be distinguished, one giving drawing 27 the highest similarity, the other favouring drawing 82 (Table 8.3.3). Examination of the drawings themselves (Fig 8.9) suggests a possible explanation - students preferring drawing 27 could have been looking primarily for outer-boundary similarity, while those favouring 82 gave more weight to similarities in inner boundary patterns.

Table 8.3.3. Query no: 175, no of subjects: 12

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5
82	0	4	5	2	0	0
27	0	5	1	1	1	1
87	0	1	3	3	3	0
37	0	2	1	1	0	5
145	0	0	0	3	4	2
31	0	0	1	0	0	0
83	0	0	0	1	0	0
24	0	0	0	0	1	0

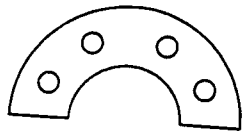
Finally, **query 32** (outer boundary only) illustrates a case where students showed little or no agreement after position 2 on the list (Table 8.3.4; Fig 8.10). The most plausible explanation for this is that there were in fact only two drawings similar to the query shape in the entire pack. Support for this view comes from the fact that 9 of the 16 students judging this query left positions 3, 4 and 5 in their lists blank. Again, however, consensus was apparent in the choice of the first two drawings selected.

Table 8.3.4. Query no: 32, no of subjects: 16

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5
33	0	13	0	0	1	0
82	0	0	9	1	0	0
87	0	0	1	2	2	0
23	0	0	2	0	0	0
145	0	0	0	1	2	1
138	0	1	0	0	0	0
161	0	0	1	0	0	0
102	0	0	0	1	0	1
66	0	0	0	1	0	0
29	0	0	0	1	0	0
24	0	0	0	0	1	0
128	0	0	0	0	1	0
118	0	0	0	0	0	1
39	0	0	0	0	0	1
65	0	0	0	0	0	1

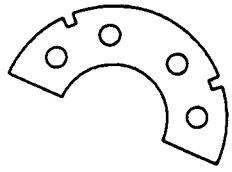
It is left for the reader to judge for him or herself whether these rankings are plausible. In the author's view they are, though in a sense this is a secondary point. The main conclusion that can be drawn is that the students' judgements were largely consistent, and therefore a reasonable base from which to derive similarity rankings for use in evaluating the prototype version of SAFARI.

Drawing no 82



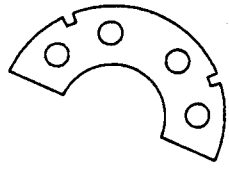
Drawing no 27

Drawing no 87

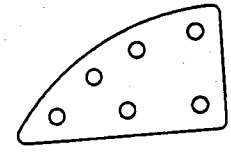


Drawing no 37

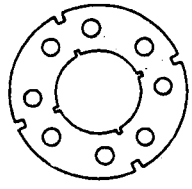
Drawing no 145



Drawing no 31



Drawing no 83



Drawing no 24

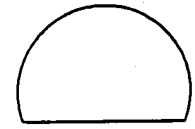
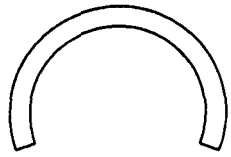
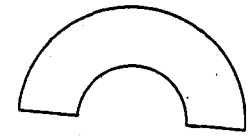


Fig 8.9 Students' judgements of the shapes most similar to query shape 175 (all boundaries)

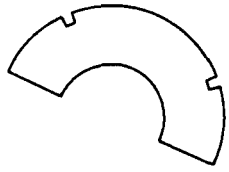
Drawing no 33



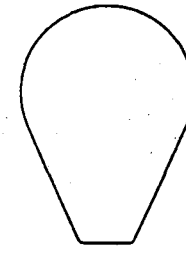
Drawing no 82



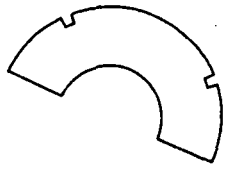
Drawing no 87



Drawing no 23



Drawing no 145



Drawing no 138

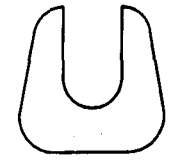


Fig 8.10 Some of the shapes students judged similar to query shape 32 (outer boundary only)

8.3.3.4 Subjects' comments on the process

Some useful insights into subjects' methods for matching query shapes can be gained from the comments that some of them recorded. Students matching shapes only on outer boundaries had fewer cues to go on, and this was reflected in some of their comments:

> "I checked all query shapes, sorting out all shapes that looked nothing like the queries, occasionally taking out 'similars' strikingly similar to the query shapes, then sorted through the remaining 'similars', picking out those most similar to the queries"

> "I looked for geometric similarities first, then subdivided the closest ones by size and number of correct features"

> "Primary shape sort; final shape sort; leave sorted shapes out of pack"

Note the principle of rapid screening to remove totally unsuitable shapes, followed by more detailed consideration of those remaining.

Students working with all-boundary queries made similar comments to those above, but also gave more specific clues to their method of working, e.g:

> "Identify major outline; major shapes, both internal and external; compare minor shapes/variations"

> "Look for major outer shape initially, then any significant internal shapes; finally narrow down slight differences"

> "Shape first, contents (circles, etc) second - unless close similarities here; external attributes last"

This suggests that the order in which SAFARI performs its matching operations (overall outer boundary shape, inner boundary pattern, then detailed shape comparison) may find parallels in the way human subjects assess shape similarity.

8.3.3.5 Derivation of standard similarity rankings

The remaining task in this phase was to derive a composite ranking of stored drawings in order of similarity to each query. In many cases (such as query 170), the ordering was obvious - one simply needed to pick out the highest-frequency rank for each drawing from the tables above. In others (such as query 175) it was not. A combined rank-frequency score was thus devised to allow a single ranking for each query. A number of alternative measures were considered. The simplest was the sum of weighted rank frequencies:

$$F = \sum_{i=0}^{5} f_i * i$$

where f_i is the number of rankings for each drawing at position i, with $i = 0$ for exact matching. This suffers from the problem that one has to assign arbitrary ranks to drawings which fail to appear in all lists in order to avoid unrealistically low scores. A variant which avoids this problem is:

$$F' = \sum_{i=0}^{5} f_i * (6 - i)$$

though it could be argued that this gives insufficient weight to exact or very similar matches. It also requires normalization to allow for variation in the number of respondents. The final measure chosen, S, defined as:

$$\frac{(f_0+0.7*f_1+0.5*f_2+0.3*f_3+0.2*f_4+0.1*f_5)}{n}$$

overcomes these problems. (Again, f_i is the number of rankings for each drawing at position i, and n is the total number of subjects performing the test). It is a true similarity measure in that a value of 1.0 indicates all subjects consider query and stored shapes to be identical, while a value of 0 indicates that no subjects consider there to be any similarity between query and stored shapes. The higher weightings for earlier rankings reflect the presumed importance of high similarity ratings. Drawings were then ranked in order of S value, as shown in Appendix A.

The final decision to be taken concerned cutoff point. As already observed, the long 'tail' of entries for query 32 is almost certainly a reflection of some subjects' desire to make sure all the boxes on their results forms were filled, rather than any real conviction that the drawings were particularly similar to the query. To use these results to judge system performance would not seem reasonable. An arbitrary cutoff was therefore applied - any drawing whose S score was less than 0.1 was deemed not to be sufficiently similar to the query and was therefore dropped from the list.

8.3.4 Measures of system performance

From the discussion above, it should be clear that the most appropriate measures of system effectiveness for use with SAFARI are precision and recall, or some parameter derived from them. Single-value measures have obvious attractions, and those defined for the SMART system would seem to be particularly appropriate. Like SMART, SAFARI is an experimental system aiming firstly to establish the fundamental validity of its approach, but eventually to throw light on the effectiveness of different methods of shape indexing and searching. Again like SMART, SAFARI ranks all output in presumed order of similarity to the query, rather than setting an arbitrary cutoff between "retrieved" and "not retrieved" items. At least at present, the shape collections it houses are small enough that users can if necessary examine every item in the database for relevance to provide the standards necessary to judge system performance. Hence the standard measures chosen to judge the retrieval effectiveness of SAFARI were the normalized recall and precision measures P_n and R_n defined in section 8.2.1.

8.4 Evaluation of SAFARI's retrieval effectiveness

8.4.1 Design of evaluation experiments

Experimental design for this phase of the study was relatively straightforward. Dummy query files were generated from the database for all 26 query shapes in the format illustrated in Fig 6.3. A batch version of program RETRIEVE was created, allowing run-time parameters to be input from a command file, thus creating a permanent record of the parameter values used. Printed output was generated in the format shown in Fig 7.6, showing similarity rankings and distance measures of all retrieved drawings. Cutoff limits for these experiments were set artificially high to ensure that all drawings identified in the human shape matching experiments were actually retrieved by the system, however low their ranking. In order to ensure that similarity rankings were directly comparable with those generated by the students, the program could be set to ignore all drawings whose identifiers were on a specified "stop list" (the electronic equivalent of removing query and duplicate cards from the drawing packs as described in section 8.3.3.2).

For each query, the similarity ranking of each drawing identified as relevant in the human shape-matching experiments described in section 8.3.3 was recorded, and entered on a spreadsheet containing the formulae required to calculate values for the P_n and R_n measures defined in section 8.2.1 above - either for individual queries, or for the entire set of queries relevant to a given set of run-time parameters. As indicated in section 6.6.1, it is possible to specify a number of different run-time parameters for each search conducted, including the matching paradigm to be used, the precise combination of feature types selected, the level of search exhaustivity, and the relative weighting of difference measures calculated by different parts of the matching process. In order to judge the potential of each matching paradigm, a large number of test runs was performed, each using a different combination of run-time parameters, in order to find the most effective level of exhaustivity and set of shape features to be used with each paradigm. For example, the most effective combination of features found for local feature matching was to use *arc angle*, *discontinuity angle* and *parent feature* distribution features as defined in section 4.4.3 to compare outer boundaries, and the *boundary pattern* features defined in section 4.4.4 to compare inner boundary positions.

Varying the level of search exhaustivity or the number of feature types used in matching in fact made very little difference to performance once a certain minimum threshold was passed. Hence no exhaustive attempts were made to identify the optimum combination of parameters once this performance plateau was reached. The main lessons to be learnt from these preliminary experiments were that to give adequate retrieval performance it was necessary (a) to use at least two feature types in the calculations, and (b) to base overall difference measures between query and stored shape boundaries on the single most closely matching pair of boundary levels, rather than averaging difference measure over all boundary levels. Detailed results of these preliminary tests are omitted from this thesis for reasons of space, but are available from the author on request. Except where otherwise indicated, the results presented below are all based on what is believed to be the most effective combination of search parameters for that search paradigm.

8.4.2 Detailed results from typical queries

Some results from a typical search are illustrated in Fig 8.11, and presented in tabular form below. The results shown in the figure are based on existence matching (as defined in section 6.6.3.4) using features based on parent feature and arc angle distribution (features 6, 8, 10 and 12 from the list in section 4.4.3), followed by segment matching as defined in section 6.6.4. This combination, though far from optimum with all queries, was found to give good all-round performance in most situations.

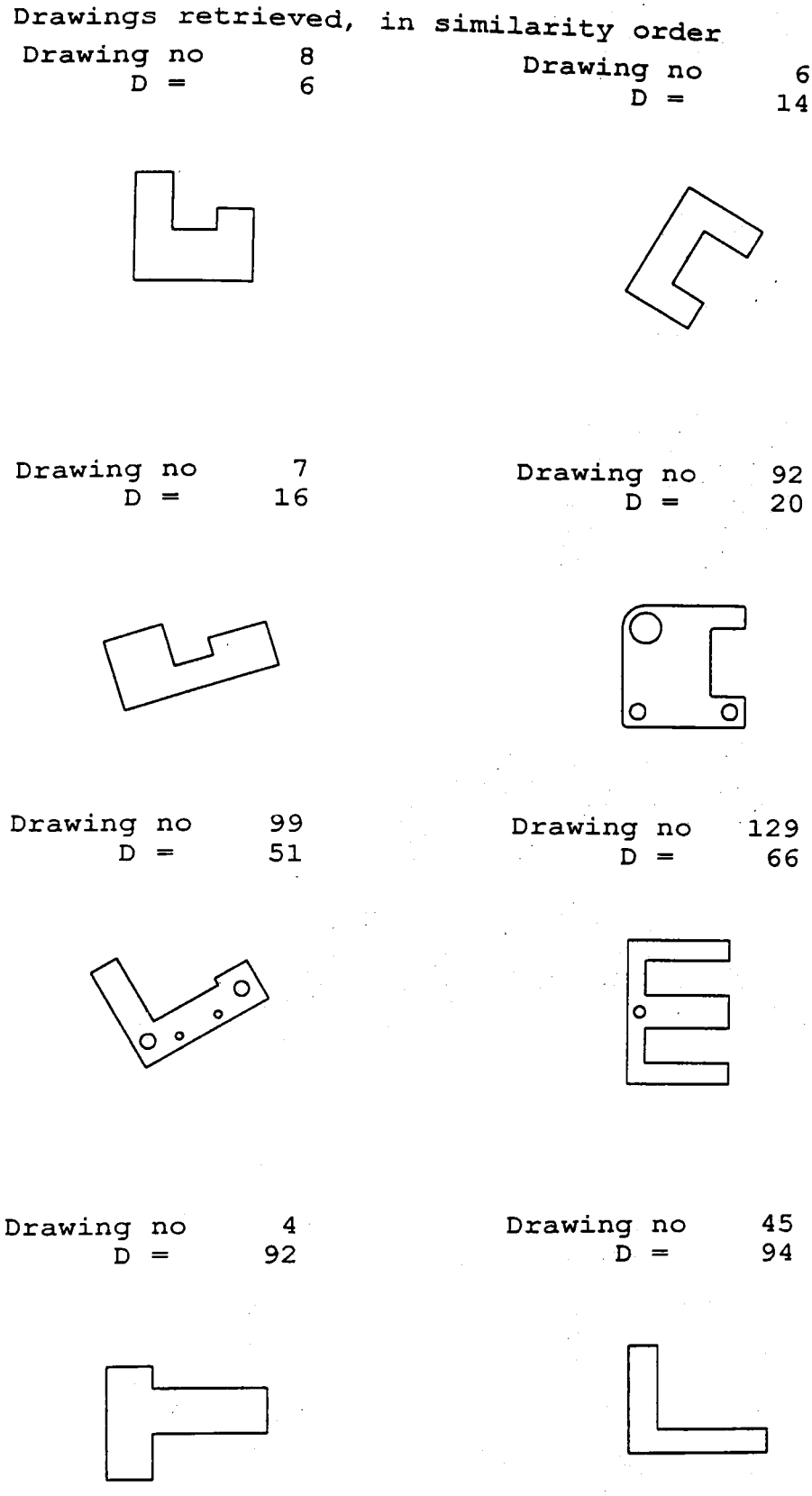


Fig 8.11 Results from SAFARI (using existence matching followed by segment matching) for query shape 9 - illustrating the eight stored shapes judged to be most similar to the query

Table 8.4.2.1 shows the first 12 drawings retrieved in response to this query, respectively using global feature matching as defined in section 6.6.2 (using all the global features listed in section 4.4.2), local feature matching (as defined in section 6.6.3.3), and existence matching, both based on the parent feature and arc angle distribution features enumerated above, followed in each case by segment matching.

Table 8.4.2.1 - system response to query shape 9

Global matching		Local matching		Existence matching	
Drawing No	Difference Measure	Drawing No	Difference Measure	Drawing No	Difference Measure
8	0.61	8	0.61	8	0.61
6	1.38	6	1.38	6	1.38
7	2.73	7	2.12	7	1.58
4	4.50	45	5.41	92	2.04
99	4.70	94	5.45	99	5.07
45	4.75	4	5.51	129	6.56
94	4.78	99	7.30	4	9.23
92	6.72	3	8.80	45	9.40
28	7.01	28	9.44	94	9.41
57	8.22	165	10.33	128	9.46
50	8.38	92	10.42	120	9.71
60	8.38	1	10.80	50	11.68

All of the 12 shapes retrieved in response to this query (an asymmetric bracket) show some family resemblance to the query, though the resemblance is perhaps less obvious with some of the later shapes. The results compare well with student judgements of relevance (Table 8.4.2.2):

Table 8.4.2.2

Drawing No	Similarity Measure
8	0.600
6	0.583
7	0.292

The three methods of matching also agree with each other remarkably well over the first three drawings retrieved, though they diverge considerably over the lower rankings. Note that the actual values of the difference measure D have no significance for the system other than as a way to rank drawings in order of presumed relevance to a query. The fact that drawings 8 and 6 have the same D values for all three methods reflects the fact that both of these drawings were judged identical to the query at the feature-matching stage. Only at the segment-matching stage (common to all three methods) was any difference detected.

Some measure of the robustness of these rankings can be gained by examining D values. In the case above, the first three rankings in each list look reasonably secure. However, the next four D values in the global matching list are all very similar, and a very small change in parameter weightings could easily have resulted in a different ordering for drawings 99, 45, 94 and 92 - emphasizing the importance of presenting the user with D values as well as actual rankings. Ideally, one might expect some step change in D values

between the last "relevant" drawing and the first "non-relevant" one, though this would be a very severe test of a system's discriminating power.

The results from another typical query are shown in Fig 8.12 and Table 8.4.2.3, though here the system's performance is not quite so good, and a higher cutoff is needed to retrieve all shapes judged relevant by students. Again, the diagram shows the results of parent feature and arc angle distribution-based existence matching followed by segment matching, while the table below compares global feature matching, local feature matching and existence matching. Most of the 20 retrieved shapes are again a plausible response to the query, though some (such as drawing 168) contain features which human judges considered undesirable.

Table 8.4.2.3 - system response to query shape 47

Global matching		Local matching		Existence matching	
Drawing No	Difference Measure	Drawing No	Difference Measure	Drawing No	Difference Measure
59	0.00	59	0.00	59	0.00
142	0.60	142	0.38	142	0.37
53	0.91	53	0.55	53	0.50
168	6.21	168	3.30	15	2.04
68	6.68	68	8.05	68	2.48
134	10.66	15	17.04	168	5.98
187	11.30	134	19.43	78	14.39
12	12.64	187	20.01	172	14.51
36	15.33	172	20.33	36	19.21
174	17.26	151	20.70	134	19.71
15	17.51	36	21.72	187	20.34
172	17.98	174	22.00	151	20.74
151	18.60	26	24.06	12	20.84
26	18.68	12	24.83	174	22.32
98	19.86	98	25.73	167	27.22
31	23.09	167	26.86	166	27.53
78	23.59	78	27.08	169	27.62
3	27.59	170	28.04	170	27.86
108	28.33	3	31.67	125	29.82
89	28.35	166	32.65	26	29.98

This compares moderately well with student judgements of relevance (Table 8.4.2.4):

Table 8.4.2.4

Drawing No	Similarity Measure
59	0.883
53	0.567
142	0.492
68	0.183
12	0.150

Again, there is a high degree of agreement between the three methods used over the first three rankings, though divergence sets in lower down the list. The system appears to be

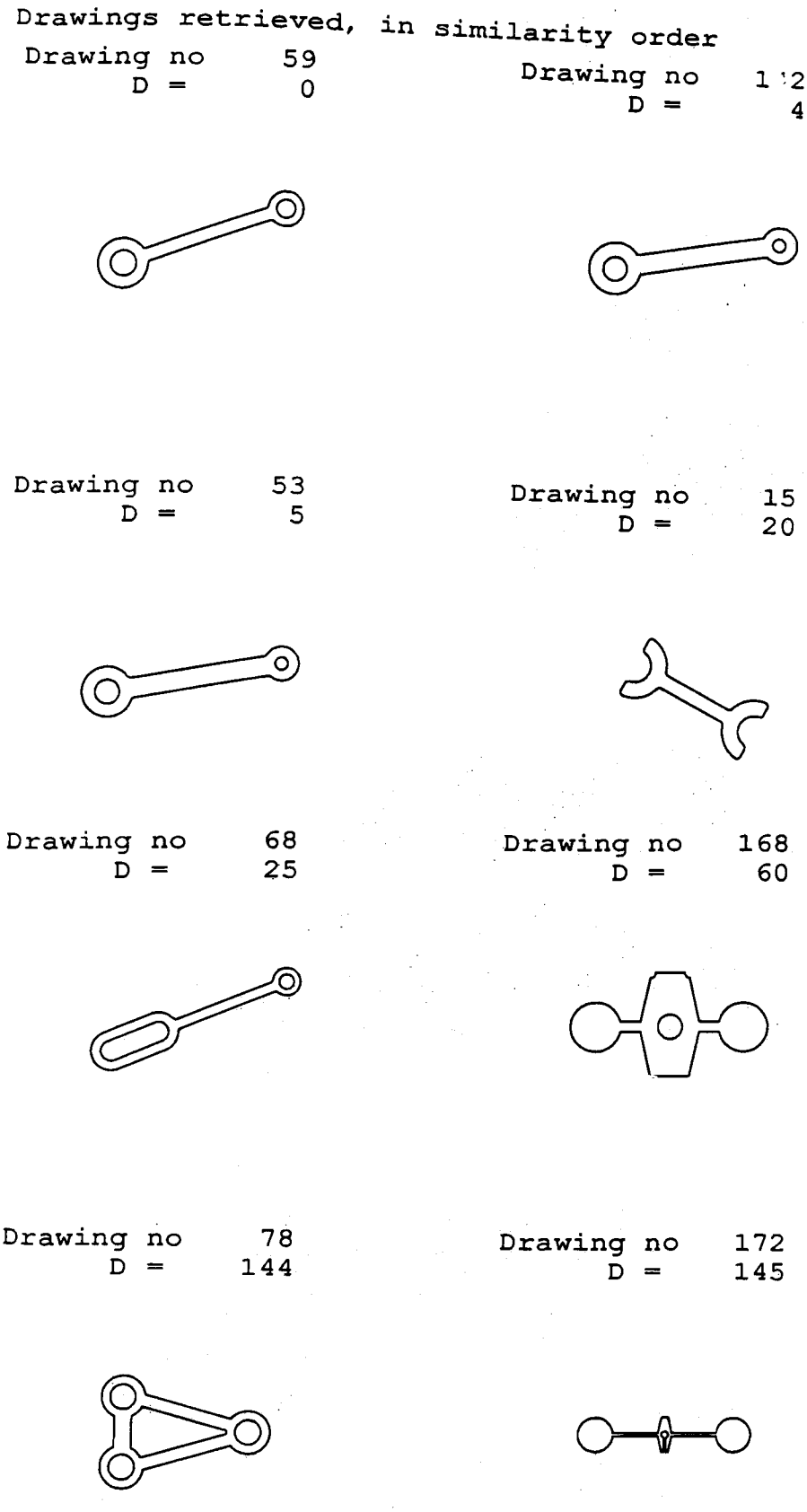


Fig 8.12 Retrieval results for query shape 47, again illustrating the eight most similar stored shapes

quite good at picking out very similar drawings, but perhaps less good at predicting how human subjects will rank less similar items. Drawing 12, for example, does not appear till position 14 on the local feature matching list, and 13 on the existence matching list. Some possible reasons for relative retrieval failures such as this are discussed below.

8.4.3 Comparative results - outer boundary queries

System rankings of drawings deemed relevant by student judges are presented below for three of the 16 queries. (Results for the remaining queries are shown in Appendix B). Four different system rankings are shown in each case, each resulting from a different method of similarity estimation, as listed below. In each case, the parameter combination chosen was that which was found to give the best overall retrieval performance. Matching was obviously limited to drawing outer boundaries for these experiments.

- global feature matching, based on all the global features listed in section 4.4.2;

- local feature matching, based on arc angle distribution, discontinuity angle distribution and parent feature distribution (features 6, 7 and 8 from the list in section 4.4.3);

- existence matching, based on the parent feature and arc angle distribution features listed earlier (6, 8, 10 and 12 from the list in section 4.4.3);

- segment matching, combining top and bottom-level θ-s matching as defined in section 6.6.4.1, with an attenuation factor of 0.3 (section 6.6.4.2).

Results are presented in tabular form for each query, showing for each method the rank at which each drawing judged relevant by human subjects was actually retrieved, and normalized recall and precision measures R_n and P_n. Retrieved drawings are listed in the order in which they were ranked by human subjects. The query shapes are illustrated in Fig 8.5.

Table 8.4.3.1 - Query shape 44

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
42	9	1	7	1
41	3	5	3	3
43	15	11	35	13
180	5	8	14	51
104	2	4	2	14
40	30	3	8	17
R_n	0.9516	0.9876	0.9459	0.9122
P_n	0.7822	0.9154	0.7693	0.7244

This was a query where local feature matching performed well, and other types of matching gave adequate results. All feature matching methods readily detected similarity between the query (a scalene triangle) and other triangles of similar shape (drawings 41, 42 and 104), even where the ends were rounded off, though global matching failed with drawing 40, where corners were rounded off to a very marked extent. The system was less successful in finding the two right-angled triangles 43 and 180; the query shape contained no right angles, so the system made no attempt to look for them.

Table 8.4.3.2 - Query shape 75

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
83	1	1	1	9
140	3	2	13	3
74	20	12	8	1
76	2	6	5	7
R_n	0.9733	0.9817	0.9717	0.9833
P_n	0.9049	0.8942	0.8183	0.8781

Generally good performance, though each method had at least one flaw. Students judged four stored shapes similar to the query, a circular disc with four rectangular notches equally spaced around its circumference. All were basically circular, but 83 (like 75) had four notches, 140 had two, 74 was a perfect circle, and 76 had two rectangular protrusions on its circumference. Both global and local feature matching performed adequately, though giving too low a rank to the perfect circle (which generated few feature types). Existence matching (based on a search for arc angle triplets and parent features) ranked drawing 140 too low, and segment matching retrieved drawings in reverse order of similarity!

Table 8.4.3.3 - Query shape 120

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
129	2	1	4	83
118	1	2	2	118
124	21	21	1	141
128	3	8	3	1
R_n	0.9717	0.9633	1.0000	0.4450
P_n	0.9021	0.8441	1.0000	0.3526

An excellent result for existence matching, fair for global and local feature matching, but hopeless for segment matching. Students' judgements here seemed to have been based purely on angular similarity. In response to a query in the form of an E-shaped bracket with the central arm significantly shorter than the others, they retrieved shapes with equal-length arms (129), with an almost non-existent central arm (128), and with a central arm *longer* than the others (124). Existence matching using arc angle triplets and parent features proved most successful here (alone proving able to retrieve drawing 124 at a reasonable rank), almost certainly because this emphasized angular similarity rather than similarity in feature size.

8.4.4 Comparative results - all-boundary queries

System rankings of three of the 16 all-boundary queries are presented below in a similar format (results for the remaining queries are presented in Appendix B). In this case, five different system rankings are shown for each query, each resulting from a different method of similarity estimation, as follows:

- global feature matching, again based on all global features;

- local feature matching, based as before on arc angle, discontinuity angle and parent feature distribution;

- existence matching, again based on arc angle and parent feature distribution, arc angle triplet and parent feature composition features;

- segment matching as defined above, combining outer boundary θ-s matching with inner boundary position matching, as defined in section 6.6.4.3.

- segment matching combining outer boundary θ-s matching with full inner boundary shape matching, as defined in section 6.6.4.3.

Results for each query are presented in the same form as for the outer-boundary queries in section 8.4.3 above, comparing system rankings and R_n and P_n measures for each shape-matching method. The query shapes themselves are shown in Fig 8.6.

Table 8.4.4.1 - query shape 49

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
61	4	2	1	8	5
88	1	3	20	14	8
56	3	1	2	12	6
110	2	4	7	7	3
R_n	1.0000	1.0000	0.9701	0.9536	0.9820
P_n	1.0000	1.0000	0.8584	0.6559	0.8040

Shape 49, a rectangular plate with one corner chamfered off, proved a useful demonstration of the system's power to distinguish between closely similar shapes. The feature matching methods (particularly global and local feature matching) had little difficulty in identifying the four desired shapes from over 60 similar straight-edged drawings. Existence matching performed less well, though the low ranking of drawing 88 (and to a lesser extent drawing 110) was in fact a problem caused by the feature set used rather than by existence matching *per se*. As shown in Fig 8.13, drawings 56 and 61 (but not 88 or 110) are fundamentally rectangular. Hence the top-level shapes of drawings 56 and 61, and the status of their low-level line segments, are quite different from those of drawings 88 and 110. The feature set used here for global and local matching emphasized line curvature and discontinuity angle, with successful results. Existence matching, by contrast, used more complex parameters such as arc angle triplet and parent feature composition, emphasizing the difference between those shapes which were basically rectangular (56 and 61) and those which were not (88 and 110). Segment matching was not conspicuously successful, though the addition of inner boundary shape matching markedly improved its performance, successfully rejecting shapes whose inner boundaries were not all circular.

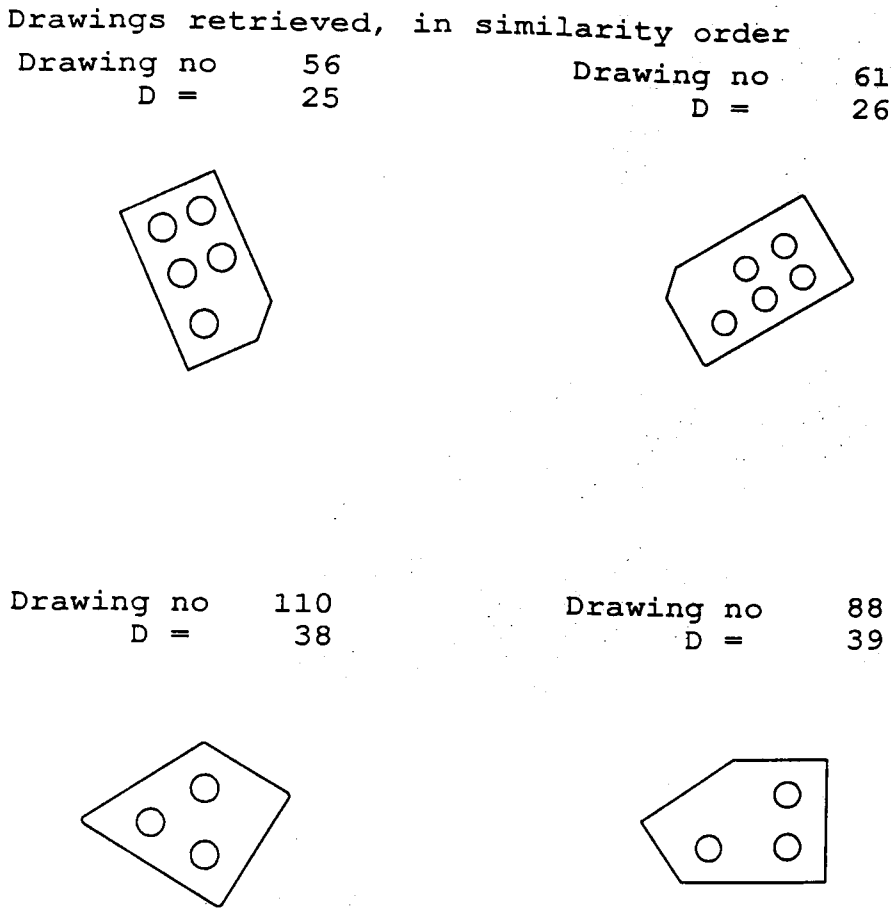


Fig 8.13 Results for query shape 49; two of the retrieved (and relevant!) drawings (61 and 56) are clearly derived from underlying rectangles, while the other two (88 and 110) are not.

Table 8.4.4.2 - query shape 100

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
105	1	2	5	92	4
181	6	1	19	33	25
178	71	4	20	32	15
R_n	0.8571	0.9980	0.9246	0.7004	0.9246
P_n	0.6869	0.9789	0.5771	0.2881	0.5945

An unusual seven-sided shape which caused the system some difficulty. Only local feature matching proved really equal to the task of identifying all three shapes (containing five, ten and eight sides respectively) deemed similar by student judges. Global matching failed dismally with drawing 178, almost certainly because this shape was much more regular than the other two, and hence had much lower values for length and discontinuity angle variances. Existence matching also performed poorly, ranking drawings 181 and 178 too low because they contained too few of the required types of arc angle triplet or parent feature composition. Segment matching proved the worst method of all, almost certainly due to differences in outer boundary starting point, at least where inner boundary matching was based solely on position. Where inner boundary shape was taken into account, performance improved markedly - to be expected given the close similarity of inner boundary shape between the query and all retrieved drawings.

Table 8.4.4.3 - query shape 154

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
149	1	1	1	2	9
171	2	2	2	1	1
79	3	3	5	5	8
83	6	7	4	19	22
74	4	4	8	6	3
R_n	0.9988	0.9976	0.9940	0.9783	0.9663
P_n	0.9913	0.9839	0.9530	0.8921	0.8237

An exercise in inner boundary matching, as illustrated in Fig 8.14. The query, and hence most retrieved shapes, were circular discs with complex patterns of inner boundaries. All three feature-matching methods performed well, and segment matching performed better than on many queries. Its only real "failure" was with drawing 83, which did not have a completely circular outer boundary, and was thus ranked below many drawings which did - possibly indicating that the relative weighting given to outer boundary shape similarity was too high here. The fact that segment matching using inner boundary shapes gave poorer results than when using position matching suggests that there may still be room for improvement in selection of the order in which inner boundaries from query and drawing are matched with each other.

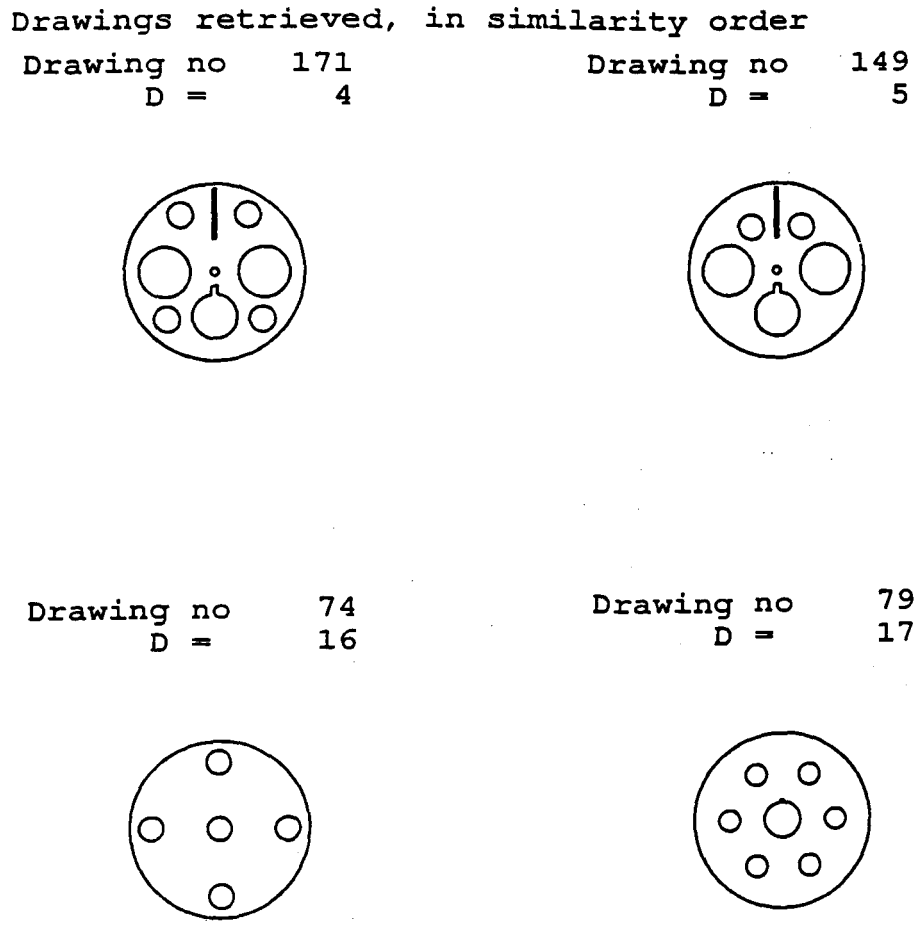


Fig 8.14 Illustration of inner-boundary shape matching with query 154

8.4.5 Comparison of matching techniques

Tables 8.4.5.1 and 8.4.5.2 below present comparative P_n and R_n scores for each query, together with the median, mean and standard deviation of all scores for each group of queries. Statistical analysis of P_n and R_n scores for each of the shape-matching methods was performed to establish whether any of the methods performed consistently better than the others. The non-parametric Wilcoxon matched-pairs, signed-rank test was used, since scores for each query for each pair of methods being compared formed comparable pairs, and no information was available about the underlying distribution of the data.

Table 8.4.5.1 - P_n and R_n scores for outer boundary queries

Query no		Global matching score	Local matching score	Existence matching score	Segment matching score
32	R_n	0.9671	0.9474	0.9375	0.8355
	P_n	0.7442	0.6372	0.6177	0.6524
44	R_n	0.9516	0.9876	0.9459	0.9122
	P_n	0.7822	0.9154	0.7693	0.7244
46	R_n	0.9906	0.9879	0.9799	0.9275
	P_n	0.9408	0.9228	0.8877	0.7529
47	R_n	0.9910	0.9820	0.9764	0.9561
	P_n	0.9505	0.9181	0.9026	0.8647
57	R_n	0.9470	0.9316	0.9581	0.8940
	P_n	0.7155	0.6674	0.7278	0.5944
62	R_n	0.9517	0.9181	0.8872	0.9517
	P_n	0.8029	0.7625	0.7116	0.8301
67	R_n	1.0000	1.0000	1.0000	1.0000
	P_n	1.0000	1.0000	1.0000	1.0000
72	R_n	0.9933	0.9867	0.9933	0.7917
	P_n	0.9049	0.8400	0.9049	0.3626
75	R_n	0.9733	0.9817	0.9717	0.9833
	P_n	0.9049	0.8942	0.8183	0.8781
103	R_n	1.0000	0.9978	0.9890	0.9978
	P_n	1.0000	0.9784	0.9058	0.9784
115	R_n	0.9934	1.0000	1.0000	0.9757
	P_n	0.9479	1.0000	1.0000	0.8436
120	R_n	0.9717	0.9633	1.0000	0.4450
	P_n	0.9021	0.8441	1.0000	0.3526
170	R_n	0.9973	0.9879	1.0000	0.3570
	P_n	0.9835	0.9420	1.0000	0.2993
175	R_n	0.9866	0.9208	0.9705	0.9651
	P_n	0.9228	0.7424	0.7991	0.8413
176	R_n	0.9603	0.9823	0.9558	0.9073
	P_n	0.8537	0.9023	0.7265	0.7477
183	R_n	0.9917	0.9983	0.9983	0.9333
	P_n	0.9368	0.9868	0.9868	0.7012
R_n	Median	0.9886**	0.9845*	0.9781*	0.9304
	Mean	0.9792	0.9716	0.9709	0.8646
	S.D.	0.0187	0.0283	0.0309	0.1901
P_n	Median	0.9139**	0.9088*	0.8951	0.7503
	Mean	0.8933	0.8636	0.8505	0.7140
	S.D.	0.0889	0.1135	0.1225	0.2152

*Asterisks indicate performance differing significantly from that for segment matching (Wilcoxon matched-pairs signed-rank test), ** :- P < 0.001, * :- P < 0.01*

Table 8.4.5.2 - P_n and R_n scores for all-boundary queries

Query no		Global matching score	Local matching score	Existence matching score	Segment matching score (position,	Segment matching score shape
9	R_n	1.0000	1.0000	1.0000		
	P_n	1.0000	1.0000	1.0000		
47	R_n	0.9976	0.9855	0.9867		
	P_n	0.9806	0.9335	0.9365		
48	R_n	1.0000	1.0000	1.0000		
	P_n	1.0000	1.0000	1.0000		
49	R_n	1.0000	1.0000	0.9701		
	P_n	1.0000	1.0000	0.8584		
57	R_n	0.9722	0.9206	0.9425		
	P_n	0.8011	0.6743	0.7157		
72	R_n	0.9940	0.9940	0.9940		
	P_n	0.9073	0.9073	0.9073		
80	R_n	0.9795	0.9904	0.9783		
	P_n	0.9205	0.9386	0.8958		
89	R_n	0.9759	0.9940	0.9807		
	P_n	0.9141	0.9668	0.8809		
100	R_n	0.8571	0.9980	0.9246		
	P_n	0.6869	0.9789	0.5771		
109	R_n	0.9861	0.9960	0.9883		
	P_n	0.9116	0.9625	0.8098		
120	R_n	1.0000	0.9940	0.9970		
	P_n	1.0000	0.9601	0.9766		
122	R_n	0.9623	0.9980	1.0000		
	P_n	0.8537	0.9789	1.0000		
154	R_n	0.9988	0.9976	0.9940		
	P_n	0.9913	0.9839	0.9530		
159	R_n	0.9538	0.9556	0.9512		
	P_n	0.8591	0.8787	0.8547		
175	R_n	0.9590	0.9735	0.9771		
	P_n	0.9015	0.8809	0.8916		
176	R_n	0.9701	0.9641	0.9596		
	P_n	0.7966	0.8879	0.7379		
R_n	Median	0.9828*	0.9940*+	0.9795*		
	Mean	0.9754	0.9851	0.9765		
	S.D.	0.0355	0.0218	0.0227		
P_n	Median	0.9128*+	0.9613*+	0.8937*		
	Mean	0.9078	0.9333	0.8747		
	S.D.	0.0911	0.0810	0.1174		

* :- performance differing significantly from that for segment (boundary position only), $P < 0.01$

\+ :- performance differing significantly from that for segment (boundary shape), $P < 0.01$

The scores obtained with these test queries suggest that the SAFARI system is capable of yielding an acceptable level of performance. R_n and P_n scores compare favourably with those obtained for document retrieval in the SMART experiments, where R_n values typically varied between 0.9 and 1.0, and P_n values between 0.6 and 1.0 (Salton, 1971). All three feature-matching methods (particularly global matching) gave performance markedly superior to the segment-matching methods, for both outer-boundary and all-boundary queries, though no one method stood out as significantly superior to either of the others. Full segment matching of all inner boundary shapes appeared to give marginally better performance with all-boundary queries than matching limited to inner boundary positions, though the differences observed with the test queries were not significant. It is perhaps worth noting that differences in median scores between segment and feature matching methods were less striking than those in mean scores - suggesting that an important aspect of feature matching's superiority is its success in avoiding disastrous results such as those seen with segment matching on queries 120 and 170.

The fact that all three feature-matching methods could be "tuned" to give comparable levels of performance (albeit with different combinations of parameters) suggests that a wide range of feature types could have been used successfully in these circumstances. It is perhaps noteworthy that the optimum levels of performance observed with local feature and existence matching were based on three and four basic feature types respectively. Adding extra feature types, or introducing more sophisticated techniques such as penumbral matching (section 6.6.3.5), caused a slight but measurable *decline* in performance for the query shapes tested. Some of the simplest methods thus gave the best results, a phenomenon not unknown in the text retrieval field (e.g. Sparck Jones, 1981).

On the basis of these results, it is hard to justify the use of segment matching as a technique on its own. Although it appears to be a valuable technique for determining whether shapes are identical, problems in determining the correct start point for θ-s matching in a heterogeneous set of shapes effectively rule it out as an effective general technique for similarity estimation. Its usefulness for similarity matching thus stands or falls on the degree to which it is effective in combination with one of the feature-matching techniques, where segment matching is used to discriminate between shapes retrieved by a preliminary feature matching step, as outlined in section 6.5. The fact that segment matching does appear to be a sensitive discriminator of closely related shapes is suggestive evidence that combined matching could prove effective.

Tests were thus run to compare the retrieval performance of each of the three feature-matching methods alone and in combination with segment matching. The same parameter sets were chosen for each type of feature matching as in previous experiments; where combined matching was used, the set of drawings retrieved by feature matching was then subjected to segment matching at the most appropriate level (outer boundary only for outer-boundary queries, inner boundary class and size (the combination giving the highest overall scores) for all-boundary queries). The results of these tests are presented in tables 8.4.5.3 and 8.4.5.4 below.

The addition of segment matching appeared to produce a modest improvement over feature matching alone with the test queries used - at least when using global or existence matching. The results with local feature matching were more equivocal. However, none of the differences observed were statistically significant, and one cannot therefore conclude from these experiments that combined feature and segment matching has any consistent advantage over feature matching alone. While it would be premature to dismiss segment matching completely as a means of identifying shapes (it remains the only reliable means of detecting shapes *identical* to a query), its overall usefulness would appear to be limited.

Table 8.4.5.3 - P_n and R_n scores for outer boundary queries

Query no		Global matching		Local matching		Exist. matching	
		Alone	With segmatch	Alone	With segmatch	Alone	With segmatch
32	R_n	0.9671	0.9967	0.9474	0.9605	0.9375	0.9441
	P_n	0.7442	0.9567	0.6372	0.7924	0.6177	0.7598
44	R_n	0.9516	0.9718	0.9876	0.9842	0.9459	0.9538
	P_n	0.7822	0.8333	0.9154	0.9022	0.7693	0.8406
46	R_n	0.9906	0.9960	0.9879	0.9960	0.9799	0.9919
	P_n	0.9408	0.9659	0.9228	0.9659	0.8877	0.9384
47	R_n	0.9910	0.9910	0.9820	0.9854	0.9764	0.9820
	P_n	0.9505	0.9489	0.9181	0.9296	0.9026	0.9173
57	R_n	0.9470	0.9735	0.9316	0.9735	0.9581	0.9868
	P_n	0.7155	0.7915	0.6674	0.7676	0.7278	0.8537
62	R_n	0.9517	0.9785	0.9181	0.9557	0.8872	0.8953
	P_n	0.8029	0.8996	0.7625	0.8480	0.7116	0.7534
67	R_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	P_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
72	R_n	0.9933	0.9867	0.9867	0.9867	0.9933	0.9433
	P_n	0.9049	0.8400	0.8400	0.8400	0.9049	0.6602
75	R_n	0.9733	0.9950	0.9817	0.9917	0.9717	0.9900
	P_n	0.9049	0.9669	0.8942	0.9368	0.8183	0.9128
103	R_n	1.0000	1.0000	0.9978	1.0000	0.9890	0.9934
	P_n	1.0000	1.0000	0.9784	1.0000	0.9058	0.9479
115	R_n	0.9934	1.0000	1.0000	1.0000	1.0000	1.0000
	P_n	0.9479	1.0000	1.0000	1.0000	1.0000	1.0000
120	R_n	0.9717	0.9400	0.9633	0.9367	1.0000	0.9983
	P_n	0.9021	0.8640	0.8441	0.8246	1.0000	0.9868
170	R_n	0.9973	1.0000	0.9879	0.9866	1.0000	0.9973
	P_n	0.9835	1.0000	0.9420	0.9384	1.0000	0.9801
175	R_n	0.9866	0.9906	0.9208	0.9624	0.9705	0.9919
	P_n	0.9228	0.9408	0.7424	0.8427	0.7991	0.9428
176	R_n	0.9603	0.9558	0.9823	0.9779	0.9558	0.9558
	P_n	0.8537	0.8468	0.9023	0.8897	0.7265	0.8468
183	R_n	0.9917	0.9933	0.9983	1.0000	0.9983	0.9983
	P_n	0.9368	0.9500	0.9868	1.0000	0.9868	0.9868
R_n	Median	0.9886	0.9922	0.9845	0.9860	0.9781	0.9910
	Mean	0.9792	0.9856	0.9716	0.9811	0.9709	0.9764
	S.D.	0.0187	0.0175	0.0283	0.0188	0.0309	0.0297
P_n	Median	0.9139	0.9494	0.9088	0.9159	0.8951	0.9279
	Mean	0.8933	0.9253	0.8636	0.9049	0.8505	0.8955
	S.D.	0.0889	0.0694	0.1135	0.0784	0.1225	0.1014

Table 8.4.5.4 - P_n and R_n scores for all-boundary queries

Query no		Global matching		Local matching		Exist. matching	
		Alone	With segmatch	Alone	With segmatch	Alone	With segmatch
9	R_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	P_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
47	R_n	0.9976	0.9952	0.9855	0.9880	0.9867	0.9892
	P_n	0.9806	0.9668	0.9335	0.9399	0.9366	0.9435
48	R_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	P_n	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
49	R_n	1.0000	1.0000	1.0000	0.9970	0.9701	0.9790
	P_n	1.0000	1.0000	1.0000	0.9706	0.8584	0.8790
57	R_n	0.9722	1.0000	0.9206	0.9921	0.9425	0.9960
	P_n	0.8011	1.0000	0.6743	0.9116	0.7157	0.9491
72	R_n	0.9940	0.9925	0.9940	0.9880	0.9940	0.9416
	P_n	0.9073	0.8967	0.9073	0.8439	0.9073	0.6286
80	R_n	0.9795	0.9976	0.9904	0.9952	0.9783	0.9928
	P_n	0.9205	0.9839	0.9386	0.9718	0.8958	0.9506
89	R_n	0.9759	0.9867	0.9940	0.9952	0.9807	0.9867
	P_n	0.9141	0.9366	0.9668	0.9718	0.8809	0.9141
100	R_n	0.8571	0.8651	0.9980	0.9940	0.9246	0.9266
	P_n	0.6869	0.6810	0.9789	0.8982	0.5771	0.5811
109	R_n	0.9861	0.9504	0.9960	0.9663	0.9683	0.9345
	P_n	0.9116	0.7607	0.9625	0.7877	0.8098	0.7410
120	R_n	1.0000	0.9940	0.9940	0.9880	0.9970	1.0000
	P_n	1.0000	0.9601	0.9601	0.9367	0.9766	1.0000
122	R_n	0.9623	0.9921	0.9980	0.9980	1.0000	1.0000
	P_n	0.8537	0.9378	0.9789	0.9789	1.0000	1.0000
154	R_n	0.9988	0.9892	0.9976	0.9867	0.9940	0.9988
	P_n	0.9913	0.9506	0.9839	0.9442	0.9530	0.9913
159	R_n	0.9538	0.9869	0.9556	0.9869	0.9512	0.9782
	P_n	0.8591	0.9493	0.8787	0.9581	0.8547	0.9444
175	R_n	0.9590	0.9554	0.9735	0.9530	0.9771	0.9530
	P_n	0.9015	0.8980	0.8809	0.8723	0.8916	0.8723
176	R_n	0.9701	0.9835	0.9641	0.9910	0.9596	0.9701
	P_n	0.7966	0.9112	0.8879	0.9472	0.7379	0.8379
R_n	Median	0.9828	0.9923	0.9940	0.9915	0.9795	0.9880
	Mean	0.9754	0.9805	0.9851	0.9887	0.9765	0.9779
	S.D.	0.0355	0.0342	0.0218	0.0125	0.0227	0.0254
P_n	Median	0.9128	0.9500	0.9613	0.9457	0.8937	0.9440
	Mean	0.9078	0.9270	0.9333	0.9333	0.8747	0.8896
	S.D.	0.0911	0.0889	0.0810	0.0583	0.1174	0.1320

8.4.6 Efficiency of matching process

As discussed in section 6.8, computational efficiency was not felt to be a major consideration in the design of the prototype version of SAFARI. No systematic attempt was therefore made either to measure performance in detail or to tune the system for greater efficiency. Nevertheless, such considerations cannot be totally ignored. Records were therefore kept of cpu usage for each run. As indicated in section 3.4 above, these figures must be treated with some caution, as they represent total cpu usage for the current process, including any required by the operating system paging the process in or out of main memory. They do however give some indication of the relative computational expense of the different matching techniques used. The figures shown in table 8.4.6.1 below (presented as ranges, in order to show the spread of values involved) represent total cpu usage (in seconds) for a complete search of the database for a single query, when run in batch mode on the VAX 8700 processor at Newcastle Polytechnic. This process involves reading in the query file from disc, and matching it successively with up to 154 stored shapes (outer boundary queries) or 171 stored shapes (all-boundary queries).

Table 8.4.6.1 - comparative cpu usage of different matching techniques

Method used	Query set used	cpu usage (s) for complete database search	
		Minimum	Maximum
Feature matching alone:			
Global	All-boundary	0.8	1.6
	Outer-boundary	0.6	1.2
Local	All-boundary	0.8	2.1
	Outer-boundary	0.6	1.5
Existence	All-boundary	1.2	17.5
	Outer-boundary	1.3	15.7
Segment matching alone:*			
Outer boundary only		13.2	14.1
Inner boundary position		15.7	17.4
Inner boundary class & size		16.1	25.5
Inner boundary shape		15.2	55.1
Combined feature and segment matching+:			
Global	All-boundary	2.6	20.0
	Outer-boundary	1.9	12.3
Local	All-boundary	2.9	19.8
	Outer-boundary	2.0	12.5
Existence	All-boundary	3.0	35.7
	Outer-boundary	2.5	26.3

* *Outer-boundary query set used for* ***outer boundary only*** *matching, all-boundary set for the other three methods.*

\+ ***Inner boundary class and size*** *segment matching used with all-boundary queries,* ***outer boundary only*** *matching with outer-boundary queries.*

Feature matching clearly seems to be more economical in its use of resources than segment matching, making it even harder to justify the use of segment matching in the

retrieval process, at least with a small collection such as this. Note that existence matching is not more resource-intensive *per se*; the cpu usage figures in the table above are based on runs using the parameter set giving the highest overall P_n and R_n scores. The most effective parameter combination for existence matching used features from both boundary level and boundary feature records, thus requiring considerably more processing than with local feature matching, where the most effective features were all located in boundary level records. Note also the small variation in cpu times different queries for segment-only matching - a reflection of the fact that no pre-screening was performed here, and hence that every shape in the database had to be searched in detail.

8.5 Conclusions

The results of this (admittedly artificial) evaluation experiment suggest strongly that the approach to shape matching adopted for SAFARI has the potential to deliver acceptable retrieval results in an operational environment, at least within its own specific domain of shapes. Further testing of the existing prototype with "real" queries would obviously be worthwhile, though this in turn requires a "real" database of shapes, rather than the present somewhat artificial test collection, if the results are to yield valid additional information. It could however be more profitable in the long run to extend SAFARI's capabilities before any further evaluation is attempted, to allow it to handle a sufficiently wide range of shapes for tests with operational drawing collections to become possible - a point discussed further in chapter 9.

It is tempting to try to draw conclusions about the relative merits of different feature sets or shape-matching techniques. However, the size and nature of the test collection is such that no generalization of this kind can be considered valid. The specific feature sets used here have proved useful with one collection of shapes. Their value in retrieval with other shape collections remains to be established. It is perhaps encouraging that all three approaches to feature-matching seem capable of yielding acceptable retrieval performance, and that performance did not seem particularly sensitive to the precise feature set chosen. This would seem to suggest a certain degree of robustness in the basic approach.

CHAPTER 9. CONCLUSIONS

9.1 Summary of findings

The basic objective of this project - to investigate the problems of developing an engineering database with shape retrieval capabilities through the construction and evaluation of a prototype system - has been achieved. A prototype system, SAFARI, has been developed, and has demonstrated an acceptable level of retrieval performance within a limited but far from trivial domain of shapes, suggesting that the principles adopted in its design could well prove useful in an operational shape retrieval system. To gauge the overall success of the project, it is perhaps worthwhile to review each of the design decisions taken on the issues listed in section 1.8 in turn.

In a sense, the first important decision concerned the project itself rather than the system. This was the decision to investigate the development process as a whole via prototype development, rather than to concentrate on a single issue such as devising improved shape matching algorithms - in other words, to adopt a breadth-first rather than a depth-first approach to the problem. The abundance of published literature concentrating on detailed aspects of the problem contrasts vividly with the paucity of reports adopting a broader view, suggesting strongly that the approach adopted for the present project is not only justified, but probably overdue!

The first group of decisions concerning the system itself involved the definition of its scope and nature. As discussed in chapter 2, it was decided to limit the domain of shapes acceptable to the prototype system to genuine 2-D objects capable of being stamped out of sheet metal. This domain was further limited by restricting boundary segments to straight lines or circular arcs - though it was noted that this still included the vast majority of machined parts. While the restriction to 2-D parts may seem unrealistic, it was almost certainly essential to the success of the project, in that the additional complexity involved in handling 2-D orthographic projections or 3-D geometric models would have prevented investigation of the full range of design issues in the time available. It is in any case a less important restriction than might be supposed - as indicated in section 9.3.3 below, the problem of generalizing the approach to 3-D geometric models of machined (as opposed to sculptured or moulded) parts should not be too onerous.

The decision to base input on a standard exchange format remains as valid now as when the project was started. The ability to accept drawings in standard format is essential for any operational system; even with a prototype, it provides a useful discipline and a clear starting-point. IGES was the only widely-used CAD exchange format at the time the project was started, and thus the automatic choice. One would expect future systems to make use of STEP (and any successor standards) in the same way.

The second group of decisions concerned shape representation (discussed in some detail in chapters 2 and 3). The majority of these decisions still seem sound in retrospect. Some of these were implicit, such as the decision to retain a complete (though condensed) representation of an object's shape, rather than just its characteristic shape features - essential for identity matching, or if retrieved shapes are ever to be displayed to the user. Others were discussed in more detail, such as the decision to represent drawings by defining their inner and outer boundaries rather than using area-filling representations such as quadtrees (section 2.4), and to represent such boundaries as sequences of straight-line or circular arc segments. Such representations have proved economical to store and efficient to process.

Perhaps the most important (and novel) aspect of shape representation within SAFARI has been the decision to view each boundary as a series of levels, and to link all extracted shape features to a specific level. This appears to be a crucial factor in ensuring successful retrieval performance (section 8.4.1). It is a design principle that has been

clearly vindicated within SAFARI, and would seem to be well worth considering for future shape retrieval systems.

The decision to derive a unique representation for each shape is more questionable. While there is clearly a need to represent shapes in a form invariant to translation, rotation and scaling, the advantages of selecting a unique starting-point for boundary traversal and a unique ordering for inner boundaries are less obvious. The rationale behind this decision (section 3.1) was that it could greatly improve the efficiency of the segment matching process, at the risk of failing to match a certain proportion of similar shapes. In practice, the proportion of shapes where the matching process fails has been higher than expected, and thus the usefulness of this form of unique representation as a basis for similarity matching must be regarded as not proven. However, it could still have a valuable role to play in a system where identity matching ("have we made this part before?") was of prime importance.

The outcome of the third group of decisions, concerning selection of retrieval features (chapter 4), is harder to evaluate. In one sense, they have proved highly successful - a relatively unsophisticated set of features, representing a very limited degree of shape understanding, has provided the basis for good retrieval performance with the test collection. The efficiency with which such features (with the possible exception of some inner boundary pattern features) could be extracted and matched was encouraging. It is noteworthy that some of the best retrieval scores were achieved by matching on three simple local features alone (arc angle class, discontinuity angle class, and parent feature type). Attempts at increasing the level of sophistication by introducing more complex features seemed to be counter-productive. Until evaluation can be performed with a wider set of drawings, however, it is impossible to determine the extent to which these features are generally useful.

The fourth group of decisions related to retrieval capability, and how it should be provided (chapters 5 and 6). The decision to concentrate on providing similarity matching in the prototype version of SAFARI was taken on purely pragmatic grounds - one had to start somewhere, and the limited evidence available suggested that this would be potentially the most useful type to provide. As discussed in section 6.4, this capability could readily be extended to cover other types of retrieval. The decision to provide a number of alternative matching paradigms in the prototype system made it possible to compare a number of different strategies. Given the minimal differences in performance between global, local and existence feature matching, this proved less worthwhile than expected. The principle of endowing the system with facilities for varying the depth and matching strategy for a given search seems generally sound, in view of the limited state of knowledge of users' retrieval needs in this area. The ability to vary run-time search parameters is unfortunately of dubious value at present, given the limited state of knowledge about their effects on system performance!

The facility to specify segment matching as a way of refining the results of a preliminary feature search is still considered potentially useful, even though none of the evaluation experiments reported above provided any conclusive evidence of their value. The shape database used for these experiments was small, and contained a very heterogeneous collection of drawings. Given a collection that was larger, more homogeneous, or both, feature matching on its own might well have produced less impressive results. In such cases, the ability of segment matching to discriminate between highly similar shapes might well come into its own. Such a facility should certainly be provided as an option in future systems. Whether segment matching under these circumstances should still be based on a unique boundary start point is an open question. Repeated segment matching, using each vertex on the drawing boundary in turn as starting point, could prove acceptably efficient if limited to a small subset of the database, and might give more reliable results.

The decision to use a database management system based on the somewhat unfashionable CODASYL model for the prototype system, justified in chapter 5, again proved sound in practice. No advantage could be taken of the power of relational query languages when implementing SAFARI's feature extraction and shape matching algorithms. The function of the DBMS was thus purely to provide access to individual stored data elements when required - a task for which CODASYL database management systems are well suited. The present investigation has not attempted to assess the suitability of alternative types of DBMS, such as the NF^2 model discussed in section 5.3.

The final group of decisions concerned interface design (chapter 7). As argued in section 8.1, it is premature to attempt any systematic evaluation of the existing interface, since this should be regarded purely as a vehicle for formulating test queries. Further work is clearly required in this area.

9.2 Further work required

Overall, therefore, the design decisions taken in the development of the prototype SAFARI system appear to have been vindicated. There is, however, scope for further development in several areas. Further evaluation studies should ideally be conducted on the current prototype with different test collections, preferably using both similarity and partial shape queries provided by would-be users of such a system. The system's similarity retrieval capabilities have been investigated in some detail *with one test collection*. The general applicability of these results must be in some doubt until they can be replicated with further collections of shapes. It can however be argued that more widespread evaluation (possibly including a comparison with manual part coding) should await the development of future versions capable of handling a wider domain of shapes. This issue is discussed in more detail in section 9.3 below.

Various aspects of fine-tuning could obviously be investigated in more detail, such as comparisons of alternative feature sets, matching techniques, and alternative ways of generating different boundary levels. It could be particularly useful to test the effect of allowing the shape hierarchy building module SKELETON to create and store alternative level hierarchies for shapes that have more than one acceptable parsing (chapter 3).

Possibly the most interesting areas for further development concern query interface design and data structuring. As discussed in section 7.4, only one of four potentially suitable types of query interface has yet been implemented. The task of implementing the remainder, integrating them with the rest of the system, and extending them to handle queries involving both shape elements and textual or numeric data, is far from trivial, even if the system remains restricted to 2-D shapes.

An operational shape database housing large collections of drawings would require more powerful data access methods than those provided in the prototype system. The question of whether these can be best provided via an underlying CODASYL, relational or even object-oriented DBMS, or through purpose-built file structures (like most bibliographic retrieval systems) should provide a fruitful area for further study.

9.3 Relevance to 3-D object retrieval

9.3.1 Introduction

As the foregoing discussion should have made clear, a system such as SAFARI will be of real use to engineers only when its scope can be enlarged to encompass 3-D shapes. An important measure of the usefulness of the Mark I prototype is thus the degree to which

the principles used in its design can be generalized to three dimensions. Two possible routes to such generalization could be taken - the system could be extended to handle *either* the 2-D projections of solid objects produced by draughting packages such as DOGS, *or* the 3-D geometric models produced by systems such as ROMULUS.

9.3.2 Orthographic projections of 3-D objects

On the face of it, extension to handle 2-D orthographic projections might seem the easier option. The system would still be dealing with two-dimensional patterns of lines following strict (though more complex) syntactical rules. Examination of typical drawings such as Fig 1.1 suggests that it would not be difficult to extend the scope of the boundary creation program LINEJOIN and the shape hierarchy builder SKELETON to cope with the situation where a line segment belonged to two or more partially-overlapping boundaries, though some degree of redundancy would inevitably be introduced, and the concept of a boundary level might need to be modified. There might also be an increase in the complexity of the algorithms involved. Reducing such a drawing to canonical form might prove a significant problem, but the evidence above suggests that there is little advantage to be gained by such a process. Feature generation would be a significantly more complex process, because many more types of relational feature as defined in section 4.2.3 would be necessary. Where each inner boundary simply represents a hole in a piece of sheet metal, a relatively rudimentary set of features representing their relative position and orientation can provide sufficient information for effective retrieval. Inner boundary patterns such as that illustrated in Fig 1.1, representing a variety of related features machined out to different depths, would clearly require a much wider feature set to reflect the relationships involved. The whole concept of an inner boundary family as defined in sections 4.4.4 and 4.5.5 would also need to be reviewed.

The major difficulty with this approach, though, lies with its overall philosophy. While engineers quite legitimately talk of retrieving a drawing from the archives, what they really require is the design specification of a particular object. The underlying object being modelled by the CAD system is the real target for retrieval, at least in the vast majority of situations. To regard a drawing representing a single projection of that object as a retrieval target in itself is a perfectly legitimate stance (see the discussion on general pictorial information retrieval systems in the next section). It is however unlikely fully to meet the needs of potential users, whose prime interest is in whether the object itself is the right shape to meet their requirements. To achieve this through the medium of 2-D orthographic projections, the system is likely to need to reconstruct the 3-D shape of the object in question before identifying and extracting features for use in retrieval. This, as indicated by Nagendra and Gujar (1988), is a major task, since three orthographic projections alone do not necessarily define an object uniquely, and some additional cues such as textual comments are often necessary to resolve possible ambiguities in 3-D shape (Yoshiura at al, 1984). While SAFARI could in principle be extended in this direction if the capability to handle libraries of 2-D orthographic projections was required, providing it with a 3-D shape reconstruction facility would be a major task.

9.3.3. 3-D geometric models

The other alternative, to extend SAFARI to handle 3-D geometric models directly, does have certain advantages, though this would again be a highly complex task. Starting with the premise (shared by all boundary representation schemes) that a solid object comprises the union of its faces, one can readily envisage a 3-D analogue of SAFARI in which each object would be represented as a set of faces, each defined in a manner similar to that described in chapter 3 above. The definition of each face would need to be extended in a similar manner to that described for orthographic projections above, since shape features

could themselves contain more detailed features, as indicated by Kyprianou (1980), though the extensions might well need to go further.

Pockets and bosses, shape features completely enclosed within a single face, would pose few problems even if nested. Shape rewriting rules similar to those used by Kyprianou could be used to generate descriptions of each face at different levels of detail, in an analogous manner to the boundary level in the present version of SAFARI. Slots, holes or any other feature spanning two or more faces would be more difficult to handle, as they would not fit neatly into SAFARI's existing type of shape hierarchy. It would probably be necessary to indicate that such features were jointly "owned" by all faces they touched, and therefore part of each face description, at least at the lowest level.

Further extensions to this representation would be needed where faces were not planar, an implicit assumption made by the present version of SAFARI. An indication would then be needed of the surface's curvature along different axes (relatively straightforward if the system was to be limited to spherical and cylindrical surfaces, in the way the present prototype is limited to circular arcs), and this information would need to be taken into account in calculating the relative position, orientation and labelling (as protrusion or depression) of shape features associated with that face.

Feature generation should not in itself be a difficult process, though considerable thought may need to be given to the task of feature selection, which would be no easier for 3-D objects than for 2-D shapes (section 4.2). Experience with SAFARI to date suggests that feature types much simpler than those identified by automatic feature recognizers for process planning (Henderson and Anderson, 1984; Lee and Fu, 1987) can be effective retrieval keys for 2-D shapes; whether this situation holds for 3-D objects remains to be established. Intuitively, one again feels (as with orthographic projections) that relational features will be more important for 3-D objects than for the simple 2-D shapes examined so far. Characterizing useful relational features may be difficult; extracting them from object descriptions is likely to prove a complex process.

Canonicalization of the kind attempted by the present version of SAFARI is unlikely to prove feasible in the 3-D context. Even assuming the degree of complexity involved in processing each face to be no more than that for a complete 2-D shape, finding a canonical ordering of n faces could require comparison of up to $n!$ alternative face orderings. As observed above, complete canonicalization is probably unnecessary. However, some partial ordering of faces could be justified on efficiency grounds if shape matching were to involve sequential matching of faces at any stage. Perhaps more importantly, the position and orientation of each local shape element (slot, boss or hole) need to be related to some invariant point and direction (possibly object centroid and principal axis - if any) before relational shape features can be generated. If partial matching of faces is to be attempted, some means of specifying an invariant point and direction on each face may be required.

Interface design for a 3-D version of SAFARI would probably present some difficulties. As discussed in section 7.3.3, few types of user interface are really suitable for formulating 3-D queries. Text-based command languages are equally unsuitable for formulating 2-D and 3-D queries. Example-based interfaces in the 3-D context are inevitably tedious to use, as the only effective way of building up an example query is to use some kind of 3-D geometric modeller. There is no 3-D equivalent of the rough hand-drawn sketch! One is thus left with menu-based interfaces (which, if based on the same principles as the feature-based design system described by Patel (1985), could prove effective if slow) and browsing along the lines suggested by Herot (1980). The latter could prove the best compromise for small to medium-sized collections, provided agreement could be reached with users on the most appropriate projection of each object to display. One might envisage this as the default interface mode, with the more cumbersome example- or menu-based interfaces available as options where specific type C or D queries were involved.

Interface problems of another kind are likely to be encountered by a true 3-D version of SAFARI. While many CAD systems can exchange 2-D drawings in standard IGES format, opportunities to exchange 3-D geometric model descriptions are much more limited, as the equivalent 3-D standard, STEP, is still in the process of development (Wilson, 1990). This has two implications. Firstly, it will be some time before STEP interfaces are a standard feature of geometric modelling systems. Obtaining significant volumes of data in standard format may thus prove difficult. Secondly, it is difficult to decide what level of translation capability to build into the next version of SAFARI until more details of the standard are known. At one level, there is the need to decide how many of the alternative ways of defining a given line or plane to support. At a deeper level, knowledge of the extent to which topology and shape feature definitions are enforced by the standard is necessary before deciding how much inferential capability is needed by the next version of SAFARI. (The situation with IGES, which made no attempt to enforce any such definitions, was fairly clear-cut). However, the general principle of basing input to SAFARI on a standard data exchange format rather than tying it to a specific modelling system is still considered valid. Note that this provides an additional reason for using boundary representation rather than CSG (see section 1.2) as the basis for any 3-D version of SAFARI. Systems based on boundary representation can accept input in either boundary representation or CSG form; CSG-based systems can in general accept input only in CSG form.

9.3.4 Conclusions

In the author's opinion, therefore, the majority of the principles adopted in the design of the Mark I version of SAFARI are applicable - with appropriate extensions - to the wider domain of engineering parts currently represented as orthographic projections or 3-D geometric models. To this must be added the caveat that, just as the initial prototype of SAFARI was limited to shapes made up of straight-line and circular-arc segments, the above discussion assumes that the 3-D version of SAFARI would be limited to objects with planar, cylindrical or spherical surfaces (which includes the vast majority of machined parts). The extent to which this approach remains valid for objects with sculptured surfaces has yet to be established.

9.4 Applicability to pictorial information systems in general

9.4.1 A taxonomy of related systems

As discussed in chapter 1, interest in what can be loosely be termed pictorial information systems has been developing for some years, particularly in the geographic field. What relevance, if any, does the present work have for the generality of such systems? To answer this question fully, it is first necessary to establish the similarities and differences between SAFARI and other types of system.

Firstly, SAFARI is a *shape* retrieval system, attempting to find the most similar objects to a given query purely on the basis of shape features. The system is believed to be unique in a number of respects; (a) subject area - with the possible exception of the ARES system (Ichikawa, 1980), for which few operational details are available, it is the only known system capable of retrieving engineering drawings by features extracted automatically from the drawings themselves (as opposed to manually-assigned index terms or classification codes), (b) boundary and feature representation - the principles of deriving a unique representation for each shape, viewing each shape boundary as a series of levels of increasing complexity, and associating extracted shape features with a specific level, do not seem to have been explicitly described elsewhere, and (c)

evaluation - it is the only pictorial information system for which any systematic evaluation of retrieval performance has been attempted.

While SAFARI can be considered to be a pictorial information system, an information retrieval system or an engineering database system, none of these titles are particularly helpful in conveying its true nature. Its closest analogues are clearly the few genuine shape retrieval systems in other areas - particularly the fingerprint matching systems now used by many police forces (IEEE, 1985). Some of the medical imaging systems which scan pictorial diagnostic aids such as X-rays (Toriwaki, 1980; Yokoya and Tamura, 1982; Frasson and Er-Radi, 1985) also fall into this category, though their image analysis capabilities as reported in the literature are rudimentary by comparison with SAFARI or fingerprint matching systems. Although not strictly *shape* retrieval systems, the image retrieval systems described by Rabitti and Stanchev (1987b, 1989) have in some ways a better claim to membership of this group, as they have implemented principles of feature extraction and classification closely related to those employed by SAFARI.

Further removed from SAFARI than this group are three very different types of system, each of which, though clearly distinct from SAFARI in overall philosophy, has influenced its design to some extent. These are geographical information systems and their analogues, feature recognition systems developed in connection with automated process planning, and image analysis systems developed for robot vision.

Geographical information systems appear at first sight to be the closest of these to SAFARI in overall objectives. The similarities between SAFARI and systems such as GRAIN (Chang et al, 1977), REDI (Chang and Fu, 1980), PICDMS (Chock et al, 1984) and PROBE (Orenstein and Manola, 1988) are obvious. Both types of system store pictorial data which can be retrieved and displayed in a number of different ways on request, and both types are based on an underlying DBMS (though using different data models). But there is one crucial difference. The geographical systems are essentially *spatial* retrieval systems, identifying objects or areas lying within identified map coordinates, or bearing specified spatial relationships to each other (adjacent to, within, northwest of, etc). The "intelligent image database system" of Chang et al (1988), though not strictly geographic, also falls into this category, as it is designed to answer questions about the relative positions of specific objects within an image. Thus, while such systems might share aspects of interface design with SAFARI, their query formulation capabilities and matching processes in the main are quite different. The only potential area of overlap is similarity retrieval, which would clearly require matching techniques analogous to those used by SAFARI. Several of the authors above describe query languages which include similarity retrieval commands - though none of them gives any indication of how to implement such commands. In the GRAIN system, for example, it is simply assumed that searchers will provide their own similarity matching procedures. Most later authors recognize that their systems need some kind of feature extraction capability to support this kind of retrieval, though the vague terms in which they discuss the problem make it quite clear that none of them have made any systematic attempt to solve it.

Feature recognition systems, aiming to recognize and extract shape features from CAD drawings or geometric models, have been the subject of increasing interest in recent years. The motivation for such studies has most often been to recognize machinable features in the design, as a prelude to automated process planning (deciding what sequence of drilling, turning or milling operations is necessary to convert an unformed piece of metal into a finished part). Examples of such work are Choi et al (1984), Henderson and Anderson (1984), Lee and Fu (1987), and Varady et al (1990). One can also perhaps class the work of Kyprianou (1980) and Kakazu and Okino (1984) under this heading. Although the motivation here was different (the generation of workpiece classification codes with group technology in mind), the techniques used were remarkably similar. In every case, syntactic analysis of object representations from a CAD system was used to uncover the presence of specific types of shape feature as a

prelude to further application-specific processing. While these systems differ clearly from SAFARI in that none of them offer any query or retrieval facilities, nor any systematic means of storing and uniquely identifying shape feature representations, the approach to shape analysis chosen for the prototype system was strongly influenced by some of the earlier systems of this type.

Computer vision systems which attempt to recognize the presence of specified objects within digitized images (such as those reviewed in Chin and Dyer, 1986) also resemble SAFARI in some important respects. The basic process of extracting features from objects detected within images, in order to match them with existing templates, has many parallels with the shape analysis and matching procedures underpinning the SAFARI system. As indicated in chapter 4, the pattern recognition literature has provided the source both of feature types (such as P^2/A ratios) and similarity estimation techniques (such as θ-s matching) adopted for the prototype system. However, such systems differ fundamentally from shape *retrieval* systems in the restricted number and type of objects they are designed to recognize. This can be most clearly shown by contrasting the way in which the two types of system handle their normal input - a digitized image containing objects to be recognized in the case of a robot vision system, a query shape for a system such as SAFARI. The vision system, though it needs to perform some highly complex analysis on the input picture, is faced with a relatively simple task when it comes to shape matching, since few systems reported in the literature hold more than about ten stored template shapes. The shape retrieval system, on the other hand, while needing to perform relatively little analysis of the query shape, has to match the query against hundreds, possibly thousands, of stored shapes. Hence the emphasis in a shape retrieval system needs to be on efficient, general-purpose shape-matching methods, coupled with appropriate data storage techniques - as opposed to a vision system where the main design effort has to be directed towards image segmentation and feature extraction from noisy images.

9.4.2 Relevance of the SAFARI project

The field of pictorial information system design is steadily growing in importance, judging by the volume and diversity of the literature (Lunin, 1987; Petrie, 1988; Chang, 1989). As well as empirical studies of the kind described in chapter 1, attempts are now being made to develop a body of underlying theory, such as the concept of the *generalized icon* (Chang, 1987) which specifies a formal mapping between logical and physical picture objects, and which might therefore be of value in picture indexing. How useful such concepts will prove in stimulating further research remains to be seen. One would perhaps have more faith in the general applicability of Chang's work had he not limited his supporting examples to Chinese ideograms.

There is another reason for questioning the validity of the generalized icon concept referred to above. The sheer diversity of picture types used to communicate ideas between human specialists in different fields raises fundamental problems about treating all types of picture in the same way. To expect humans to view maps, diagrams, press photographs, engineering drawings, X-rays, chemical structures, trade marks, and cartoons as a single homogeneous set of visual objects, which can all be interpreted in the same way, does not on the face of it seem reasonable. Engineering and architectural drawings are clearly recognizable as design documents specifying the size and shape of certain artefacts. Maps are occasionally used as specification documents, but most often simply record the presence and location of certain types of natural or man-made object. Diagnostic images such as X-ray photographs are used in a completely different way; only those parts of the image showing some kind of abnormality are normally of interest. Photographs in general are virtually impossible to interpret at more than the most trivial level without the aid of cues outside the picture, such as explanatory text. Studies on picture comprehension performed nearly 20 years ago by Firschein and Fischler (1971,

1972), contrasting the way in which different subjects attempted to interpret identical scenes, illustrate this point admirably.

Since the user is seldom, if ever, really concerned with the *physical* image except as a vehicle for conveying underlying ideas which are more easily represented graphically than textually, it is necessary to base image retrieval systems on *logical* picture objects and the specific features which can be derived from them. Such characteristic features are remarkably application-specific. Thus SAFARI specifically exploits the underlying regularity of machined components; fingerprint matching systems rely on identification of characteristic *minutiae*; geographical information systems link related types of data via spatial coordinates. The likelihood of a general picture retrieval system achieving successful results without being able to rely on application-specific cues of this kind is not great. An analogous situation can be observed in the artificial intelligence field (Winston, 1984), where real progress in expert systems development came about only when the search for a general-purpose problem solver was abandoned in favour of application-specific systems such as MYCIN and PROSPECTOR.

If this view is valid, its implications are clear. It is impossible to apply many of the application-specific techniques adopted for SAFARI - *or for any other such system* - to pictorial information systems in general. Some design principles are likely to be generally applicable, though many of these are already well-established, such as the need to extract characteristic features from each stored object and use these as retrieval keys. One might further expect the domain of drawings where the complete SAFARI approach remained applicable to be those which were created and interpreted in the same way as engineering drawings. This effectively includes only two further classes - architectural plans and drawings, and technical illustrations such as the drawings of new inventions included in patent specifications (normally single isometric or perspective views, which require considerable interpretation to reconstruct the original 3-D object).

Some aspects of the SAFARI approach might well be useful outside this area. While its relevance for geographic information systems is limited because spatial rather than shape retrieval is involved, aspects of its approach to feature extraction and storage would certainly seem to be valid for queries involving similarity matching, if only to the extent of recognizing that a relatively simple feature set might well provide adequate retrieval performance. Its relevance for process planning feature recognition systems is probably minimal, because the level of feature description required by these systems is much more detailed than the level that appears to be needed for simple shape retrieval. The most promising related area might well be image analysis - the principle of viewing each 2-D boundary identified as a series of levels, each with its own characteristic feature set, could well improve recognition performance in systems with relatively large numbers of reference shapes; and the principle of θ-s matching of canonical shape descriptions, though of limited usefulness for similarity retrieval, could be extremely valuable in searching for exact matches between image and reference objects, and significantly more efficient than published methods such as Perkins (1978).

In a way this limited applicability is disappointing, as it implies that progress in the pictorial information retrieval area will continue only on a piecemeal basis, with new systems inevitably tied to specific applications areas, with little to offer to related areas. On a more positive note, one can perhaps observe that this merely emphasizes the variety and richness of the field, and the possibilities it offers for further work, both to the practical systems designer and the theorist. There may well be further underlying design principles to discover, as in the expert systems field, where knowledge representation techniques such as the rule and the frame have gained widespread acceptance, despite the bewildering variety of knowledge they handle. Whether such advances stem from Chang's mathematical approach, Fischler and Firschein's psychological investigations, or simply the steady progression of empirical studies such as SAFARI, time alone will tell.

9.5 Epilogue

Twenty years of research into automated image processing have left most observers with the inescapable conclusion that it is not an inherently suitable application area for the digital computer. The human eye and brain still outperform even the best computer systems with contemptuous ease. In many ways, automated image processing systems are an apt target for Dr Johnson's famous comparison with a dog walking on its hind legs: "it is not done well; the wonder is that it is done at all". It is perhaps in this light that the SAFARI project should be judged. A successful attack has been made on a problem (that of extracting, storing and matching drawing features efficiently and reliably enough to incorporate into an operational database) which many writers have described, but few have attempted to solve. The demonstration that acceptable retrieval performance can be achieved with relatively unsophisticated feature extraction and matching techniques will, one hopes, encourage similar developments in other areas of pictorial information retrieval.

REFERENCES

Adamson, G W et al (1973) "Strategic considerations in the design of a screening system for substructure searches of chemical structure files" *Journal of Chemical Documentation* **13**(3), 153-157

Akin, O (1978) "How do architects design?", pp 65-98 in *Artificial intelligence and pattern recognition in computer-aided design*, (ed Latombe, J), North-Holland, Amsterdam

Asada, H and Brady, M (1986) "The curvature primal sketch" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(1), 2-14

Assmann, K et al (1984) "The ISQL language - a uniform tool for managing images and non-image data in an image data base management system" pp 42-45 in *Proceedings of IEEE Conference on Medical Images and Icons*, Arlington, Virginia, July 1984

Ballard, D H (1981) "Strip trees: a hierarchical representation for curves" *Communications of the ACM* **24**(5), 310-321

Ben-Bassat, M and Zaidenberg, L (1984) "Contextual template matching: a distance measure for patterns with hierarchically dependent features" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**(2), 201-211

Bhanu, B and Faugeras, O D (1984) "Shape matching of two-dimensional objects" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**(2), 137-155

Braid, I C (1973) "Designing with volumes" Ph.D. Thesis, University of Cambridge

Burgess, C G (1986) "A graphical database interface" *Computers and Industrial Engineering* **11**, 355-359

Burton, W (1977) "Representation of many-sided polygons and polygonal lines for rapid processing" *Communications of the ACM* **20**(3), 166-171

Castelli, E (1989) "Symmetry-based approach to mechanical drawings retrieval, an AI application" pp 405-413 in *Visual Database Systems* (ed Kunii, T L) Elsevier, Amsterdam

Chang, N S and Fu, K S (1980) "A relational database system for images" pp 288-321 in *Pictorial Information Systems*, (ed Chang, S K and Fu, K S), Springer-Verlag, Berlin

Chang, N S and Fu, K S (1981) "Picture query languages for pictorial database systems" *IEEE Computer* **14**(11), 23-33

Chang, S K et al (1977) "A relational database system for pictures" pp 142-149 in *Proceedings of the IEEE Workshop on Picture Data Description and Management*, Chicago, April 1977

Chang, S K et al (1980) "A generalized zooming technique for pictorial database systems" pp 257-287 in *Pictorial Information Systems*, (ed Chang, S K and Fu, K S), Springer-Verlag, Berlin

Chang, S K and Liu, S H (1984) "Picture indexing and abstraction techniques for pictorial databases" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**(4), 475-484

Chang, S K (1987) "Iconic semantics - towards a formal theory of icons" *International Journal of Pattern Recognition and Artificial Intelligence* **1**(1), 103-120

Chang, S K at al (1988) "An intelligent image database system" *IEEE Transactions on Software Engineering* **14**(5), 681-688

Chang, S K (1989) "Principles of Pictorial Information Systems Design" Prentice-Hall, Englewood Cliffs, NJ

Chen, C H (1973) "Statistical Pattern Recognition" Hayden, Washington

Chen, P S (1976) "The entity-relationship model - towards a unified view of data" *ACM Transactions on Database Systems* **1**(1), 9-36

Chin, R T and Dyer C R (1986) "Model-based recognition in robot vision" *ACM Computing Surveys* **18**(1), 67-108

Chock, M et al (1984) "Database structure and manipulation capabilities of a picture database management system (PICDMS)" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**(4), 484-492

Choi, B K et al (1984) "Automatic recognition of machined surfaces from a 3-D solid model" *Computer Aided Design* **16**(2), 81-86

Cleverdon, C W et al (1966) "Factors determining the performance of indexing systems" College of Aeronautics, Cranfield

CODASYL (1971) "Report of Data Base Task Group of CODASYL Programming Language Committee" ACM, New York

Codd, E F (1970) "A relational model of data for large shared data banks" *Communications of the ACM* **13**(6), 377-387

Codd, E F (1972) "Further normalization of the data base relational model" in *Data Base Systems*, (ed Rustin, R) Prentice-Hall, Englewood Cliffs, New Jersey

Codd, E F (1979) "Extending the database relational model to capture more meaning" *ACM Transactions on Database Systems* **4**(4), 397-434

Dadam, P at al (1986) "A DBMS prototype to support extended NF^2 relations: an integrated view on flat tables and hierarchies" *SIGMOD Record* **15**(2), 356-367

Dietrich, F (1985) "Visual intelligence: the first decade of computer art" *IEEE Computer Graphics and Applications* **5**(7), 32-41

Dittrich, K and Dayal, U, eds. (1986) "Proceedings of International Workshop on Object-oriented Database Systems" IEEE Computer Society, Pacific Grove, CA

Dong, X and Wozny, M (1988) "FRAFES, a frame-based feature extraction system" pp 296-305 in *Proceedings of International Conference on CIM* IEEE Computer Society, Pacific Grove, CA

Dubois, S R and Glanz, F H (1986) "An autoregressive model approach to two-dimensional shape classification" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(1), 55-66

Duda, R O and Hart, P E (1973) "Pattern Classification and Scene Analysis" Wiley, New York

Dunham, J G (1986) "Optimum uniform piecewise linear approximation of planar curves" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(1), 67-75

Eberlein, W and Wedekind, H (1982) "Design databases in integrated engineering systems", pp 3-37 in *File structures and data bases for CAD* (ed Encarnacao, J and Krause, F L), North-Holland, Amsterdam

Everitt, B (1980) "Cluster Analysis", 2nd edition. Heinemann, London

Firschein, O and Fischler, M A (1971) "Describing and abstracting pictorial structures" *Pattern Recognition* 3, 421-443

Firschein, O and Fischler, M A (1972) "A study in descriptive representation of pictorial data" *Pattern Recognition* 4, 361- 377

Fischler, M A and Bolles, R C (1986) "Perceptual organization and curve partitioning" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(1), 100-105

Frasson, C and Er-Radi, M (1986) "Principles of an icons-based command language" *SIGMOD Record* **15**(2), 144-152

Freeman, H (1974) "Computer processing of line-drawing images" *ACM Computing Surveys* 6(1), 57-97

Frei, H P and Jauslin, J F (1983) "Graphical presentation of information and services: a user-oriented interface" *Information Technology: Research and Development* 2, 23-42

Fu, K S (1982) "Syntactic Pattern Recognition and Applications" Prentice-Hall, Englewood Cliffs, New Jersey

Gardan, Y and Lucas, M (1983) "Interactive graphics in CAD" Kogan Page, London

Gaschnig, J (1983) "Evaluation of expert systems: issues and case studies" pp 241-280 in *Building Expert Systems* (ed Hayes-Roth, F et al). Addison-Wesley, Reading, Mass.

Gips, J (1974) "Shape grammars and their uses" Stanford University Computer Science Department report STAN-CS-74-413

Goldberg, A and Robson, D (1983) "Smalltalk-80: the language and its implementation" Addison-Wesley, Reading, Mass

Greanias, E C et al (1963) "The recognition of handwritten numerals by contour analysis" *IBM Journal* 7, 14-21

Hayes-Roth, F et al (1983) "Building Expert Systems" Addison-Wesley, Reading, Mass.

Henderson, M R and Anderson, D C (1984) "Computer recognition and extraction of form features: a CAD/CAM link" *Computers in Industry* 5, 329-339

Herot, C F (1980) "Spatial management of data" *ACM Transactions on Database Systems* 5(4), 493-514

Hough, P V C (1962) "Methods and means for recognizing complex patterns" U.S. Patent 3069654

Howard, H C and Rehak, D R (1986) "Expert systems and CAD databases" pp 236-248 in *Proceedings of CAD-86, the 7th International Conference on the Computer as a Design Tool* Butterworths, London

Ichikawa, T et al (1980) "A query manipulation system for image data retrieval by ARES" pp 61-67 in *Proceedings of IEEE Workshop on Picture Data Description and Management*, August 1980

IEEE (1985) "Computer graphics in the detective business" *IEEE Computer Graphics and Applications* 5(4), 14-17

Ives, B (1982) "Graphical user interfaces for business information systems" *MIS Quarterly*, special issue, Dec 1982, 15-47

Johnson, R H and Dewhirst, D L (1982) "The product structured data base: a schema for design of mechanical systems", pp 155-163 in *File structures and data bases for CAD* (ed Encarnacao, J and Krause, F L), North-Holland, Amsterdam

Kakazu, Y and Okino, N (1984) "Pattern recognition approaches to GT code generation on CSG" pp 10-18 in *Proceedings of 16th CIRP International Seminar on Manufacturing Systems*, Tokyo

Kalay, Y E (1983) "A relational database for non-manipulative representation of solid objects" *Computer-Aided Design* **15**(5), 271-276

Kato, O et al (1982) "Interactive hand-drawn input system" pp 544-549 in *Proceedings of IEEE Computer Society Conference on Pattern Recognition and Image Processing (PRIP 82)*, Las Vegas, June 1982

Kemper, A and Wallrath M (1987a) "An analysis of geometric modelling in database systems" *ACM Computing Surveys* **19**(1), 47-91

Kemper, A and Wallrath M (1987b) "An object-oriented database system for engineering applications" *SIGMOD Record* **16**(3), 299- 310

Ketabchi, M A and Berzins, V (1987) "Modeling and Managing CAD Databases" *IEEE Computer* **18**(2), 93-102

Kim, H J et al (1988) "PICASSO: a graphical query language" *Software Practice and Experience* **18**(3), 169-203

Kimura, F et al (1982) "Construction and uses of an engineering data base in design and manufacturing environments", pp 95-111 in *File structures and data bases for CAD* (ed Encarnacao, J and Krause, F L), North-Holland, Amsterdam

Klinger, A and Dyer, C R (1976) "Experiments in picture representation using regular decomposition" *Computer Graphics and Image Processing* **5**, 68-105

Koriba, M (1983) "Database systems: their applications to CAD software design" *Computer-Aided Design* **15**(5), 277-287

Kruger, R P et al (1972) "Automated radiographic diagnosis via feature extraction and classification of cardiac size and shape descriptors" *IEEE Transactions on Biomedical Engineering* **BME-19**(3), 174-186

Kunii, T L et al (1975) "An interactive fashion design system INFADS" *Computers and Graphics* **1**, 297-302

Kyprianou, L K (1980) "Shape classification in CAD" Ph.D. Thesis, University of Cambridge

Lafue, G (1978) "A theorem prover for recognizing 2-D representations of 3-D objects", pp 391-401 in *Artificial intelligence and pattern recognition in computer-aided design*, ed Latombe, J. North-Holland, Amsterdam

Lee, E T (1980) "Similarity retrieval techniques", pp 128-176 in *Pictorial Information Systems*, (ed Chang, S K and Fu, K S), Springer-Verlag, Berlin

Lee, H C and Fu, K S (1972) "A stochastic syntax analysis procedure and its application to pattern classification" *IEEE Transactions on Computing* **21**, 660-666

Lee, Y C and Fu, K S (1987) "Machine understanding of CSG: extraction and unification of manufacturing features" *IEEE Computer Graphics and Applications* 7(1), 20-32

Lee, Y C and Jea, K F J (1987) "PAR: a CSG-based unique representation scheme for rotational parts" *IEEE Transactions on Systems, Man and Cybernetics* **SMC-17**(6), 1039-1049

Leong, M K et al (1989) "Towards a visual language for an object-oriented multi-media database system" pp 465-495 in *Visual Database Systems* (ed Kunii, T L) Elsevier, Amsterdam

Leou, J J and Tsai, W H (1987) "Automatic rotational symmetry determination for shape analysis" *Pattern Recognition* **20**(6), 571-582

Liardet, M et al (1978) "Input to CAD systems: two practical examples", pp 403-414 in *Artificial intelligence and pattern recognition in computer-aided design*, ed Latombe, J. North-Holland, Amsterdam

Liewald, M H and Kennicott, P R (1982) "Intersystem data transfer via IGES" *IEEE Computer Graphics and Applications* 2(5), 55-63

Lin, W C and Fu, K S (1984) "A syntactic approach to 3-D object representation" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**, 350-364

Lunin, L F (1987) "Electronic Image Information" *Annual Review of Information Science and Technology* **22**, 179-224

Lynch, M F (1977) "Variety generation - a reinterpretation of Shannon's mathematical theory of communication and its implications for information science" *Journal of the American Society for Information Science* **28**(1), 19-25

Max, N L (1983) "Computer representation of molecular surfaces" *IEEE Computer Graphics and Applications* 3(5), 21-29

Mason, H (1985) "Searching for standards", *Engineering*, March 1985, 154-156

Moayer, B and Fu, K S (1976) "A tree system approach for fingerprint pattern recognition" *IEEE Transactions on Computing* **25**, 262-274

Mokhtarian, F and Mackworth, A (1986) "Scale-based description and recognition of planar curves and two-dimensional shapes" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-8**(1), 34-43

Mui J K et al (1977) "Automated classification of blood cell neutrophils" *Journal of Histochemistry and Cytochemistry* **25**(7), 633-640

Myers, G J (1979) "The art of software testing" Wiley, New York

Nagendra, I V and Gujar, U G (1988) "3-D objects from 2-D orthographic views - a survey" *Computers and Graphics* 12(1), 111-114

Nagy, G (1985) "Image database" *Image and Vision Computing* 3(3), 111-117

Nagy, G and Wagle, S (1979) "Geographic data processing" *ACM Computing Surveys* 11(2),139-181

National Bureau of Standards (1983) "Initial Graphics Exchange Specification, Version 2.0" Report NBSIR-82-2631-AF, National Engineering Laboratory, National Bureau of Standards, Washington, DC

Opitz, H et al (1969) "Workpiece classification and its industrial application" *International Journal of Machine Tool Design Research* 9, 39-50

Patel, R (1985) "A mechanical engineering design interface for geometric modelling" Ph.D. Thesis, Council for National Academic Awards

Orenstein, J A and Manola, F A (1988) "PROBE spatial data modelling and query processing in an image database application" *IEEE Transactions on Software Engineering* 14(5), 611-629

Pavlidis, T (1980) "Algorithms for shape analysis of contours and waveforms" *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-2(4), 301-312

Perkins, W A (1978) "A model-based vision system for industrial parts" *IEEE Transactions on Computers* 27(2), 126-143

Petrie, J H (1988) "An overview of image processing and image management systems and their application" British Library Research Paper 40, British Library, London

Pratt, M J (1984) "Solid modelling and the interface between design and manufacture" *IEEE Computer Graphics and Applications* 4(7), 52-59

Rabitti, F and Stanchev, P (1987a) "An Approach to Image Retrieval from Large Image Databases" pp 284-295 in *Proceedings of 10th Annual ACM-SIGIR Conference*, New Orleans, June 1987

Rabitti, F and Stanchev, P (1987b) "Graphical Image Retrieval from Large Image Databases" vol 2, pp 69-89 of *Proceedings of AICA conference*, Trento

Rabitti, F and Stanchev, P (1989) "GRIM_DBMS: a GRaphical IMage DataBase Management System" pp 415-430 in *Visual Database Systems* (ed Kunii, T L) Elsevier, Amsterdam

Requicha, A A G (1980) "Representation for rigid solids: theory, methods, and systems" *ACM Computing Surveys* 12(4), 437-464

Requicha, A A G (1988) "Solid Modelling - a 1988 update" pp 3-22 in *CAD Based Programming for Sensory Robots* Springer-Verlag, Berlin

Requicha, A A G and Chan, S C (1986) "Representation of geometric features, tolerances, and attributes in solid modellers based on constructive geometry" *IEEE Journal of Robotics and Automation* RA-2(3), 156-166

van Rijsbergen, C J (1979) "Information Retrieval", 2nd edition. Butterworths, London

Rogers, G (1980) "Computer Graphics and Architecture" in *Eurographics 80*, ed C E Vandoni. North-Holland, Amsterdam

Rosenthal, A et al (1984) "An example of knowledge-based query processing in a CAD/CAM DBMS" pp 363-370 in *Proceedings of 10th International Conference on Very Large Databases*, Singapore, 1984

Salton, G (1971) "The SMART retrieval system - experiments in automatic document processing" Prentice-Hall, Englewood Cliffs, New Jersey

Shaw, A C (1970) "Parsing of graph-representable pictures" *Journal of the Association for Computing Machinery* **17**(3), 453- 481

Sokal, R R and Sneath, P H A (1963) "Principles of numerical taxonomy" Freeman, San Fransisco

Sparck Jones, K, ed. (1981) "Information retrieval experiment" Butterworths, London

Spooner, D L et al (1985) "Abstract data types for CAD systems" pp 359-364 in *Proceedings of International Conference on Robotics and Automation* IEEE Computer Society, Pacific Grove, CA

Staley, S M et al (1983) "Using syntactic pattern recognition to extract feature information from a solid geometric database" *Computers in Mechanical Engineering* 2(2), 61-66

Staley, S M and Anderson, D C (1986) "Functional specification for CAD databases" *Computer-Aided Design* **18**(3), 132-138

Stamper, R (1973) "Information in business and administrative systems" Batsford, London

Stockman, G et al (1982) "Matching images to models for registration and object detection via clustering" *IEEE Transactions on Pattern Recognition and Machine Intelligence* **PAMI-4**(3), 229- 241

Sutherland, I E (1965) "SKETCHPAD: a man-machine graphical communication system" MIT Lincoln Lab Technical Report 296

Swets, J A (1963) "Information retrieval systems" *Science* **141**, 245-250

Tamura, H (1980) "Image database management for pattern information processing studies", pp 198-227 in *Pictorial Information Systems*, (ed Chang, S K and Fu, K S), Springer-Verlag, Berlin

Tamura, H and Yokoya, N (1984) "Image database systems: a survey" *Pattern Recognition* **17**(1), 29-43

Tiller, W (1983) "Rational B-splines for curve and surface representation" *IEEE Computer Graphics and Applications* 3(9), 61-69

Toriwaki, J et al (1980) "Pictorial information retrieval of chest X-ray image database using pattern recognition techniques", pp 1116-1119 in *Proceedings of the Third World Conference on Medical Informatics*, Tokyo, October 1980 (ed Lindberg, D A B and Kaihara, S), North-Holland, Amsterdam

Turner, J U and Wozny, M J (1987) "Tolerances in computer-aided geometric design" *Visual Computer* 3, 214-226

Turney, J L et al (1985) "Recognizing partially occluded parts" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-7**(4), 410-421

Ulfsby, S et al (1982) "TORNADO: a DBMS for CAD/CAM systems", pp 335-346 in *File structures and data bases for CAD* (ed Encarnacao, J and Krause, F L), North-Holland, Amsterdam

Umetani Y and Taguchi K (1982) "Discrimination of general shapes by psychological feature properties" *Digital Systems for Industrial Automation* **1**(2-3), 179-198

Varady, T et al (1990) "Identifying features in solid modelling" *Computers in Industry* **14**, 43-50

Voelker, H B and Requicha, A A G (1977) "Geometric modelling of mechanical parts and processes" *IEEE Computer* **10**(12), 48-57

Voelker, H B et al (1978) "The PADL-1.0/2 system for defining and displaying solid objects" *Computer Graphics* **12**(3), 257-263

Voelker, H B et al (1990) "Computer applications in manufacturing" *Annual Review of Computer Science* 3, 349-387

Weiler, K (1985) "Edge-based data structures for solid modelling in curved-surface environments" *IEEE Computer Graphics and Applications* 5(1), 21-40

Wilson, P R (1987) "A short history of CAD data transfer standards" *IEEE Computer Graphics and Applications* 7(6), 64-67

Wilson, P R (1989) "PDES STEPs forward" *IEEE Computer Graphics and Applications* 9(2), 79-80

Wilson, P R (1990) "STEP ballot results" *IEEE Computer Graphics and Applications* 10(5), 79-82

Winston, P H (1984) "Artificial Intelligence", 2nd edition. Addison-Wesley, Reading, Mass.

Woo, T C (1982) "Feature extraction by volume decomposition", pp 76-94 in *Proceedings of Conference on CAD/CAM Technology in Mechanical Engineering*, Cambridge, Mass

Wu, C T (1987) "GLAD: graphics language for database" pp 164- 170 in *Proceedings of 11th International Conference on Computer Software and Applications, Tokyo*. IEEE Computer Society, New York

Yachida, M and Tsuji, S (1977) "A versatile machine vision system for complex industrial parts" *IEEE Transactions on Computers* C-26(9), 882-894

Yamaguchi, K et al (1980) "ELF: extended relational model for large, flexible picture structures" pp 95-100 in *Proceedings of the IEEE Workshop on Picture Data Description and Management*, August 1980

Yokoya, N and Tamura, H (1982) "A database system of microscopic cell images", pp 471-476 in *Proceedings of ISM III '82, the first IEEE Computer Society International Symposium on Medical Imaging*, Berlin

Yoshiura, H et al (1984) "Top-down construction of 3-D mechanical object shapes from engineering drawings" *IEEE Computer* **18**(12), 32-40

You, K C and Fu, K S (1979) "A syntactic approach to shape recognition using attributed grammars" *IEEE Transactions on Systems, Man and Cybernetics* **SMC-9**(6), 334-345

Zahn, C T and Roskies, R Z (1972) "Fourier descriptors for plane closed curves" *IEEE Transactions on Computers* **C-21**(3), 269-281

Zakia, R D (1975) "Perception and photography" Prentice-Hall, Englewood Cliffs, NJ

Zloof, M M (1975) "Query-by-Example" pp 431-438 in *Proceedings of the 1975 Spring National Computer Conference*. AFIPS Press, Arlington, Virginia

Zobrist, A L and Bryant, N A (1980) "Designing an image based information system", pp 177-198 in *Pictorial Information Systems*, (ed Chang, S K and Fu, K S), Springer-Verlag, Berlin

APPENDIX A - RESULTS OF STUDENT SIMILARITY RANKING EXPERIMENTS

Table A1. Similarity ratings for OUTER-BOUNDARY queries

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 32 (16 subjects) -							
33	0	13	0	0	1	0	0.581
82	0	0	9	1	0	0	0.300
87	0	0	1	2	2	0	0.094
23	0	0	2	0	0	0	0.063
145	0	0	0	1	2	1	0.050
138	0	1	0	0	0	0	0.044
161	0	0	1	0	0	0	0.031
102	0	0	0	1	0	1	0.025
66	0	0	0	1	0	0	0.019
29	0	0	0	1	0	0	0.019
24	0	0	0	0	1	0	0.013
128	0	0	0	0	1	0	0.013
118	0	0	0	0	0	1	0.006
39	0	0	0	0	0	1	0.006
65	0	0	0	0	0	1	0.006
Query no 44 (16 subjects) -							
42	5	10	0	1	0	0	0.769
41	1	3	7	2	2	0	0.475
43	1	1	4	5	3	2	0.375
180	0	0	2	7	4	1	0.250
104	0	0	2	1	3	2	0.131
40	0	2	0	0	0	3	0.106
38	0	0	1	0	0	0	0.031
78	0	0	0	1	0	0	0.019
31	0	0	0	0	0	1	0.006
Query no 46 (16 subjects) -							
52	6	8	0	1	0	0	0.744
146	5	4	5	1	0	0	0.663
51	1	1	5	2	1	1	0.319
18	0	0	2	4	3	1	0.181
13	0	0	2	1	4	0	0.131
28	0	1	0	0	2	0	0.069
10	0	1	0	0	0	1	0.050
16	0	1	0	0	0	0	0.044
14	0	0	0	2	0	1	0.044
148	0	0	0	0	0	1	0.006

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 47 (16 subjects) -							
59	14	1	0	0	0	1	0.925
53	2	11	1	1	1	0	0.669
142	3	1	9	1	0	0	0.531
68	0	1	3	4	2	1	0.244
12	0	0	2	3	5	0	0.181
174	0	1	0	3	2	1	0.131
134	0	0	1	1	0	0	0.050
15	0	1	0	0	0	0	0.044
18	0	0	0	1	0	0	0.019
Query no 57 (16 subjects) -							
94	1	13	2	0	0	0	0.694
3	1	1	7	3	0	1	0.388
99	0	0	2	4	1	1	0.156
4	0	0	1	1	1	1	0.069
55	0	0	0	2	0	1	0.044
50	0	0	1	0	1	0	0.044
5	0	0	0	1	2	0	0.044
1	0	1	0	0	0	0	0.044
9	0	1	0	0	0	0	0.044
155	0	0	0	0	1	3	0.031
6	0	0	0	0	1	0	0.013
8	0	0	0	0	1	0	0.013
7	0	0	0	0	0	1	0.006
Query no 62 (16 subjects) -							
143	9	6	1	0	0	0	0.856
158	4	8	3	1	0	0	0.713
160	0	1	5	1	1	1	0.238
81	0	0	2	5	3	0	0.194
152	0	0	2	2	6	0	0.175
156	0	0	1	3	1	0	0.100
97	0	1	0	1	0	1	0.069
56	0	1	0	0	0	0	0.044
77	0	0	0	1	1	1	0.038
63	0	0	0	0	1	0	0.013
58	0	0	0	0	1	0	0.013
162	0	0	0	0	0	1	0.006
16	0	0	0	0	0	1	0.006
Query no 67 (16 subjects) -							
71	12	4	0	0	0	0	0.925
144	9	5	1	0	0	0	0.813
106	0	2	0	0	0	0	0.088
113	0	0	1	0	0	0	0.031
107	0	0	1	0	0	0	0.031
83	0	0	0	1	0	0	0.019
111	0	0	0	1	0	0	0.019
153	0	0	0	0	1	0	0.013

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 72 (16 subjects) -							
65	1	6	5	0	2	0	0.506
69	0	4	6	2	1	0	0.413
70	0	4	2	5	4	0	0.381
139	0	2	1	5	5	0	0.275
66	0	1	0	0	0	1	0.050
54	0	0	0	0	0	1	0.006
Query no 75 (16 subjects) -							
83	6	10	0	0	0	0	0.813
140	0	4	7	3	0	0	0.450
74	0	1	3	2	3	1	0.219
76	0	1	1	5	3	0	0.206
106	0	0	2	0	0	1	0.069
87	0	0	1	1	0	0	0.050
164	0	0	0	0	3	1	0.044
127	0	0	0	1	0	2	0.031
145	0	0	0	0	2	0	0.025
111	0	0	0	0	1	0	0.013
35	0	0	0	0	0	1	0.006
54	0	0	0	0	0	1	0.006
Query no 103 (16 subjects) -							
105	0	7	3	0	0	1	0.406
100	0	3	6	0	0	0	0.319
86	0	0	1	3	2	0	0.113
123	0	0	1	1	3	1	0.094
132	0	0	1	2	1	0	0.081
40	0	0	1	1	1	1	0.069
11	0	1	0	0	1	2	0.069
88	0	1	0	1	0	0	0.063
177	0	1	0	0	0	1	0.050
85	0	1	0	0	0	0	0.044
135	0	1	0	0	0	0	0.044
109	0	0	0	2	0	1	0.044
1	0	0	1	0	0	0	0.031
136	0	0	0	1	0	1	0.025
78	0	0	0	1	0	0	0.019
119	0	0	0	1	0	0	0.019
Query no 115 (16 subjects) -							
90	1	11	2	0	0	0	0.606
119	0	5	7	1	1	0	0.469
114	0	0	2	3	1	3	0.150
165	0	0	1	2	1	0	0.081
89	0	0	2	0	1	0	0.075
108	0	0	0	2	1	0	0.050
84	0	0	0	1	1	0	0.031
91	0	0	0	1	0	0	0.019
121	0	0	0	0	0	1	0.006

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 120 (16 subjects) -							
129	0	10	3	1	0	1	0.556
118	0	4	8	1	0	0	0.444
124	0	2	3	1	1	0	0.213
128	0	0	1	4	2	0	0.131
126	0	0	0	4	1	1	0.094
66	0	0	0	0	2	0	0.025
6	0	0	0	0	1	1	0.019
5	0	0	0	0	0	1	0.006
Query no 170 (16 subjects) -							
167	1	13	1	1	0	0	0.681
169	0	0	13	2	1	0	0.456
166	0	3	1	12	0	0	0.388
168	0	0	1	1	5	7	0.156
172	0	0	0	0	9	6	0.150
Query no 175 (16 subjects) -							
27	15	0	0	1	0	0	0.956
24	0	11	5	0	0	0	0.638
37	0	4	8	2	1	0	0.475
102	0	0	2	7	0	1	0.200
82	0	1	1	1	2	1	0.125
87	0	0	0	1	1	2	0.044
31	0	0	0	0	1	1	0.019
25	0	0	0	1	0	0	0.019
161	0	0	0	0	1	0	0.013
145	0	0	0	0	1	0	0.013
Query no 176 (16 subjects) -							
179	0	10	6	0	0	0	0.625
21	0	6	10	0	0	0	0.575
177	0	0	0	5	2	0	0.119
88	0	0	0	1	2	0	0.044
56	0	0	0	2	0	0	0.038
181	0	0	0	0	1	2	0.025
95	0	0	0	0	1	1	0.019
26	0	0	0	1	0	0	0.019
Query no 183 (16 subjects) -							
182	0	10	2	2	1	0	0.550
184	0	2	10	2	0	0	0.438
107	0	4	3	2	3	0	0.344
185	0	0	0	10	4	0	0.238
111	0	0	0	0	4	4	0.075
106	0	0	0	1	0	1	0.025
164	0	0	0	0	0	2	0.013

Table A2. Similarity ratings for ALL-BOUNDARY queries

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 9 (12 subjects) -							
8	0	6	6	0	0	0	0.600
6	0	6	5	1	0	0	0.583
7	0	0	1	10	0	0	0.292
18	0	0	0	0	3	0	0.050
92	0	0	0	0	2	0	0.033
94	0	0	0	0	0	1	0.008
131	0	0	0	0	0	1	0.008
13	0	0	0	0	0	1	0.008
155	0	0	0	0	0	1	0.008
Query no 47 (12 subjects) -							
59	8	3	1	0	0	0	0.883
53	0	5	6	1	0	0	0.567
142	0	5	3	3	0	0	0.492
68	0	0	1	2	4	3	0.183
12	0	0	0	4	3	0	0.150
174	0	0	0	1	1	5	0.083
15	0	0	0	0	1	0	0.017
Query no 48 (12 subjects) -							
54	0	7	4	1	0	0	0.600
147	0	6	5	0	1	0	0.575
164	0	0	0	2	0	2	0.067
140	0	0	0	0	2	0	0.033
38	0	0	0	1	0	0	0.025
35	0	0	0	1	0	0	0.025
185	0	0	0	1	0	0	0.025
79	0	0	0	1	0	0	0.025
76	0	0	0	1	0	0	0.025
161	0	0	0	0	1	0	0.017
78	0	0	0	0	1	0	0.017
106	0	0	0	0	1	0	0.017
34	0	0	0	0	1	0	0.017
30	0	0	0	0	0	1	0.008
Query no 49 (12 subjects) -							
61	0	6	2	2	0	0	0.483
56	0	2	6	1	0	0	0.392
88	0	3	2	3	2	0	0.367
110	0	1	0	1	3	0	0.133
85	0	0	0	2	1	2	0.083
58	0	0	1	0	0	0	0.042
50	0	0	0	1	0	0	0.025
141	0	0	0	0	1	0	0.017
91	0	0	0	0	0	2	0.017
180	0	0	0	0	1	0	0.017

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 57 (13 subjects) -							
45	0	7	2	1	0	0	0.477
94	0	3	7	1	0	0	0.454
99	0	4	0	2	0	0	0.262
155	0	0	1	1	2	0	0.092
3	0	0	1	2	0	0	0.085
63	0	0	0	1	1	0	0.038
2	0	0	0	0	1	1	0.023
180	0	0	0	1	0	0	0.023
131	0	0	0	0	0	1	0.008
56	0	0	0	0	0	1	0.008
Query no 72 (13 subjects) -							
65	0	9	1	1	1	0	0.562
69	0	3	4	5	1	0	0.446
139	0	1	3	6	2	0	0.338
70	0	1	3	1	7	1	0.308
137	0	0	1	0	0	0	0.038
66	0	0	0	0	0	4	0.031
Query no 80 (13 subjects) -							
140	10	1	0	0	0	0	0.823
76	0	5	5	1	0	1	0.492
101	0	5	5	0	1	0	0.477
75	0	1	1	5	0	0	0.208
83	0	0	1	4	4	0	0.192
79	0	0	0	1	3	0	0.069
145	0	1	0	0	0	0	0.054
74	0	0	0	0	1	1	0.023
96	0	0	0	0	1	0	0.015
149	0	0	0	0	0	1	0.008
Query no 89 (13 subjects) -							
108	11	1	0	0	0	0	0.900
114	0	12	1	0	0	0	0.685
115	0	0	4	5	1	0	0.285
119	0	0	3	1	4	0	0.200
90	0	0	2	3	3	1	0.200
150	0	0	0	0	0	1	0.008
Query no 100 (13 subjects) -							
105	0	10	1	1	1	0	0.615
181	0	4	2	3	1	0	0.377
178	0	0	8	2	0	0	0.354
103	0	0	0	1	1	0	0.038
136	0	0	1	0	0	0	0.038
102	0	0	0	1	0	0	0.023
123	0	0	0	0	0	1	0.008

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 109 (13 subjects) -							
22	0	8	1	1	0	0	0.492
23	0	4	3	0	0	0	0.331
64	0	1	1	1	2	0	0.146
97	0	0	1	2	0	0	0.085
98	0	0	1	1	0	0	0.062
186	0	0	1	0	0	0	0.038
140	0	0	1	0	0	0	0.038
181	0	0	0	1	0	1	0.031
26	0	0	0	1	0	1	0.031
25	0	0	0	0	2	0	0.031
178	0	0	0	1	0	1	0.031
103	0	0	0	1	0	0	0.023
101	0	0	0	1	0	0	0.023
105	0	0	0	0	1	0	0.015
Query no 120 (13 subjects) -							
118	0	9	2	0	0	0	0.562
129	0	4	6	3	0	0	0.515
124	0	2	0	4	4	0	0.262
128	0	1	2	3	4	0	0.262
125	0	1	0	0	0	0	0.054
155	0	0	1	0	0	2	0.054
152	0	0	0	1	0	0	0.023
2	0	0	0	0	0	1	0.008
Query no 122 (13 subjects) -							
117	0	12	1	0	0	0	0.685
116	0	1	10	1	0	0	0.462
121	0	0	2	9	0	0	0.285
81	0	0	0	0	1	1	0.023
14	0	0	0	0	1	1	0.023
165	0	0	0	0	1	0	0.015
188	0	0	0	0	1	0	0.015
77	0	0	0	0	0	1	0.008
153	0	0	0	0	0	1	0.008
Query no 154 (12 subjects) -							
149	3	7	1	0	0	0	0.700
171	2	4	6	0	0	0	0.650
79	0	1	2	3	1	0	0.233
83	0	0	1	4	0	0	0.142
74	0	1	0	0	2	3	0.117
101	0	1	0	0	0	0	0.058
76	0	0	0	0	2	0	0.033
140	0	0	0	0	1	1	0.025
96	0	0	0	0	0	1	0.008

Drawing No	Exact matches	Frequency at ranking position 1	2	3	4	5	Overall rating
Query no 159 (12 subjects) -							
163	3	9	0	0	0	0	0.775
158	0	2	2	1	3	2	0.292
77	0	2	4	0	0	0	0.283
162	0	0	4	0	1	1	0.192
160	0	1	0	4	0	0	0.158
143	0	0	2	0	2	1	0.125
62	0	0	0	4	0	0	0.100
81	0	1	0	0	0	0	0.058
157	0	0	0	0	0	3	0.025
152	0	0	0	0	1	0	0.017
Query no 175 (12 subjects) -							
82	0	4	5	2	0	0	0.492
27	0	5	1	1	1	1	0.383
87	0	1	3	3	3	0	0.308
37	0	2	1	1	0	5	0.225
145	0	0	0	3	4	2	0.158
31	0	0	1	0	0	0	0.042
83	0	0	0	1	0	0	0.025
24	0	0	0	0	1	0	0.017
Query no 176 (12 subjects) -							
179	0	12	0	0	0	0	0.700
21	0	0	6	1	0	0	0.275
177	0	0	3	2	1	0	0.192
88	0	0	0	3	1	3	0.117
61	0	0	0	1	1	0	0.042
85	0	0	0	0	2	1	0.042
110	0	0	0	0	2	0	0.033
56	0	0	0	1	0	0	0.025
180	0	0	0	0	0	1	0.008

APPENDIX B - DETAILED SHAPE RETRIEVAL RESULTS

Results are presented here for all shape queries in the same format as in sections 8.4.3 and 8.4.4. For each query, the rank at which each drawing judged relevant by human subjects was actually retrieved is tabulated for each method, together with the P_n and R_n measures defined in section 8.2.1.

B1. Outer boundary queries

Table B1.1 - Query shape 32

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
33	11	4	4	1
82	2	15	18	52
R_n	0.9671	0.9474	0.9375	0.8355
P_n	0.7442	0.6372	0.6177	0.6524

The performance of all four types of matching is mediocre here, with global feature matching the best even though it retrieves the two relevant drawings in the wrong order. All three types of feature matching have the same problem - the 'horseshoe' shape of query 32 is made up from six straight-line segments and two circular arcs, while both drawings 33 and 82 consist of two straight-line and two circular arc segments. Matching based on local feature counts is thus at a considerable disadvantage. One might expect θ-s matching to be less susceptible to this problem - which it is with the more similar of the two drawings.

Table B1.2 - Query shape 44

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
42	9	1	7	1
41	3	5	3	3
43	15	11	35	13
180	5	8	14	51
104	2	4	2	14
40	30	3	8	17
R_n	0.9516	0.9876	0.9459	0.9122
P_n	0.7822	0.9154	0.7693	0.7244

This was a query where local feature matching performed well, and other types of matching gave adequate results. All feature matching methods readily detected similarity between the query (a scalene triangle) and other triangles of similar shape (drawings 41, 42 and 104), even where the ends were rounded off, though global matching failed with drawing 40, where corners were rounded off to a very marked extent. The system was less successful in finding the two right-angled triangles 43 and 180; the query shape contained no right angles, so the system made no attempt to look for them.

Table B1.3 - Query shape 46

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
52	2	2	1	1
146	1	1	2	2
51	5	9	6	26
18	4	8	7	27
13	10	4	14	13
R_n	0.9906	0.9879	0.9799	0.9275
P_n	0.9408	0.9228	0.8877	0.7529

All three feature-based methods perform successfully here. The rectangular shape (probably a cross-section of a girder) is obviously a good source of sufficiently characteristic features. Note that segment matching detects the two very similar drawings (compare with query 32), but is much less successful at picking out shapes with a lower level of similarity.

Table B1.4 - Query shape 47

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
59	1	1	1	1
53	3	3	3	3
142	2	2	2	2
68	5	5	5	5
12	7	15	17	29
174	11	11	14	20
R_n	0.9910	0.9820	0.9764	0.9561
P_n	0.9505	0.9181	0.9026	0.8647

This 'dumb-bell' shape (as it appears in silhouette) was handled reasonably effectively by all four methods - particularly by global feature matching, where performance is quite impressive. All methods were able to detect similarity as long as at least one end of the object contained a circular arc of approximately the same relative dimensions as the query (drawings 53, 59, 68 and 142). Drawings 12 and 174, with differently-shaped end-pieces, have fewer specific features in common with the query, and hence present the system with more of a problem.

Table B1.5 - query shape 57

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
94	6	10	7	2
3	22	25	16	22
99	2	2	2	30
R_n	0.9470	0.9316	0.9581	0.8940
P_n	0.7155	0.6674	0.7278	0.5944

An apparently simple query, for an L-shaped bracket, that nearly defeated the system - none of the four methods gave really adequate performance! The shape's simplicity lay at the root of the problem; since the query shape generated only a few, very common, features, it matched with virtually every rectangular part in the database. The poor performance of segment matching was equally disappointing, though this was almost certainly because the relative dimensions of drawings 3, 94 and 99 were very different from those of the query shape.

Table B1.6 - query shape 62

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
143	1	1	2	1
158	2	2	1	2
160	11	7	12	4
81	12	29	40	25
152	25	37	44	19
R_n	0.9517	0.9181	0.8872	0.9517
P_n	0.8029	0.7625	0.7116	0.8301

One of the few cases where segment matching yielded better results than feature matching. The system matched the query shape (a simple rectangle with rounded corners) reasonably well with drawings of similar length/width ratio (143, 158 and 160), but was markedly less successful with much narrower (152) or wider (81) shapes.

Table B1.7 - query shape 67

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
71	2	1	1	1
144	1	2	2	2
R_n	1.0000	1.0000	1.0000	1.0000
P_n	1.0000	1.0000	1.0000	1.0000

A relatively easy test of the system - the only two drawings remotely similar to the query shape were readily retrieved by all methods. Note that drawings 71 and 144, respresenting the same part, were made on two different CAD systems. The fact that *D* values for both drawings were almost identical is encouraging evidence that SAFARI is indeed capable of retrieving drawings from a variety of sources.

Table B1.8 - query shape 72

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
65	4	5	5	32
69	3	4	4	31
70	2	3	3	47
139	5	6	2	25
R_n	0.9933	0.9867	0.9933	0.7917
P_n	0.9049	0.8400	0.9049	0.3626

This query shape, made up entirely of smooth curves, provided some unexpected results. Since there were no sharp corners, boundary segments merging imperceptibly into each other, one might expect feature matching based on the number and type of individual segments to produce poor results, while θ-s matching would be less sensitive to variations in the size and curvature of individual segments. In fact the reverse effect was seen - feature matching proved surprisingly robust in this situation, while segment matching failed hopelessly. In retrospect, the problem with segment matching was obvious; when a boundary is made up of a large number of smooth curved segments, the position of the longest segment is somewhat arbitrary. Even if query and stored shapes are virtually identical, their similarity will not be recognized if they have their longest segments (always chosen as the start point for similarity matching) in different positions.

Table B1.9 - Query shape 75

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
83	1	1	1	9
140	3	2	13	3
74	20	12	8	1
76	2	6	5	7
R_n	0.9733	0.9817	0.9717	0.9833
P_n	0.9049	0.8942	0.8183	0.8781

Generally good performance, though each method had at least one flaw. Students judged four stored shapes similar to the query, a circular disc with four rectangular notches equally spaced around its circumference. All were basically circular, but 83 (like 75) had four notches, 140 had two, 74 was a perfect circle, and 76 had two rectangular protrusions on its circumference. Both global and local feature matching performed adequately, though giving too low a rank to the perfect circle (which generated few feature types). Existence matching (based on a search for arc angle triplets and parent features) ranked drawing 140 too low, and segment matching retrieved drawings in reverse order of similarity!

Table B1.10 - query shape 103

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
105	3	1	7	4
100	2	4	1	2
86	1	2	3	1
R_n	1.0000	0.9978	0.9890	0.9978
P_n	1.0000	0.9784	0.9058	0.9784

This pentagonal shape gave the system little trouble - though existence matching was a little less successful than the other methods in finding drawing 105, probably because (unlike the query shape) it contained no right angles.

Table B1.11 - query shape 115

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
90	1	3	3	4
119	6	1	1	1
114	2	2	2	12
R_n	0.9934	1.0000	1.0000	0.9757
P_n	0.9479	1.0000	1.0000	0.8436

A complex - though basically rectangular - shape, well handled by the three feature matching methods. Segment matching successfully retrieved the two most similar shapes, though it ranked the less complex drawing 114 rather too low.

Table B1.12 - Query shape 120

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
129	2	1	4	83
118	1	2	2	118
124	21	21	1	141
128	3	8	3	1
R_n	0.9717	0.9633	1.0000	0.4450
P_n	0.9021	0.8441	1.0000	0.3526

An excellent result for existence matching, fair for global and local feature matching, but hopeless for segment matching. Students' judgements here seemed to have been based purely on angular similarity. In response to a query in the form of an E-shaped bracket with the central arm significantly shorter than the others, they retrieved shapes with equal-length arms (129), with an almost non-existent central arm (128), and with a central arm *longer* than the others (124). Existence matching using arc angle triplets and parent features proved most successful here (alone proving able to retrieve drawing 124 at a reasonable rank), almost certainly because this emphasized angular similarity rather than similarity in feature size.

Table B1.13 - query shape 170

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
167	1	1	2	1
169	2	2	1	158
166	3	3	3	157
168	7	13	5	111
172	4	5	4	67
R_n	0.9973	0.9879	1.0000	0.3570
P_n	0.9835	0.9420	1.0000	0.2993

This query retrieved a group of shapes originally drawn as a family, sharing similar features and differing only in relative dimensions. Just as with the previous query, existence matching (with its emphasis on angular similarity) proved the most successful, and segment matching (sensitive to changes in boundary start point as relative segment lengths alter, as for query 72) the least. Segment matching in fact gave *worse* results than one would expect by pure chance.

Table B1.14 - query shape 175

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
27	1	1	3	1
24	6	26	12	14
37	4	14	11	6
102	12	31	9	18
82	2	2	2	2
R_n	0.9866	0.9208	0.9705	0.9651
P_n	0.9228	0.7424	0.7991	0.8413

Like other simple shapes, query 175 (a semicircle) proved quite tricky for the system. There were few problems in retrieving drawings closely based on a semicircle (27, 82), but shapes based on more (24, 102) or less (37) than a complete semicircle proved quite hard for local feature matching in particular to recognize as similar. In this case, the more general approach of global feature matching (less dependent on the presence or absence of specific features) provided the best results.

Table B1.15 - query shape 176

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
179	1	1	4	1
21	2	2	3	4
177	21	11	19	43
R_n	0.9603	0.9823	0.9558	0.9073
P_n	0.8537	0.9023	0.7265	0.7477

The system readily retrieved two of the three most similar shapes to this query, an irregular hexagonal shape, whichever method was used. The third shape, an almost regular hexagon, proved much more difficult to find, almost certainly because of its regularity. Almost the only feature it shared with the query shape was the number of sides!

Table B1.16 - query shape 183

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank
182	1	2	2	18
184	2	1	1	10
107	7	3	5	21
185	5	5	3	1
R_n	0.9917	0.9983	0.9983	0.9333
P_n	0.9368	0.9868	0.9868	0.7012

This 'daisywheel' query proved very easy to distinguish from the majority of circular shapes, though none of the methods was quite able to reproduce human judgements. As usual, segment matching performed less well than feature matching.

B2. All-boundary queries

Table B2.1 - query shape 9

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
8	1	1	1	1	1
6	2	2	2	7	7
7	3	3	3	17	17
R_n	1.0000	1.0000	1.0000	0.9623	0.9623
P_n	1.0000	1.0000	1.0000	0.7806	0.7806

All three feature-matching methods performed well with this query (already illustrated in Fig 8.11), though segment matching failed to identify the less similar members of the family with any reliability - a situation very similar to queries 120 and 170, above.

Table B2.2 - query shape 47

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
59	1	1	1	1	1
53	3	3	3	3	3
142	2	2	2	2	2
68	5	5	5	5	6
12	6	16	15	30	19
R_n	0.9976	0.9855	0.9867	0.9687	0.9807
P_n	0.9806	0.9335	0.9366	0.9034	0.9166

This query occurred in both outer-boundary and all-boundary lists, and it is perhaps instructive to compare the way in which its two versions were handled. Student similarity rankings were almost, but not quite, identical in the two cases (students shown only the outer boundary silhouettes also included drawing 174 in their lists of similar drawings). SAFARI performed marginally better when matching on the entire shape than when matching outer boundaries only, whichever method was used, suggesting that information on inner boundary position and shape was being constructively used. The relative performance of different matching methods was again similar; so too was the relative difficulty experienced by all methods except global feature matching in retrieving drawing 12.

Table B2.3 - query shape 48

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
54	1	1	1	1	1
147	2	2	2	2	2
R_n	1.0000	1.0000	1.0000	1.0000	1.0000
P_n	1.0000	1.0000	1.0000	1.0000	1.0000

This query, a shape in the form of a circular gasket, was readily matched to the only two really close shapes in the database by all five methods - not a very severe test of the system, but a successful one nevertheless.

Table B2.4 - query shape 49

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
61	4	2	1	8	5
88	1	3	20	14	8
56	3	1	2	12	6
110	2	4	7	7	3
R_n	1.0000	1.0000	0.9701	0.9536	0.9820
P_n	1.0000	1.0000	0.8584	0.6559	0.8040

Shape 49, a rectangular plate with one corner chamfered off, proved a useful demonstration of the system's power to distinguish between closely similar shapes. The feature matching methods (particularly global and local feature matching) had little difficulty in identifying the four desired shapes from over 60 similar straight-edged drawings. Existence matching performed less well, though the low ranking of drawing 88 (and to a lesser extent drawing 110) was in fact a problem caused by the feature set used rather than by existence matching *per se*. As shown in Fig 8.13, drawings 56 and 61 (but not 88 or 110) are fundamentally rectangular. Hence the top-level shapes of drawings 56 and 61, and the status of their low-level line segments, are quite different from those of drawings 88 and 110. The feature set used here for global and local matching emphasized line curvature and discontinuity angle, with successful results. Existence matching, by contrast, used more complex parameters such as arc angle triplet and parent feature composition, emphasizing the difference between those shapes which were basically rectangular (56 and 61) and those which were not (88 and 110). Segment matching was not conspicuously successful, though the addition of inner boundary shape matching markedly improved its performance, successfully rejecting shapes whose inner boundaries were not all circular.

Table B2.5 - query shape 57

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
45	9	22	18	2	5
94	10	23	16	3	6
99	1	1	1	38	29
R_n	0.9722	0.9206	0.9425	0.9266	0.9325
P_n	0.8011	0.6743	0.7157	0.7328	0.6345

This shape again figured in both "outer-boundary" and "all-boundary" query lists, though student similarity rankings in the two cases showed significant differences (drawing 3, for example, was well below the threshold score of 0.1 in the "all-boundary" rankings). The presence of inner boundaries clearly influenced students' judgement, though not in any obvious way - neither drawings 94 nor 3 contained inner boundaries, yet one was chosen, the other rejected. SAFARI performed almost as poorly with the all-boundary version of this query as with the outer-boundary version discussed earlier, for the same reasons. It is interesting to note that the feature matching methods performed best on the shapes on which segment matching performed worst, suggesting that for this query at least, a combination of both methods might prove beneficial.

Table B2.6 - query shape 72

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
65	4	5	5	40	42
69	3	3	4	41	48
139	5	4	2	37	39
70	2	2	3	55	63
R_n	0.9940	0.9940	0.9940	0.7560	0.7275
P_n	0.9073	0.9073	0.9073	0.3176	0.2948

The results of matching this query in its all-boundary form were almost identical to those obtained by outer boundary matching: good with the three feature matching methods, poor with both forms of segment matching. The fact that inner boundary shape matching yielded worse results than position matching alone suggests that the system may be giving too much weight to inner boundary shape similarity, over-emphasizing the differences in inner boundary shape between the query shape and the four retrieved drawings.

Table B2.7 - query shape 80

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
140	2	1	2	1	1
76	1	3	3	18	16
101	21	9	16	2	2
75	3	2	11	10	6
83	5	8	1	11	10
R_n	0.9795	0.9904	0.9783	0.9675	0.9759
P_n	0.9205	0.9386	0.8958	0.8324	0.8671

All shape matching methods handled this query (a notched circular disc) adequately, with local feature matching performing particularly well. Global and existence matching failed to rank drawing 101 (with a completely circular outer boundary, which generated few feature types) as high as human judges, a problem similar to that seen with query shape 75. Segment matching readily retrieved drawing 101, though (perhaps reasonably) it gave a low ranking to drawing 76, which contained local shape features standing proud of the circumference.

Table B2.8 - query shape 89

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
108	1	1	1	1	1
114	2	2	3	2	2
115	5	3	6	22	17
119	3	4	5	12	8
90	24	10	16	29	26
R_n	0.9759	0.9940	0.9807	0.9386	0.9530
P_n	0.9141	0.9668	0.8809	0.7676	0.8046

All methods successfully identified the two drawings most similar to the query, a rectangular part with a fairly simple pattern of features machined out of one side. The remaining drawings, similar but with a much more complex pattern of features, proved more of a test of the system, though the three feature-matching methods readily retrieved all but drawing 90, the most complex. Segment matching failed to rank any of these three shapes sufficiently highly.

Table B2.9 - query shape 100

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
105	1	2	5	92	4
181	6	1	19	33	25
178	71	4	20	32	15
R_n	0.8571	0.9980	0.9246	0.7004	0.9246
P_n	0.6869	0.9789	0.5771	0.2881	0.5945

An unusual seven-sided shape which caused the system some difficulty. Only local feature matching proved really equal to the task of identifying all three shapes (containing five, ten and eight sides respectively) deemed similar by student judges. Global matching failed dismally with drawing 178, almost certainly because this shape was much more regular than the other two, and hence had much lower values for length and discontinuity angle variances. Existence matching also performed poorly, ranking drawings 181 and 178 too low because they contained too few of the required types of arc angle triplet or parent feature composition. Segment matching proved the worst method of all, almost certainly due to differences in outer boundary starting point, at least where inner boundary matching was based solely on position. Where inner boundary shape was taken into account, performance improved markedly - to be expected given the close similarity of inner boundary shape between the query and all retrieved drawings.

Table B2.10 - query shape 109

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
22	1	1	1	49	50
23	2	2	5	38	35
64	10	5	16	163	159
R_n	0.9861	0.9960	0.9683	0.5159	0.5278
P_n	0.9116	0.9625	0.8098	0.2045	0.2109

A partly rectangular, partly circular shape, which again showed local feature matching to advantage. Global and existence matching ranked drawing 64 much lower than student judges, and segment matching gave rankings little different from those expected by pure chance. Investigation of these discrepancies suggested that the causes were the same as for query 100 above.

Table B2.11 - query shape 120

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
118	1	1	2	116	86
129	3	2	6	107	9
124	4	8	1	124	42
128	2	3	3	1	1
R_n	1.0000	0.9940	0.9970	0.4940	0.8084
P_n	1.0000	0.9601	0.9766	0.3622	0.5844

Comparison of these results with those for query 120 in outer-boundary mode is quite instructive. Drawing 124, previously difficult to retrieve by all except existence matching, is now readily retrieved at a reasonable rank by all three methods of feature matching because its pattern of inner boundaries is very similar to that of the query shape. There is even some improvement in segment matching performance, particularly when inner boundary shapes as well as positions are taken into consideration.

Table B2.12 - query shape 122

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
117	1	1	1	1	1
116	2	2	2	2	2
121	22	4	3	5	3
R_n	0.9623	0.9980	1.0000	0.9960	1.0000
P_n	0.8537	0.9789	1.0000	0.9625	1.0000

Another case where all three retrieved drawings were very similar to the query shape, and all matching methods performed well. The only exception was the failure of global matching to rank drawing 121 (the most complex shape of the three retrieved) sufficiently highly. The greater complexity of this shape meant that values of global parameters such as segment length and arc angle variances differed markedly from those for the query or the other two retrieved shapes. Local and existence matching proved more robust here, as positive matches could be made on the features which query and stored shapes *did* have in common.

Table B2.13 - query shape 154

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
149	1	1	1	2	9
171	2	2	2	1	1
79	3	3	5	5	8
83	6	7	4	19	22
74	4	4	8	6	3
R_n	0.9988	0.9976	0.9940	0.9783	0.9663
P_n	0.9913	0.9839	0.9530	0.8921	0.8237

An exercise in inner boundary matching, as illustrated in Fig 8.14. The query, and hence most retrieved shapes, were circular discs with complex patterns of inner boundaries. All three feature-matching methods performed well, and segment matching performed better than on many queries. Its only real "failure" was with drawing 83, which did not have a completely circular outer boundary, and was thus ranked below many drawings which did - possibly indicating that the relative weighting given to outer boundary shape similarity was too high here. The fact that segment matching using inner boundary shapes gave poorer results than when using position matching suggests that there may still be room for improvement in selection of the order in which inner boundaries from query and drawing are matched with each other.

Table B2.14 - query shape 159

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
163	1	1	4	1	1
158	2	4	1	2	2
77	38	46	41	18	15
162	13	9	16	4	3
160	20	14	17	5	4
143	3	2	2	7	8
62	4	3	3	8	7
R_n	0.9538	0.9556	0.9512	0.9852	0.9895
P_n	0.8591	0.8787	0.8547	0.9239	0.9493

This query, again largely an exercise in inner boundary matching (though this time within rectangular outer boundaries) was another of the few cases where segment matching seemed superior to feature matching. While successfully retrieving rectangles of similar length/width ratio (163, 158, 143), the feature-matching methods were relatively less successful with rectangles having markedly different length/width ratios (77), or where additional inner boundaries were present (160, 162). The segment-matching strategy of matching inner boundaries individually, rather than simply looking for the presence or absence of the inner boundary pattern features defined in section 4.4.4, seems to be more appropriate here.

Table B2.15 - query shape 175

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
82	1	1	1	2	2
27	3	3	3	1	1
87	2	2	2	3	3
37	4	15	12	12	12
145	39	16	16	149	121
R_n	0.9590	0.9735	0.9771	0.8169	0.8506
P_n	0.9015	0.8809	0.8916	0.7846	0.7946

It is interesting to note that students' similarity rankings for this query differed markedly from those for its outer-boundary version above. The presence of a distinctive inner-boundary pattern (four circular holes with centres lying on the same circular arc) in the query shape clearly influenced students' choice, moving drawing 82 up from fifth to first position, and including drawings 87 and 145 (all of wich shared the same pattern of inner boundaries) in the list of those retrieved in place of 24 and 102. On the whole, the system reflected this change in emphasis well. Drawings 82 and 87 were retrieved high on all lists; so too was drawing 27, lacking the required inner boundary pattern but virtually identical in external shape. Drawing 37 was less well recognized, for the reasons discussed in the previous section.

The system's failure to retrieve drawing 145 was more unexpected. It was eventually traced to a problem with the original drawing. The outer boundary had been badly drawn, preventing program SKELETON from building the expected shape feature hierarchy, and hence preventing later programs from generating the required range of shape features. While the drawing could simply have been corrected and the shape-matching experiments re-run, it was decided to leave it in place, as a demonstration of the severe problems that could occur if stored shapes were incorrectly drawn. Some form of integrity checking would clearly be needed for any operational shape database.

Table B2.16 - query shape 176

Drawing No	Global rank	Local rank	Exist rank	Segmatch rank (position)	Segmatch rank (shape)
179	1	1	3	1	1
21	7	2	4	3	4
177	13	28	21	58	27
88	9	3	9	7	2
R_n	0.9701	0.9641	0.9596	0.9117	0.9641
P_n	0.7966	0.8879	0.7379	0.7737	0.8734

Unlike the previous query, students' similarity rankings were very similar for both outer-boundary and all-boundary versions, the only difference being that drawing 88 was included in the all-boundary list, presumably on the strength of its inner boundary pattern. The system's performance was little different from that described for the outer-boundary version of the query above; drawings 179, 21 and 88 were retrieved reasonably easily, but 177 again proved more difficult, for the reasons outlined in the previous section.