A LOWER JURASSIC AMMONITE IMAGE DATABASE AND ITS APPLICATIONS

by

BO LIANG

B.Eng., Xiangtan Mining Institute, 1984
M.Eng., China University of Geosciences, 1987

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

GEOLOGICAL SCIENCES

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

May 1994

© Bo Liang, 1994

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Geological Sciences

The University of British Columbia
Vancouver, Canada

Date Oct. 13, 1994

DE-6 (2/88)

Abstract

Ammon is the first computerized image database for ammonites that allows automated measurement of morphological parameters from images. The database contains 7790 specimens representing 15 families, 179 genera and 1319 species. Each specimen has 102 descriptors covering taxonomy, quantitative morphology, qualitative morphology, stratigraphy, locality information and general comments.

Species density diagrams are plotted with respect to basic shell geometry. The data show strong correlations among basic geometric parameters which may be related to restrictions imposed by whorl shape and size. Buckman's Law of Covariation is shown to be applicable to Lower Jurassic ammonite stocks and new covariations are demonstrated among tuberculation, ribbing, whorl shape and ventral features, all of which have functional implications.

Morphological and stratigraphical data from Ammon suggest a possible correlation between basic shell geometry, morphological variability and taxa duration. Low within-population morphological variabilities and short durations are associated with taxa that inhabit the most densely populated geometric region. Away from this peak, there is a steady increase in both morphological variability and taxa longevity implying a decrease in selective pressure.

The taxonomic diversity of Lower Jurassic ammonites peaks first at the family level in the Ibex Zone, followed by generic data one zone later, and finally species in the Margaritatus Zone implying a delay in radiation for lower levels of ammonite taxa. The Late Sinemurian and Early Pliensbachian have high family origination rates and relatively low species diversities; species reach their highest diversity level in the Late Pliensbachian which also has the highest family extinction rate and an almost zero origination rate.

Two morphological diversity indexes are used to study changes through time. The range index describes the range of the morphospace occupied. The Shannon index measures both the range and the evenness of the occupation. Both indexes indicate a similar pattern. There is a rapid rise to an Early Sinemurian maximum through morphological radiation,

especially in terms of the basic shell geometry, followed by a pronounced fall in the middle Sinemurian with diversity remaining relatively high and stable thereafter.

It is evident by comparing taxonomic and morphological diversity changes through time that initial morphological diversification in Lower Jurassic ammonites exceeded the proliferation of lower-level taxa. For example, the morphological diversity of the Bucklandi Zone with 46 species is slightly higher than that of the Margaritatus Zone with 177 species. It demonstrates that morphological diversity and taxonomic diversity are not simply correlated but that taxa initially occupy discrete positions within a broad morphospace followed by the occupation of intervening areas.

It is observed that all Lower Jurassic ammonite families radiating into numerous species in the first zone of their occurrence have durations of less than 4 zones; in other words, rapid origination is associated with short duration.

Table of Contents

Abstract ii

Table of Contents iv

Acknowledgments vii

Chapter 1 Introduction 1

1.1 Introductory Statement 1

1.2 Previous Work 2

1.3 Purpose and Scope 3

Chapter 2 The Lower Jurassic Ammonite Image Database -- Ammon 5

2.1 Introduction 5

2.2 The Database Structure 6

2.3 SQL -- The Query Language 18

2.3.1 Data entry 19

2.3.2 Updating the database 19

2.3.3 Querying the database 20

2.3.4 Changing the structure of the database 23

2.4 Data Sources 24

Chapter 3 Image Analytical Techniques 27

3.1 Introduction 27

3.2 Digitization 27

3.3 Interactive Image Measurement Module 28

3.4 Automatic Image Measurement 33

3.4.1 Preprocessing 34

3.4.2 Edge detection 34

3.4.3 Thinning and line tracing 35

3.4.4 Locating coiling axis 35

3.4.5 Classification of coiling curves and ribs 36

3.4.6 Deriving morphological parameters 38
3.5 Fourier Analysis of Whorl Shape 38
Chapter 4 Morphological Variation and Covariation 45
4.1 Introduction 45
4.2 Lower Jurassic Ammonite Shell Morphology 45
4.2.1 Background 45
4.2.2 The Sample of Observations 46
4.2.3 Basic Shell Geometry 47
4.2.4 An interpretation of Buckman's Law of Covariation 51
4.2.5 Morphological Variability and Taxon Longevity 54
4.2.6 Correlation between Basic Shell Geometry and Other Morphological Features 60
4.3 Covariation 69
4.3.1 Introduction 69
4.3.2 Buckman's Law 71
4.3.3 Volution 74
4.3.4 Whorl Shape 77
4.3.5 Ribs 80
4.3.6 Tuberculation 82
4.3.7 Venter 84
4.3.8 Summary & Speculations 86
4.4 Ontogenetic Variation 89
4.4.1 Umbilical Ratio (U) 89
4.4.2 Shell Expansion Rate (W) 91
4.4.3 Ratio of Whorl Width to Whorl Height (WWWH) 92
4.4.4 Primary Rib Density (PRHW) 94
4.4.5 Superfamily 95

Chapter 5 Changes through Time 96
5.1 Introduction 96
5.2 Stratigraphic Division 97
5.3 Choice of Characters 97
5.4 Geometry through Time 98
5.5 Taxonomic Diversity 101
5.6 Morphological Diversity 104
5.6.1 Morphological Range Diversity Index (MRDI) 104
5.6.2 Shannon Diversity Index (SDI) 108
5.7 Discrepancies between Morphological and Taxonomic Diversities 113
5.8 Other Interesting Observations 114
Chapter 6 An Aid to Identification 119
6.1 Introduction 119
6.2 User Interface 121
6.3 Examples 128
Chapter 7 Summary and Conclusions 151
References 154
Appendix 1 Various Morphological Parameter Value Combinations and their Abundance in the Database Ammon 164
Appendix 2 Source Code for the Graphics Interface of Ammon 200
Appendix 3 Source Code for the Program--imageEdit 259
Appendix 4 Source Code for the Interactive Image Measurement Module .. 276
Appendix 5 Source Code for the Automated Image Measurement Module . 321 (without the Canny edge detector and the Pavlidis thinning algorithm)
Appendix 6 Source Code for Fourier Analysis 407
Appendix 7 Source Code for the Animation of Whorl Shape Changes 422

Acknowledgments

The completion of this thesis is made possible, in large part, to the guidance, encouragement and financial support of my thesis advisor Dr. P. L. Smith whose help often extended beyond research. His comments and suggestions on the earlier drafts have greatly improved the manuscript. In addition, his research library is the major source of data making up the database Ammon.

Special thanks are due to Dr. R. J. Woodham (Computer Science Department, UBC) who kindly provided the program implementation of the Canny edge detector in the initial stage of this project and made valuable suggestions as a member of my supervisory committee.

I wish to thank Dr. T. H. Brown for his critical comments on Chapter 4 which led to a major improvement. I would also like to thank Dr. W. C. Barnes for his encouragement and detailed review of the manuscript.

This study was supported by a University Graduate Fellowship from the University of British Columbia and grants to P. L. Smith from the Natural Science and Engineering Research Council, which are gratefully acknowledged.

I owe a great deal to my fellow graduate students at the Paleontology Lab of UBC. Giselle Jakobs helped updating the information for Toarcian ammonites in the database and provided some reference material. Genga Nadaraju and Jozsef Palfy were always there to help with my word-processing related questions. Jozsef Palfy also provided advice with Sinemurian ammonites and the thesis layout. I benefited from discussions with Gary Johannson. A great deal of thanks must go to Tomas Clemo and Joann Bessler from Hydrology Lab of UBC for their help.

Dr. H. W. Tipper of the Geological Survey of Canada at Vancouver kindly sent me a well preserved specimen of *Fanninoceras fannini* for study. He also helped updating the information for Hettangian ammonites from North America.

I am indebted to Mr. Stewart Kingdon (Computer Science Department, UBC) who provided the program implementation of the Pavlidis thinning algorithm and helped digitizing a few ammonite images. I thank Dr. Andre-Pierre Benguerrel (UBC) for his encouragement, support and valuable advice.

My sincerest thanks go to P. L. Smith and K. Gordanier-Smith for making me feel at home as I adapted to a different environment.

Finally I would like to thank my wife Cindy and my parents for their dedicated support that I can always count on.

Chapter 1 Introduction

1.1 Introductory Statement

Jurassic ammonites are commonly regarded as excellent index fossils, evolving rapidly and being geographically widespread. The Early Jurassic is a 30 Ma interval and rocks of this age are widespread in Western North America both in allochthonous sequences and on the craton. "The high resolving power of ammonite biochronology offers the best method of correlating between the suspect terranes and comparing their disparate geological histories" (Smith 1988). Obtaining biochronologic information requires the recognition of associations of ammonite shell morphologies (species), associations which can be demonstrated to maintain their superpositional relationships throughout their entire geographic distribution. Each species shows variation within the population and also during ontogeny, variation which must be documented and understood before a species can be identified with confidence. The description of morphology and the documentation of variation can be approached in three ways:

(a) The use of geometric models to describe morphology quantitatively (theoretically capable of producing a universal set of all possible morphologies).

(b) The use of images of specimens to obtain quantitative data and qualitative descriptors (determining which parts of the universal set are actually occupied); a computerized image database is best suited for this task .

(c) The use of a database management system to manipulate the data from (b) to recognize species.

In addition, the data from (b) can be used to test hypotheses on evolution such as the claim for greater early disparity (Whittington 1985; Gould 1989), to explore possible correlations between taxon longevity and morphological variability, to detect subtle links between morphological features, to more rigorously document extinction events and so on.

Continuing advances in computer hardware and software technologies have made it feasible to store and manipulate fossil images digitally. Access speeds and storage capacities are progressively increasing while costs decrease. An image scanner with 256 gray shades that cost $5000 when this project started in 1990, now costs less than $1000 (1994). Image databases for other fossil groups are likely to emerge which, when networked by the "Information Superhighway", will make a huge impact on the way that paleontologists work.

1.2 Previous Work

The first attempt to mathematically describe coiled shells was made in the nineteenth century by Moseley (1838) who established a geometric model for mollusk shells based on the logarithmic spiral. Raup (1966, 1967) reexamined and modified Moseley's model and was able to express the basic forms of most mollusc shells using four geometric parameters. Raup (1967) showed that ammonoids occupy only a portion of the theoretical range of possibilities for shell geometry, and that some geometric combinations have been utilized more than others. Raup's approach was subsequently used by Ward (1980) to compare the shell geometry of Jurassic and Cretaceous ammonoids to Mesozoic and Cenozoic nautiloids. Chamberlain (1981) went further and related ammonoid and nautiloid shell geometry to hydrodynamic efficiency by showing that species with coiled shells cluster on adaptive peaks that optimize streamlining and stability. More recently, Saunders and Swan (1984,1987) have used Raup's parameters to analyze the morphologic diversity of mid-Carboniferous ammonoids. These studies demonstrated the usefulness of Raup's approach in evaluating and graphically portraying the distribution of ammonoid shell geometry.

During the past twenty years progress has been made towards storing fossil information on computer databases. Price (1984) described how a hierarchical database management system was designed for material in the Sedgewick Museum, Cambridge. The prototype Ammon was initiated to systematically handle data accruing from stratigraphic studies of Jurassic sedimentary basins in Nevada, Oregon and British Columbia (Smith

1986). GONIAT, a computer database for Paleozoic ammonoids was developed by Kullmann, Korn, Kullmann and Petersen (1991).

Images are of great importance in paleontology. For example, the Paleontological Cataloging System Development Committee of the American Association of Stratigraphic Palynologists and Micropaleontology Press have been working together to design an image database (Huber 1990). Similarly a major oil company has been developing a textual database with an image library (Rich 1989). Fossil images in these systems are for visual comparisons only, a big step forward would be to derive morphologcal parameters directly from images in the database.

Numerous attempts have been made to interpret images since the advent of digital computers. Most progress has been made in systems that perform a particular task in a controlled environment which provides strong constraints on the recognition process. On the other hand, less progress has been made in the areas where computers have been called upon to extract ill-defined information from images (Horn 1985). The major elements in an ammonite image are line features, such as ribs and the coiling curve. Several attempts to extract line work from natural images have been published. Cleynenbreugel et al (1988) described initial experience with a system that extracts road networks from SPOT satellite images for example, and a knowledge-based system was used for automatic segmentation of coronary vessels from digital angiograms (Stansfield 1986).

1.3 Purpose and Scope

This thesis focuses on Lower Jurassic ammonites and the primary objectives are :

1. To incorporate images into the textual Ammon database and expand the database to give a comprehensive coverage of Lower Jurassic ammonites.

2. To design a computer system for measuring morphological parameters from images in the Ammon database and to assess the feasibility of a fully automated approach.

3. To quantify the description of whorl shape so that intraspecific variations can be documented objectively and comparisons can be made among different taxa.

4. To quantify and document intra-taxon variation in morphology which serves to define taxa more objectively.

5. To map the distribution of Lower Jurassic ammonites in terms of basic shell geometry so that relative densities of occurrence in different geometric regions can be evaluated and functional factors governing the distribution can be explored.

6. To detect covariations among morphological features and give interpretations whenever possible.

7. To explore possible correlation between taxon longevity and morphological variability.

8. To quantitatively document taxonomic diversity changes and morphological diversity changes through geological time, and study the relation between the two.

9. To design a computer system to aid in the identification of Lower Jurassic ammonites which may also serve as an education tool.

Chapter 2 The Lower Jurassic Ammonite Image Database - Ammon

2.1 Introduction

Ammonite species normally show variation within the population which must be determined before a species can be identified with confidence. The retrieval of information for any species requires searching a large literature which limits the speed of identification and makes direct comparison of specimens difficult.

During the last two decades there is increasing need to evaluate global biotic patterns in time. Plate tectonic and paleobiogeographic research require paleontological input, and yet our established practices are hardly appropriate for integrating and evaluating fossil data on a global scale, particularly for narrow time intervals (Spicer 1986). Maintaining up to date revisions of taxa based on specimens and literature from different geographical areas is also time consuming.

Computerized databases have the potential to solve these problems. During the past twenty years progress has been made towards storing fossil catalogue information on computer databases. Price (1984) described how a hierarchical database management system was designed for material in the Sedgewick Museum, Cambridge. GONIAT, a computer database for Paleozoic ammonoids was introduced at the International Cephalopod Symposium in 1990. GONIAT utilizes Dbase software supplemented with additional programming which provides a simplified user interface (Kullmann, Korn, Kullmann and Petersen 1991). Images are of great importance in paleontology. For example, the Paleontological Cataloging System Development Committee of the American Association of

Stratigraphic Palynologists and Micropaleotology Press have been working together to design a paleontological catalog database where, according to Huber (1990) : "Palynologists are evaluating systems that can capture images of fossils as digital data directly from a microscope, and then store, manipulate and retrieve them from an image database stored on optical disk". Similarly a major oil company has been developing a textual database and an image library in a system that allows one or more images of a particular fossil type to be combined with textual information (Rich 1989). Klapper and Foster (1993) described a shape analysis system that identifies conodont species with its digitized outline. The research that leads to this thesis is a continuation of Ammon, a Lower Jurassic ammonite database which was initiated to systematically handle data accruing from stratigraphic studies of Jurassic sedimentary basins in Nevada, Oregon, and British Columbia (Smith 1986).

The computerization of paleontological data is not without difficulty. Conventionally descriptions are prepared in a format which has evolved over many years of taxonomic practice. Within this framework presentation may vary and, for each individual description, authors are free within limits to choose whatever phraseology seems most appropriate. For information-retrieval purposes all elements of descriptions should be strictly comparable requiring that terminology be used consistently. Other areas of difficulty include character selection, standardization of data and amount of data that needs to be entered into database.

2.2 The Database Structure

The Ammon was originally designed by Smith (1986) on a mainframe computer using TAXIR as the database management program. Subsequently it was transferred to a

Item\Descriptor	Descriptor 1	Descriptor 2	Descriptor 3
Item 1	State 1	State 3	State 2
Item 2	State 3	State 2	State 1
Item 3	State 2	State 4	State 5

Figure 2-1 A table in the relational model

workstation (SparcStation1) and converted to work under Oracle Relational Database Management System.

In the relational model, data are represented in tabular form (Fig. 2-1). Each row of the table is referred to as an item and each column is called a descriptor. The rows of the table link together related values, and different tables may be linked implicitly if they include values of a common item, e.g. specimen number.

"The first step in designing a database is to establish a list of descriptors that will describe the items in every possible way that might be useful now and in the future but with a minimum of information redundancy. In the database Ammon described below, an item is an ammonoid and the descriptors are categories of information describing those ammonoids" (Smith 1986). There were 96 descriptors proposed by Smith (1986) and the author has added another 11 descriptors which include relative location of image file and scale of illustration. These descriptors are divided into six categories, viz., taxonomy, quantitative morphology, qualitative morphology, stratigraphy, locality (including catalogue information) and general comments (Fig. 2-2). The following notes describing the states of these descriptors are taken from Smith's original paper (1986) with a few modifications.

Taxonomy	1 SUBORDER		2 SUPERFAMILY		3 FAMILY	
	4 SUBFAMILY		5 GENUS		6 SUBGENUS	
	7 QUALIFIER		8 SPECIES		9 SUBSPECIES	
	10 TAXAUTHYEAR		11 REFAUTHYEAR		12 SYNONYMY	
	13 SYNSPECIES					
Quantitative Morphology	14 DMAX	15 DPHRAG	16 D	17 UD	18 U	19 W
	20 WH	21 WHMAX	22 WHD	23 WW	24 WWD	25 WWWH
	26 PRHW	27 RIBWIDTH	28 SRHW	29 THW	30 BISPACE	31 FURCPOS0
	32 UNITUBPOS0	33 CHW	34 SF	35 VOLUTION0	36 AH	37 APPROX
Qualitative Morphology	38 VOLUTION	39 WHORL SHAPE	40 EXPANSION	41 UWALL	42 UWALLHT	43 UWALLANG
	44 USHOULD	45 FLANKS	46 VENTER	47 VENTPROF	48 KEEL	49 SULCI
	50 PRIBD	51 PTREND	52 PFORM	53 PPROF	54 FURC	55 FURCPOS
	56 SRIBD	57 STREND	58 SFORM	59 SPROF	60 TUBERC	61 UNITUBPOS
	62 CONSTRD	63 CTREND	64 CFORM	65 APERTURE	66 SUTURE	67 ONTOGENY
Stratigraphy	68 STAGE	69 SUBSTAGE	70 EURZONE	71 EURSUBZONE	72 ZONE	73 SUBZONE
	74 HORIZON	75 FORMATION	76 MEMBER	77 LITHOLOGY	78 DATUM	79 RELDATUM
	80 SITU	81 ASSOCSPEC				
Locality and Catalogue Information	82 AREA	83 COUNTRY	84 PROVINCE	85 LONG	86 LAT	87 Q
	88 SECTNAME	89 SECTNO	90 LOCNO	91 OTHERNO	92 SUBLOCNO	93 SUPERLOCNO
	94 REPOSITORY	95 TYPE	96 COLLECTORYR		97 GENERALOC	98 SPECNO
	99 IMAGE	100 SCALE				
Others	101 SPECFEATURE		102 REMARKS			

Figure 2-2 The descriptors of the database Ammon (modified from Smith 1986)

Taxonomic descriptors. -- Ammonoid taxonomy has been the subject of a prolific literature culminating in the Treatise on Invertebrate Paleontology (Arkell, Kummel and Wright, 1957) and subsequent attempts to update this information (for the Jurassic see Donovan and Forsey, 1973, and Donovan, Callomon and Howarth, 1981).

Descriptors 1-4 (SUBORDERS, SUPERFAMILIES, FAMILIES AND SUBFAMILIES) are self explanatory. Since this database concerns only the Ammonoidea it is not necessary to go above the sub-ordinal level.

Descriptors 5, 6, 8 and 9 should be the names used by the author of the publication if the information is derived from the literature.

Descriptor 7 (QUALIFIER): 1. CF (confer); 2. AFF (affinis); 3.? (uncertain generic assignment); 4. EX GR (ex grupo); 5. SS (sensu stricto); 6. SL (sensu lato); 7. SP? (uncertain species assignment).

Descriptor 10 (TAXAUTHYEAR) is the author of the taxon together with the year of publication (all in parentheses if the species has been transferred from the original genus).

Descriptor 11 (REFAUTHYEAR) is the reference (author and year) in which the figure of the specimen appears (if applicable).

Descriptor 12 (SYNONYMY) and descriptor 13 (SYNSPECIES) are, of course, the valid genus and species name as judged by the person designing the database.

Quantitative morphology. - Quantitative approaches to morphology were ignored in the Treatise (Arkell, Kummel and Wright, 1957) and it has been left to individual workers to define their own variables with each publication so that a lack of consistency has resulted. The primary variables used in Ammon are descriptors 14-17, 19-21, 23, 26-33, 35-36. The variables derived from them could be calculated but the advantage of improved storage efficiency would be offset by slower retrieval.

Descriptor 14 (DMAX) is the maximum diameter measurable for that specimen. If the specimen is mature this will be indicated by information recorded in REMARKS (e.g., approximated sutures) or it will be reflected in the information for the APERTURE (descriptor 65).

Descriptor 15 (DPHRAG) is the shell diameter at the adoral end of the phragmocone.

Descriptor 16-18, 20, 22-29 and 33 are illustrated and explained in Fig. 2-3.

Descriptor 19: $W = (LR/L)^{2\pi/r}$ where LR and L are the same linear feature separated by an angular distance about the coiling axis of r radians (a version of the expansion rate of Raup, 1966, 1967). This variable can be used to extrapolate linear measurements (D, UD, etc.) backwards or forwards in ontogeny, assuming that the expansion rate remains constant during growth.

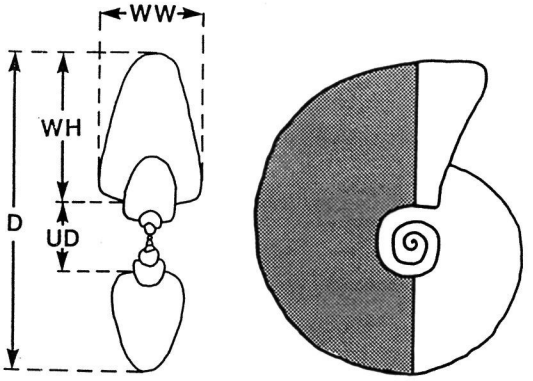


Figure 2-3 The quantitative descriptors used in Ammon (from Smith 1986). D (Descriptor 16) = the diameter at which all measurements and qualitative observations are recorded. UD (Descriptor 17) = the umbilical diameter. U (Descriptor 18) = UD/D. WH (Descriptor 20) = whorl height. WHD (Descriptor 22) = WH/D. WW (Descriptor 23) = whorl width. WWD (Descriptor 24) = WW/D. WWWH (Descriptor 25) = WW/WH. The shaded half-whorl represents the interval at diameter D over which counts of primary ribs, secondary ribs, tubercles and constrictions are made (Descriptors PRHW (26), SRHW (28), THW (29) and CHW (33) respectively).

Descriptor 21 (WHMAX) is the maximum whorl height measurable for the specimen. This descriptor was not included in the original Ammon but added by the author to deal with fragmentary specimens.

Descriptor 30 : BISPACE = Y/D, where Y is the distance between tubercles in bituberculate ammonoids.

Descriptor 31 : FURCPOS0 = FD/WH, where FD is the distance between furcation point and umbilical seam. FURCPOS0 is the quantitative equivalent of FURCPOS and is added by the author.

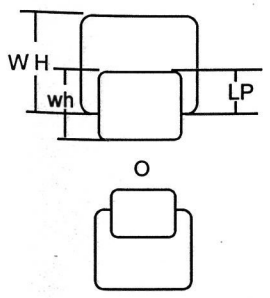


Figure 2-4 Descriptors 35, 36

VOLUTION0 = LP/wh.

AH = LP/WH.

Descriptor 32 : UNITUBPOS0 = TD/WH, where TD is the distance between tubercles and umbilical seam in unituberculate ammonoids. UNITUBPOS0 is the quantitat--ive equivalent of UNITUBPOS and is added by the author.

Descriptor 34 (SF) is the sutural formula for the specimen at that diameter. A specimen may be entered into the computer several times at different shell diameters so that the ontogenetic development of a sutural pattern may be recorded.

Descriptor 35: VOLUTION0 = LP/wh where LP is the whorl overlap and wh is the inner whorl height (Fig. 2-4). This descriptor is the quantitative equivalent of VOLUTION

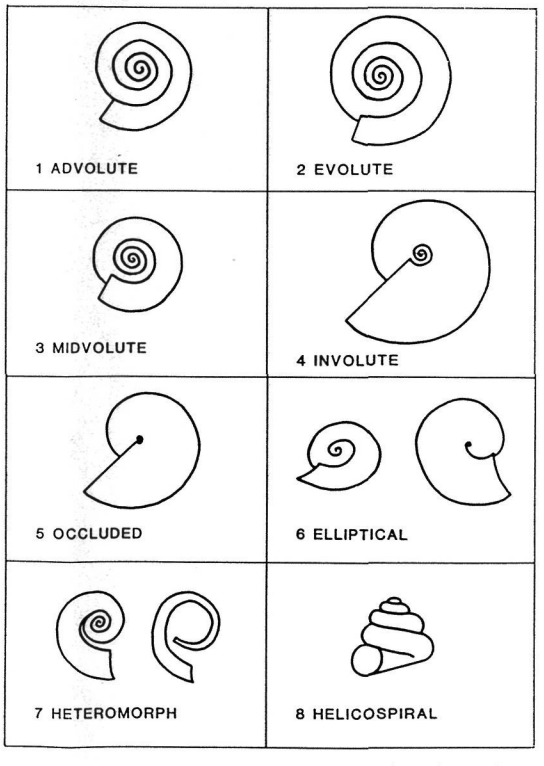


Figure 2-5 The states for the descriptor VOLUTION (38) (from Smith 1986)

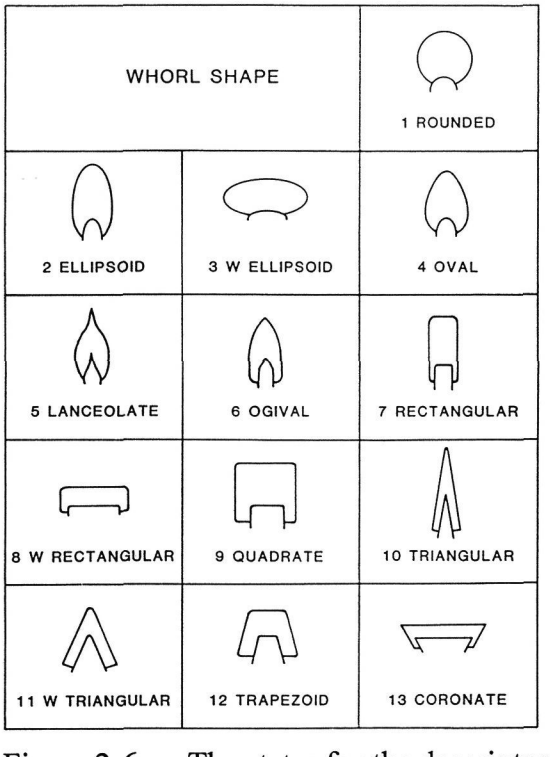


Figure 2-6 The states for the descriptor WHORL_SHAPE (39) (from Smith 1986)

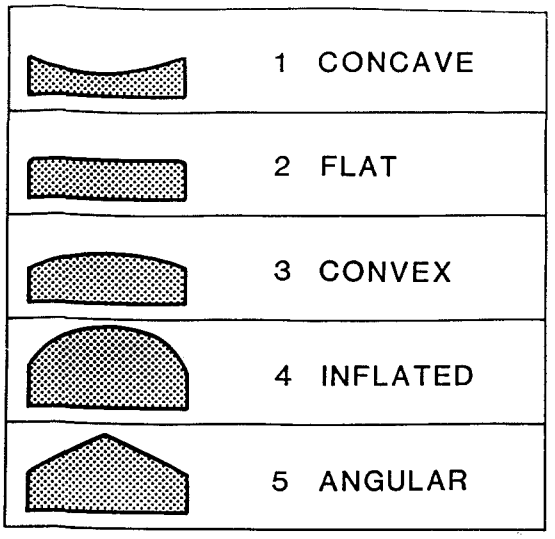


Figure 2-7 States used for describing profiles of the UWALL, the FLANKS, the VENTPROF, the PPROF, the SPROF. (from Smith 1986)

and is not included in the original Ammon.

Descriptor 36: AH = LP/WH where LP is the whorl overlap and WH is the outer whorl height (Fig. 2-4). This descriptor seems important in interpreting the covariation between W and U (see section 4.2.4).

Descriptor 37 (APPROX): Those primary variables that are approximate are listed in descriptor 30 so that they can be eliminated from subsequent analyses when necessary.

Qualitative morphology. -- Descriptor 38 (VOLUTION) : In the descriptive terms illustrated in Fig. 2-5, the umbilical seam of the whorl progresses from barely touching the preceding whorl (advolute) to overlapping the upper (evolute), middle (midvolute) and lower (involute) third of the flank of the preceding whorl. when the shell is occluded, the outer whorl completely covers the inner whorls. States 6 to 8 describe the situation when growth deviates from a regular, logarithmically accreting planispire. Some ammonoids, particularly in the Paleozoic, show an elliptical growth pattern throughout ontogeny. More commonly the mature whorl will egress to the elliptical pattern or even lose contact with the preceding regular whorls altogether and become heteromorphic (state 7). Some ammonoids have a

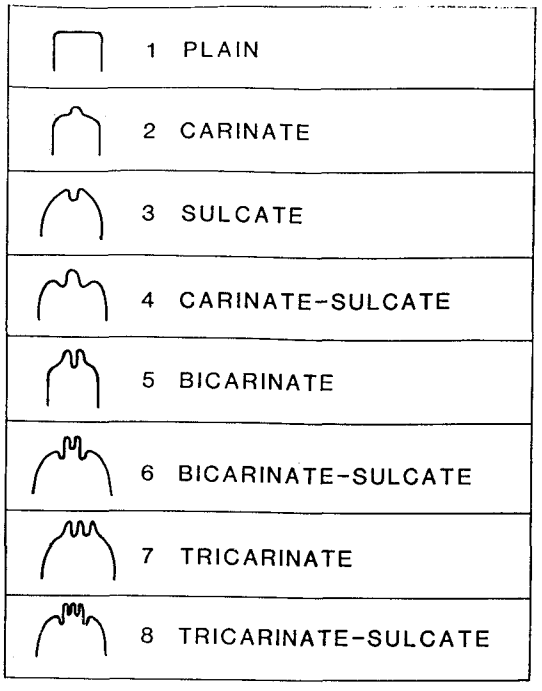


Figure 2-8 States for the descriptor VENTER (from Smith 1986)

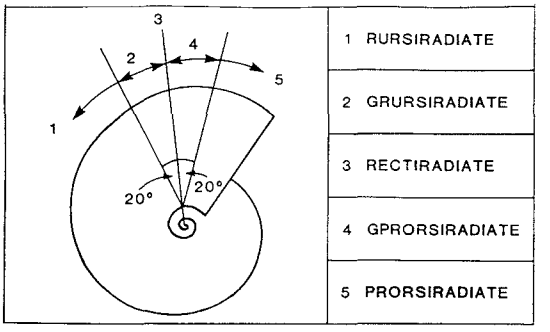


Figure 2-9 States for the descriptors PTREND and CTREND (from Smith 1986)

translational component to their growth and appear gastropod-like (state 8). States 7 and 8 never occur in Lower Jurassic ammonites.

Descriptor 39 (WHORL_SHAPE): see Fig. 2-6.

Descriptor 40 (EXPANSION) describes how fast the shell expands in the following terms: slow, moderate and rapid.

Descriptor 41 (UWALL) describes the profile of the umbilical wall using the terms shown in Fig. 2-7.

Descriptor 42 (UWALLHT) is the height of the umbilical wall (low; high).

Descriptor 43 (UWALLANG) describes the slope of a flat umbilical wall in the following terms: undercut, vertical, steep and shallow.

Descriptor 44 (USHOULD) is the profile of the umbilical shoulder (rounded; angular).

Descriptor 45 (FLANKS) describes the profile of the whorl flanks in the terminology of Fig. 2-7.

Descriptor 46 (VENTER) describes the appearance of the venter as shown in Fig. 2-8.

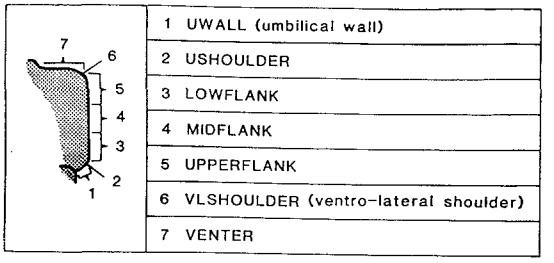


Figure 2-10 States for describing position on the profile of a whorl. Used in PFORM, FURCPOS, UNITUBPOS and CFORM

Descriptor 47 (VENTPROF) is the profile of the venter in cross-section as shown in Fig. 2-7.

Descriptor 48 (KEEL) refers to carinate forms and describes the prominence (high, low) and profile (flat-topped, rounded, angular or crenulate) of the keel as well as noting the presence of a floor.

Descriptor 49 (SULCI) describes sulci (descriptor 47, states 3, 4, 6, 8) by the terms deep or shallow.

Descriptor 50 (PRIBD) describes the ribbing density by the terms distant or dense.

Descriptor 51 (PTREND) describes the average trend of the primary ribs across the flank of the whorl (Fig. 2-9).

Descriptor 52 (PFORM) : The form of the primary ribbing has been described by a number of terms (e.g., Schlegelmilch, 1976, fig. 7, Arkell, Kummel and Wright, 1957, fig. 132, Smith, 1986, fig. 11). The terms used in the Treatise (Arkell, Kummel and Wright, 1957, fig. 132) are adopted here for simplicity

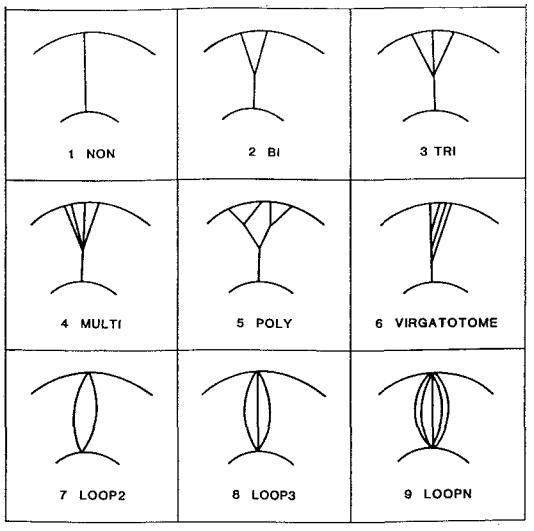


Figure 2-11 States for the descriptor FURC

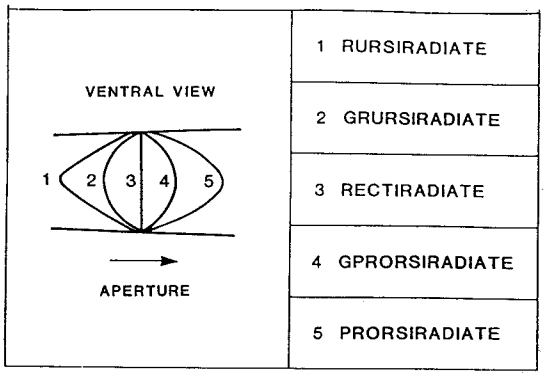


Figure 2-12 States for the descriptor STREND showing the trend of secondary ribs across the venter of a coronate ammonoid (from Smith 1986).

Descriptor 53 (PPROF) indicates the cross-sectional profile of the primary ribs in the terms shown in Fig. 2-7.

Descriptor 54 (FURC) describes the pattern of rib furcation as shown in Fig. 2-11.

Descriptor 55 (FURCPOS) indicates the position of the furcation (descriptor 54, states 2-6; the first furcation is indicated for states 5 and 6) using the terminology of Fig. 2-10. Fasciculation can be described using a combination of descriptors 54 and 55.

Descriptor 56-59 describe characters of the secondary ribbing in the terminology applied to the primary ribs (analogous descriptors 50-53) with the following modifications: descriptor 57 (STREND) is shown in Fig. 2-12 and descriptor 58 (SFORM) are prefixed with an I when the ribs are intercalated.

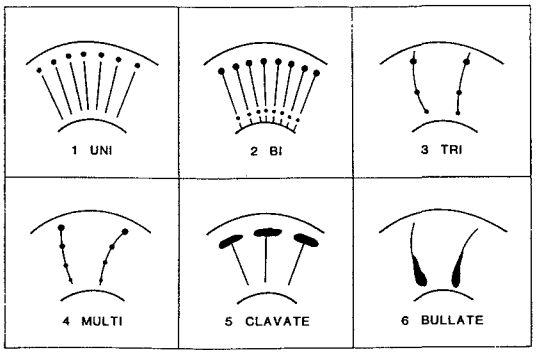


Figure 2-13 States for the descriptor TUBERC (from Smith 1986)

Descriptor 60 (TUBERC) and 61 (UNITUBPOS) describe the tuberculation (Fig. 2-13) and the position of the tubercles in unituberculate forms in the terminology of Fig. 2-10.

Descriptors 62-64 describe the density, trend and form of constrictions in the same terminology applied to primary ribs (descriptors 50-52).

Descriptor 65 (APERTURE) describes the aperture using the states illustrated in Fig. 2-14.

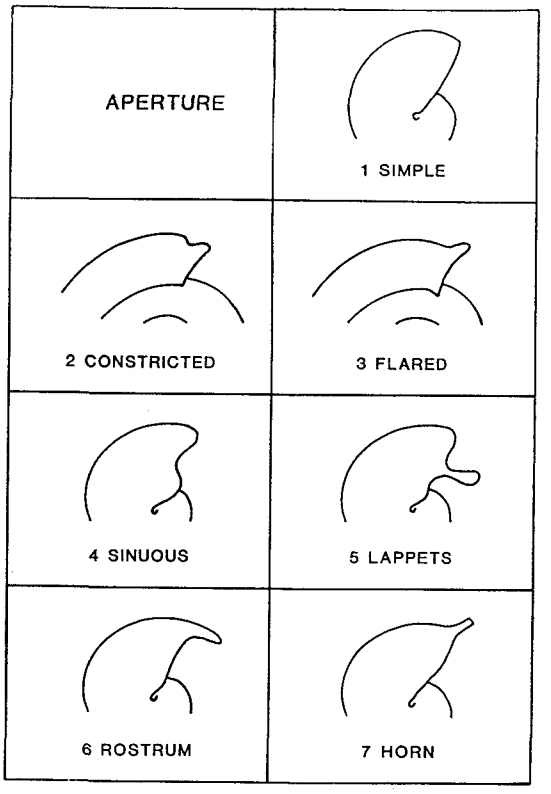


Figure 2-14 States for the descriptor APERTURE (from Smith 1986)

Descriptor 66 (SUTURE) is a field for comments on features of the suture.

Descriptor 67 (ONTOGENY) indicates whether there is a distinct change in morphology during ontogeny (yes or no). This descriptor is used to tell the user to scan the database at a variety of shell diameters to look for multiple entries of data for the same specimen at different diameter (each entry will have the same specimen number (descriptor 98)) and to check the general remarks.

Stratigraphy. -- The descriptors for stratigraphy are largely self explanatory.

Descriptors 70 (EURZONE) and 71 (EURSUBZONE) are standard zones and subzones of the Northwest European Province (Dean, Donovan and Howarth, 1961).

Descriptors 72-74 record a zonation other than that of northwest Europe.

Descriptor 78-79: For ammonite localities from measured sections, the point of reference such as the contact between two formations is listed in descriptor 78 with measurements in meters above or below (negative) that datum shown in descriptor 79. This information permits the rapid compilation of the stratigraphic range of a species in any section.

Descriptor 81 (ASSOCSPEC) : The number of specimens of this species at this locality that are not entered separately into the database.

Locality and catalogue information.-- Descriptor 82 (area) divides the world into a few broad geographic regions such as the Western Pacific, Western Tethyan, North America, South America, Northwest Europe. Descriptor 84 (PROVINCE): Province, state, or county. Descriptors 85-87 give the geographic location of the specimen in terms of longitude and latitude with descriptor 87 giving the position relative to the equator (N or S) and the Greenwich meridian (E or W).

Descriptors 88-91 are the section name, section number, locality number (may include letters) and any other number, e.g., a field number that is associated with the specimen apart from its official catalogue number (descriptor 98).

Descriptor 92 (SUBLOCNO) and 93 (SUPERLOCNO) record the subjacent and superjacent localities in a measured section so that a worker can retrieve information in stratigraphic order.

Descriptor 95 (TYPE) has the following states: holotype, paratype, syntype, lectotype, paralectotype, neoholotype, newparatype, topotype, hypotype.

Descriptor 96 (COLLECTORYR) lists the collector and year of collection to facilitate retrieval of information from field notebooks.

Descriptor 97 (GENERALOC) gives non-standardized locality information, e.g., Graham Island, Silvies River, etc.

Descriptor 98 (SPECNO) is the unique label that characterizes this specimen.

Descriptors 99 (IMAGE) and 100 (SCALE) give the relative path of image files in the file system and the scale of the illustration. The absolute path of the image is determined by combining the relative path with a base directory checked at run time which enables the user to move the image file to the desired location without modifying the user interface described in Chapter 6.

Descriptor 101 is added by the author for the following reason: Some taxonomic groups have unique feature that is important for the identification of the group yet does not justify a separate descriptor in the database, for example, *Hildoceras* and *Hildaites* have strong median lateral groove; *Amaltheus*, *Aegoceras*, and *Pleuroceras* have serrated keel; they are lumped into SPECFEATURE which can be scanned for a specific feature when needed.

Descriptor 102 (REMARKS) is a free format text field, remarks that does not fit into any of the descriptors can be put here.

2.3 SQL -- The Query Language

The Oracle relational database management system allows the operator to use the industry standard language SQL to store, retrieve and manipulate data. SQL is an English-

like language consisting of several "layers" of increasing complexity and capability. End users with little or no experience with databases can learn SQL's basic features very quickly, yet SQL provides information professionals with the powerful and complete set of facilities required (Anonymous 1990a).

2.3.1 Data entry

The user adds new specimens to Ammon by issuing the INSERT command, for example,

INSERT INTO Ammon (genus, species, zone, specno, d)
VALUES ("Badouxia", "canadensis", "Bucklandi", "C-201453", 4.2);

inserts a new specimen of *Badouxia canadensis* with a specimen number of C-201453 from Bucklandi zone. If more than one specimen need to be entered, a more efficient way is to use one's favorite word processing program or text editor and save the data into an ASCII file, then use SQL*Loader to load data in external files into Ammon (see Anonymous 1990b for detail). In fact, most of the data in Ammon are entered using SQL*Loader.

2.3.2 Updating the database

The **UPDATE** command changes values stored in Ammon. Descriptors such as WWWH, U, WWD in Ammon are represented as ratios but some ammonite workers prefer to present these parameters as percentages. This can be achieved by issuing an **UPDATE** command:

UPDATE Ammon
SET wwwh = wwwh∗100, u = u∗100, wwd = wwd∗100

which converts WWWH, U, WWD of all specimens in Ammon into percentage values. A **WHERE** clause can be used to specify specimens that need to be updated. The image measurement module described in Chapter 3 can also be used to update morphological data in Ammon. The **DELETE** command removes specimens from Ammon.

2.3.3 Querying the database

Retrieving data from the database is the most common SQL operation. A database retrieval is called a *query* and to issue a query the operator uses the **SELECT** command. The basic **SELECT** command has three parts:

SELECT some descriptors
FROM a table (such as Ammon) or some tables
WHERE search condition

A question like "what species are involute, have a carinate venter or more than 30 primary ribs per half whorl" is addressed by

```
SELECT    synonymy, synspecies
FROM      Ammon
WHERE     volution='involute' AND (venter='carinate' OR prhw>30);
```

The user can also select items that do *not* meet search conditions. For example, the following query retrieves all species that are not involute:

```
SELECT    synonymy, synspecies
FROM      Ammon
WHERE     volution != 'involute';
```

!= means not equal. The user may combine AND, OR and != in the same query to connect as many search conditions as are necessary to retrieve the data.

The IN operator lets the user select items that contain a value that matches *one* of the values in a list. For example, the following query lists all species with a carinate venter, or a carinate-sulcate venter or sulcate venter:

```
SELECT    synonymy, synspecies
FROM      Ammon
WHERE     venter IN ('carinate','carinate-sulcate','sulcate');
```

Because some species may have many specimens in the database and each specimen is an independent item in the database, there would be lots of duplicate species names in the query result which can be eliminated by specifying DISTINCT in the **SELECT** clause:

```
SELECT    DISTINCT synonymy, synspecies
```

The user can also select items that match a specified *pattern* of characters or numbers. For examples, the user may want to find all species that have "der" in their names:

```
SELECT    DISTINCT synonymy, synspecies
FROM      Ammon
WHERE     synspecies LIKE '%der%';
```

the SQL operator **LIKE** means that species name contains the string "der" and the percent sign (%) signifies any string of zero or more characters. This feature allows multiple states in the same entry. For example, for a specimen that has both bi-furcate and tri-furcate ribs, the value of the descriptor FURC can be input as "bi,tri" which will meet the search condition for specimens with bi-furcate ribs as well as specimens with tri-furcate ribs.

SQL also supports the following arithmetic operators, string manipulation and group functions:

Arithmetic	Operators	String	Function	Group	Function
+	Add	|	Concatenation	AVG	Average
-	Subtract	DECODE	Translate	SUM	Sum
*	Multiply	LENGTH	String Length	MAX	Maximum
/	Divide	SUBSTR	Substring	MIN	Minimum
POWER	Exponentiation	INSTR	Instring	STDDEV	Standard Deviation
ROUND	Rounding	UPPER	Upper Case	VARIANCE	Variance
TRUNC	Truncation	LOWER	Lower Case	COUNT	Number of Items
ABS	Absolute Value	SOUNDEX	Sound Matching		

This is only a partial list, most relevant to Ammon. The SQL Language Reference Manual (Anonymous 1990c) has a complete set of the operators and functions.

Group functions which let the user select summary information from groups of items demonstrate the power of SQL. Suppose, for instance, that the user wants to know the average shell expansion rate (W) and degree of intraspecific variation of W for every species that has at least 6 specimens in the Ammon database, the user can issue the following query:

```
SELECT    synonymy,synspecies,AVG(w),STDDEV(w)
FROM      Ammon
GROUP by  synonymy,synspecies
HAVING    COUNT(*) >= 6;
```

In the example above, every species is a group, average and standard deviation values of W are calculated for each species. Just as specify search conditions for individual items using

the **WHERE** clause, the SQL clause **HAVING** is used to specify search conditions for groups of items.

One of the reasons that SQL is so powerful is that complex queries can be built out of several simple queries. A **WHERE** clause of one query may contain another query called a *subquery*. Suppose that the user wants to retrieve all specimens (specno) collected from the same zone as specimen X with a specimen number of GSC20122 :

```
SELECT   specno
FROM     Ammon
WHERE    zone =
         (SELECT   zone
         FROM      Ammon
         WHERE     specno = 'GSC20122');
```

Subqueries may be just as complex as main queries -- they may contain subqueries of their own.

2.3.4 Changing the structure of the database

The SQL command **ALTER** can be used to add new descriptors to Ammon or make columns of existing descriptors wider to accommodate long text strings.

ALTER TABLE Ammon **ADD** (specfeature CHAR (100));

In this command a new descriptor specfeature is added to Ammon with data type (CHAR) and maximum number of characters in the descriptor (100). Oracle will not waste storage by reserving space for the maximum number of characters allowed for each descriptor. For example, even though the maximum length of a value in descriptor specfeature is 100, most specimens do not have any entry for this descriptor therefore occupy no storage space at all.

SQL is the industry standard relational database language which assures compatibility across different database systems including IBM's DB2 and SQL/DS. In addition, Oracle runs on an impressive range of computers, from PCs to minis to mainframes. All versions of Oracle are identical which enables the application be moved across hardware and software boundaries with virtually no modifications.

More importantly, SQL commands can be embedded in standard programming languages such as C, FORTRAN or Pascal. For example, a graphics user interface has been implemented for Ammon using C, embedded SQL, Sunview and a graphics library called Pixrect. This graphics user interface gives easy access to the database as well as image display and manipulation capabilities (see Chapters 3 and 6 for details).

2.4 Data Sources

The following references were used in compiling the Ammon image database:

Arkell, Kummel and Wright, 1957
Arthur, 1985, 1987
Blasco, 1978
Blasco, Levy and Nullo, 1978
Crickmay, 1925, 1928
Dagis, 1968, 1974, 1976
Dean, Donovan and Howarth, 1961
Erben, 1956
Frebold, 1951, 1958, 1959, 1960, 1964, 1966, 1967, 1970, 1975
Frebold, Mountjoy et al., 1967
Gabb, 1869
Gabilly, 1976a, 1976b
Geczy, 1976
Geyer, 1974, 1979
Guex, 1980, 1981, 1989

Hall, 1987
Hall and Howarth, 1983
Hall and Westerman, 1980
Hillebrandt, 1973, 1981, 1982, 1987, 1989, 1990
Hirano, 1971, 1973
Imlay, 1968, 1981
Jakobs, 1992
Kalacheva, 1980
Mclearn, 1932
Meister, 1986
Novedades, 1988
Obrien, 1985
Palfy, 1991
Poulton, 1987, 1991
Poulton and Tipper, 1991
Quinzio, 1987
Repin 1974, 1968
Riccardi and Westermann, 1991a, 1991b
Sanborn, 1960
Schlatter, 1991
Schlegelmilch, 1976
Smith, 1976, 1981
Smith and Tipper, 1986, 1988
Stankevich, 1964
Taylor, 1988
Thompson and Smith, 1988, 1992
Tilmann, 1917
Tipper and Richards, 1976
Tipper, Smith et al., 1991
Tozer, 1982
Wang and Smith, 1986
Westermann, 1972
Westermann and Riccardi, 1972, 1979
White, 1889
Whiteaves, 1884

Wiedenmayer, 1977, 1980
Willard, 1963

The prototype Ammon had information on about 1400 specimens; the present greatly expanded version has 7790 specimens from over 25 countries. The number of specimens sampled from each country is not uniform; the vast majority are from the following countries in descreasing order of abundance: Canada, Russia, United States, Switzerland, Argentina, France, Chile, Germany, Britain, Hungary, Austria, Mexico and Japan.

Chapter 3 Image Analytic Techniques

4.1 Introduction

A number of interesting hypotheses in paleobiology can only be addressed using quantitative morphological data. Claim for greater early disparity, for example, cannot be confidently established until we develop quantitative techniques for the characterization of morphospace and its differential filling through time (Gould 1991). Subtle links between morphological features can hardly be recognized without the power of the computer to isolate and quantify all aspects of variation and covariation (Smith 1986). Some ammonite morphological parameters such as shell expansion rate and volution show continuous spectra, quantification of which is needed to differentiate intraspecific variations from interspecific variations. Although quantification is highly desired, deriving data from actual specimens is time consuming. Even for qualitative morphological characters, typing numerous character states into the computer is a slow and tedious process. The following computer image analytic techniques are used to alleviate the problem.

4.2 Digitization

Illustrations of selected ammonites from published literature are scanned into a Sun SparcStation using a MicroTek image scanner with 256 gray shades. The highest resolution available from the scanner is 400 dpi (dots per inch), but after some experiments, it is found that 300 dpi resolution gives the best results for illustrations on good quality paper and 200 dpi resolution is enough for illustrations on old and poor quality paper. To make images displayed on screen (with 100 dpi resolution) the same size as their original forms on paper, a program was written to shrink scanned image to one half original size (for illustrations scanned with 200 dpi resolution), or one third of original size (for illustrations scanned with 300 dpi resolution) by averaging adjacent pixels. This greatly reduces storage requirements since a half sized image only occupies one fourth of the original disk space.

Fossil plates in journal articles normally have multiple specimens figured in each plate. To speed up the digitization process, a whole plate is scanned into the computer and the program "imageEdit" (Appendix 3) is used to pick up individual specimens, clear irrelevant components and save the selected part of the image as a separate file.

An image is saved as a Sun raster file which is named after the author, year and abbreviation of the publication followed by plate and figure number. For example, plate 1, figure 1 of Hall 1987 (Canadian Journal of Earth Science) is named as: hall87cjes1-1.ras (lateral view), hall87cjes1-1.1.ras (ventral view), the corresponding descriptor IMAGE in the Ammon is coded as hall87cjes1-1|1.1. "|" is a separator which indicates the start of another image with the same base name as the preceding one.

3.3 Interactive Image Measurement Module

A front-end user interface is built on top of the Ammon database to facilitate measurement of morphological characters from images of specimens. The interface is implemented in C, embedded SQL, Sunview and a graphics library called Pixrect (see Appendix 4). The basic idea of this module is to derive morphological characters by combining the human ability to visually locate features with the computer ability to memorize and compute. When a specimen is entered into the database for the first time, morphological data are often not available and have to be measured from images of the specimen.

Suppose a specimen with a SPECNO of 43252a is entered into the Ammon without morphological data. Later it is called to screen (Fig. 3-1, see Chapter 6 for detail of query by user interface) to calculate the shell expansion rate (W) and the umbilical ratio (U), a procedure requiring four control points: the coiling axis (O), two points (P_3, P_4) on the outer coiling curve and one point on the inner coiling curve (P_2), another point (P_1) is given to enable simulation of the inner whorl spiral and to help in locating the coiling axis more precisely since fitting the inner curve relies even more heavily on correct location of the coiling axis. P_1 and P_2 are separated by one whorl. By definition (Fig. 3-1)

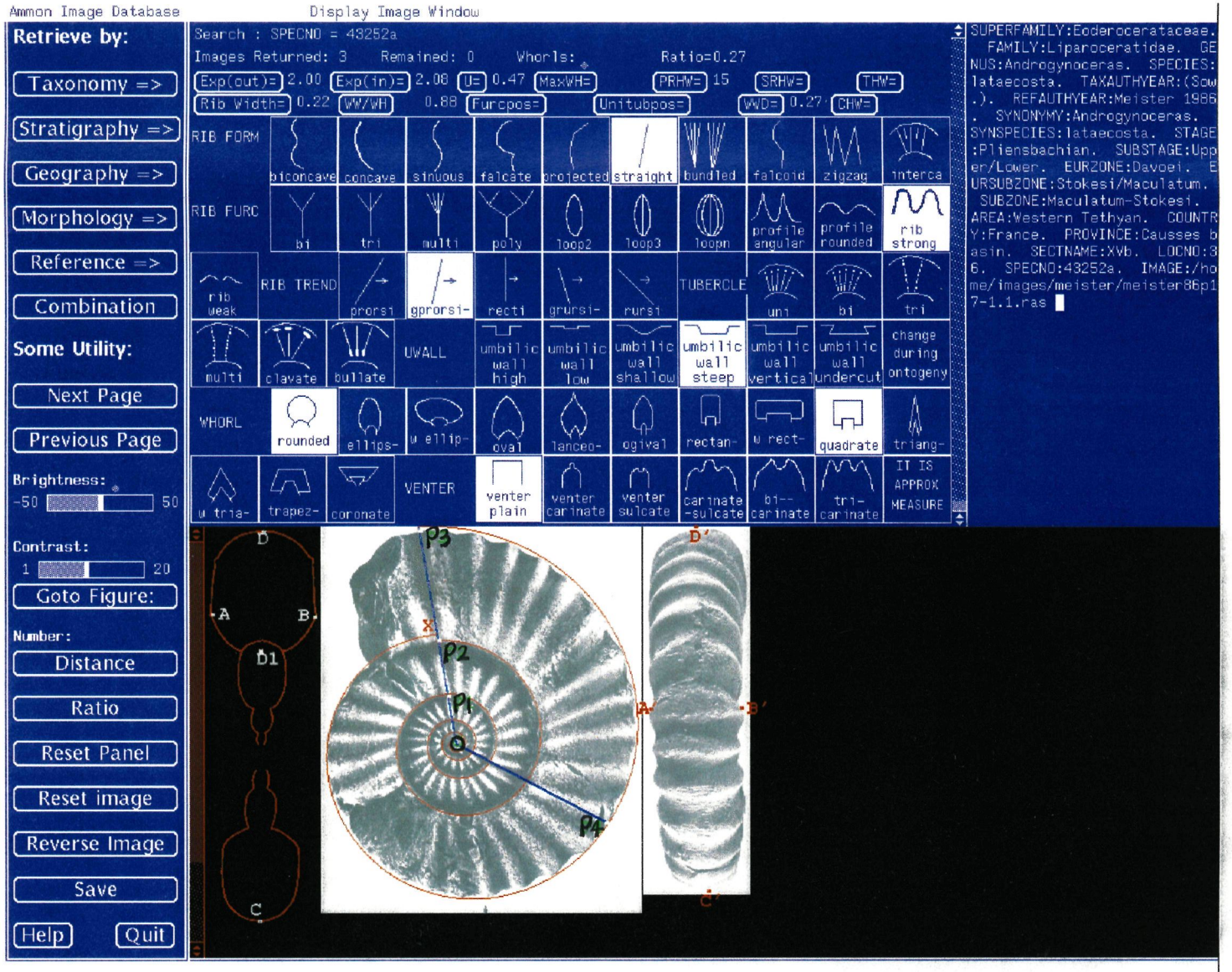


Figure 3-1 Interactive image measurement program

$$W = (OP_3/OP_4)^{2\pi/r}$$

where OP_3 and OP_4 are separated by an angular distance of r radians which must be less than π to make coiling direction unequivocal to the program.

$$U = OP_2/OP_3$$

For rare ammonites that have an undercut umbilical wall, the umbilical shoulder rather than the umbilical seam is taken as the edge of the umbilicus (Smith 1986). An equivalent measurement of W for inner whorl is

$$WI = OP_2/OP_1$$

where OP_2 and OP_1 are separated by one whorl. WI can be used to simulate the inner whorl spiral. The diameter of the shell and the whorl height at point P_3 are

$$D = OP_3 + OP_3/\sqrt{W}, \qquad WH = P_2P_3$$

The absolute values for D and WH in centimeters is computed by combining the image scale recorded in the Ammon and monitor resolution coded in the program. The whorl overlap ratios (Fig. 2-4) with respect to the inner whorl and the outer whorl are (Fig. 3-1):

$$VOLUTION0 = P_2X/P_1P_2, \quad AH = P_2X/P_2P_3$$

To calculate the above parameters, all the user needs to do is to mark the five control points with the middle mouse button then press one of the "W(out)=", the "W(in)=" or the "U=" command buttons in the top window, the program will superimpose the simulated inner and outer coiling curves on the image and show the parameter values in the text fields following the command buttons. If a specimen has parts broken or damaged and the control points cannot be located with confidence, the user can proceed with "try and see". If the fitting is not satisfactory the operation can be undone by pressing the "Reset image" button in the left window. After the locations of the control points have been adjusted the user can try fitting again. Once the location of the coiling axis is settled, the primary rib density (PRHW) can be counted by clicking on a number of consecutive ribs with the middle mouse button. Suppose that N ribs are marked then

$$PRHW = N*\pi/r$$

where r is the angular distance between the first and last ribs which must be less than π to avoid ambiguity about coiling direction yet large enough to justify extrapolation of the number of ribs for half whorl (PRHW). This can be a handy tool for exploring primary rib density change during ontogeny; the user simply clicks on a few ribs and the program does the counting and extrapolation. Fig. 3-1 shows that the PRHW for this specimen gradually increases from 10 in the innermost whorl to 12 in the middle whorl to 15 in the last whorl. The secondary rib density (SRHW) if applicable can be determined in a similar way.

Another group of parameters which are related to whorl width can be measured from a cross-section (Fig. 3-1, left):

WW = AB = 2.5 cm, WWWH = AB/DD1 = 0.88, WWD = AB/CD = 0.27

To measure WW the user clicks on points A and B then press the "Distance" button in the left window; to measure WWWH and WWD the user clicks on the four control points A through D and then press the "Ratio" button.

For most specimens, cross-sections are not available so the above parameters have to be derived from ventral view images. Since the ventral view image is subject to photographic distortion at both ends, the two control points for whorl width (A' and B' in Fig. 3-1, right) are located in the center of the view and line A'B' is perpendicular to the plane of bilateral symmetry. To determine the shell diameter and whorl height of the specimen at this locality: first, find a point on the lateral view where the shell diameter is equal to C'D', suppose that the point is P_3; second, find a point Y on the line OP_3 so that P_3Y is equal to the distance between point D' and line A'B'; finally draw a line from Y which is perpendicular to OP3 and the line intercepts the ventral edge at two points, the point which is closer to P3 along the venter is the projection of A' and B' where the shell diameter and whorl height can be measured appropriate for the whorl width A'B'. The above process is done by the Interactive Image Measurement program, the user only needs to click on the four control points (A', B', C', D') then press the "WWD=" button in the top window. The text fields show the same results as given by the cross section : WWD = 0.27 and WWWH = 0.88. To

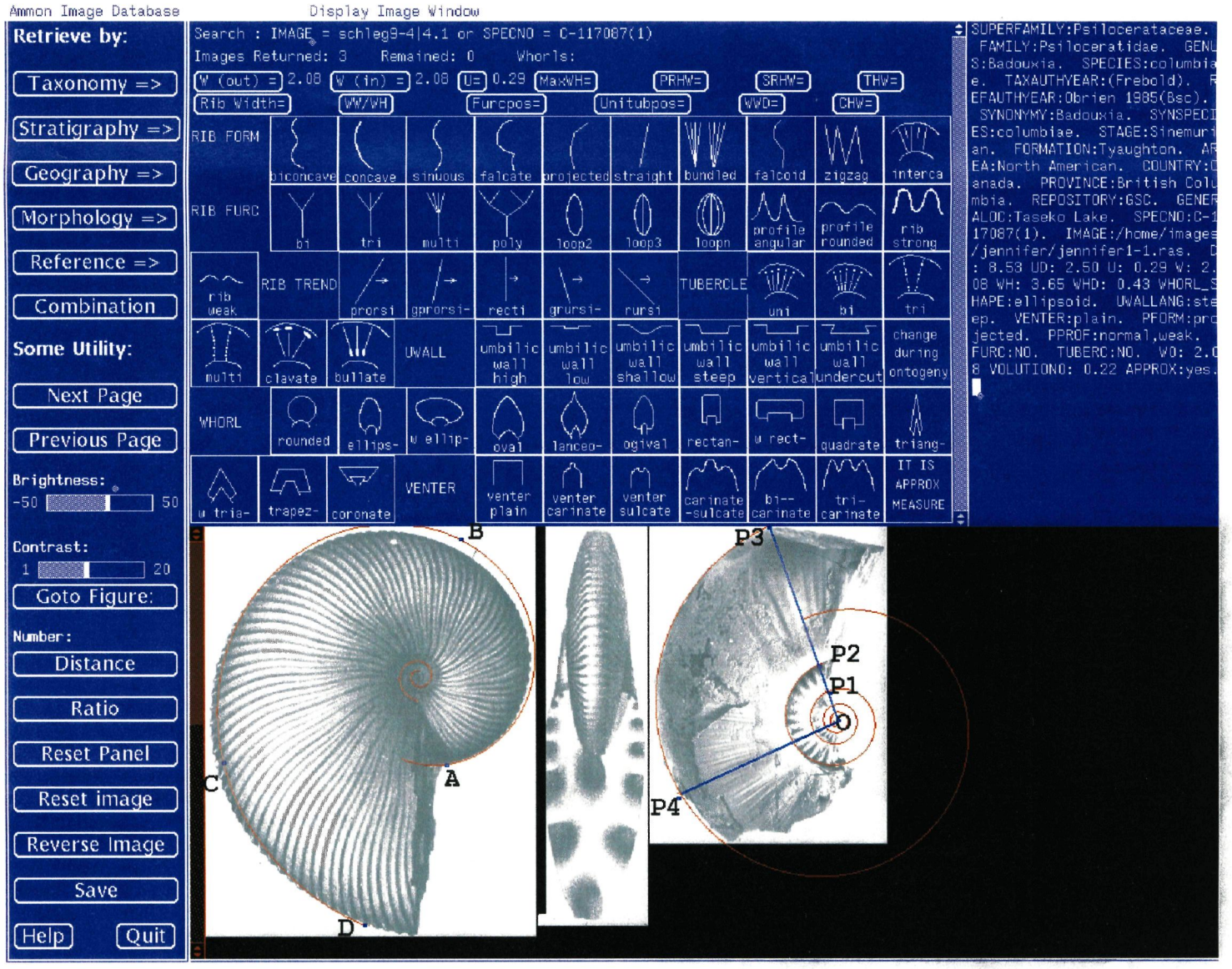


Figure 3-2 Coiling curve simulation and shell reconstruction

evaluate the effect of possible photographic distortion on the WWD and the WWWH values, the program has been used to process 46 specimens with both cross-sections and ventral views. The average difference between the two sources is 4.3% of the measurements from cross-sections and the differences are usually well within 10% of the measurements therefore the alternative method to derive the WWD and the WWWH from ventral views are considered reliable.

The icon panel in Fig. 3-1 is designed to facilitate the input of qualitative morphological data into Ammon. It is evident from the images of the specimen that it has strong, straight and gently prorsiradiate ribs; a steep umbilical wall; a quadrate to rounded whorl section (the Ammon database allows multiple states for any character of a specimen) and a plain venter. The user clicks on the corresponding icons which the program highlights. Finally press the "Save" button in the left window and all qualitative, as well as quantitative, morphological data are put into the database without the user typing a single word or having to memorize any character states.

The interactive measurement module can also be used to detect subtle changes of parameters during growth, or to reconstruct a shell from a fragment. Fig. 3-2 (left) shows that the shell expansion rate from A to B is below the average of the entire whorl since the coiling curve of the actual shell lags behind the hypothetical curve. From B to C, the shell expands rapidly to close the gap and surpass the hypothetical curve followed by a slowing down from C to D to complete the cycle. Fig. 3-2 (right) shows reconstruction of a shell from a fragment with five control points : coiling axis (O), P_1 ,P_2 , P_3 and P_4. The vast majority of morphological data of the Ammon are gathered using this interactive image measurement module.

3.4 Automatic Image Measurement

To explore the possibility of a fully automatic approach, a program (Appendix 5) was written to measure shell expansion rate and rib density from lateral view images of specimens without human intervention. Other morphological features such as tuberculation and rib

furcation are ignored to make the task tractable. The morphology of ammonites has a high degree of regularity, e.g. the coiling curve usually follows a logarithmic spiral; ribs and other ornamentation occur periodically following the coiling curve. This regularity is the basis of image interpretation. The program involves six steps:

3.4.1 Preprocessing

Images are cleared of irrelevant components such as text labels and remnants of other images using the program imageEdit. To focus the subsequent analyses on the ammonite in the image, the average (X) and standard deviation (S) of the background brightness intensity are estimated from a three-pixel wide margin of the image (assuming each image has a background margin around it) and the program reassigns brightness intensity value to each pixel in the 256 gray shades image:

$$Z = \begin{cases} 0 & Z < X+S \text{ and } X < 128 \text{ (black background)} \\ 256 & Z > X-S \text{ and } X > 128 \text{ (white background)} \\ Z & \text{Otherwise} \end{cases}$$

where Z is the brightness intensity of the pixel.

3.4.2 Edge detection

The major elements in an ammonite image are coiling curve and ribs, both are edges where brightness shows significant changes. Edge detection serves to simplify the analysis of images by drastically reducing the amount of data to be processed, while at the same time preserving useful structural information about object boundaries. The Canny (from John Canny) operator was chosen for this purpose. Canny based his design on the following criteria (Canny 1986):

1. Good detection: there should be a low probability of failing to mark real edge points and a low probability of falsely marking non-edge points;

2. Good localization: the points marked as edge points by the operator should be as close as possible to the center of the true edge;

3. Only one response to a single edge;

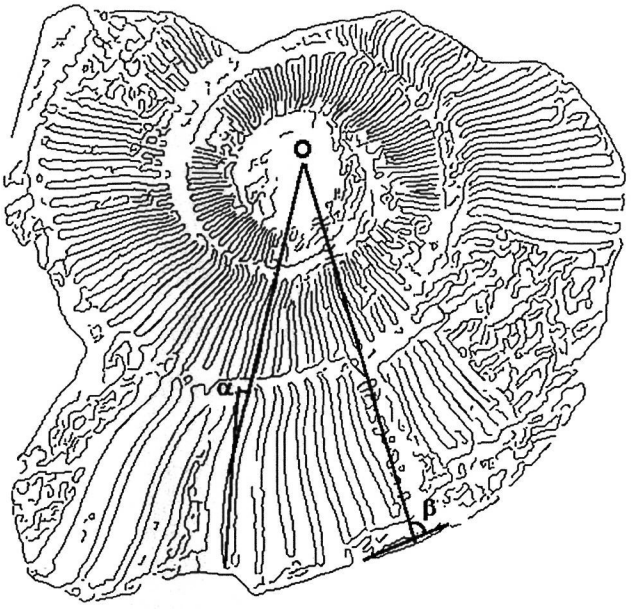


Figure 3-3 An output from Canny operator Locate coiling axis using possible rib and coiling curve segments.

Canny (1986) proved that the operator derived could be approximated by the first derivative of a Gaussian :

G'(x) where $G(x) = \exp(-x^2/2\sigma^2)$

3.4.3 Thinning and line tracing

The output of the Canny operator consists of separate pixels which are then linked into line segments according to 2-D spatial proximity and collinearity. Each line segment has three parts : orientation, length in terms of number of edge points in the segment and coordinates of these points. Before line tracing, a thinning algorithm (Pavlidis 1982) is used to skeletonize edge segments potentially wider than one pixel into one-pixel wide segments.

3.4.4 Locating coiling axis

The most important feature of ammonite morphology is that ribs distribute around coiling axis radially and coiling curves follow a logarithmic spiral starting from coiling axis. Assume β is the angle made by the radius vector to any tangent to the spiral which describes how rapidly coiling curves move away from coiling axis. Lower Jurassic ammonites have β values between 75° and 90° even when there are changes during ontogeny. A statistical parameter (SC) based on this observation is used to locate the coiling axis :

$$SC = \sum_{i=1}^{m} CoilingLen[i] + \sum_{i=1}^{n} RibLen[i]$$

where *CoilingLen[i]* is the length of the *ith* line segment which has an β value between 75° and 90° with respect to the hypothetical coiling axis O (Fig. 3-3); *RibLen[i]* is the length of

the *ith* line segment with $\alpha < 15°$ so the line segment is in a radial position to the hypothetical coiling axis, α is calculated from the middle point of the segment (Fig. 3-3). The SC value provides a measurement of support that the hypothetical coiling axis gets from all line segments in the image using the underlying geometric pattern.

Coiling axis is practically always located within a central region in the image which is half of the image size in terms of width and height. The program samples this region from left to right, top to bottom computing the SC values with an increment of 4 pixels which corresponds to 1 mm if the image scale is 1 and the monitor resolution is 100 dpi. The pixel which has the highest SC value is considered to be the location of the coiling axis. This algorithm gives very robust results.

3.4.5 Classification of coiling curves and ribs

This is the most challenging part of the program since coiling curves and ribs often are broken by various kinds of noise and there may be lots of line segments resulting from local brightness changes which are not related to coiling curves and ribs (Fig. 3-3). Again the underlying geometric constraints are used to eliminate non-rib and non-coiling-curve :

1. Ribs distribute radially around coiling axis;
2. Coiling curves are nearly normal to radius vectors ($\beta > 75°$);
3. The distance between adjacent ribs should not change abruptly, otherwise there must be missing ribs or false ribs between them;
4. The length of each rib should be consistent with its neighboring ribs;
5. The "white to black" rib edges should be sandwiched by two "black to white" rib edges and vice versa;
6. Empirically, the range of shell expansion rate (W) for Lower Jurassic ammonites is between 1.2 and 5, possible coiling curve segments and geometric relations among them should meet this constraint;

Generally speaking, statistical data on geometric features are more robust than individual measurements. The image is therefore divided into 12 equiangular sectors around

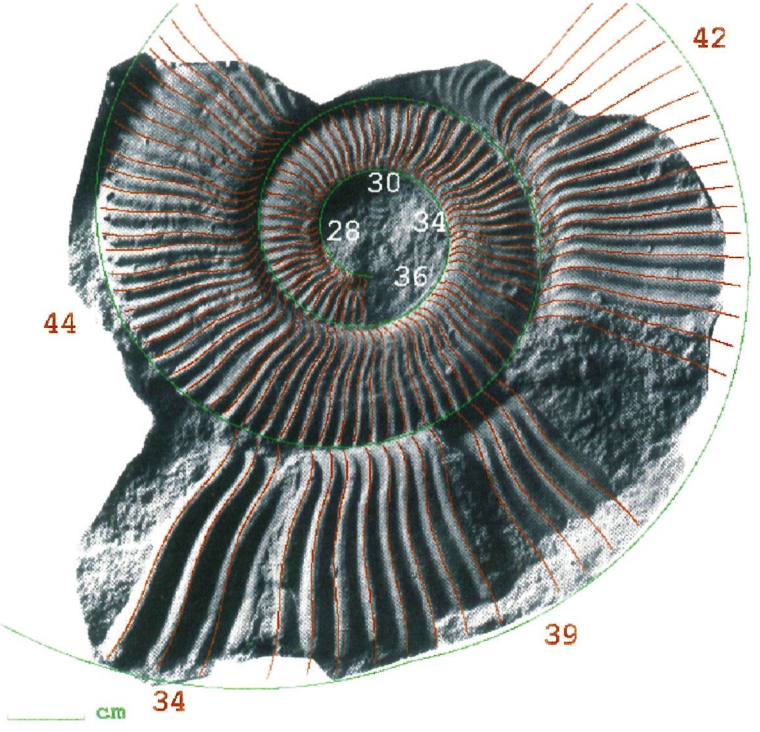


Figure 3-$4 Automatic image measurement
Computer classified and simulated ribs
and coiling curve on original image

the coiling axis. Rib and coiling curve segments are verified in each sector and then rechecked during the final assembling stage according to the above constraints.

3.4.6 Deriving morphological parameters

Once the coiling axis, coiling curves and ribs are determined, shell expansion rate (W), rib density (PRHW), volution (VOLUTION0), AH and their changes during ontogeny can easily be computed. Nonlinear regression can be used to formulate rib forms.

Fig. 3-4 shows ribs and coiling curves simulated by the program superimposed on the original image; ribs were matched using the least squares technique. This program can be a handy tool to explore ontogenetic changes. A user may determine the shell expansion rate between any two points by clicking on them with the mouse. Similarly, a user may determine the rib density between any two points by clicking on them; the program will display how many ribs between the two points and how many ribs there would be for half a whorl. By working from the inner to the outer whorls, the user can get a clear picture of how the shell expansion rate and rib density change during growth. In this example (Fig. 3-4) the rib density gradually increases from 28 -> 30 ->34 -> 36 -> 44 from the visible innermost whorl to the outer whorl then start to decrease to 42->39->34 in the last half whorl. Unfortunately the accuracy demonstrated in this example cannot be achieved in most cases. It is very difficult to classify all ribs and coiling curve segments correctly and locate coiling curve accurately.

3.5 Elliptic Fourier Analysis of Whorl Shape

Whorl shape is an important aspect of ammonite shell morphology. A simple descriptor in the Ammon database is the ratio of whorl width to whorl height (WWWH) which does not define the shape uniquely since a circle and a square would give the same ratio. Another descriptor WHORL_SHAPE uses descriptive terminology such as "quadrate", "rounded", "subrounded" etc, but workers do not necessarily agree on the exact meaning of a given descriptive phrase such as "subrounded" and it is difficult to compare variations across different taxonomic groups. These considerations create a need for a method like Fourier analysis which has been successfully used in many paleontological studies for the

characterization of closed curves (Kaesler and Waters, 1972; Anstey and Delmet, 1973; Christopher and Waters, 1974; Younker and Ehrlich, 1977; Canfield and Anstey 1981; Foote 1989).

One commonly used Fourier method, polar Fourier analysis (Kaesler and Waters, 1972), is limited to simple curves without multiple reentrants and is therefore not considered here. The more sophisticated elliptic Fourier analysis (Kuhl and Giardina, 1982) and perimeter-based Fourier analysis (Foote 1989) were compared in a preliminary study. Experiments with whorl shape data show that the elliptical Fourier analysis gives better results, and there is also the advantage that there is no need to determine an artificial centroid for the shape. The Elliptical Fourier analysis is therefore used in the following study.

Whorl shape outline is approximated by a continuous sequence of digitized points in an X-Y coordinate system. The outline can be considered to be generated by a point moving around the boundary. Suppose l is the arc length along the curve, l_p is the arc length at point p; L is the total perimeter length and K is the total number of points on the curve. The elliptical Fourier series are (Kuhl and Giardina, 1982):

$$x(l) = A_0 + \sum_{n=1}^{\infty}(a_n \cos\frac{2n\pi l}{L} + b_n \sin\frac{2n\pi l}{L})$$

$$y(l) = C_0 + \sum_{n=1}^{\infty}(c_n \cos\frac{2n\pi l}{L} + d_n \sin\frac{2n\pi l}{L})$$

where a_n and b_n are the nth harmonic:

$$a_n = \frac{L}{2n^2\pi^2}\sum_{p=1}^{K}\frac{\Delta x_p}{\Delta l_p}[\cos\frac{2n\pi l_p}{L} - \cos\frac{2n\pi l_{p-1}}{L}]$$

$$b_n = \frac{L}{2n^2\pi^2}\sum_{p=1}^{K}\frac{\Delta x_p}{\Delta l_p}[\sin\frac{2n\pi l_p}{L} - \sin\frac{2n\pi l_{p-1}}{L}]$$

$$c_n = \frac{L}{2n^2\pi^2}\sum_{p=1}^{K}\frac{\Delta y_p}{\Delta l_p}[\cos\frac{2n\pi l_p}{L} - \cos\frac{2n\pi l_{p-1}}{L}]$$

$$d_n = \frac{L}{2n^2\pi^2}\sum_{p=1}^{K}\frac{\Delta y_p}{\Delta l_p}[\sin\frac{2n\pi l_p}{L} - \sin\frac{2n\pi l_{p-1}}{L}]$$

$$\Delta x_p = x_p - x_{p-1},\ \Delta y_p = y_p - y_{p-1} \qquad p = 1, 2, \ldots K$$

The constant components in these Fourier series are as follows:

$$A_0 = \frac{1}{L}\sum_{p=1}^{K}\frac{\Delta x_p}{2\Delta l_p}(l_p^2 - l_{p-1}^2) + \xi_p(l_p - l_{p-1})$$

$$C_0 = \frac{1}{L}\sum_{p=1}^{K}\frac{\Delta y_p}{2\Delta l_p}(l_p^2 - l_{p-1}^2) + \delta_p(l_p - l_{p-1})$$

where

$$\xi_p = \sum_{j=1}^{p-1}\Delta x_j - \frac{\Delta x_p}{\Delta l_p}\sum_{j=1}^{p-1}\Delta l_j$$

$$\delta_p = \sum_{j=1}^{p-1}\Delta y_j - \frac{\Delta y_p}{\Delta l_p}\sum_{j=1}^{p-1}\Delta l_j$$

and

$$\xi_1 = \delta_1 = 0$$

The following normalization procedure results in Fourier descriptors which are invariant with rotation, dilation and translation of the contour and also with the starting point on the contour. The basic idea is to rotate the first harmonic phasor until it is aligned with a semi-major axis of its locus:

$$\begin{bmatrix} a_n^{**} & b_n^{**} \\ c_n^{**} & d_n^{**} \end{bmatrix} = \begin{bmatrix} \cos\psi_1 & \sin\psi_1 \\ -\sin\psi_1 & \cos\psi_1 \end{bmatrix}\begin{bmatrix} a_n & b_n \\ c_n & d_n \end{bmatrix}\begin{bmatrix} \cos n\theta_1 & -\sin n\theta_1 \\ \sin n\theta_1 & \cos n\theta_1 \end{bmatrix}$$

where $\theta_1 = 2\pi\lambda_1/L$ and λ_1 is the displacement of the starting point, ψ_1 is the spatial rotation angle:

$$\theta_1 = \frac{1}{2}\arctan\left[\frac{2(a_1b_1+c_1d_1)}{a_1^2+c_1^2-b_1^2-d_1^2}\right]$$

$$\begin{bmatrix} a_1^* & c_1^* \\ b_1^* & d_1^* \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix}\begin{bmatrix} a_1 & c_1 \\ b_1 & d_1 \end{bmatrix}$$

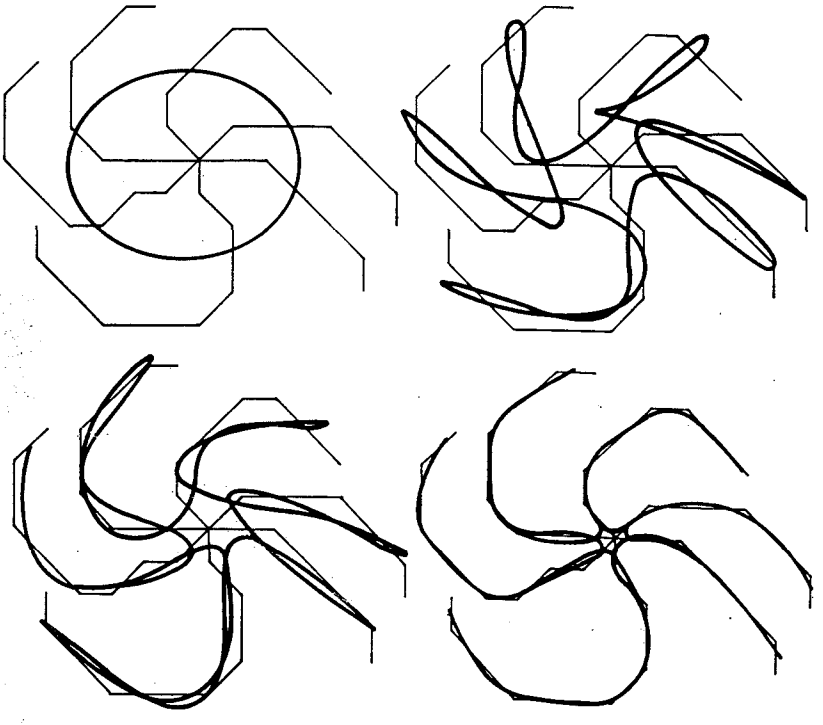


Figure 3-5 1-, 9-, 12-, and 30- harmonic representations of a geometric shape (from Kuhl and Giardina 1982)

The Fourier coefficients can be made independent of size by dividing each of the coefficients by the magnitude of the semimajor axis and independent of translation by ignoring the bias terms A_0 and C_0. The first harmonic content of the size-normalized Fourier series is always characterized by $a_1^{**} = 1.0$, $b_1^{**} = 0.0$, $c_1^{**} = 0.0$, $\left|d_1^{**}\right| < 1.0$. Elliptic Fourier series can be used to represent very irregular closed contours as demonstrated by Kuhl and Giardina (Fig. 3-5).

Whorl shape drawings are scanned into the computer from published work. A thinning algorithm (Pavlidis 1982) is used to skeletonize outline parts potentially wider than one pixel and a contour tracing program (as used in section 3.4) links separate pixels into a continuous outline ready for analysis.

Fig. 3-6 shows the ontogenetic variations of *Arnioceras ceratitoides.* the number of

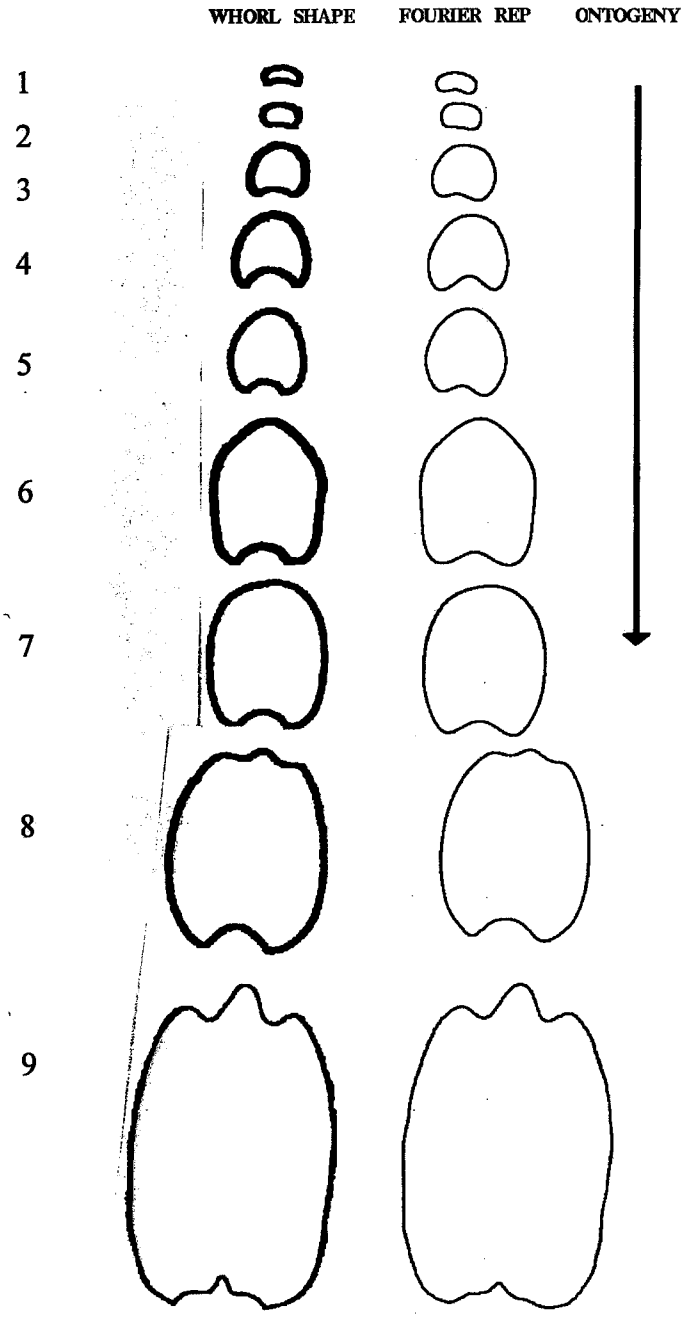


Figure 3-6 Ontogenetic change in whorl shape for *Arnioceras ceratitoides*

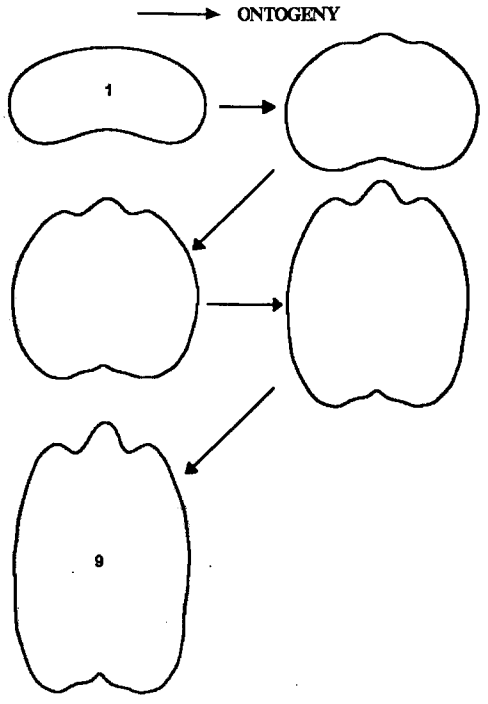


Figure 3-7 Computer generated hypothe--tical sequence from interpolation of whorl 1 and 9 in Fig. 3-6.

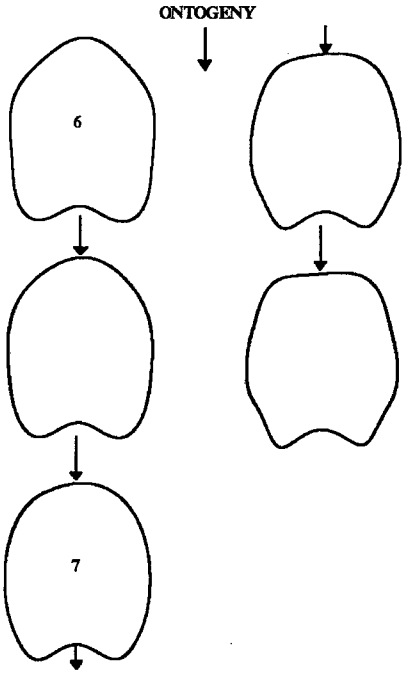


Figure 3-8 Computer generated hypo--thetical sequence from extrapolation of whorl 6 and 7 in Fig. 3-6.

From top to bottom, numbers of harmonic used in the Fourier series are:
4, 3, 4, 5, 6, 6, 8, 15, 23 (root mean square error < 0.01)

harmonics required to reduce root mean square error to 0.01 or less increases with ontogeny. This is consistent with the visual observation of the general increase in whorl shape complexity. The most significant change occurs from whorl 7 to 8 where the number of harmonics increases from 8 to 15 which corresponds to the development of a keel.

One of the principal advantages of Fourier analysis is that it allows the construction of intermediate whorl shapes and extensions, or reductions, of morphologic trends beyond the forms available. If the first whorl in Fig. 3-6 is allowed to evenly evolve (in terms of Fourier coefficients) into the last whorl, it produces a hypothetical sequence of whorl shapes which is different from the real sequence (Fig. 3-7). Fig. 3-8 (right) shows an extrapolation sequence of whorl 6 to 7. A program (Appendix 7) has been written to visualize and animate the detail

of how one whorl shape can gradually evolve into another. All shapes in the sequence are normalized and later whorls are superimposed on the earlier whorls so that subtle change in shape can be detected. These examples demonstrate the potential value of elliptic Fourier analysis in morphologic simulation.

In addition, Fourier analysis can detect and quantify whorl shape variations within and among taxa. Ontogeny can be directly compared with phylogeny to gain insight into heterochrony. It would be interesting to map the whorl shape distribution of naturally occurring ammonites with respect to their Fourier representations so the relative density of occurrence can be evaluated, perhaps shedding light on functional factors governing whorl shape. Finally, current models of 3-D simulation of ammonites shells all assume a circular or ellipsoid whorl section. Fourier representation of whorl sections make it possible to model real ammonites. All these are open areas for future research.

Chapter 4 Morphological Variation and Covariation

4.1 Introduction

The diverse shell morphologies of ammonites have promoted much speculation on relationships between shell form and function, shell geometry and taxon duration. In this chapter the Ammon database is used to analyze Lower Jurassic ammonites in terms of shell morphology, possible correlation between taxon longevity and its morphological variability. In addition to documenting the range and diversity of ammonite morphology, I hope to demonstrate the existence of patterns of shell morphology that are independent of taxonomy at the superfamily level. This concerns possible functional relationships between morphological variables.

It is well known that the information in paleontological databases is incomplete (Raup 1991): only a fraction of the taxa that ever lived have been sampled, new fossil material is being collected all the time and taxa are subject to revision with new discoveries. Despite many potential sources of error, Sepkoski (1993) demonstrated that errors and changes in paleontological databases are not necessarily impediments to statistical analysis so long as the data are sufficiently numerous.

4.2 Lower Jurassic Ammonite Shell Morphology

4.2.1 Background

As is shown in one outstanding paper by Raup (1967), the basic geometry of planispiral ammonites can be described by three parameters: W (whorl expansion rate), U (umbilical ratio), WWWH (shape of the whorl). Plots of W, U and WWWH can be used effectively to map the distribution of basic shell geometry. Raup (1967) showed that ammonoids occupy only a portion of the theoretical range of possibilities for shell geometry, and that some geometric combinations have been utilized more than others. Raup's approach was subsequently used by Ward (1980) to compare the shell geometry of Jurassic and

Cretaceous ammonoids to Mesozoic and Cenozoic nautiloids. Chamberlain (1981) went further and related ammonoid and nautiloid shell geometry to hydrodynamic efficiency by showing that species with coiled shells cluster on adaptive peaks that optimize streamlining and stability. More recently, Saunders and Swan (1984,1987) have used this approach to analyze the morphologic diversity of mid-Carboniferous ammonoids. These studies demonstrated the usefulness of Raup's approach in evaluating and graphically portraying the distribution of ammonoid shell geometry. By design, the studies of Raup, Ward, and Chamberlain were generalized in their approach; this was necessary in order to assemble the amount of data required for the types of analyses undertaken. However, geometric analysis at the taxonomic level of genus and stratigraphic level of period, while acceptable for a preliminary survey, is increasingly regarded as being too coarse-grained to resolve many critical questions. Database Ammon makes it possible to work at the species and zonal levels. As a result of more detailed analysis, some of Raup's conclusions, particularly concerning geometric trends through time and ontogenetic variations are now subject to revision.

4.2.2 The Sample of Observations

The data used in this study comprise observations made on 6784 specimens, representing 15 families, 179 genera, and 1319 species. This comprises virtually complete coverage for the ammonite genera known to occur in the Lower Jurassic as catalogued by Donovan, Callomon, and Howarth (1981). All species are represented in the sample by averaging the measurements made on specimens with shell diameters (D) larger than 4 cm. This reduces the impact of ontogenetic change on the data since most Lower Jurassic ammonite species that undergo such change do so early in ontogeny. Selected analyses at D > 8 cm show no appreciable differences from analyses conducted at D > 4 cm.

Ontogenetic change will be considered as a separate problem with appropriate biometrical data in section 4.5. Variations between each species are also calculated which show that morphological variations between species are much higher than within species, for basic shell geometry even though intraspecific variations are widespread (Table 4-1).

Table 4-1 Between species variance and within species variance

Parameter	Mean	Between Species Variance BSV (Num. of Species)	Within Species Variance WSV (Num. of Species)	F= BSV /WSV	F test α=0.01
U	0.4192	0.015502 (1251)	0.000915 (817)	16.99	1.32
W	2.0231	0.106973 (1248)	0.015257 (670)	7.01	
WWWH	0.8822	0.082893 (1045)	0.009702 (639)	8.53	

4.2.3 Basic Shell Geometry

The W, U, and WWWH data for the whole sample have been plotted as scatter diagrams with one point for each species. The density of points in the scatter has been contoured and the results shown in Fig. 4-1, where successively higher contours indicate increase in points per unit area on the plot. The shell geometry of these ammonites is restricted to the low W and high U region of the morphological spectrum with the greater portion clustering in the vicinity of W=1.9, U=0.45, which coincides with one of the two peaks of hydrodynamic efficiency determined by Chamberlain (1981). Fig. 4-2 shows that WWWH is strongly correlated with W; as W increases, WWWH decreases. Figures 4-3, 4-4, 4-5 show the standard deviation contours of W, U, and WWWH with respect to W and U, which serve as "error bars" for Fig. 4-1 and Fig. 4-2, only species with at least 6 occurrences in the database are used, the number is chosen somewhat arbitrarily, any fewer than this seems to me to be too small a number for calculating standard deviation for a species. The curved line shown in these figures follows the equation W=1/U which marks the point of minimal whorl contact. The density contour and WWWH contour are strikingly similar to those for Mesozoic ammonoids in general being largely concentrated on a small portion of the theoretical range of possibilities for shell geometry (Fig. 4-6) and suggesting the same geometrical restraints. For example, high values of U tend to be associated with low W and high WWWH, which forms an important part of Buckman's Law of Covariation.

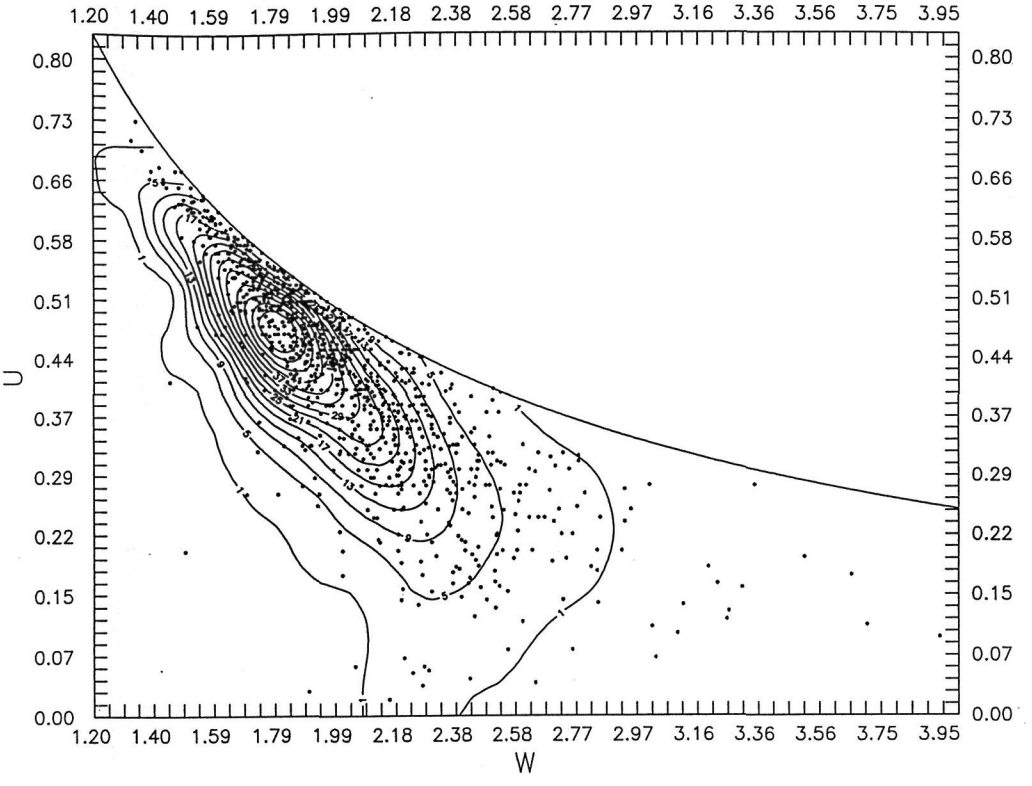


Figure 4-1 Density diagram with respect to W and U (whole sample)

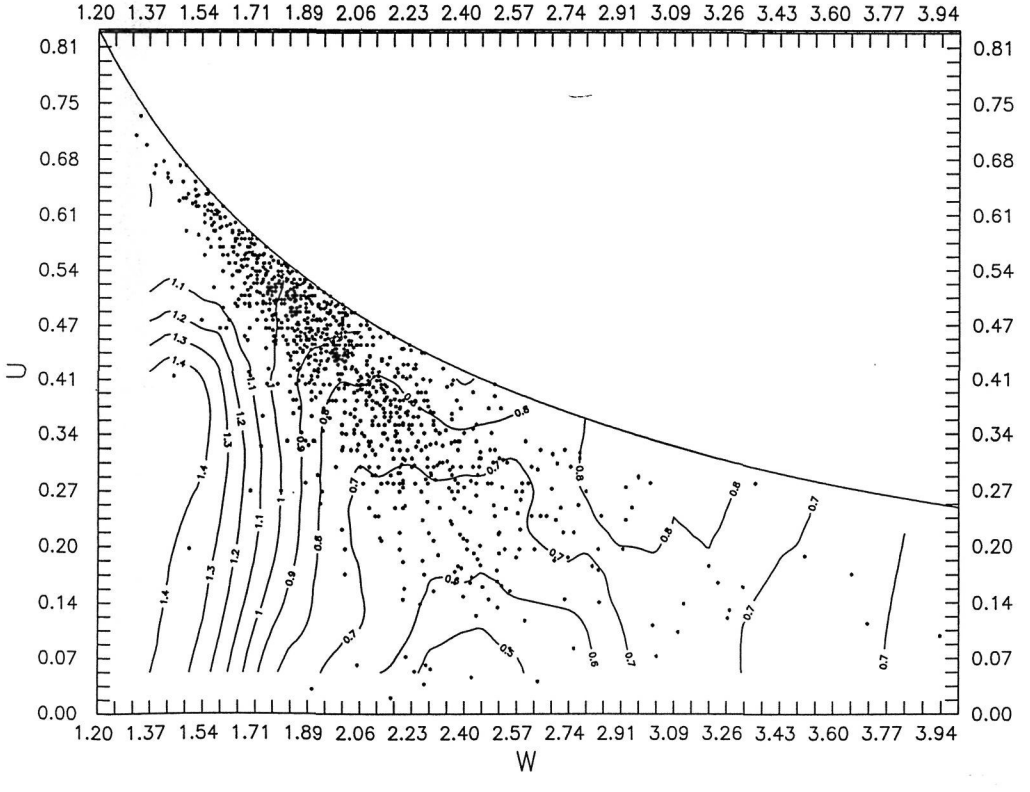


Figure 4-2 WWWH contour with respect to W and U (whole sample)

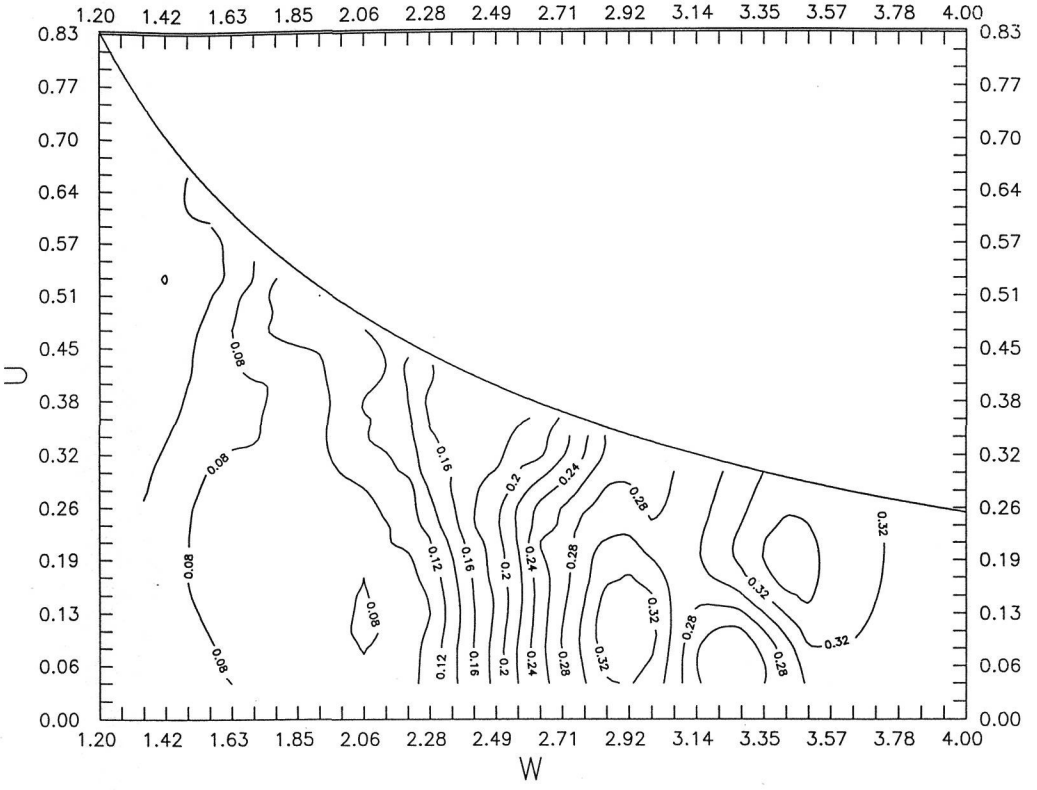


Figure 4-3 Standard deviation contours of W with respect to W and U (964 species)

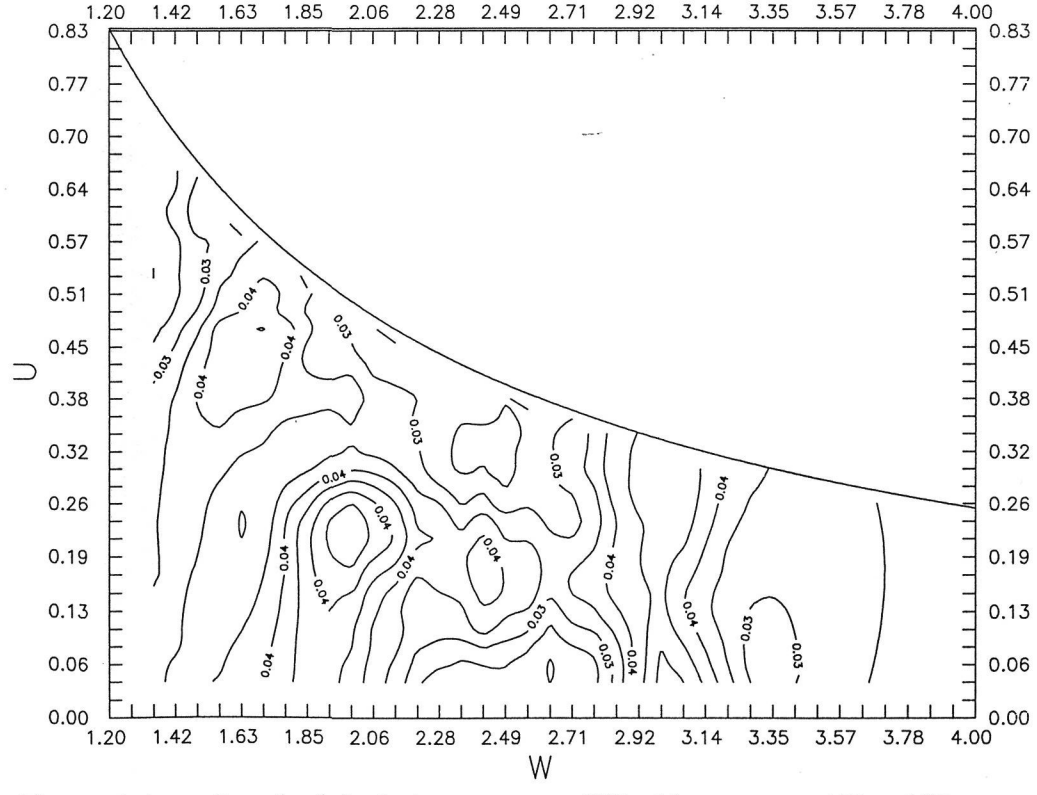


Figure 4-4 Standard deviation contour of U with respect to W and U (964 species)

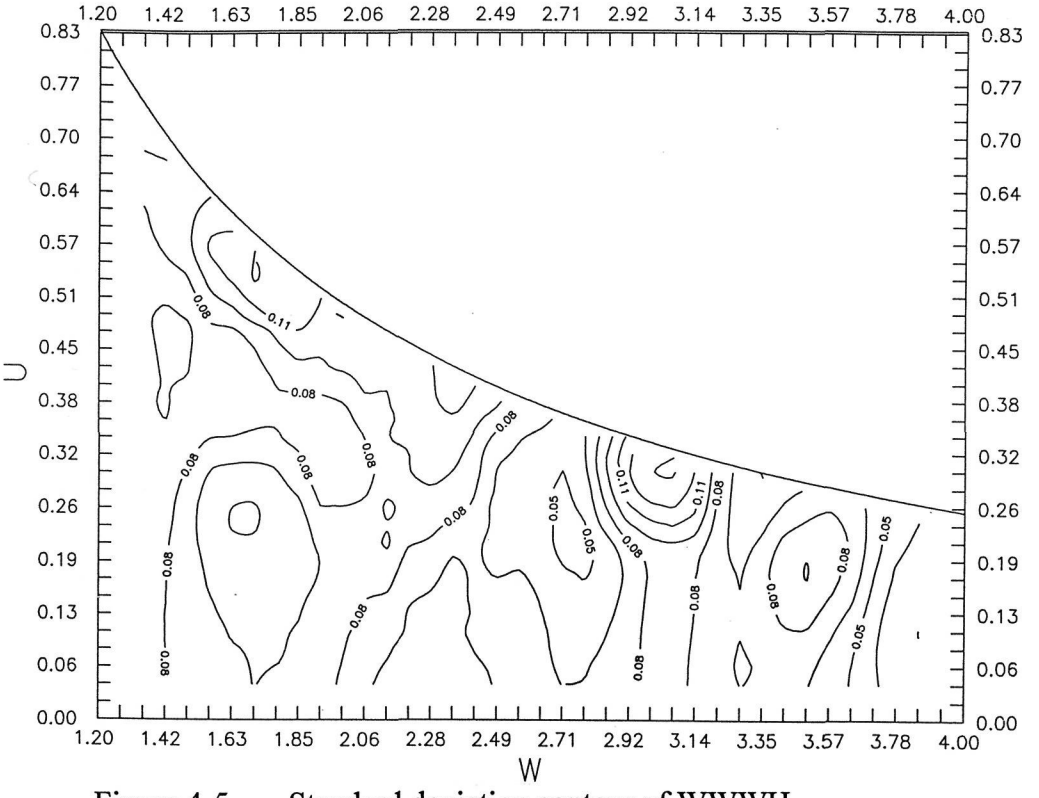


Figure 4-5 Standard deviation contour of WWWH with respect to W and U (780 species)

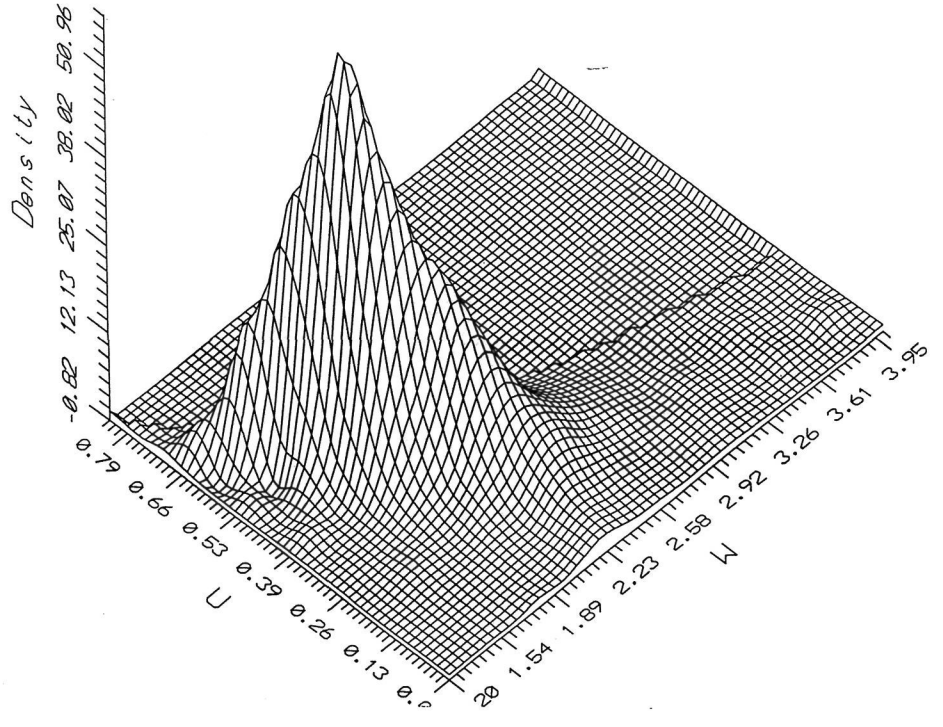


Figure 4-6 W and U density topography (whole sample)

4.2.4 An interpretation of Buckman's Law of Covariation

Swan and Saunders (1987) made the important observation that the covariation among W, U, and WWWH may be related to restrictions imposed by the shape and relative size of the whorl.

If AH is the ratio of the internal cross sectional height of the whorl (IH) to the external height of the whorl (EH) and W <= 1/U (Fig. 4-7 , inset) then:

$$AH=(1-1/W)/(1-U) \quad (1)$$

When the family of curves for successive values of AH is superimposed on the W-U scatter plot, there appears to be a relationship (Fig. 4-7). The actually occurring combinations of W and U appear to be concentrated on AH of between 0.7 and 1, whereas the preferred AH for mid-Carboniferous ammonoids is between 0.5 and 0.7. Swan and Saunders (1987) stated

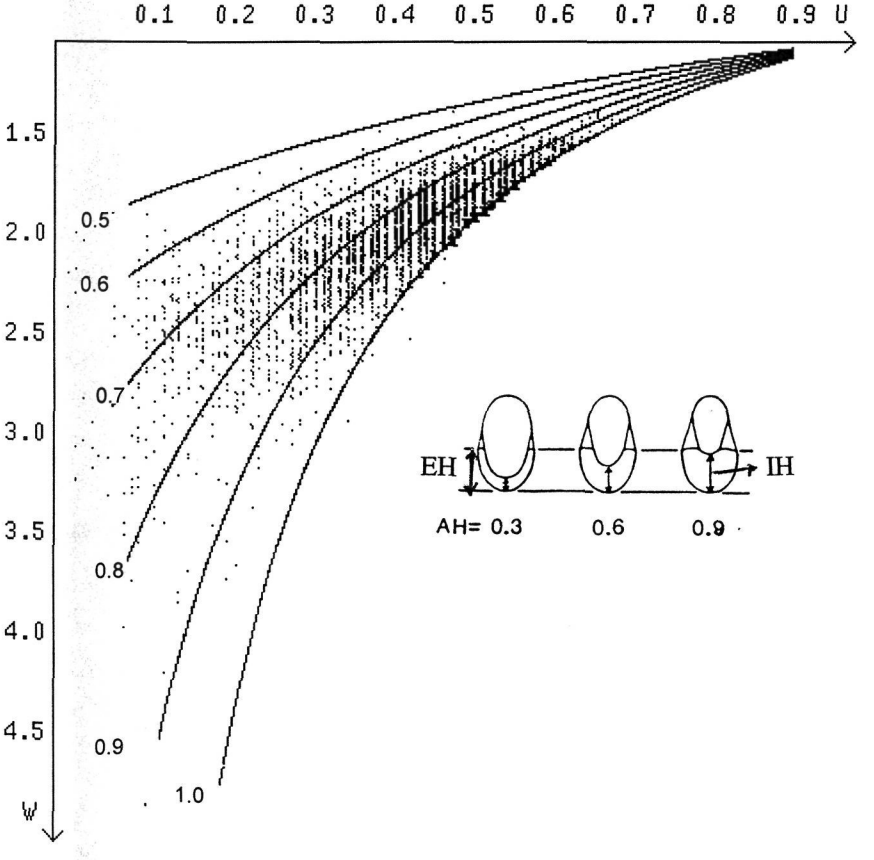


Figure 4-7 Family of curves for AH in terms of W and U

"In terms of function, we speculate that the distinctly U-shaped cross sections that result from low AH values might prohibit the deployment of strong propulsive musculature, and might even restrict ingestion. It is not so obvious why high AH values should have been unfavorable". In contrast, high AH values are popular among Lower Jurassic ammonites. From equation (1) above, it is apparent that as expansion rate (W) increases, U has to decrease if AH is to remain relatively constant as is observed in the Carboniferous and Jurassic samples.

Furthermore, if AA is the cross sectional area of the whorl, as a first order approximation that does not consider the overlap area between two successive whorls, then for unit shell radius:

$$AA=K*WWWH*(1-U)^2 \qquad (2)$$

where $K=\begin{cases} 1 & \text{cross section is rectangular or quadrate} \\ \pi/4 & \text{cross section is circular (Swan and Saunders 1987 equation)} \\ \pi/4 & \text{cross section is elliptic} \\ 1/2 & \text{cross section is triangular} \end{cases}$

In real world ammonites, both rectangular and triangular whorl sections have some degree of rounding, the actual range for constant K is somewhere between 0.6 and 0.9 instead of 0.5 and 1, and the most common forms have K values around $\pi/4$. Assuming $K=\pi/4$, the family of curves for equation (2) in terms of WWWH and U again shows a parallelism with the distribution of Lower Jurassic ammonite data. Fig. 4-8 illustrates a preference for AA values between 0.1 and 0.3 whereas the preferred AA for mid-Carboniferous ammonoids is between 0.5 and 0.7 (Swan and Saunders 1987). From equations (1) and (2), one can infer that, as expansion rate (W) increases, U decreases in order to keep AA constant and consequently WWWH decreases. This explains the previously observed pattern that shells with higher expansion rates tend to be more involute and compressed, and involute shells with low expansion rates tend to be depressed.

I can demonstrate the negative correlation between U and WWWH holds, even if the overlap area between two successive whorls is considered.

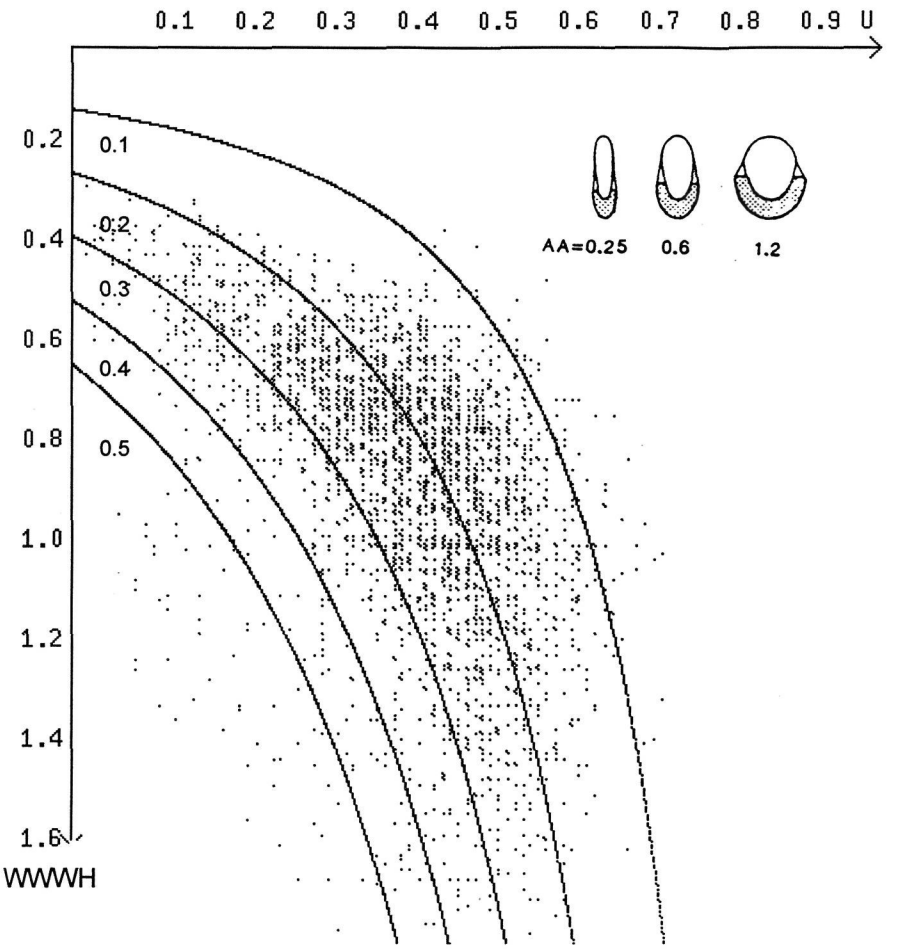


Figure 4-8 Family of curves for AA in terms of WWWH and U

1. If the whorl cross section is rectangular or quadrate, then for unit shell radius:

AA=WWWH*(1-U)*(1-U-(1/W-U)/W) (3)

Let F(U)=(1-U-(1/W-U)/W) (4)

Combining Eqs (1) and (4), we have

$F(U)=(AH-AH^2)*U^2+(2*AH^2-3*AH)*U+2*AH-AH^2$

whose derivative is

$F'(U)=2*(AH-AH^2)*U+2*AH^2-3*AH$

Since U<1,

$F'(U)<-AH$

and AH is always positive, therefore

$F'(U)<0$

The derivative is negative, so the function F(U) decreases as U increases, Finally, because

AA=WWWH*(1-U)*F(U)

I conclude as U increases , WWWH decreases in order to keep AA constant.

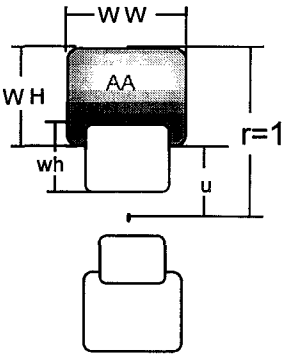


Figure 4-9 Cross section with unit shell radius, so u equals to U of Ammon, WWWH = WW / WH
W = WH /wh

2. If cross section is triangular, then for unit shell radius:

$AA=0.5*WWWH*((1-U)^2-(1/W-U)^2)$ (5)

Let $F(U)=(1-U)^2-(1/W-U)^2$ (6)

Combining Eqs (1) and (6), we have

$F(U)=(2*AH-AH^2)*U^2+(2*AH^2-4*AH)*U+2*AH-AH^2$

whose derivative is

$F'(U)=2*(2*AH-AH^2)*U+2*AH^2-4*AH$

Since $U<1$,

$F'(U)<0$

The derivative is negative, so the function F(U) decreases as U increases, Finally, because

AA=0.5*WWWH*F(U)

I conclude as U increases , WWWH decreases in order to keep AA constant.

3. Other shapes: since the vast majority of whorl shapes stand between the two extremes, that is between triangular and rectangular shapes, we can reasonably expect the negative correlation between U and WWWH would also hold for them.

4.2.5 Morphological Variability and Taxon Longevity

The relative durations of taxa and the possible correlation between morphology and duration are intriguing puzzles of evolution (Ward 1983). The standard deviation contours of W, U, and WWWH (Figs. 4-3, 4-4, 4-5 respectively) not only serve as the "error bars" of Fig.

4-1 and Fig. 4-2, but also show the morphological variability of species at the corresponding location in the morph space. The magnitude of the standard deviation is at least in part controlled by the magnitude of the mean value; to facilitate the comparison of morphological variability between different species, the direct influence by the magnitude of the mean value is eliminated by calculating the *coefficient of variation* (C_V) where

C_V = standard deviation * 100% / average

It is evident from Figs. 4-10, 4-11, 4-12 that the coefficients of variation for W, U, and WWWH show the same pattern of changes (only species with at least 6 occurrences and stratigraphic information are used); small C_V values are associated with the most densely populated region of low W and high U. Moving away from this adaptive peak to less densely populated regions of high W and low U, the coefficients of variation gradually increase, which may indicate that less selective pressure is being applied in these adaptive "lowlands".

Jurassic ammonites are commonly regarded as excellent index fossils that offer fine time resolution. To examine whether there is a correlation between basic shell geometry and taxa duration, the range data of Lower Jurassic ammonite species and genera have been assembled.

Table 5-1 presents the well-established northwest European zonation scheme used as the primary standard in this study; the Lower Jurassic is divided into 4 stages and 20 zones. In the discussion that follows concerning the duration of taxa in units of zones, the following points should be borne in mind:

(1) In constructing time scales, absolute ages are established for points within sequences and encompassed zones are assumed to be equal in duration for interpolation purpose (Palmer et al 1983; Harland et al 1989).

(2) If a taxon is found within a zone, it is assumed to range throughout the zone which may inflate the range of some taxa.

(3) Attention is confined to taxa whose range can be resolved to the zonal level and that are well represented in the database (arbitrarily by at least 6 occurrences).

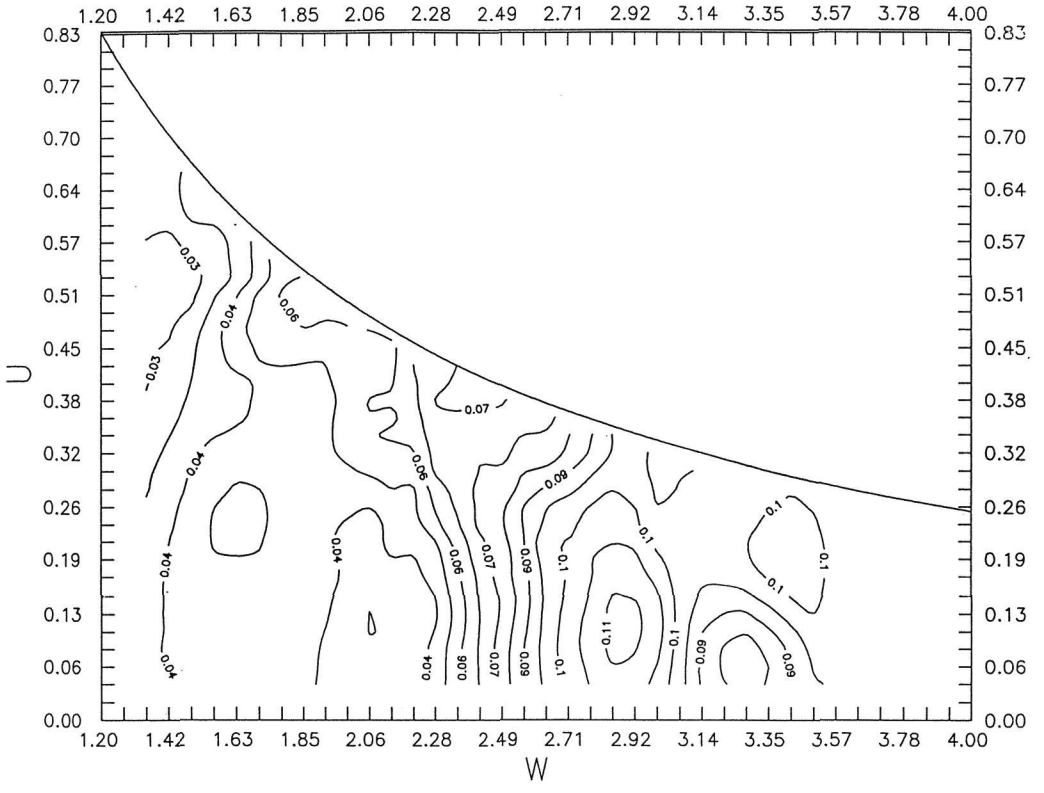


Figure 4-10 The coefficient of variation contours for W
728 species, contour interval = 0.07

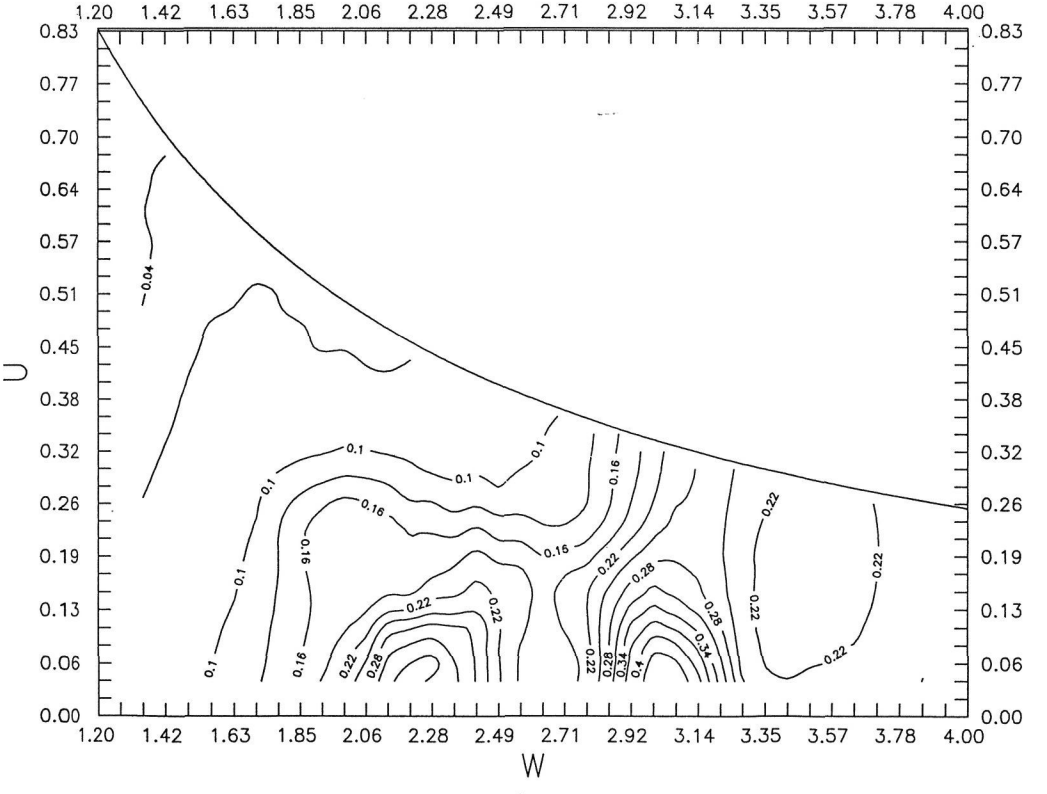


Figure 4-11 The coefficient of variation contours for U
728 species, contour interval = 0.03

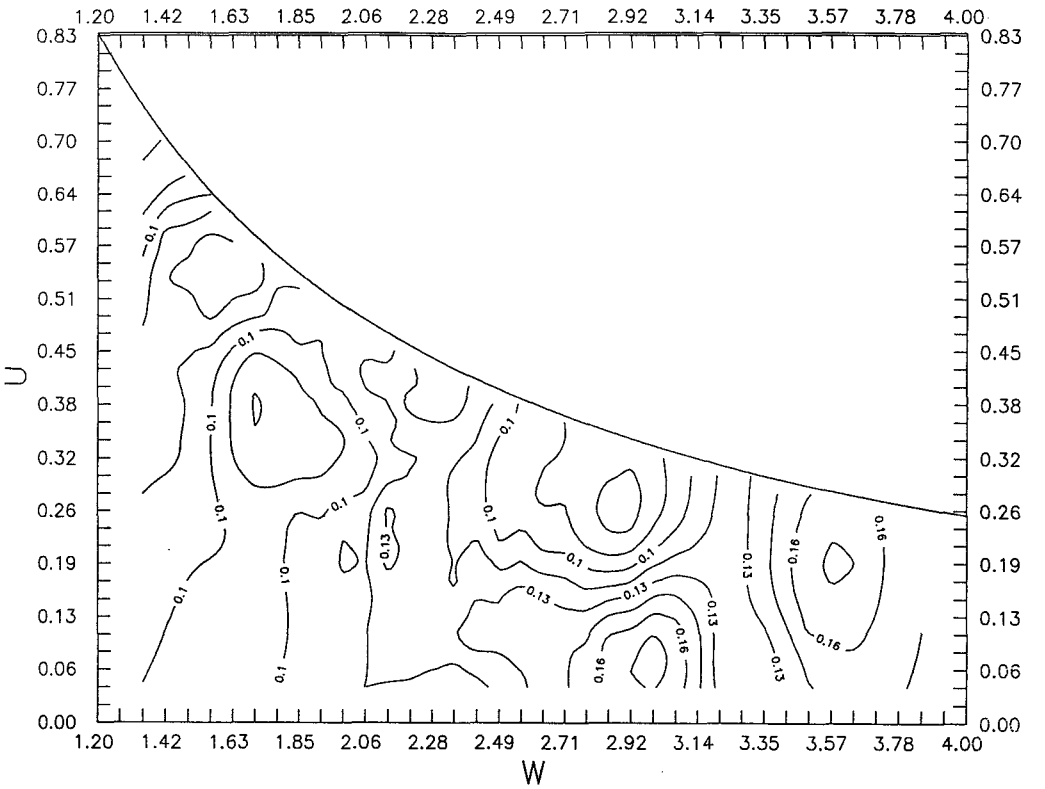


Figure 4-12 The coefficient of variation contours for WWWH 728 species, contour interval = 0.015

The duration contour for the same species as in Fig. 4-10 (Fig. 4-13) shows resemblances to the coefficient of variation contour plots (Figs. 4-10, 4-11, 4-12). Ammonites occupying the adaptive peak have low within-population variability and short duration. Away from this peak in faster expanding regions, there is a steady increase in both morphological variability and taxa longevity implying a decrease selective pressure. Patterns at the generic level need not be identical to those at the species level, since long-lived genera, for example, could include many short-lived species. Fig. 4-14 shows the duration contour for 117 genera to whom the above 964 species belong, which is very similar to the one for the species. This result suggests that generic and species longevities of Lower Jurassic were related to their morphology. One possibility is that the correlation between taxa duration and geometric variability could be the result of unequal taxonomic splitting. The concept that taxonomy and ranges are influenced by complexity of hard-part morphology was advocated

by Schopf et al. (1975) who suggested that a fossil with a great deal of describable morphology will normally show a shorter range than a nondescript form because evolutionary change in the nondescript taxon will have less chance of being recognized. Following this reasoning, the least ornamented ammonites should have longer durations. It is interesting to see whether the above observed correlation holds once the influence of ornamentation is eliminated. Subsamples for both species and genera with smooth shells or weak ornamentation were extracted from the whole sample. Figs. 4-15, 4-16 show their duration contours and it is clear that both figures still reflect the trends seen in Figs. 4-13, 4-14.

The analysis presented here is primitive, and the conclusion that taxa duration may be related to morphological variability within the population through selective pressure is tentative. To be more confident with this conclusion, a more refined time scale calibration, a better correlation between different ammonite zonation schemes, and more specimens for under-represented species are needed.

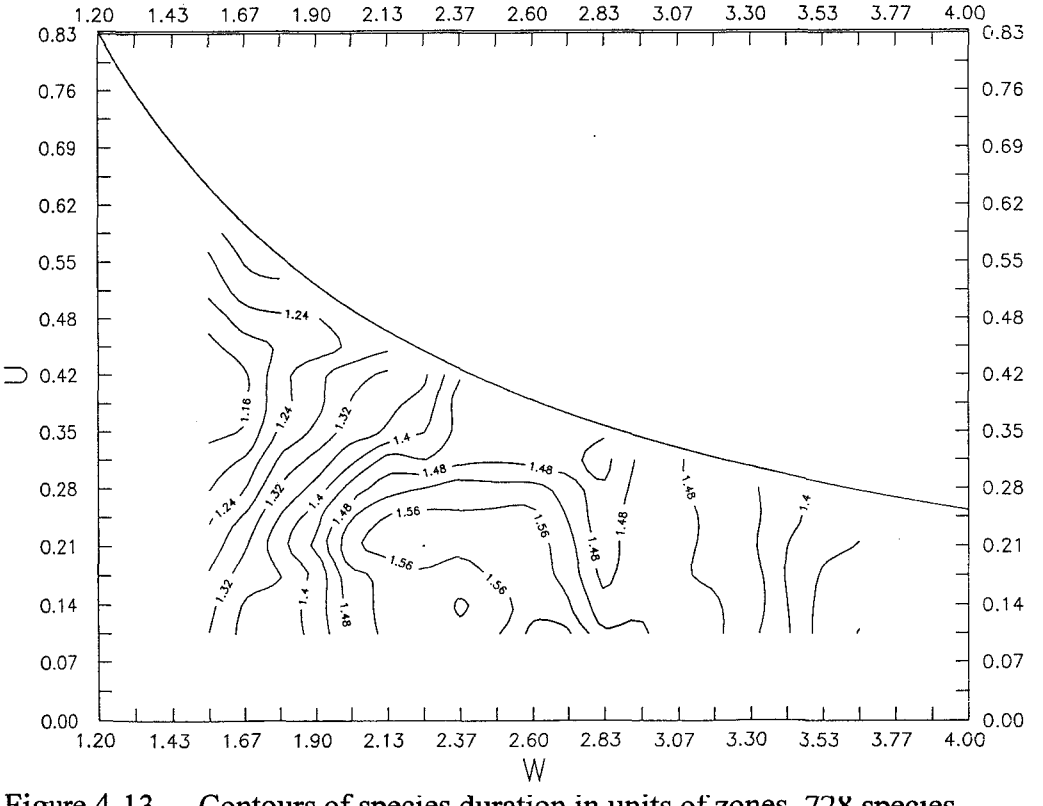


Figure 4-13 Contours of species duration in units of zones, 728 species, contour interval = 0.04

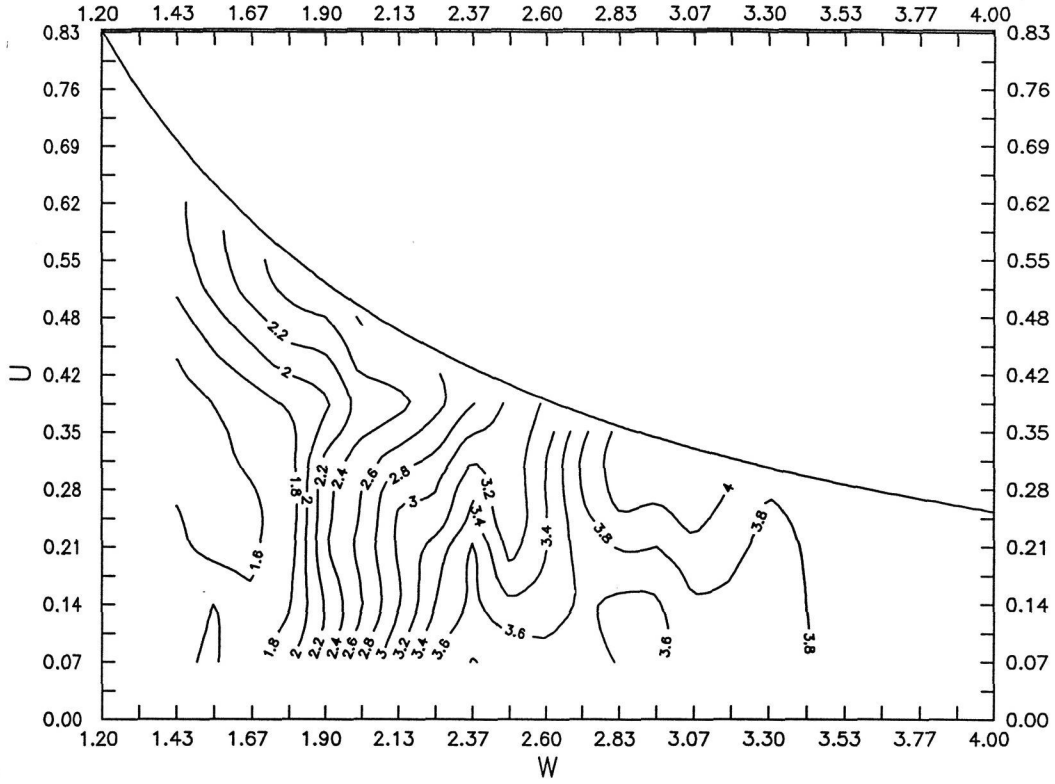


Figure 4-14 Contours of genus duration in units of zones, 117 genera, contour interval = 0.2

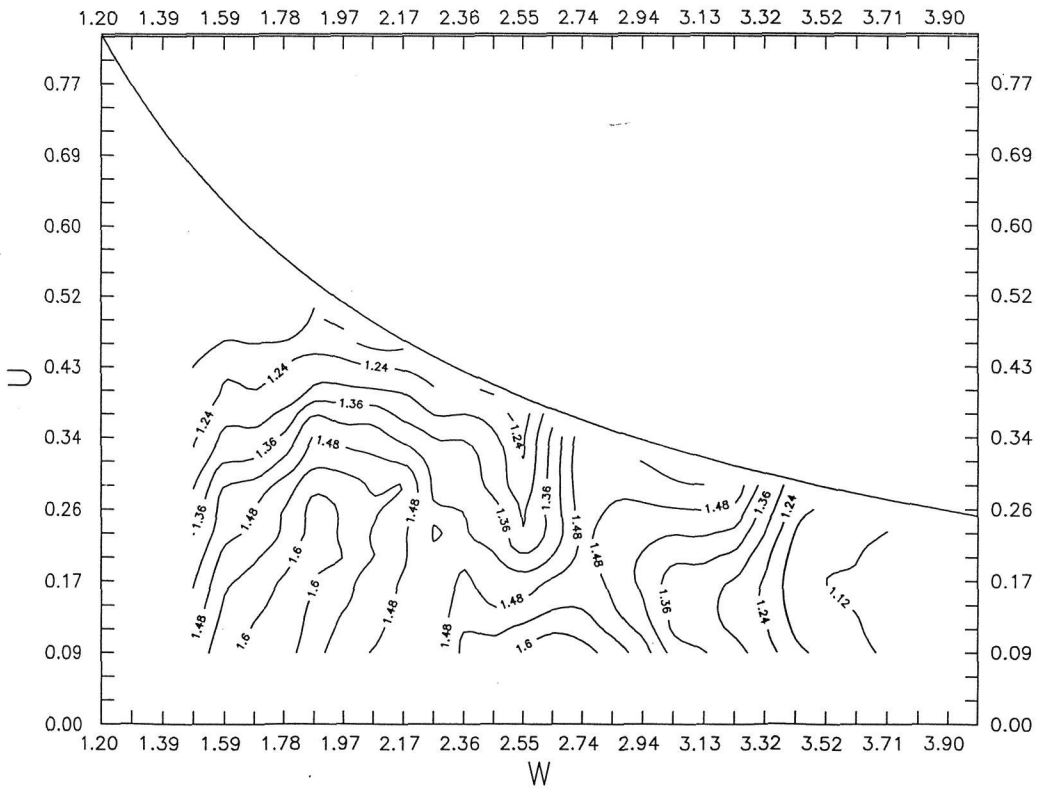


Figure 4-15 Contours of species duration in units of zones for species with weak ornamentation, 258 species, contour interval = 0.06

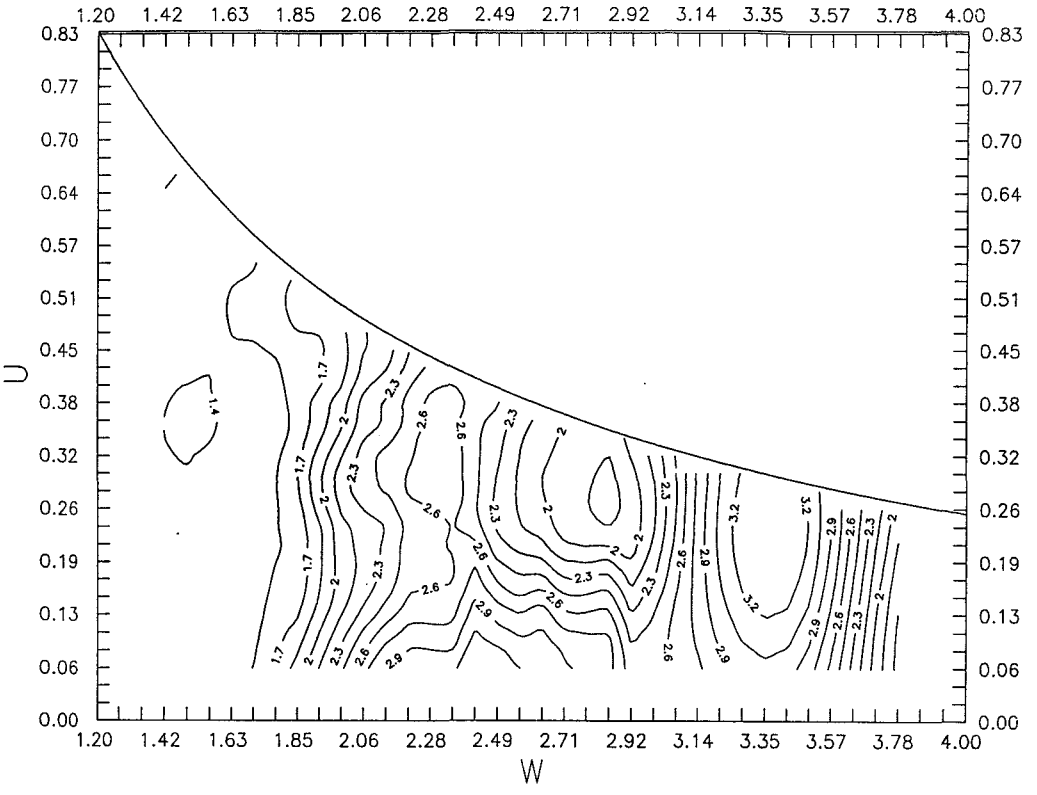


Figure 4-16 Contours of genus duration in units of zones for genera with weak ornamentation, 53 genera, contour interval = 0.15

4.2.6 Correlation between basic shell geometry and other morphological features

The analysis thus far has considered only the essentials of overall form and has ignored ornamentation and ventral features. In the following analyses I explore the correlation between basic geometry (W and U) and 10 variables characterizing ornamentation and ventral features. The most significant correlations with shell geometry involve tubercles, rib-furcation, rib strength and the presence of a keel.

The first attempt is related to shell ornamentation, and particularly to the development of tubercles. Three subsamples were extracted from the total sample. Fig. 4-17 shows the W-U distribution of ammonites with tubercles on the upper flank, Fig. 4-18 shows the W-U distribution of ammonites with tubercles on the lower flank, neither of which differ substantially from the distribution for the whole sample (Fig. 4-1) except for the general absence of tuberculate forms in the low U region. Fig. 4-19 showing the W-U distribution of

ammonites with two rows of tubercles is more or less a combination of Fig. 4-17 and Fig. 4-18.

Figs. 4-20, 4-21, 4-22 show the W-U distributions of ammonites with simple ribs, ribs furcated on upper flank, and ribs furcated on lower flank respectively. The W-U distribution of ammonites with simple ribs is similar to the distribution for the whole sample (Fig. 4-1), whereas ammonites with ribs furcated on upper flank concentrate on the low W and high U region, and ammonites with ribs furcated on lower flank tend to be more involute and have higher expansion rates.

Similar analysis of subsamples was made in relation to the coarseness of ribs. Three subsamples were extracted from the total sample. Figs. 4-23, 4-24, 4-25 show the W-U distribution of ammonites with weak ribs, average ribs, and strong ribs respectively. Ammonites with weak ribs show more diversity in terms of basic shell morphology, as the rib strength increases, the shells tend to be more and more restricted to Chamberlain's (1981) low C_D peak region with high hydrodynamic efficiency. It is difficult to explain,

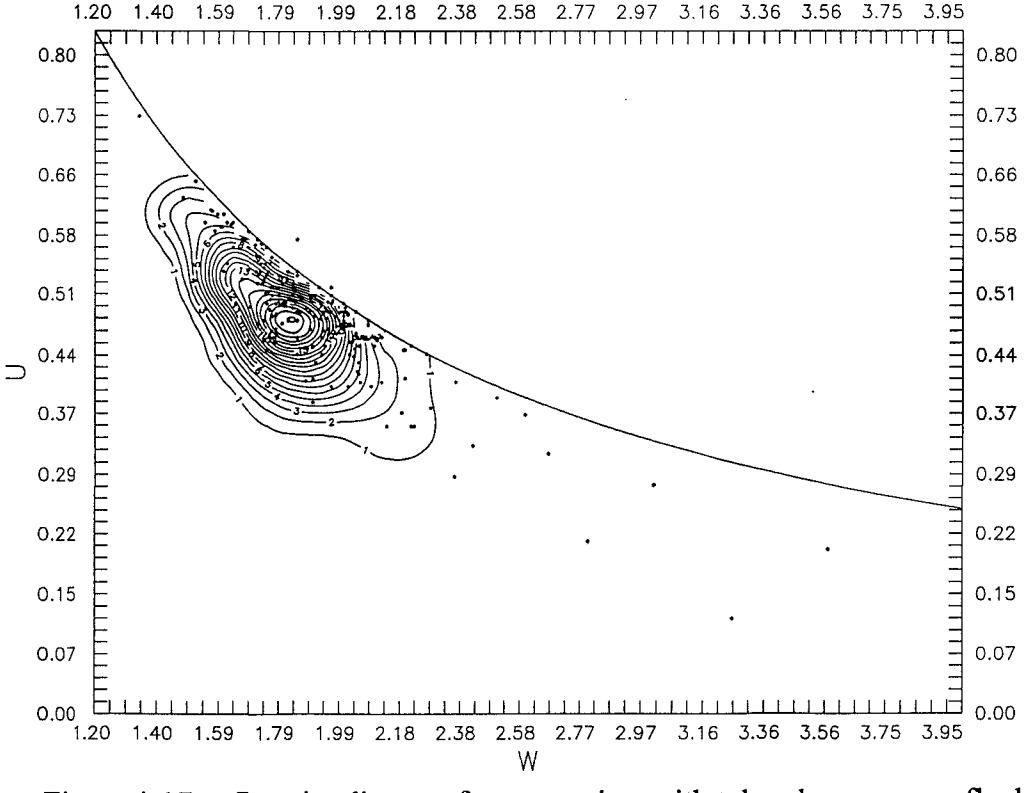


Figure 4-17 Density diagram for ammonites with tubercles on upper flank (208 species)

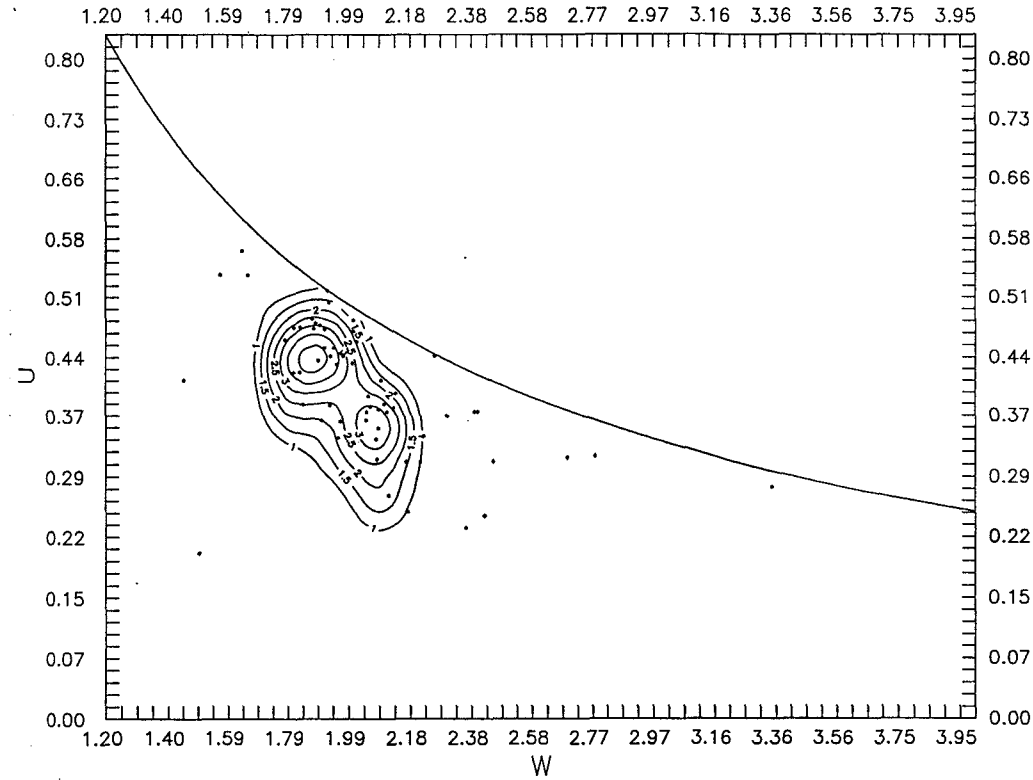


Figure 4-18 Density diagram for ammonites with tubercles on lower flank (64 species)

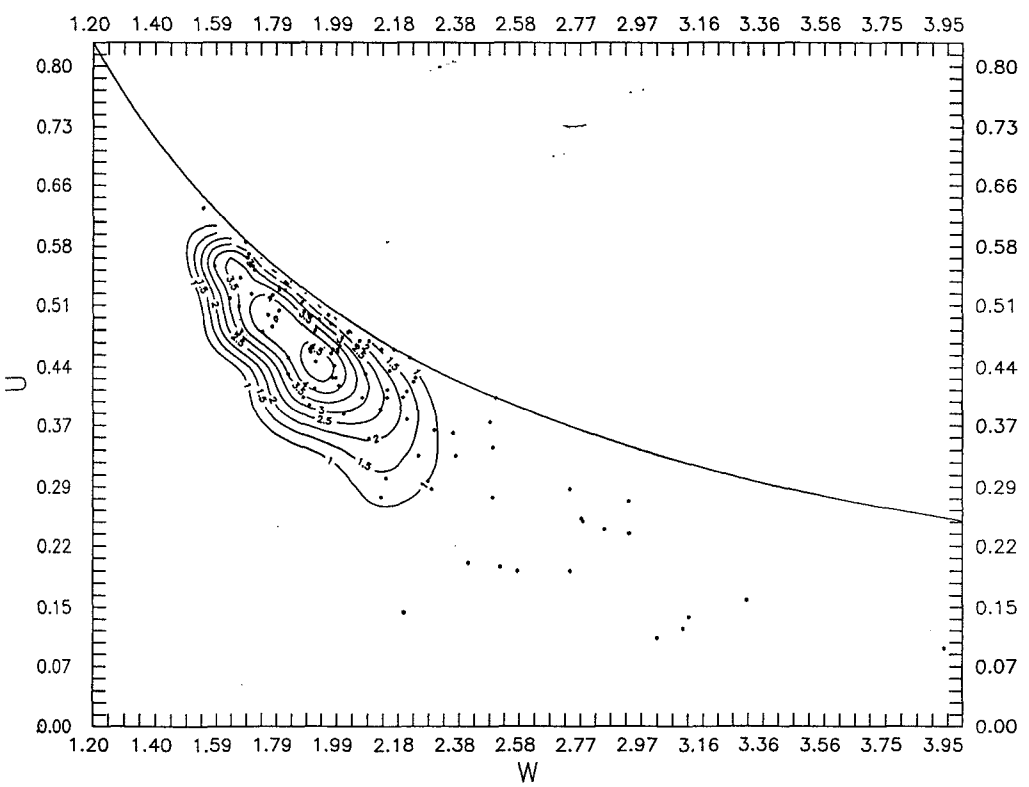


Figure 4-19 Density diagram for ammonites with two rows of tubercles (74 species)

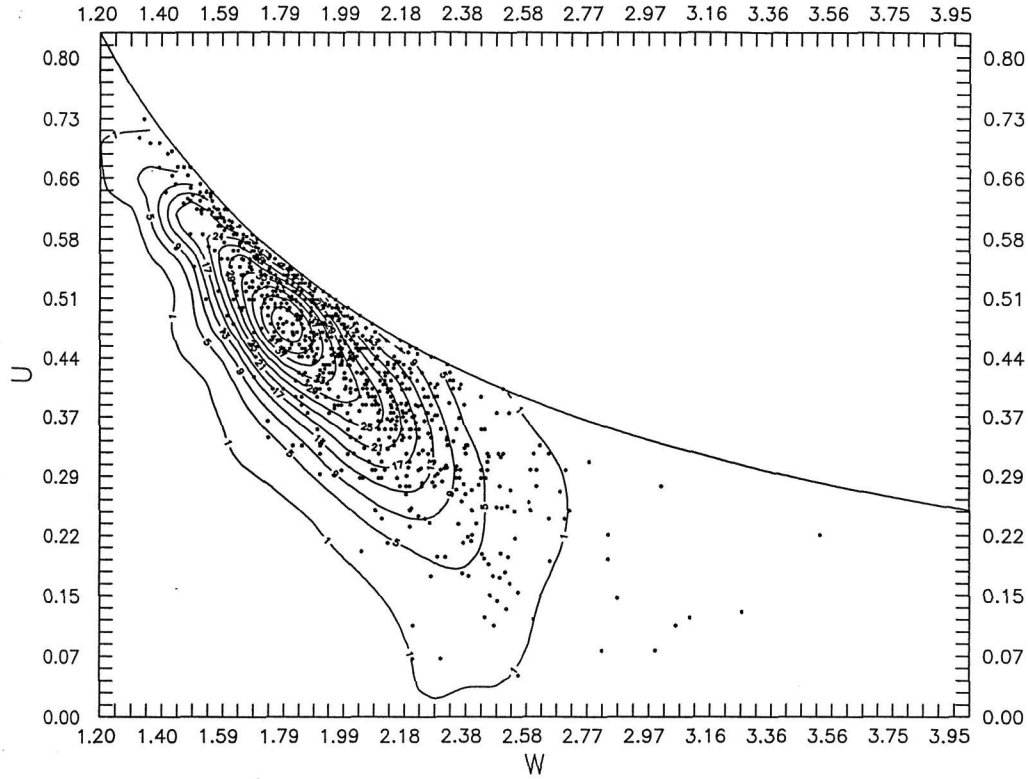


Figure 4-20 Density diagram for ammonites with simple ribs
(784 species)

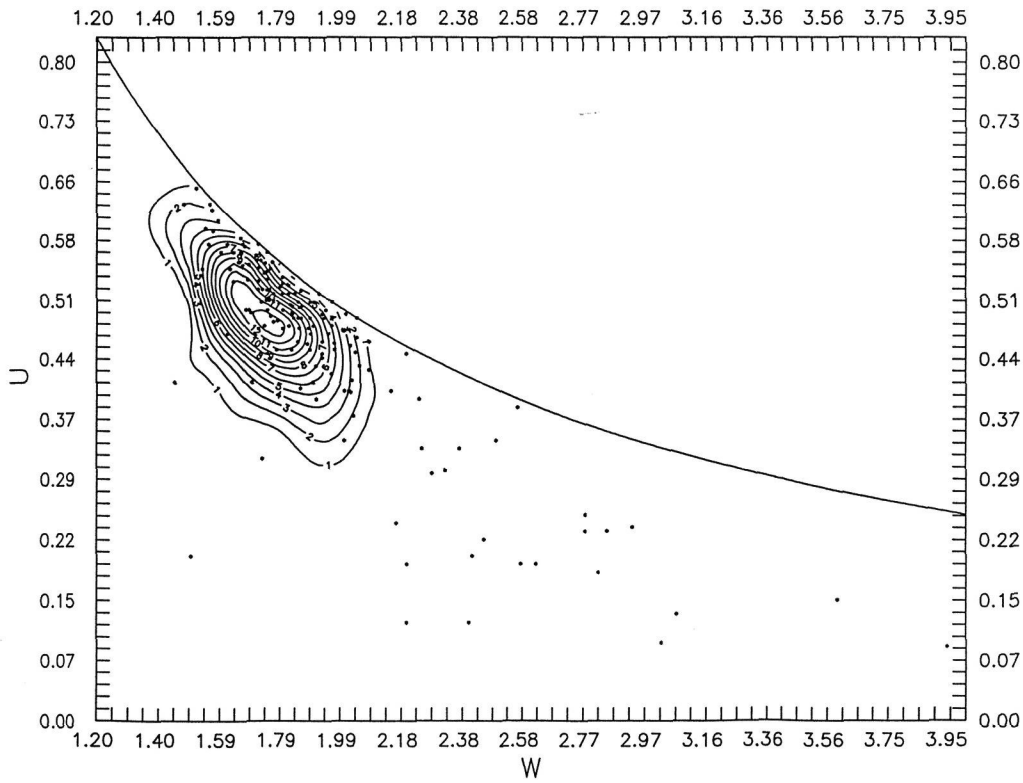


Figure 4-21 Density diagram for ammonites with ribs furcated on upper flank
(150 species)

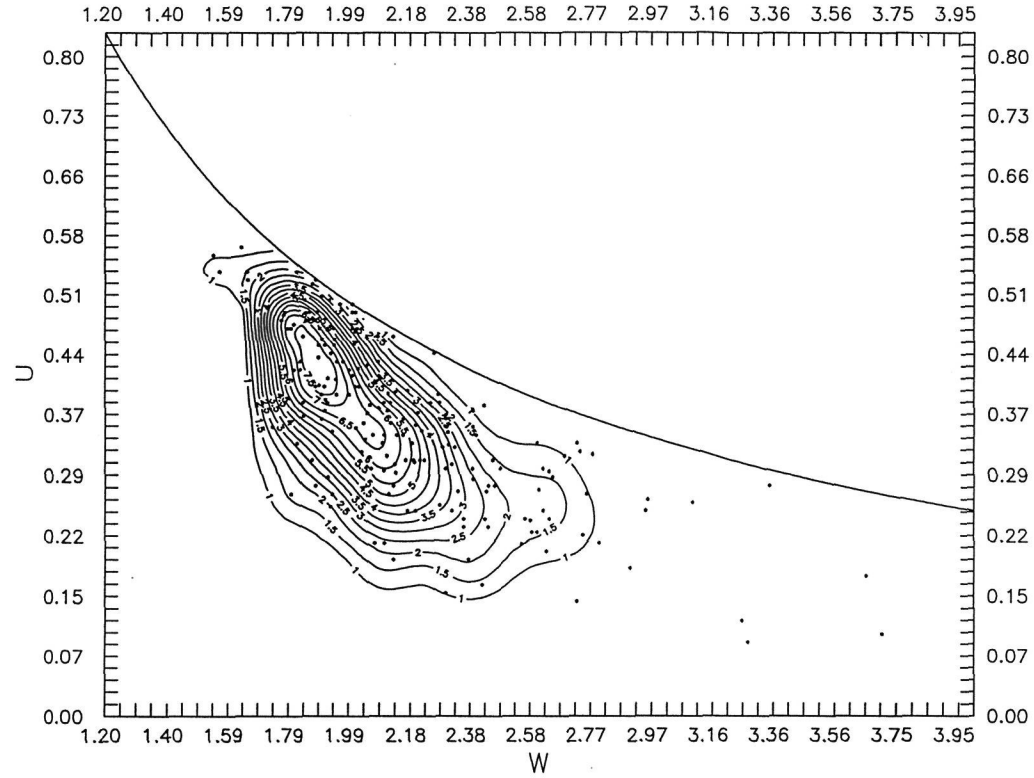


Figure 4-22 Density diagram for ammonites with ribs furcated on lower flank (178 species)

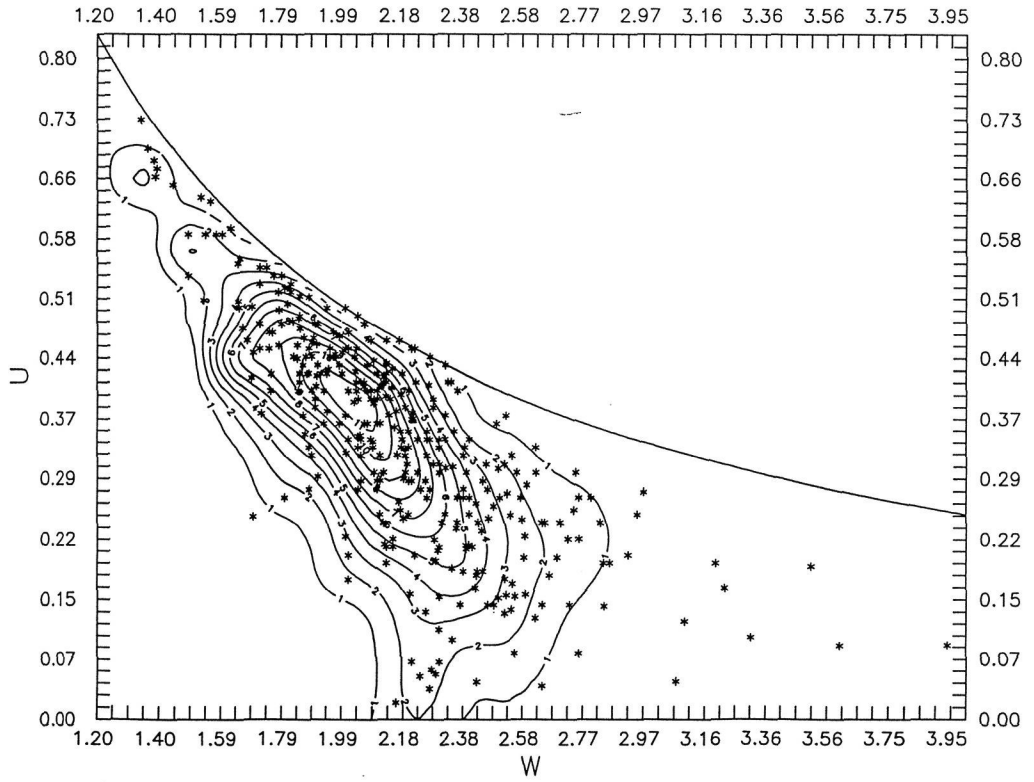


Figure 4-23 Density diagram for ammonites with weak ribs (278 species)

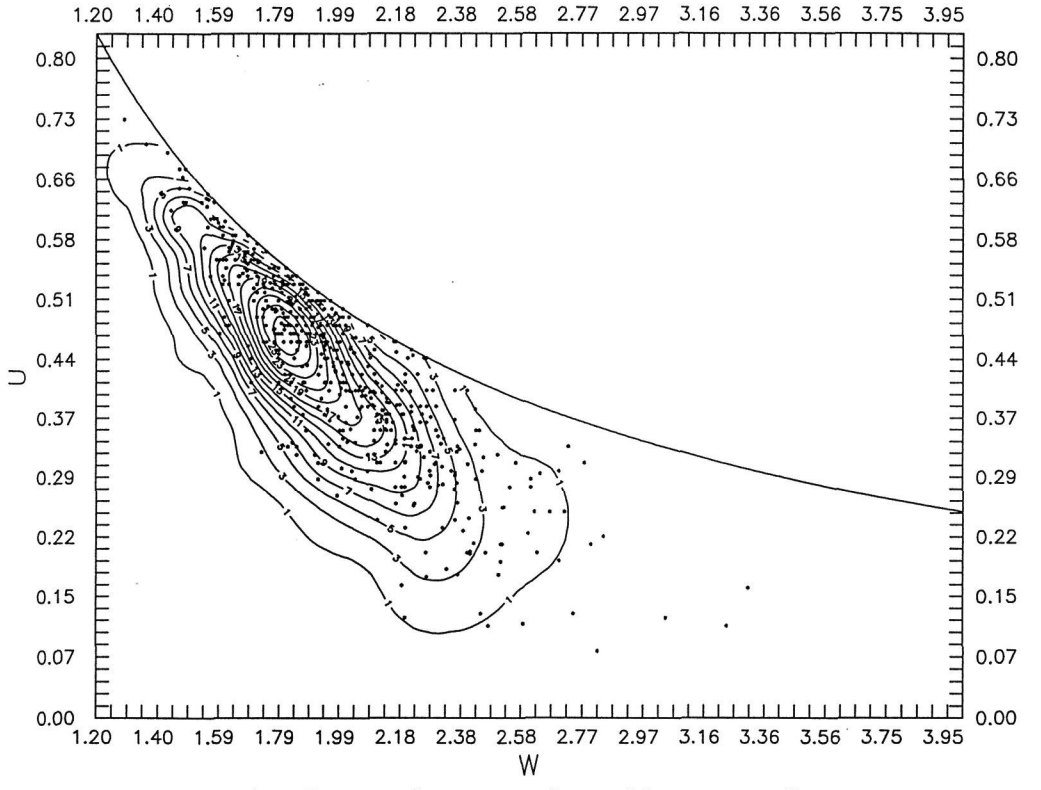


Figure 4-24 Density diagram for ammonites with average ribs (701 species)

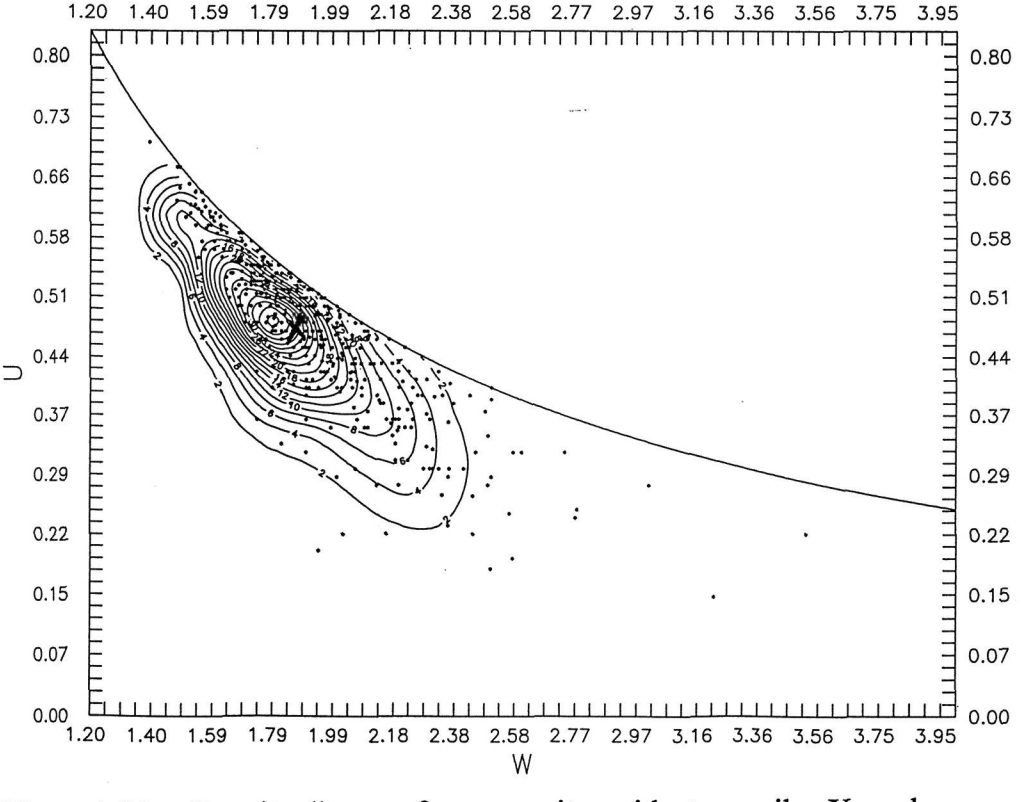


Figure 4-25 Density diagram for ammonites with strong ribs, X marks low C_D peak of Chamberlain (1981), 351 species

considering ammonites with strong ornamentation are generally regarded as poor swimmers (Swan and Saunders 1987). One possibility is that if ribs are coarse for reasons not connected with hydrodynamic efficiency, then it makes sense to offset their detrimental impact on the drag coefficient by adopting the most hydrodynamically efficient shell geometry. Since Chamberlain's C_D contour map was based on shells having circular whorl cross sections (WWWH=1), I used a subsample of specimens with 0.89<=WWWH<=1.11. Fig. 4-26 shows the same phenomenon, namely that shell forms with the coarsest ribs are concentrated around the low C_D peak region.

A similar analysis of subsample was made in relation to the presence or absence of a well-defined keel on the venter. Fig. 4-27 shows ammonites with a plain venter are most abundant in the low W and low U region. Fig. 4-28 shows ammonites with a keel are more evenly distributed.

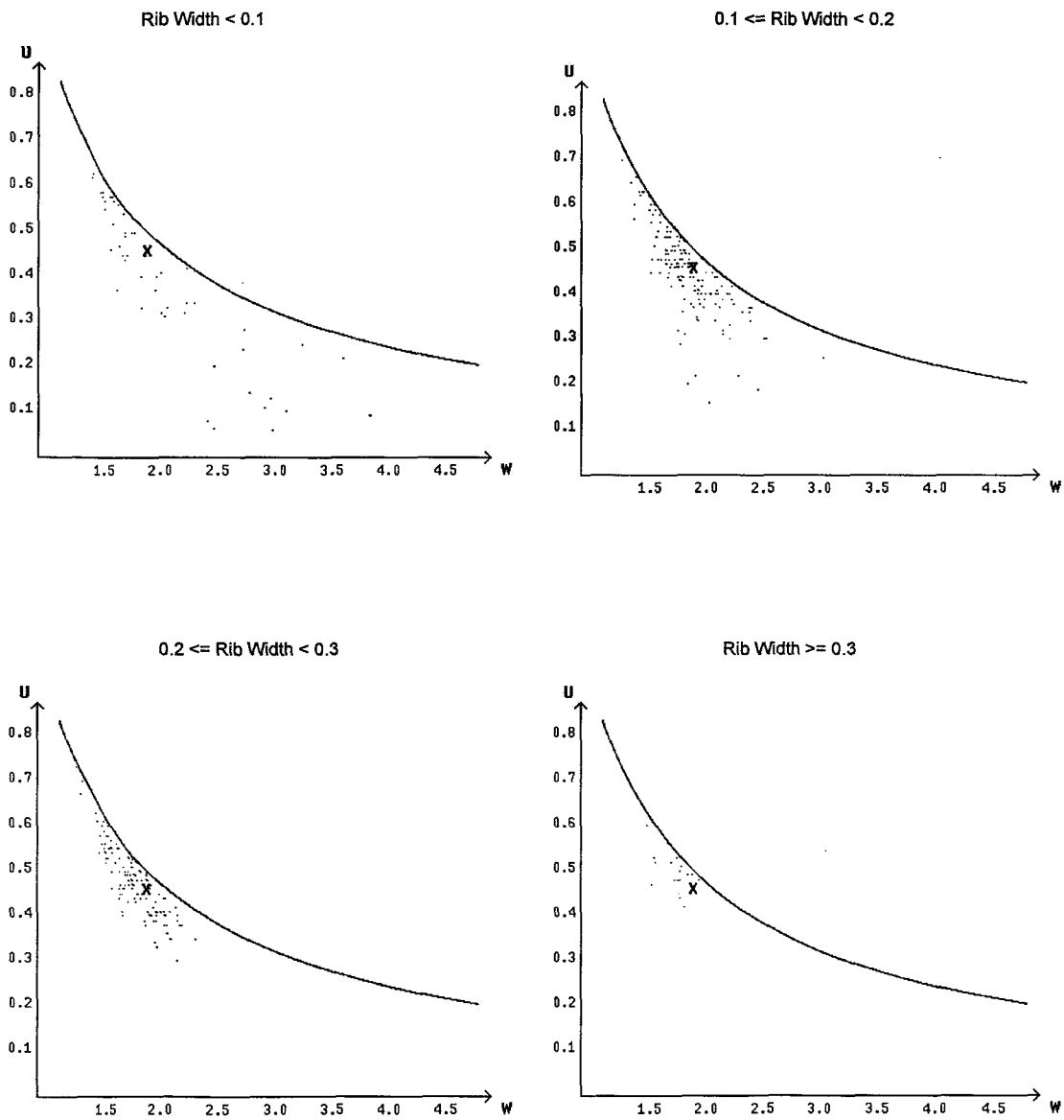


Figure 4-26 W and U distribution with respect to rib width

X marks low C_D peak of Chamberlain (1981)

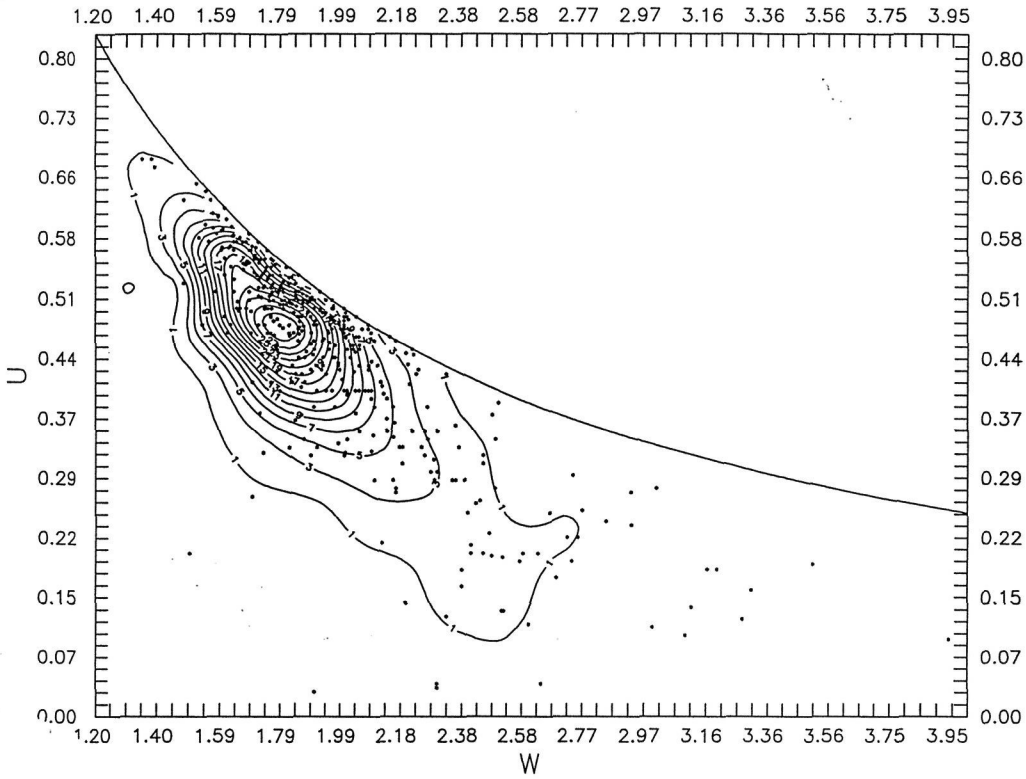


Figure 4-27 Density diagram for ammonites with plain venter
(409 species)

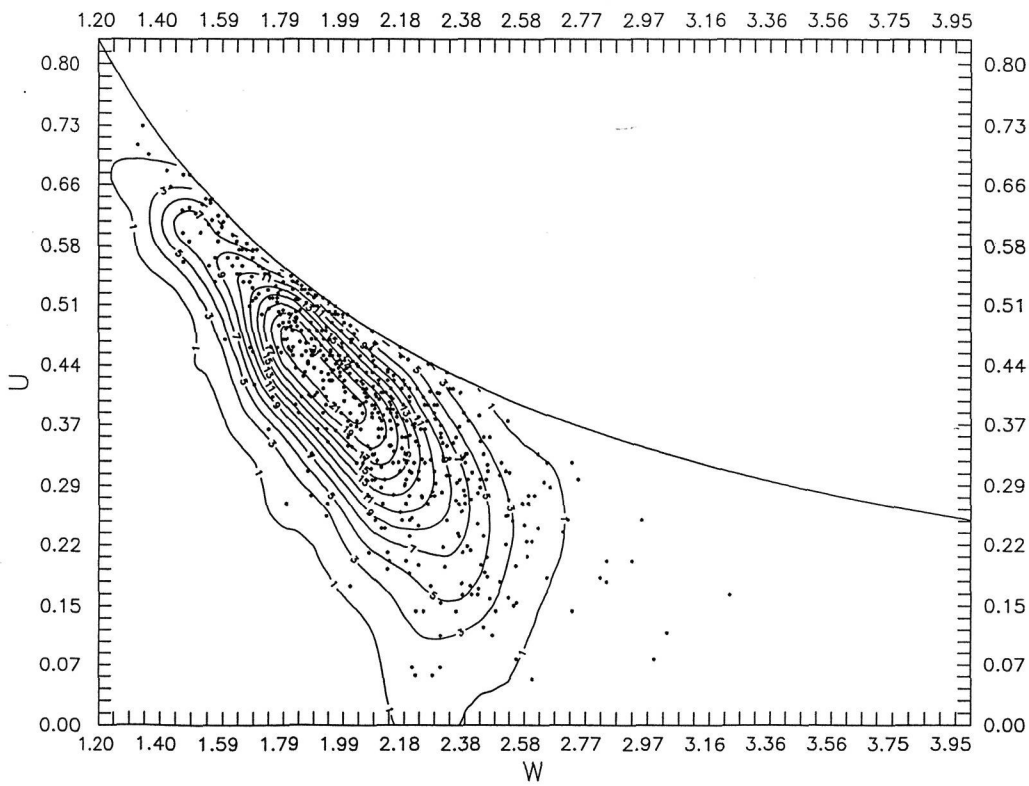


Figure 4-28 Density diagram for ammonites with carinate venter
(540 species)

4.3 Covariation

4.3.1 Introduction

Westermann (1966) has observed that several ammonite stocks show a persistent intercorrelation of morphological features: more involute and compressed forms are less ornate than more evolute and inflated forms. Such a correlation was probably first recognized by Buckman (1892) and was named 'Buckman's Law of Covariation' by Westermann (1966). Buckman (1892) identified 69 new species from a single bed of the Sowerbyi Zone in the Bajocian (Middle Jurassic) of Dorset, England. Based on covariation and morphologic intergradation, Westermann was able to group 64 of Buckman's species into a single palaeospecies. Subsequently, similar phenomena have been widely documented, for example, by Bayer and McGhee (1984), and Kennedy and Cobban (1976), who cited examples of intra-specific variation in Cretaceous ammonites.

In Jurassic ammonites the superfamily has long been used as the principal high level taxon (Donovan, Callomon, and Howarth 1981). There are three superfamilies in the Lower Jurassic. The oldest is the Psiloceratacеae which appears at the base of Jurassic and is of uncertain origin. The Eoderoceratacеae appears four zones later than the Psiloceratacеae and ranges four zones higher. The Hildoceratacеae arises from the Eoderoceratacеae in the lowest Pliensbachian and persists into the Middle Jurassic where it gives rise to all other Jurassic ammonites (Fig. 4-29). In the following sections, I will try to demonstrate that 'Buckman's Law of Covariation applies equally well to the three superfamilies of Lower Jurassic ammonites and in so doing, give a more rigorous demonstration of Buckman's Law than has been previously possible. I will then go on to explore other patterns of covariation. To focus on functional rather than phylogenetic constraints, I will test the patterns for each of the three superfamilies independently. To avoid bias by some frequently represented species in the

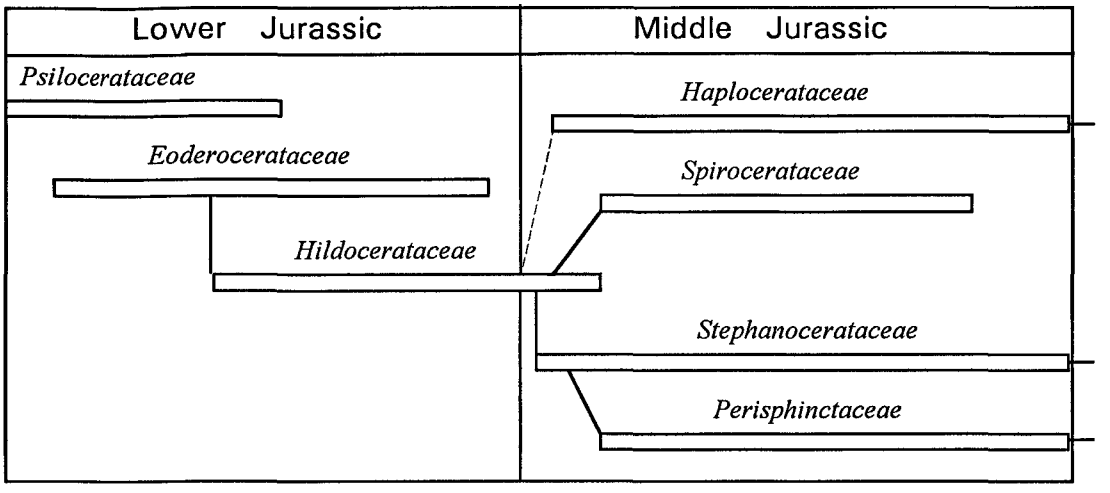


Figure 4-29 Ranges and possible phylogenetic relationships of ammonite superfamilies in the Lower-Middle Jurassic (after Donovan, Callomon and Howarth 1981)

database, species rather than specimens serve as the basic calculation unit. To incorporate the morphological variation during ontogeny, a species might be represented more than once. For example, the species *Haugia jugosa* shows strong variation during ontogeny and has four entries in the database:

entry	W	V	Diameter
1	2.62	0.59	4.71
2	2.18	0.47	15.47
3	1.87	0.43	23.60
4	1.64	0.36	31.73

When studying how the V values are related to W values, the data are divided into three groups with W <1.8 (group 1), 1.8 <=W< 2.2 (group 2), and W >=2.2 (group 3). Entry 1 falls into group 3 with a V value of 0.59; entries 2 and 3 fall into group 2 with an average V value of 0.45; and entry 4 falls into group 1 with a V value of 0.36 so there are three data points for *Haugia jugosa* in the covariation analysis.

For quantitative morphological parameters, the data are plotted as 3-D bar diagrams.

Table 4-2 Abbreviations for statistical and morphological terms

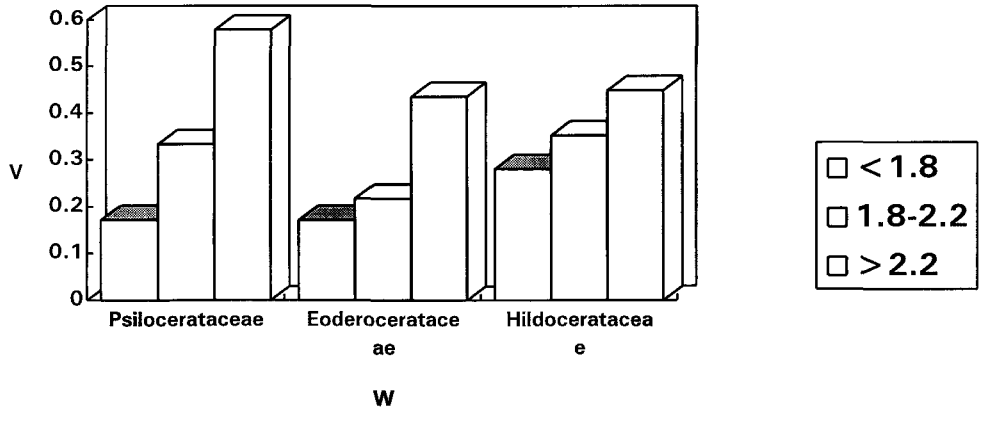
M -- Mean	S -- Standard Deviation
Sig. -- Significance Level, T Test	F -- Frequency
PRHW -- Primary Rib Density per half whorl	N -- Number of Species
RW -- Ratio of Maximum Rib Width to Whorl Height	RF -- Rib Form
WWWH -- Ratio of Whorl Width to Whorl Height	D -- Shell Diameter
V -- Percentage of Interior Whorl being Overlapped	WWD -- Ratio of Whorl Width to D

difference between two means is relatively small comparing with the standard deviations yet the means are significantly different due to large number of specimens used in the calculation

4.3.2 Buckman's Law

(1) Shells with higher expansion rate (W) tend to be more involute (V).

Superfamily	Psiloceratacea				Eoderoceratacea				Hildoceratacea			
W\V	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
<1.8	0.173	0.126	152		0.172	0.105	145		0.281	0.134	77	
1.8-2.2	0.334	0.220	163	0.001	0.218	0.136	208	0.001	0.353	0.173	321	0.001
>2.2	0.578	0.239	95	0.001	0.436	0.222	71	0.001	0.449	0.193	217	0.001

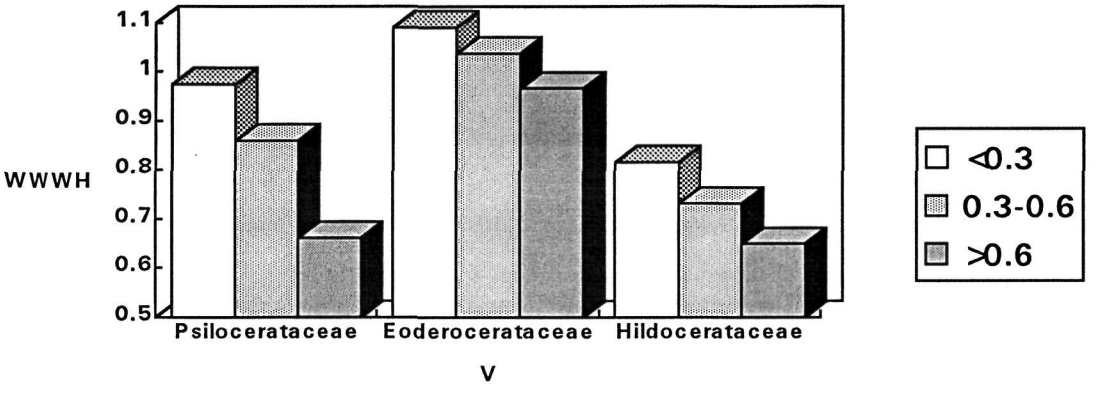


This covariation can be explained by equation (1) in section 4.2, that is, the restrictions imposed by the ratio of the interior height of the whorl (IH) to the exterior height of the whorl (EH).

(2) Involute shells are more compressed than evolute shells.

(2) Involute shells are more compressed than evolute shells.

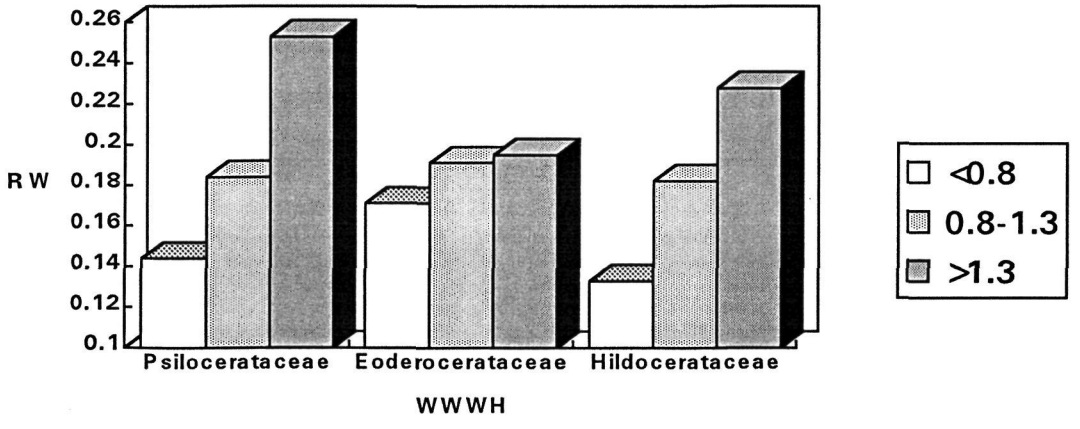
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
V\WWWH	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
<0.3	0.975	0.183	123		1.091	0.293	217		0.816	0.154	183	
0.3-0.6	0.860	0.218	86	0.001	1.038	0.401	100	0.09	0.731	0.2	206	0.001
>0.6	0.661	0.370	53	0.001	0.967	0.379	33	0.10	0.649	0.190	91	0.001



This covariation can be explained by equation (2) in section 4.2, that is, the restrictions imposed by the relative cross sectional area of the whorl (AA).

(3) Compressed shells have finer ribs.

Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
WWWH\RW	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
<0.8	0.144	0.059	79		0.171	0.056	92		0.133	0.047	271	
0.8-1.3	0.184	0.057	152	0.001	0.191	0.067	216	0.005	0.182	0.048	165	0.001
>1.3	0.253	0.057	13	0.001	0.195	0.073	104	0.15	0.228	0.076	7	0.001



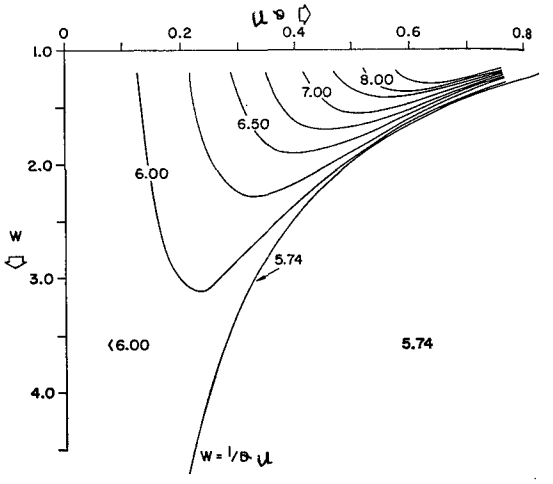


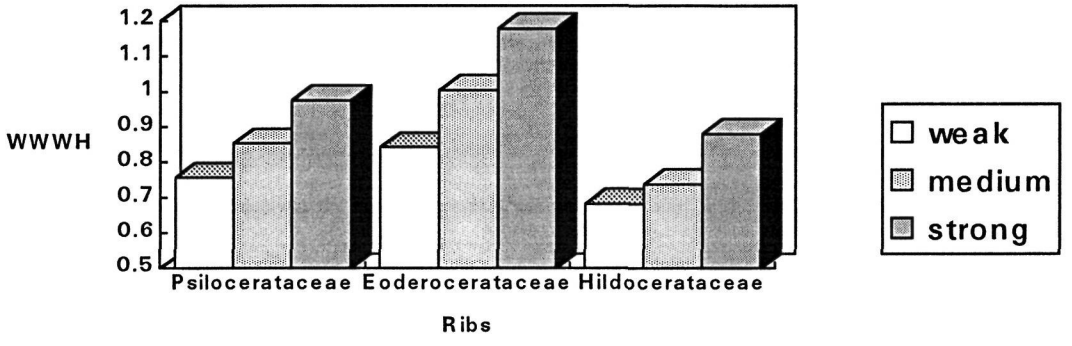
Figure 4-30 Variation in the ratio between internal shell volume and volume of shell material, as a function of W and U assuming a shell thickness equal to 0.0772 of the radius of the generating curve(From Raup 1967).

Buoyancy control and carbonate efficiency may have played major roles here. Assume (1) a circular generating curve and (2) no variation in relative shell thickness during ontogeny, Fig. 4-30 shows the distribution of the ratio: *internal volume/volume of shell material* with respect to W and U (Raup 1967). The buoyancy and relative secretion efficiency of the shell as measured by this ratio decrease toward the region of high W and low U. In a real example, Swan and Saunders (1987) measured the shell thickness on 129 whorl sections from 49 Middle Carboniferous ammonoid specimens and showed that there is an inverse relationship between W and shell thickness: forms with higher W tend to have thinner shells. Three possibilities arise: (a) weak ornamentation could minimize the loss of buoyancy due to the rapidly expanding shell. Fig. 4-22 and 24 in the preceding section also support this reasoning, since ammonites with strong ornamentation are restricted to the low W region whereas ammonites with weak ornamentation could have either slow or fast expanding shells; (b) it may be that shells with high W simply could not afford extra shell ornamentation due to the low carbonate efficiency; (c) weak ornamentation gives compressed and fast expanding shells

better streamlining which may lessen the role of the shell as a protective device. No matter which of these interpretations is correct, covariations (1) and (2) can lead to (3).

(4) Conversely from (3), shells with weak ribs are more compressed.

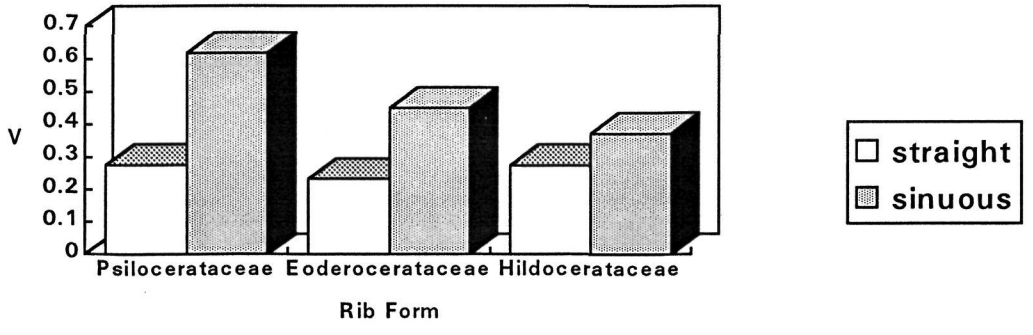
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Rib\WWWH	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
weak	0.757	0.253	45		0.843	0.215	28		0.683	0.166	121	
medium	0.854	0.288	139	0.02	1.005	0.311	150	0.004	0.739	0.201	215	0.004
strong	0.976	0.219	56	0.002	1.178	0.307	188	0.001	0.882	0.160	106	0.001



4.3.3 Volution

(1) Shells with sinuous ribs are more involute than shells with straight ribs.

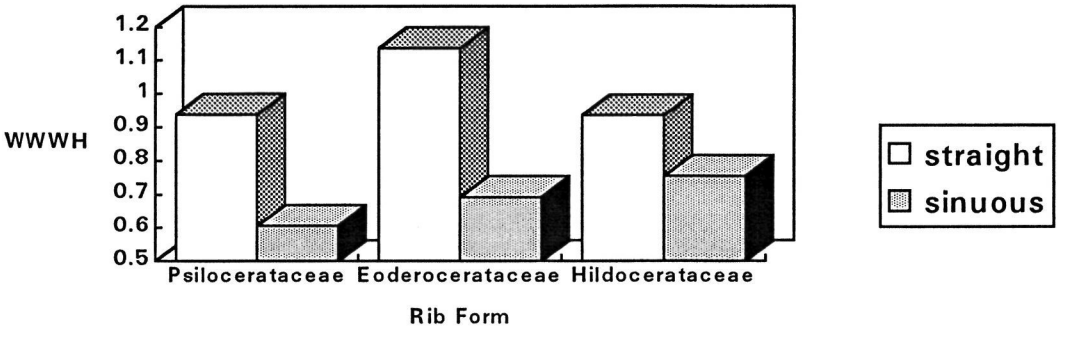
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Rib Form\V	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
straight	0.277	0.209	176		0.233	0.170	279		0.275	0.161	105	
sinuous	0.618	0.231	20	0.001	0.450	0.117	15	0.001	0.373	0.177	231	0.001



From (1) and Buckman's law, we can infer:

(2) Shells with sinuous ribs are more compressed than shells with straight ribs.

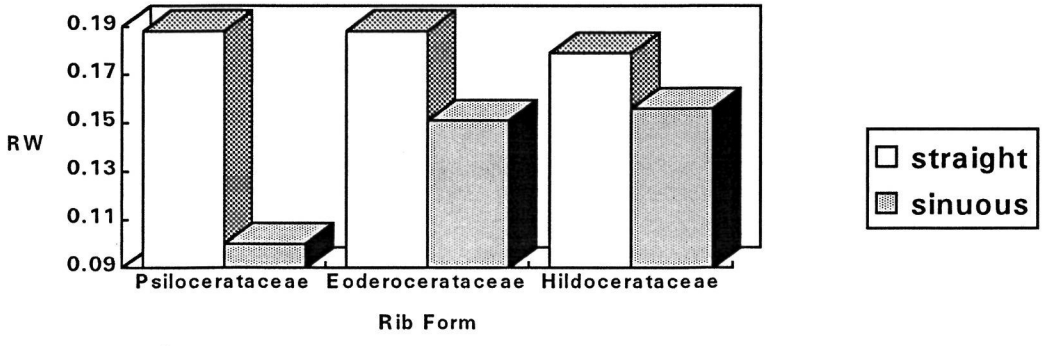
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Rib\WWWH	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
straight	0.938	0.253	137		1.137	0.300	289		0.935	0.246	93	
sinuous	0.606	0.134	13	0.001	0.692	0.095	17	0.001	0.754	0.154	169	0.001



From (1) and Buckman's law, we can also infer:

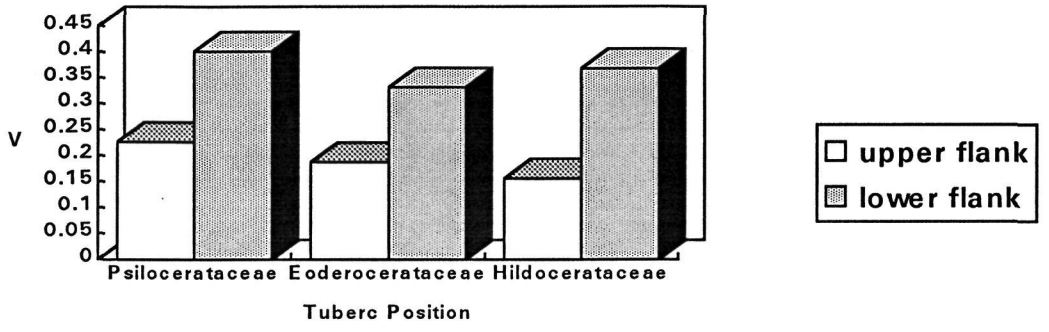
(3) Straight ribs tend to be coarser than sinuous ribs.

Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
RF\RW	M	S	N	Sig.	M	S	N	Sig	M	S	N	Sig.
straight	0.188	0.061	207		0.188	0.068	325		0.179	0.053	127	
sinuous	0.1	0.042	22	0.001	0.151	0.056	19	0.1	0.156	0.047	240	0.001



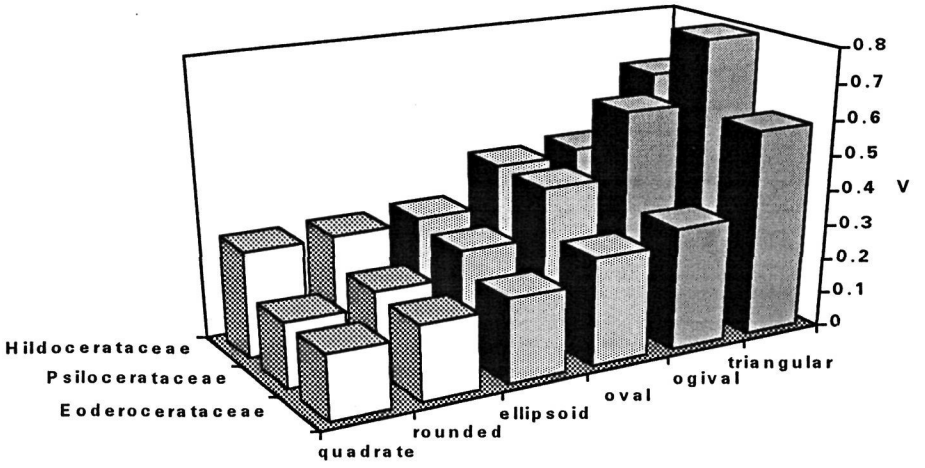
(4) Shells with tubercles on the upper flank tend to be more evolute than shells with tubercles on the lower flank.

Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Tuberc\V	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
upper flank	0.226	0.162	29		0.187	0.110	152		0.156	0.102	12	
lower flank	0.401	0.149	4	0.025	0.331	0.175	7	0.001	0.368	0.169	60	0.001



(5) Shells tend to be more involute as whorl shape changes from quadrate, rectangular->rounded->ellipsoid->oval,ogival->triangular.

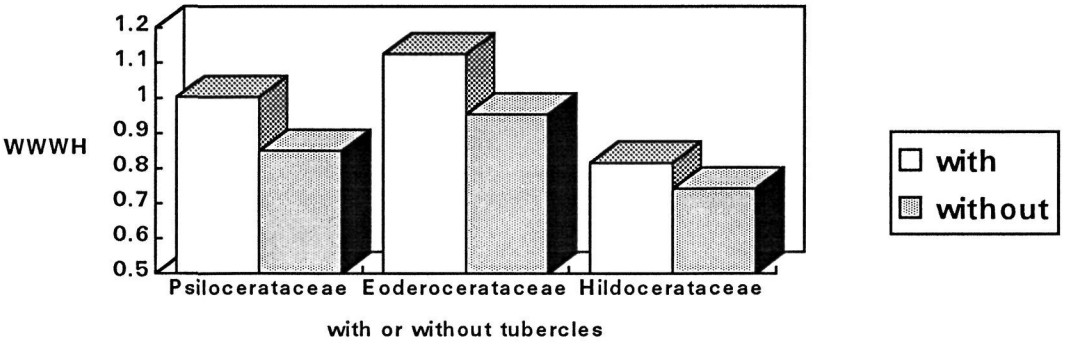
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Whorl\V	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
quadrate	0.186	0.137	52		0.184	0.131	96		0.306	0.133	152	
rounded	0.227	0.167	73	0.07	0.214	0.165	71	0.09	0.312	0.151	22	No
ellipsoid	0.299	0.225	79	0.01	0.240	0.169	117	0.12	0.324	0.163	127	No
oval	0.436	0.214	39	0.001	0.308	0.169	26	0.03	0.442	0.215	75	0.001
ogival	0.618	0.251	20	0.003	0.350	0.166	36	0.13	0.457	0.182	131	0.2
triangular	0.794	0.166	18	0.008	0.588	0.150	7	0.001	0.651	0.165	27	0.001



4.3.4 Whorl Shape

(1) Tuberculate shells are more depressed than nontuberculate shells.

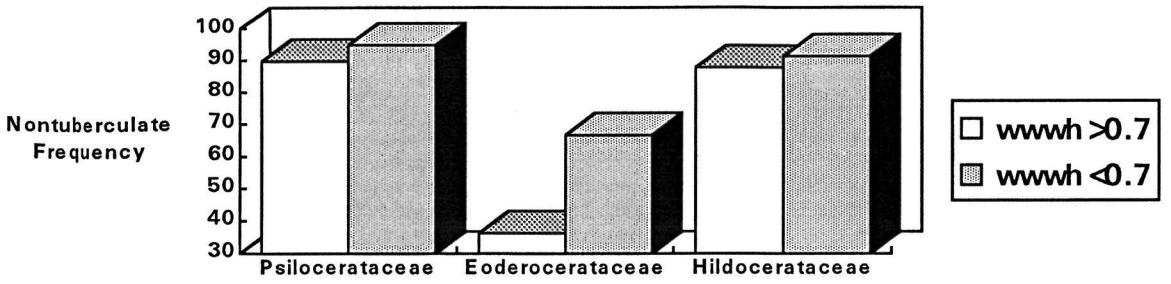
Superfamily	Psiloceratace ae				Eoderoceratace ae				Hildoceratace ae			
Tub\WWWH	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
with	1.003	0.29	29		1.125	0.309	226		0.816	0.17	60	
without	0.849	0.246	242	0.001	0.954	0.303	136	0.001	0.742	0.203	355	0.004



(2)

Conversely the frequency of shells without tubercles is higher in highly compressed forms (WWWH<0.7).

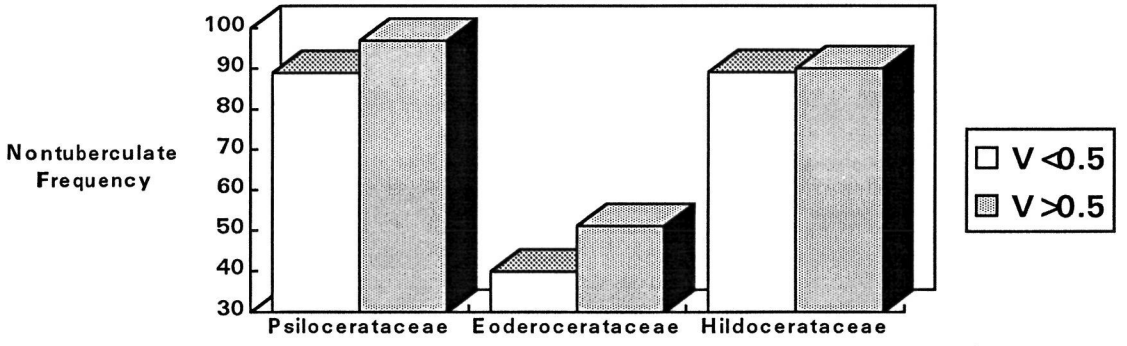
Superfamily	Psiloceratace ae		Eoderoceratace ae		Hildoceratace ae	
WWWH\F%	M	N	M	N	M	N
WWWH > 0.7	89.74	302	36.34	322	87.85	288
WWWH < 0.7	94.93	79	66.66	60	91.32	219



From (1) and Buckman's law, we can infer:

(3) Frequency of shells without tubercles is greater in involute forms (V>0.5).

Superfamily	Psiloceratace ae		Eoderoceratace ae		Hildoceratace ae	
V\F%	M	N	M	N	M	N
V < 0.5	88.81	286	39.76	337	88.99	327
V> 0.5	96.84	95	51.11	45	90	180



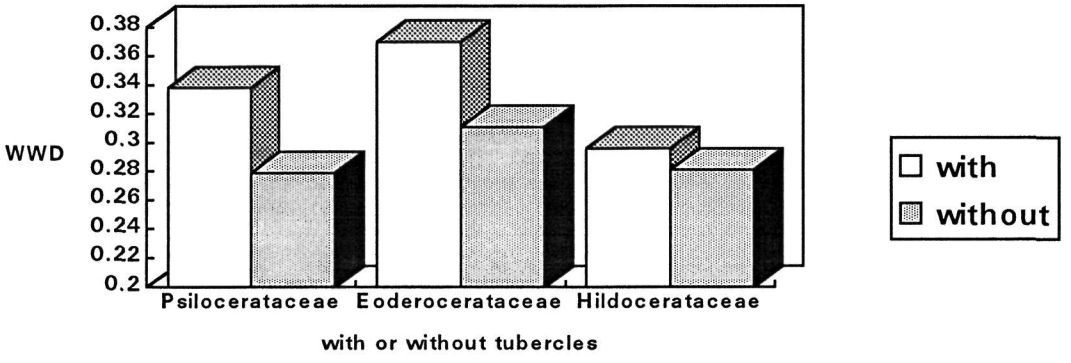
(2) and (3) are related by Buckman's law. It is an interesting question to find out which covariation predominates. Take a subsample of rapidly expanding forms (W>2.2) from superfamily Eoderoceratace ae, calculate the average WWWH and V against tuberculation:

Average	V			WWWH		
Tuberc	M	S	N	M	S	N
nontuberculate	0.438	0.156	30	0.778	0.241	28
unituberculate	0.282	0.164	14	0.970	0.237	18
bituberculate	0.522	0.222	27	1.114	0.241	27

There is an increase in WWWH as the number of tubercles increases, whereas, a correlation is not evident for V. It seems WWWH is directly correlated with tuberculation, and V is more or less related to tuberculation through Buckman's law.

(4) WWD increases if the shells are tuberculate.

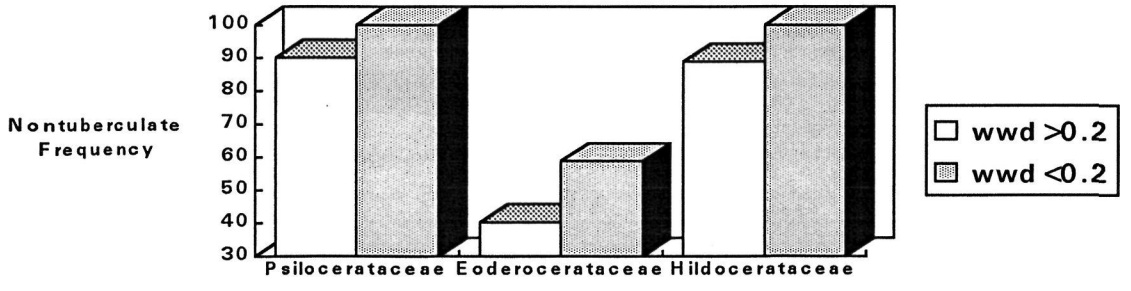
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Tub\WWD	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
with	0.338	0.105	22		0.370	0.118	213		0.296	0.072	59	
without	0.279	0.091	218	0.002	0.311	0.081	127	0.001	0.281	0.065	321	0.05



From this covariation, we can infer:

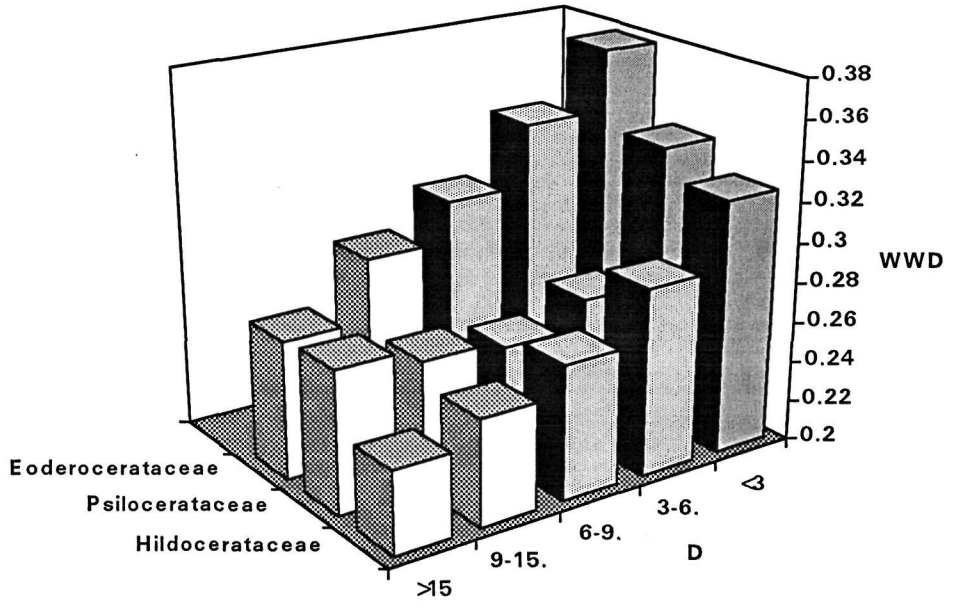
(5) Frequency of shells without tubercles increases if the fineness ratio of the shell is small (WWD<0.2).

Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
WWD\F%	M	N	M	N	M	N
WWD > 0.2	90.06	352	40.27	365	88.93	488
WWD < 0.2	100	29	58.82	17	100	19



(6) WWD decreases as shell diameter(D) increases.

Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
D(cm)\WWD	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
<3	0.340	0.130	63		0.378	0.104	121		0.326	0.086	90	
3-6	0.273	0.073	105	0.001	0.348	0.102	184	0.006	0.293	0.069	202	0.001
6-9	0.260	0.079	53	0.12	0.318	0.109	105	0.009	0.267	0.044	136	0.001
9-15	0.264	0.07	50	No	0.299	0.127	56	0.13	0.253	0.055	85	0.02
>15	0.271	0.065	43	No	0.269	0.078	20	0.14	0.241	0.048	29	0.11

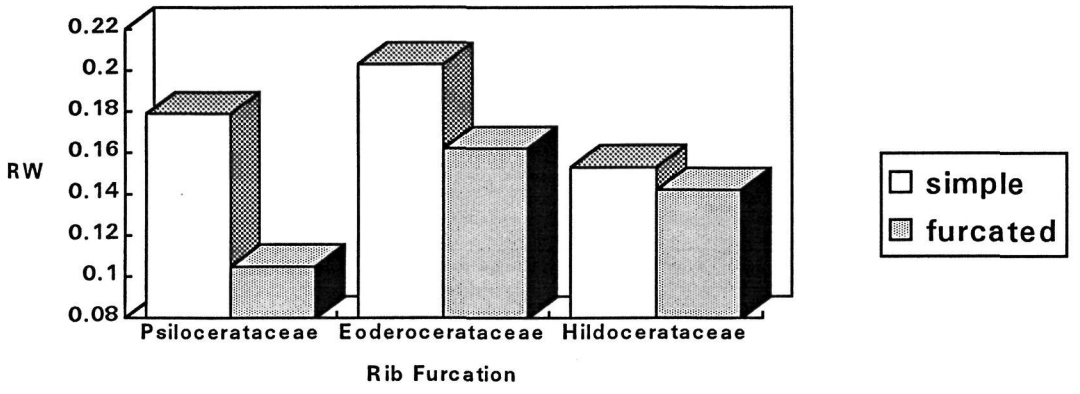


This covariation is most significant when shell diameter D is relatively small, which indicates significant change in WWD during the early stage of the growth.

4.3.5 Ribs

(1) Simple primary ribs are coarser than primary ribs that furcate.

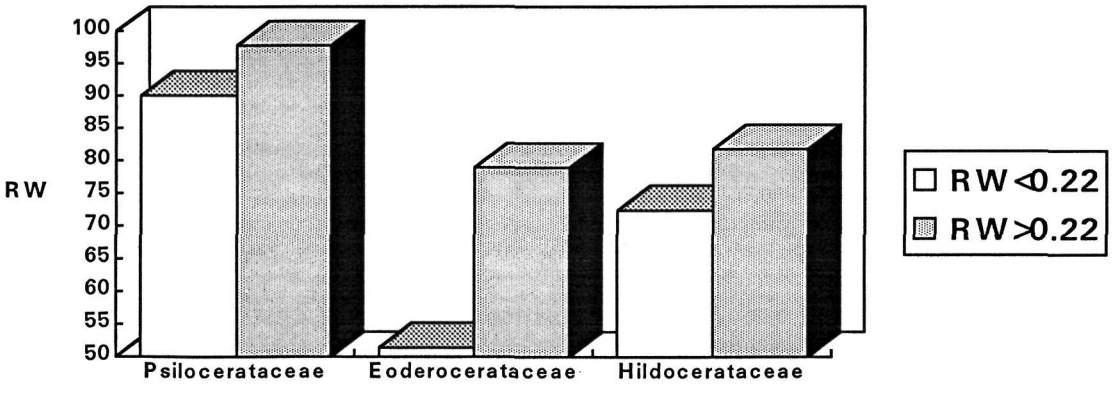
Superfamily	Psiloceratacеae				Eoderoceratacеae				Hildoceratacеae			
Rib\RW	M	S	N	Sig.	M	S	N	Sig.	M	S	N	Sig.
simple	0.179	0.059	300		0.203	0.065	217		0.153	0.056	349	
furcated	0.105	0.042	34	0.001	0.162	0.061	155	0.001	0.142	0.042	153	0.015



The reverse is also evident:

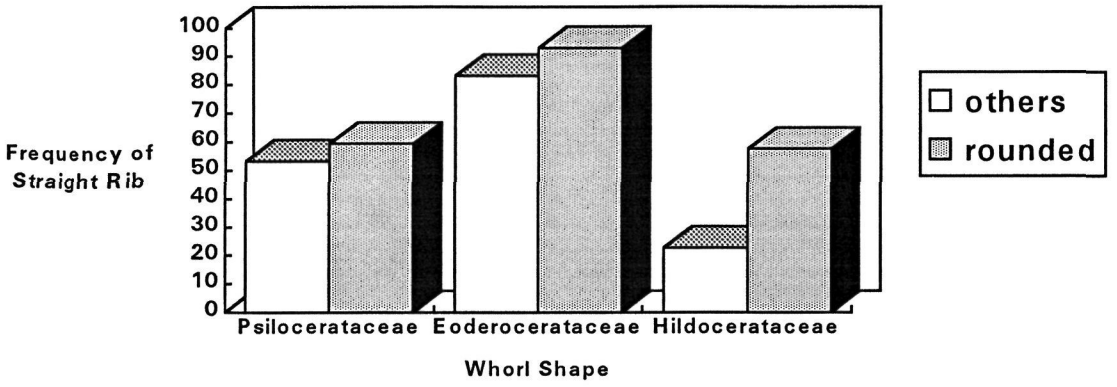
(2) Frequency of shells with simple ribs increases if the shells are coarsely ribbed (Rib Width>0.22).

Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
RW\F%	M	N	M	N	M	N
RW < 0.22	90.07	292	51.38	218	72.43	428
RW > 0.22	97.75	89	78.94	152	81.81	77



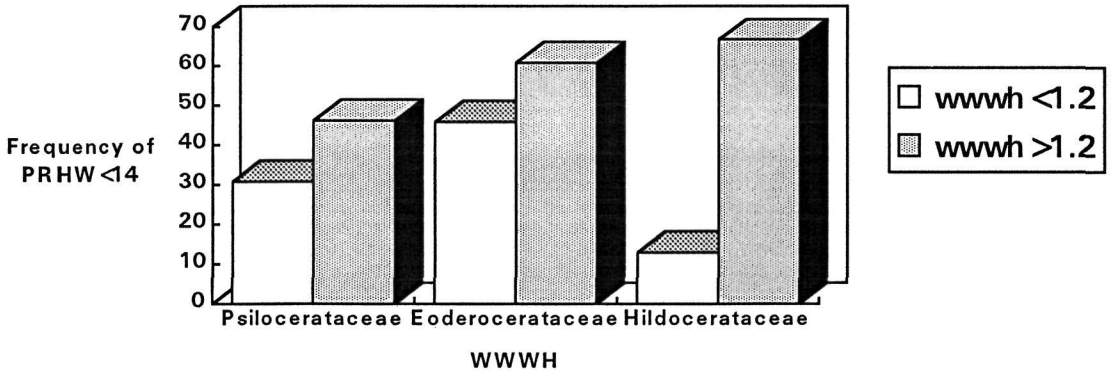
(3) Frequency of shells with straight ribs increases if whorl shape is rounded.

Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
Whorl Shape\F%	M	N	M	N	M	N
others	53.36	298	83.16	297	22.87	494
rounded	59.52	84	93.10	87	57.69	26



(4) Frequency of sparsely ribbed (PRHW<14) forms increases if the shells are depressed (WWWH>1.2).

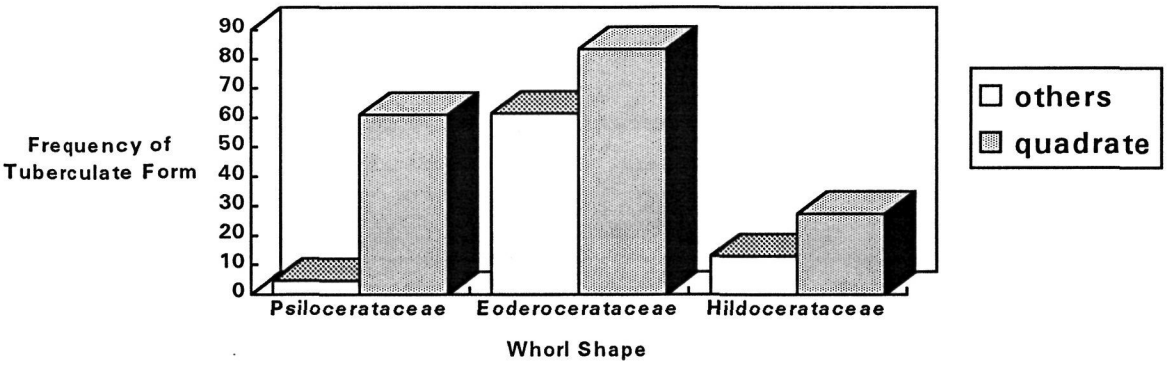
Superfamily	Psiloceratataceae		Eoderoceratataceae		Hildoceratataceae	
WWWH\F%	M	N	M	N	M	N
wwwh < 1.2	30.82	318	45.76	177	12.95	363
wwwh > 1.2	46.15	26	60.78	51	66.66	6



4.3.6 Tuberculation

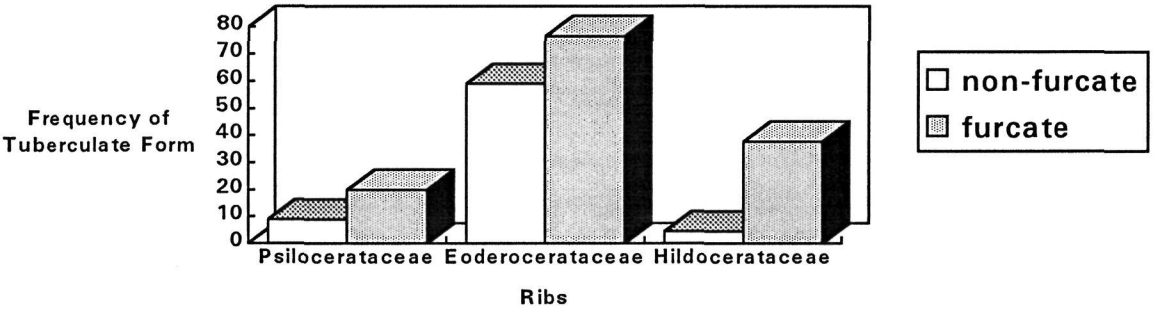
(1) Frequency of shells with tubercles increases if the whorl shape is quadrate.

Superfamily	Psiloceratataceae		Eoderoceratataceae		Hildoceratataceae	
Whorl Shape\F%	M	N	M	N	M	N
others	4.64	345	61.39	303	12.94	456
quadrate	61.11	36	83.54	78	27.45	51



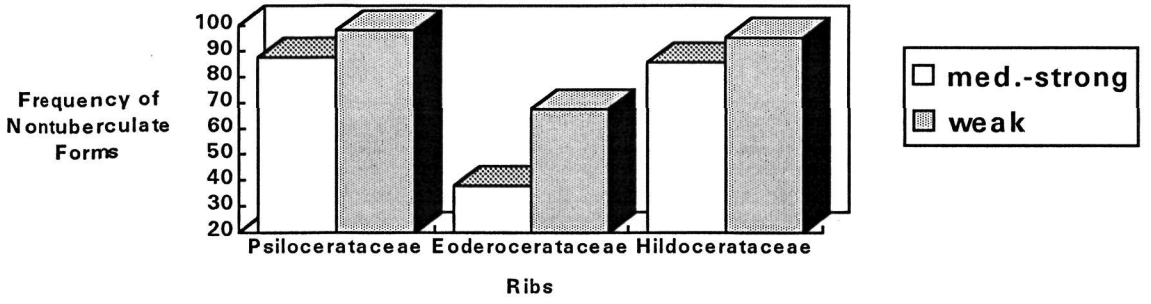
(2) Frequency of shells with tubercles increases if ribs furcate.

Superfamily	Psiloceratacea		Eoderoceratacea		Hildoceratacea	
Rib\F%	M	N	M	N	M	N
non-furcate	8.96	346	58.95	229	4.49	356
furcate	20	35	76.47	153	37.74	151



(3) Frequency of shells without tubercles increases if ribs are weak.

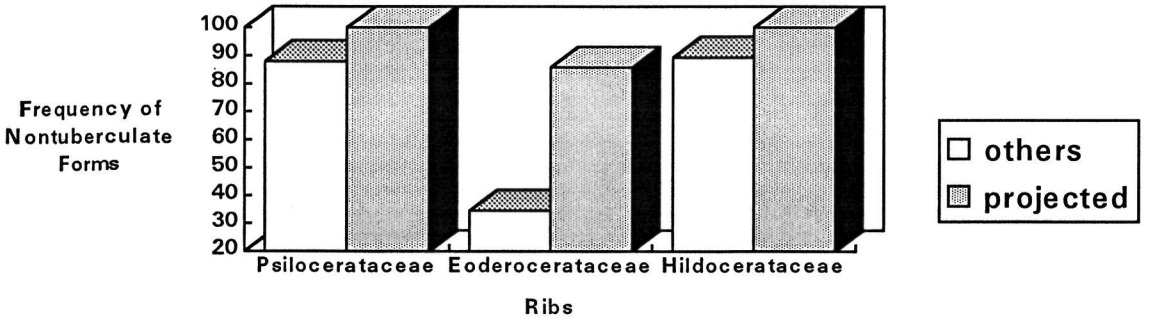
Superfamily	Psiloceratacea		Eoderoceratacea		Hildoceratacea	
Rib\F%	M	N	M	N	M	N
med.-strong	87.64	267	37.76	339	85.62	313
weak	98.24	114	67.44	43	95.36	194



This positive correlation between tuberculation and rib strength may point to the importance of buoyancy control and shell building material supply during growth. Both tuberculate forms and strongly ribbed forms are concentrated in the low W and high U geometric region which provides high buoyancy and carbonate efficiency, whereas nontuberculate and weakly ribbed forms are concentrated more in the high W and low U region which has low buoyancy and carbonate efficiency.

(4) Frequency of shells without tubercles increases if the ribs are projected.

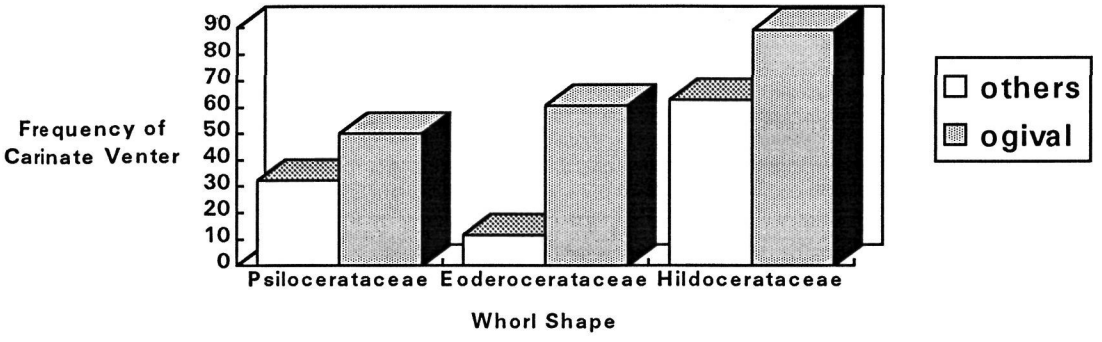
Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
Rib\F%	M	N	M	N	M	N
others	87.89	289	34.53	333	89.20	500
projected	100	92	85.71	49	100	7



4.3.7 Venter

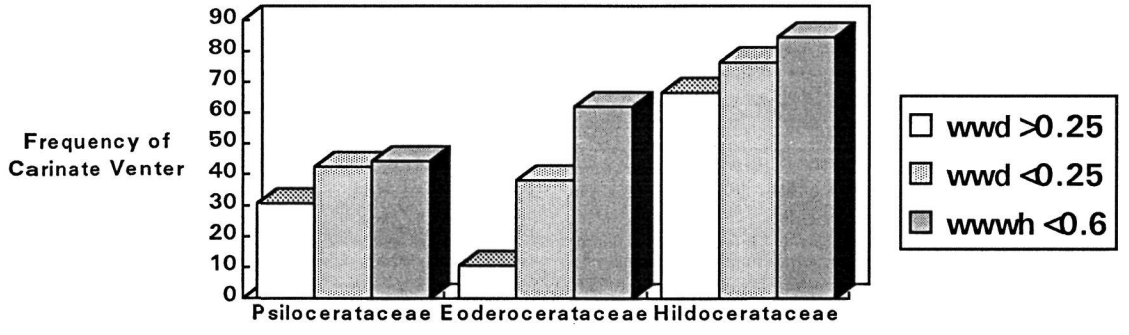
(1) Frequency of shells with carinate venters increases if whorl shape is ogival.

Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
Whorl Shape\F%	M	N	M	N	M	N
others	32.23	363	11.61	353	62.72	405
ogival	50	26	60.52	38	89.13	138



(2) Frequency of shells with carinate venter increases if the shells are highly compressed (WWWH<0.6), or have small fineness ratios(WWD<0.25).

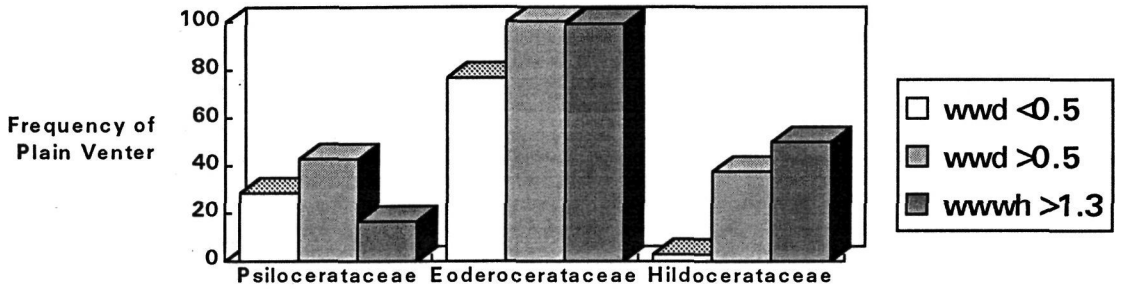
Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
Whorl Shape\F%	M	N	M	N	M	N
WWD > 0.25	30.79	302	10.65	310	66.67	390
WWD < 0.25	42.52	87	38.27	81	76.47	153
WWWH < 0.6	44.44	54	62.06	47	84.67	124



(3)

Frequency of shells with plain venters increases if the shells have high fineness ratios (WWD>0.5).

Superfamily	Psiloceratacеae		Eoderoceratacеae		Hildoceratacеae	
Condition\F%	M	N	M	N	M	N
WWD < 0.5	28.53	382	76.74	344	2.80	530
WWD > 0.5	42.85	7	100	47	37.5	8
WWWH > 1.3	16.66	18	99.08	108	50	6



One interesting point here is that depressed shells (WWWH>1.3) do not show an increase in frequency of plain venters for all three superfamilies, whereas high fineness ratios do. It may be that WWD is directly correlated with ventral morphology, and WWWH is related through WWD.

4.3.8 Summary & Speculations

It is generally agreed among ammonite workers that the shell of ammonite served two prime functions: as a protection for the soft parts and as a hydrostatic device. Ornamentations such as ribs and tubercles were highly functional features of the shell, as is demonstrated by the common covariations documented above. Their significance has been thought of as (Kennedy and Cobban 1976): (1) Mechanical; (2) Protective, serving for both physical protection and as an aid to camouflage; (3) An aid to stability; (4) A means of regulating buoyancy; (5) A means of improving streamlining; (6) Sexually selected display features; (7) Sensory.

The inter-relationships of shell expansion rate, degree of involution, whorl shape, ribs, tubercles, and venter have been studied using Ammon database. The major covariation patterns are:

(a) The faster expanding shells tend to be more involute, the involute shells are more compressed than evolute shells, and the compressed shells have finer ribs. Possible interpretations for this covariation, also known as Buckman's Law, are the restrictions imposed by overlap ratio and relative area of the whorl (equations 1, 2 in section 4.2), buoyancy control and carbonate efficiency (Fig. 4-29).

(b) Frequency of shells with tubercles increases if ribs furcate. Westermann (1978) suggested that a point of rib division would appear to be a mechanically good location for lateral tubercles which may exert localized stress on the shell, this would explain the repeated pattern of rib furcation and tuberculation during ontogeny that bundling ribs and prominent tubercles on the inner whorls fade to simple ribs at maturity when the wall has gained sufficient thickness. The division of ribs may also be due to the increase of circumference with increasing radius (Westermann 1978).

(c) Tubercles of compressed shells tend to be on the lower flank whereas tubercles of depressed shells tend to be on the upper flank, which may be a reflection of Westermann's observation that tubercles are usually situated on the shoulders or the maximal whorl width which are often in the lower flank of compressed shells. This would also explain the fact that the frequency of shells with tubercles increases if whorl shape is quadrate.

(d) Shells with sinuous ribs are more involute and more compressed than shells with straight ribs.

(e) There is a positive correlation between tuberculation and rib strength which may point to the importance of buoyancy regulation and shell building material supply during growth. Both tuberculate forms and strong ribbed forms are concentrated on low W and high U geometric region which provides high buoyancy and carbonate efficiency whereas

nontuberculate and weak ribbed forms are concentrated more on high W and low U region which has low buoyancy and carbonate efficiency;

(f) As expected, the frequency of shells with a keels increases in shells that are compressed, or have a small fineness ratio.

It should be noted that the observed covariation patterns may be unique to the data set used except for Buckman's Law which has been recognized in other ammonoid stocks. To be confident that these covariation patterns are due to functional rather than phylogenetic constraints would require examining earlier and later ammonoid records.

4.4 Ontogenetic Variation

The geometric data used in the previous analyses were gathered from specimens with shell diameters larger than 4 cm to reduce the impact of ontogenetic variation. However, the problem of ontogenetic change cannot be ignored and an attempt is made here to describe such changes using measurements from different shell diameters for each species. The sample is restricted to species with at least 6 occurrences in the database and the shell diameters of the specimens have a large enough range to show ontogenetic variations, if present. Because the number of specimens for each species in the database is limited, they are not subdivided into groups based on geographic areas. This may blur true ontogenetic change or create false patterns if morphological variations for different geographic regions are large. The limitations of the data should be kept firmly in mind while interpreting the results of these inquires.

4.4.1 Umbilical Ratio (U)

"Ontogenetic variations have been found in practically all ammonoid groups. In some instances the ontogenetic change has the effect of making the shell more evolute but in the majority of cases the change is toward involute" (Raup 1967). This generalization is probably true for Paleozoic ammonoids, especially goniatites, but Lower Jurassic ammonite data show otherwise: 235 out of 670 species studied become more evolute during ontogeny, only 49 species tend to be more involute and the rest remain more or less constant. Ontogenetic change is determined by (1) visually examining the U-D plots like Fig. 4-31, 4-32; (2) computing the difference between the U value measured at the largest diameter of the species and the one at the smallest diameter. If it is larger than the within species mean standard deviation of U which is 0.043 in this data set, a significant change on U during ontogeny is inferred. The top ten species to undergo significant either increasing (+) or decreasing (-) ontogenetic change in U are: *Fanninoceras carlottense* (-) , *Fanninoceras disciforme* (-),

Fanninoceras fannini (-), *Hildoceras bifrons* (+), *Haugia jugosa* (+), *Psiloceras planorbis* (+), *Psiloceras johnstoni* (+), *Vermiceras scylla* (+), *Aveyroniceras acanthoides* (+), *Dubariceras silviesi* (+).

It is noteworthy that increase in U does not necessarily indicate less whorl overlap because it could also be the result of a reduction in shell expansion rate during ontogeny. This is common in the genus *Vermiceras*.

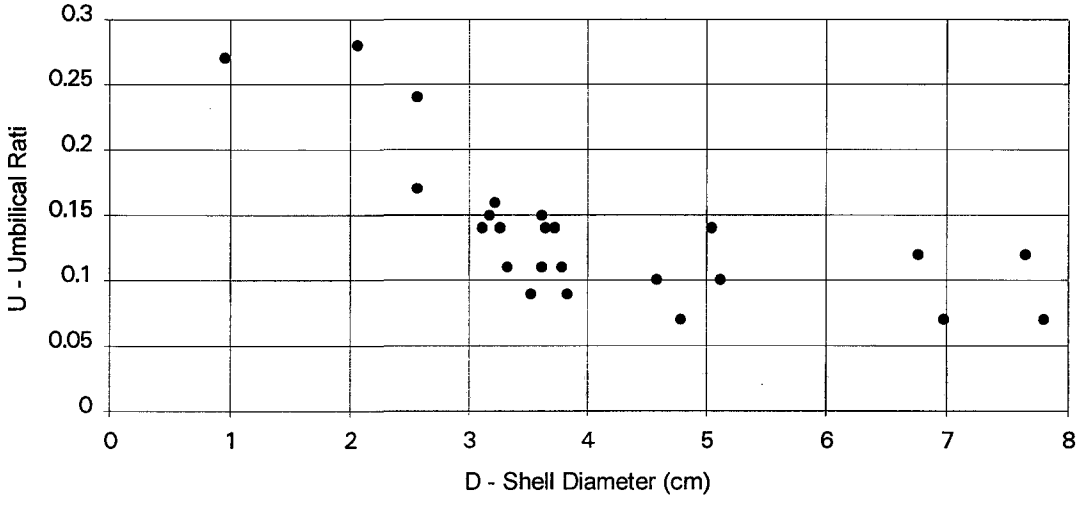


Figure 4-31 Plot of ontogenetic change in U for *Fanninoceras fannini*

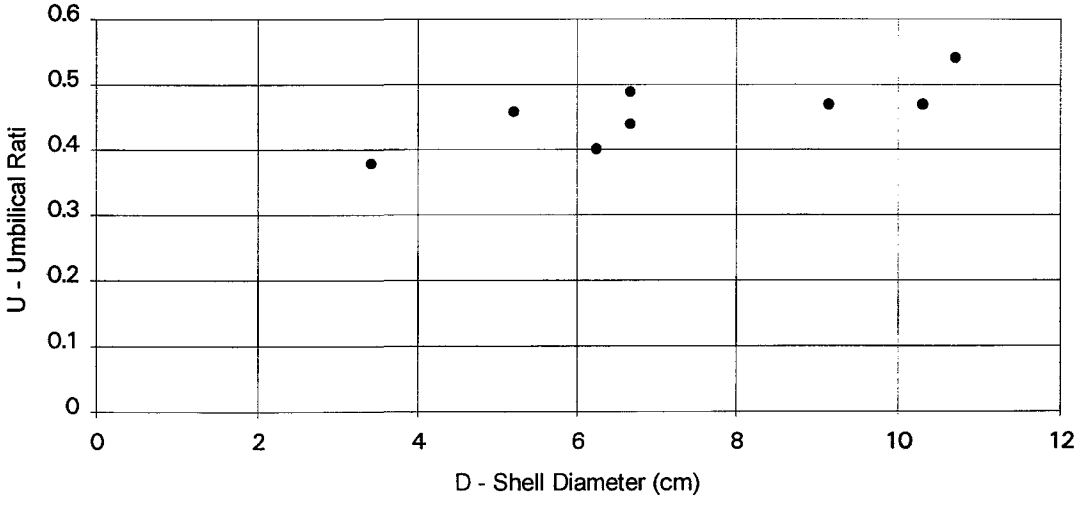


Figure 4-32 Plot of ontogenetic change in U for *Dubariceras silviesi*

4.4.2 Shell Expansion Rate (W)

Using the same method as described above, it was found that 181 out of 670 species studied show an expansion rate decrease during ontogeny, only 40 species show increase and the majority remain more or less constant. The top ten species that undergo significant ontogenetic change on W are: *Pleuroceras solare* (-), *Hammatoceras speciosum* (-), *Dubariceras silviesi* (-), *Phricodoceras taylori* (+), *Pseudogrammoceras muelleri* (-), *Arnioceras ceratitoides* (-), *Metaderoceras talkeetnaense* (-), *Vermiceras scylla* (-), *Vermiceras rursicostatum* (-), *Haugia variabilis* (-). Fig. 4-33 shows that expansion rate decreases during ontogeny for *Dubariceras silviesi* and the negative correlation between U and W is evident by comparing Fig. 4.33 with Fig. 4-32. Fig. 4-34 shows that expansion rate increases during ontogeny for *Phricodoceras taylori.*

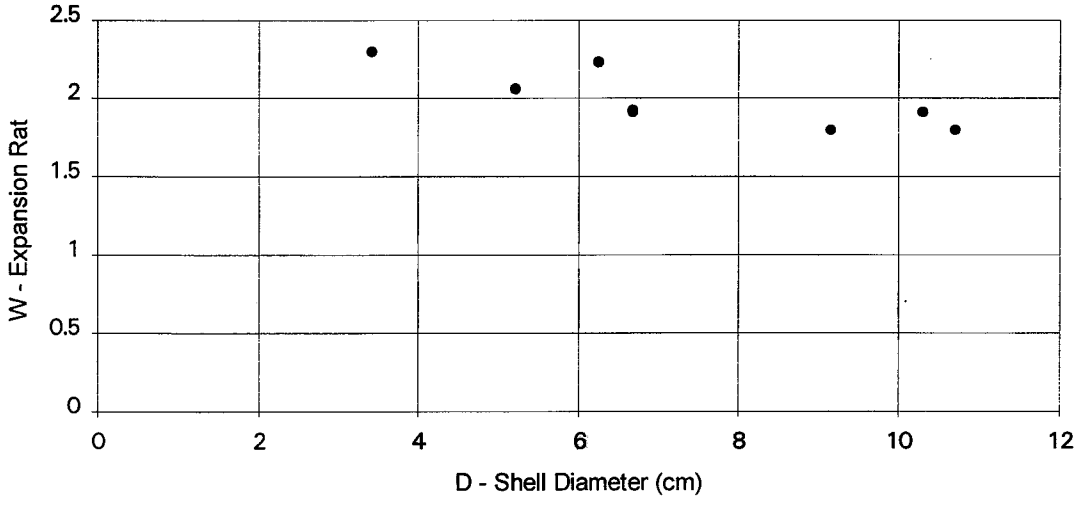


Figure 4-33 Plot of ontogenetic change in W for *Dubariceraas silviesi*

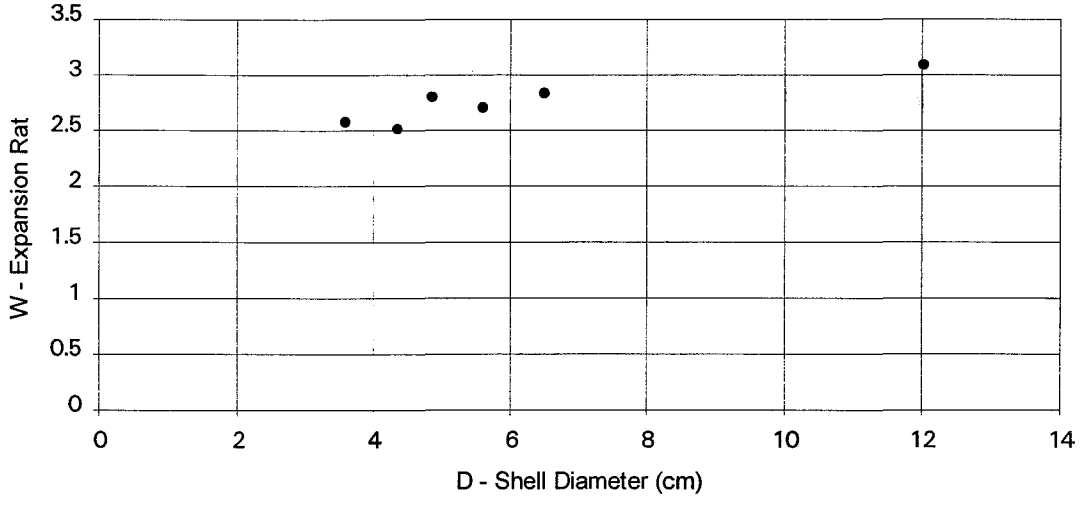


Figure 4-34 Plot of ontogenetic change in W for *Phricodoceras taylori*

4.4.3 Ratio of Whorl Width to Whorl Height (WWWH)

It is found that 224 out of 623 species studied become more compressed during ontogeny, only 68 species become more depressed and the rest remain more or less constant. The top ten species in the database that undergo significant ontogenetic change in WWWH are: *Amaltheus margaritatus* (-), *Androgynoceras maculatum* (-), *Aegoceras luridum* (+), *Dactylioceras anguinum* (-), *Phricodoceras urcuticum* (-), *Phricodoceras taylori* (-), *Prodactylioceras davoei* (-), *Coeloceras incertum* (+), *Phymatoceras rude* (-), *Phymatoceras tumefacta* (-). Fig. 4-35 shows that *Phricodoceras taylori* becomes more compressed during ontogeny and the negative correlation between W and WWWH is evident by comparing Fig. 4.35 with Fig. 4-34. Fig. 4-36 shows that *Aegoceras luridum* becomes more depressed.

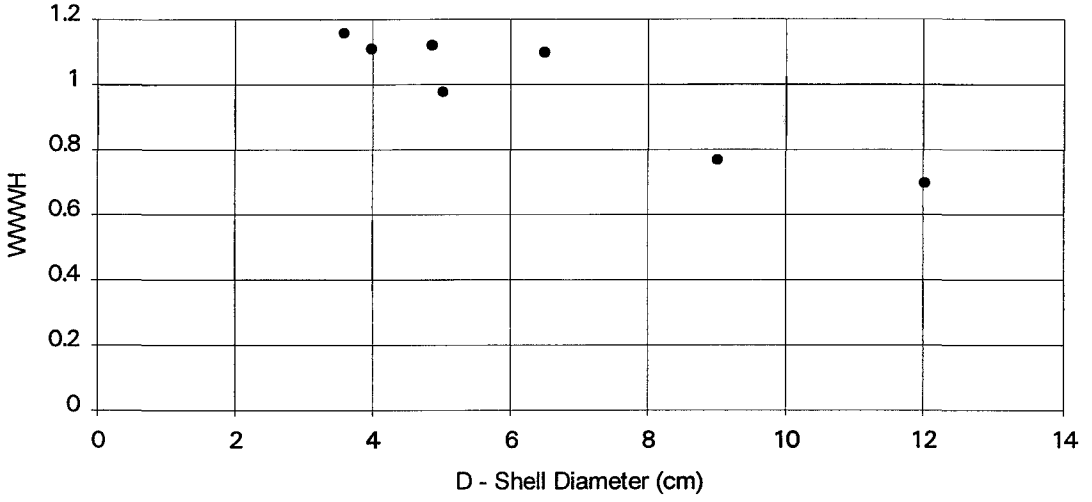


Figure 4-35 Plot of ontogenetic change in whorl shape for *Phricodoceras taylori*

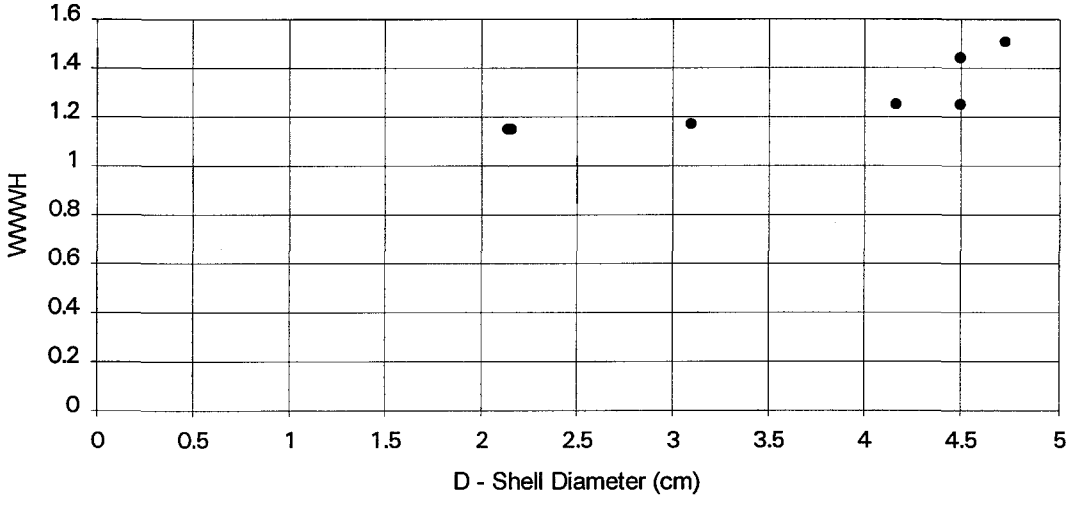


Figure 4-36 Plot of ontogenetic change in whorl shape for *Aegoceras luridum*

4.4.4 Primary Rib Density (PRHW)

It is found that primary rib density for virtually all species remains more or less constant (61%) or increases (39%) during ontogey, none show significant decrease in primary rib density during ontogeny although the weakening of ornamentation is common. The top ten species that undergo significant significant ontogenetic change on primary rib density are: *Amaltheus conspectus, Amaltheus margaritatus, Amaltheus extremus, Arieticeras algovianum, Harpoceras chrysanthemum, Harpoceras falciferum, Protogrammoceras praecurionii, Pseudolioceras lythense, Reynesoceras italicum, Schlotheimia angulata.* Fig. 4-37 shows that the rib density of *Amaltheus conspectus* increases during ontogeny and Fig. 4-38 shows the rib density change for *Arieticeras algovianum.*

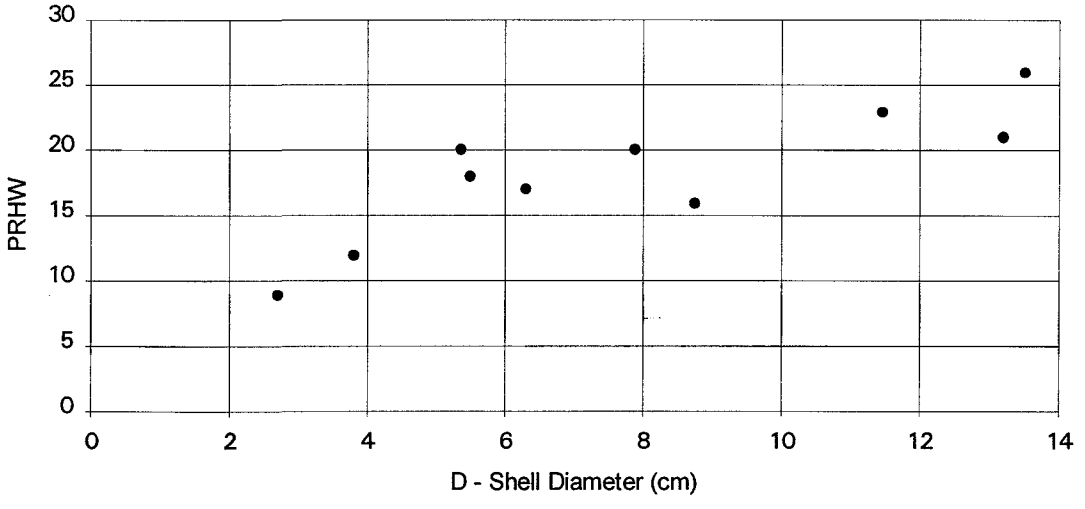


Figure 4-37 Plot of ontogenetic change in PRHW for *Amaltheus conspectus*

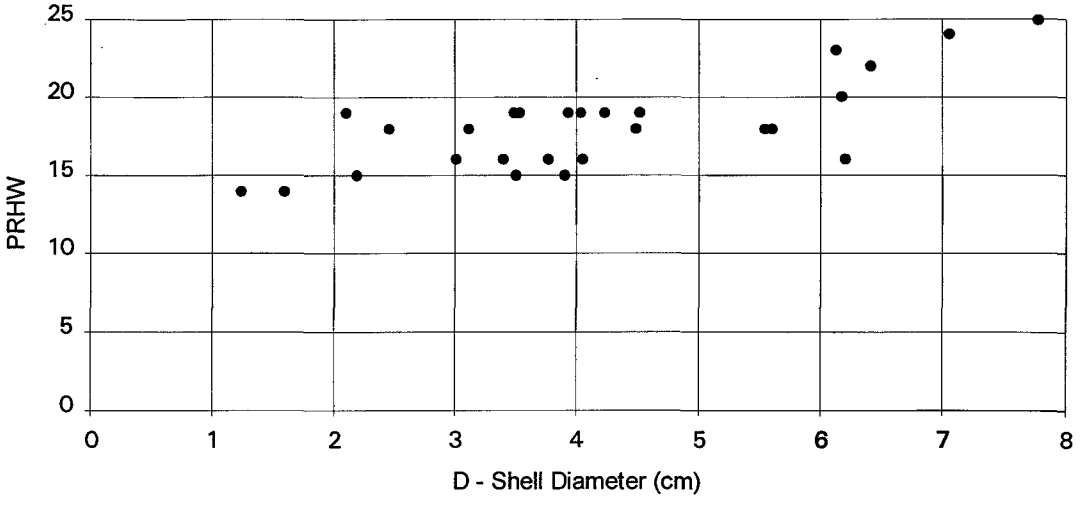


Figure 4-38 Plot of ontogenetic change in PRHW for *Arieticeras algovianum*

4.4.5 Superfamily

The difference of parameter values on the largest and smallest specimens is used to represent the degree of ontogenetic variation for each species with more than four specimens. Table 4-3 shows the average ontogenetic changes of species within superfamilies. the Psiloceratacеae has the largest ontogenetic variation in U whereas the Eoderoceratacеae shows greater ontogenetic variation in WWWH. Members of the Hildoceratacеae are more likely to undergo ontogenetic change on W

Table 4-3 Average ontogenetic changes of species within superfamilies

Superfamily	U	W	WWWH
Psiloceratacеae	0.071	0.165	0.128
Eoderoceratacеae	0.049	0.173	0.205
Hildoceratacеae	0.053	0.186	0.135

Chapter 5 Changes through Time

5.1 Introduction

Evolutionary change is not evenly distributed over time but is concentrated in episodes of evolutionary radiation as clearly indicated by the fossil record (Foote 1991). For ammonites the early Jurassic represents the most important episode because the ammonoid stem barely escaped extinction at the close of the Triassic (Donovan, Callomon and Howarth 1981). Yet, despite its significance, a limited number of approaches have been used to study this diversification, most notably the analysis of taxonomic data (Hallam 1987). The analysis of diversification by 'taxon counting' assumes either that morphologic diversity can be measured by taxonomic diversity or that the number of taxa reflects the number of objectively discernible morphotypes (Foote 1991).

We know that taxonomic data and morphologic data do correlate, but taxonomic and morphologic approaches are not simply redundant (Foote 1991). If taxa are consistently defined, then taxonomic data can tell us about the number of biological units at a given time. But if we want to know the nature of these units, how they originate, how they evolve once established and how discrete they are from each other then morphological data are clearly necessary. "Since form represents the raw data of paleobiology, it is important to document significant events in the history of life from the standpoint of morphology" (Foote 1991).

This study represents an attempt to analyze in detail the Lower Jurassic ammonite record, in terms of morphologic and taxonomic diversity and their relationship at 20 successive levels (zones) through an approximately 30-Ma interval (Harland et al. 1990).

Table 5-1 Lower Jurassic zonal scheme for northwest Europe

Series	Age at base	Stage	Zone
Middle Jurassic	179 Ma	Aalenian	Opalinum
Lower Jurassic	187 Ma	Toarcian	Levesquei
			Thouarsense
			Variabilis
			Bifrons
			Falciferum
			Tenuicostatum
	196 Ma	Pliensbachian	Spinatum
			Margaritatus
			Davoei
			Ibex
			Jamesoni
	203 Ma	Sinemurian	Raricostatum
			Oxynotum
			Obtusum
			Turneri
			Semicostatum
			Bucklandi
	208 Ma	Hettangian	Angulata
			Liasicus
			Planorbis

5.2 Stratigraphic Division

The traditional stratigraphic division of the Lower Jurassic into 4 stages and 20 primary standard zones is adopted here with the age at the base of each stage taken from Harland et al. (1990)(Table5-1).

5.3 Choice of Characters

I have attempted to represent ammonite morphologic diversity in terms of basic shell geometry and ornamentation, which are the major describable morphological characters for species identification. Basic shell geometry includes parameters: W, U, WWWH; as well as

venter for each species. Ornamentation includes PFORM (primary rib form), PRHW (primary rib density), FURC (a combination of pattern and position of rib furcation), and tuberculation (a combination of pattern and position of tubercles). All parameters are defined in Chapter 2. Aperture shape is not included in this study due to limited data available. This study is restricted to external morphology rather than internal characters, such as the septal suture. In strong contrast to earlier systems, the septal sutures of Jurassic ammonites are seldom useful for defining major groups and are hardly ever employed at lower levels of classification (Donovan, Callomon and Howarth 1981).

The data used in this study comprise observations made on 6784 specimens, representing 15 families, 179 genera, and 1319 species. This comprises virtually complete coverage of the ammonite genera known to occur in the Lower Jurassic as catalogued by Donovan, Callomon and Howarth (1981).

5.4 Geometry through Time

It is tempting to search the biometrical data for evidence of correlation between geometry and stratigraphic time. W-U plots have been made for the four stages. A bimodal distribution characterizes the Hettangian (Fig. 5-1), with one peak corresponding to evolute and slowly expanding forms, and another peak corresponding to moderately involute forms. Sinemurian ammonites fully explored the region of the first peak (Fig. 5-2) but the W-U distribution for the Pliensbachian concentrates on the second more involute peak (Fig. 5-3), a trend that continues for the Toarcian (Fig. 5-4).

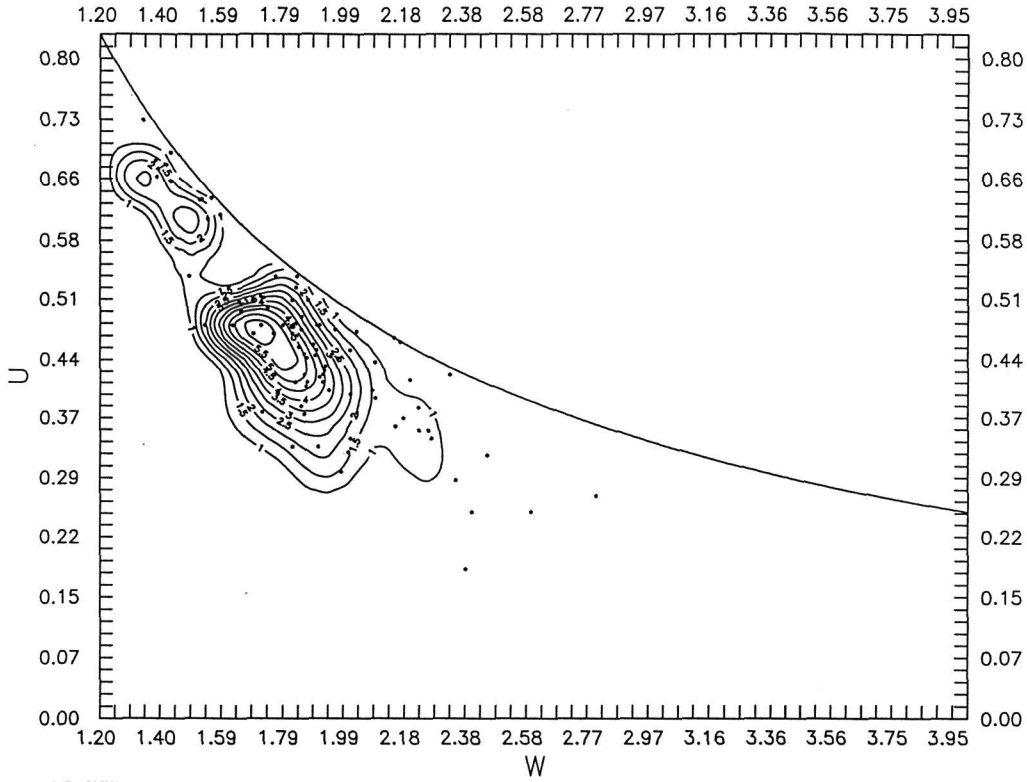


Figure 5-1 Density diagram for Hettangian ammonites
(101 species)

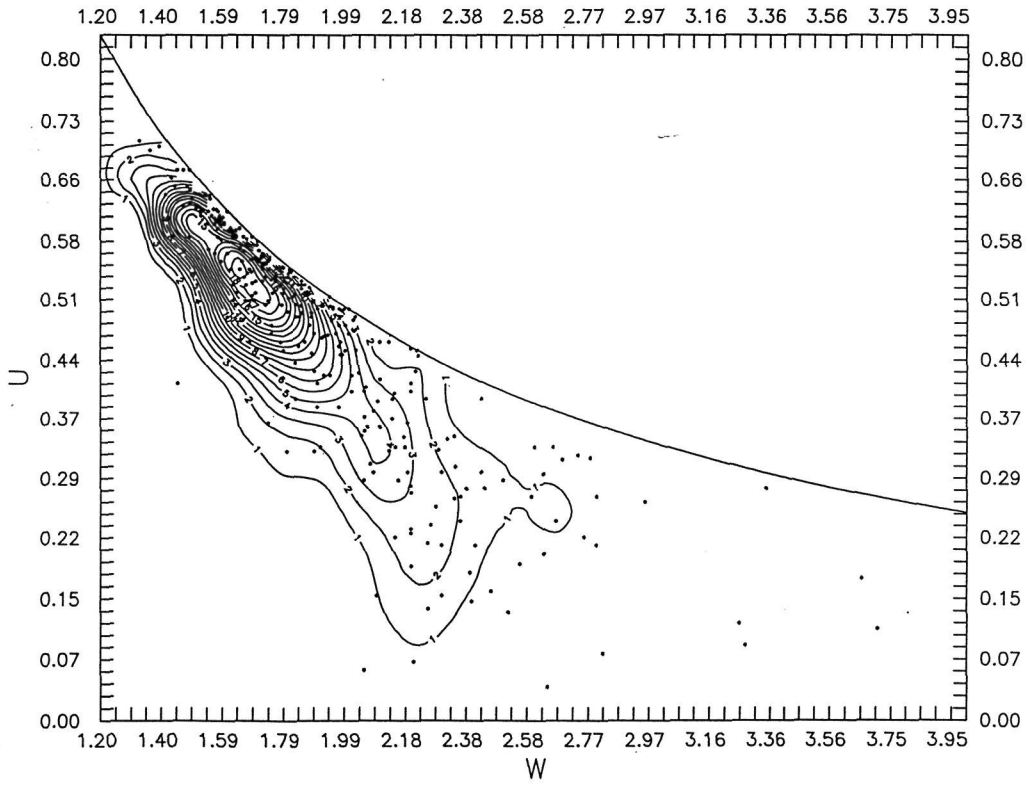


Figure 5-2 Density diagram for Sinemurian ammonites
(366 species)

1993). I will use as a starting point the sum of the morphological range as a measure of morphological diversity.

Figure 5-7A shows the morphological diversity for each of the 20 successive zones using the four basic characters of shell geometry. Each bar in the graph represents a zone and each shaded portion represents the morphological diversity for a character with all characters having the same weight. Figure 5-8A adds ornamentation to the four shell geometry characters. In Chapter 4, I demonstrated that covariation is widespread in ammonites and covariation will introduce a certain amount of redundancy into the measures of morphologic diversity, as is evident in Table 5-2. The shaded boxes in Table 5-2 indicate pairs of characters with correlation coefficients greater than 0.5. the most significant correlation occurs among the three parameters describing rib density, form and furcation (coefficient > 0.8) which are therefore combined (averaged) into one parameter (RIB) for this study. The covariations among W, U, and WWWH demonstrated by Eqs (1) and (2) in section 4.2 are also evident here. As can be expected, there is a significant correlation between ornamentation and whorl shape, and a correlation between rib furcation and tuberculation. To eliminate the redundancy caused by covariations, a multiple linear regression was performed for each parameter using the rest of the parameters. The ratio of residual to total variance which indicates the degree of independence of each parameter with respect to the rest (Table 5-3) can then be used as a weight in the summing up of the MRDI values for each zone as shown in Fig.5-7B, 5-8B.

Table 5-2 MRDI correlation matrix (Shaded values have correlation coefficients > 0.5)

	W	U	WWWH	VENTER	PRHW	PFORM	FURC	TUBERC
W	1							
U	0.635982	1						
WWWH	0.559241	0.6933	1					
VENTER	0.066713	0.421967	0.252447	1				
PRHW	0.096465	0.208958	0.565855	0.156352	1			
PFORM	0.365184	0.326135	0.601697	0.19651	0.841871	1		
FURC	0.203162	0.387015	0.641858	0.257381	0.922446	0.842275	1	
TUBERC	0.440936	0.366407	0.492782	0.119988	0.34703	0.521663	0.539111	1

5.5 Taxonomic Diversity

Ammonites have served as classic examples of biostratigraphic index fossils. Their fossil record is exceedingly rich and their taxonomy is mature in the sense that it no longer shows an exponential increase in description of new taxa each year (rather, the reverse is happening) which makes them well suited for taxa counting (Ward 1983). The data sets become more comprehensive moving up the taxonomic and stratigraphic hierarchy as sampling failure becomes less likely. Even so, there is no reason to assume any non-random bias in sampling even down to the level of species and zones, so that the data can be confidently used to assess relative, if not absolute, changes in diversity and turnover through time. The method of taxonomic diversity analysis adopted here is similar to that presented for a number of invertebrate groups in the Lower Jurassic of northwest Europe by Hallam (1987). The diversity graphs are simple plots of numbers of taxa for each successive zone; the graphs of times of family first and last appearances are self explanatory. The major difference is my greatly expanded database which enables me to work at the species as well as the generic level.

As illustrated in Fig. 5-5A, species diversity shows a slow increase from the oldest Planorbis Zone to the Bucklandi Zone, followed by a slight decline to the Turneri Zone, and rapid rise to a middle Pliensbachian maximum followed by a spectacular crash through the Tenuicostatum Zone to a Falciferum Zone minimum. Thereafter there is a further rise to the Bifrons Zone to match Sinemurian but not Pliensbachian levels followed by a decline to the Thouarsense Zone and a final rise to the Levesquei Zone. It is evident from Fig. 5-5B,C that diversity curves for genera and families closely resemble the one for species. The overall similarity among the three diversity curves suggests that they reflect a common underlying pattern which supports the general claim made by Bambach and Sepkoski (1992) that large-scale spatial and temporal diversity patterns at the level of species, genus, and family are well correlated. It is interesting to note that diversity peaks first at the family level in the Ibex

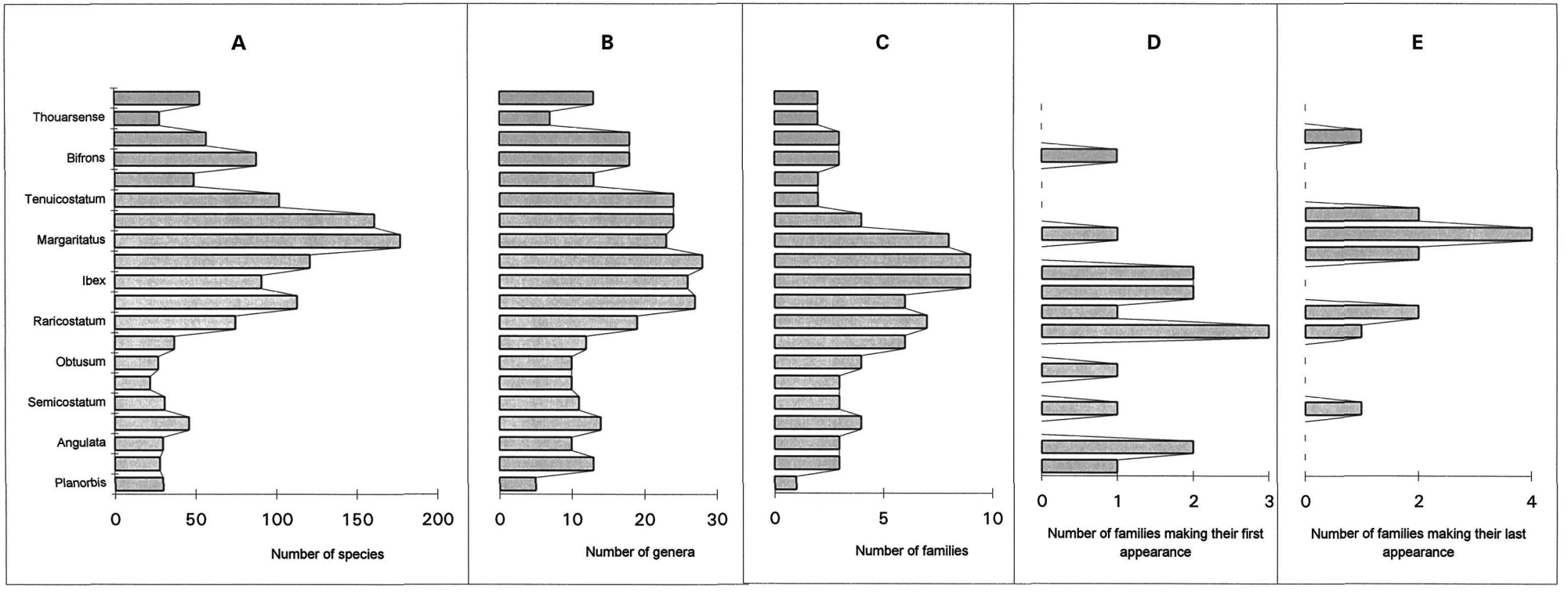


Figure 5-5 Lower Jurassic Ammonite Taxonomic Diversity Changes through Time

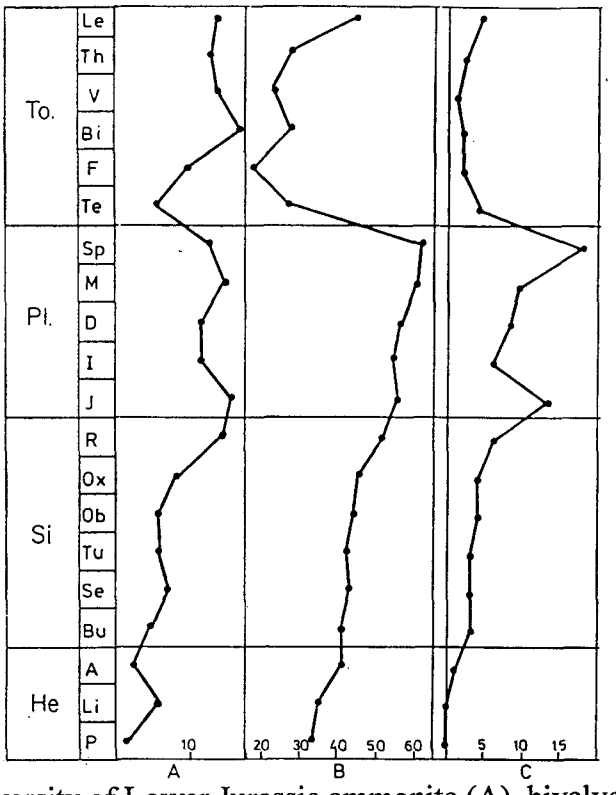


Figure 5-6 Diversity of Lower Jurassic ammonite (A), bivalve (B), and brachiopod (C) diversity. Number of genera for ammonites, number of species for the rest (Hallam 1987).

Zone, followed by generic data one zone later, and finally species which implies significant delay in radiation for lower levels of ammonite taxa. The observed genus diversity change pattern is in general agreement with the one shown for ammonite genera (Fig. 5-6A) by Hallam (1987) where there is a more or less steady rise in diversity from a very low level in the early Hettangian through to the Pliensbachian followed by a marked decline into the early Toarcian after which it tends once more to increase. The major difference between the two is that ammonite genera reach their highest diversity level in the Pliensbachian instead of the middle Toarcian in my data set. My ammonite species diversity peak coincides with Hallam's bivalve and brachiopod diversity peaks of the late Pliensbachian (Fig. 5-6B, C). Fig. 5-5 D, E shows the Oxynotum Zone has the highest family origination rate and one of the lowest species diversity. The late Sinemurian and early Pliensbachian also have high family origination rates and relatively low species diversities; species reach their highest diversity

level in the late Pliensbachian which happens to have the highest family extinction rate and an almost zero origination rate. Raup and Sepkoski (1986) identified the Pliensbachian as a time of mass extinction at the family level that was part of a 26 Ma periodic series. The average ammonite family extinction rate for the Upper Pliensbachian is 50% which is significantly higher than 7% -- the average for the rest of Lower Jurassic zones (Fig. 5-5). The claim of periodic extinction has provoked speculation about an ultimate cause, such as sea level changes (Hallam 1987), climatic shifts, volcanic eruptions (Officer et al. 1987) or extraterrestrial events (Raup 1986). A central issue has been how rapidly the extinction occurred. It is evident from Fig. 5-5C, E that the Pliensbachian extinction event was gradual for ammonites at the family level, it started at Davoei Zone with a family extinction rate of 22% and followed by high extinction rate of 50% in the Margaritatus Zone and the Spinatum Zone.

5.6 Morphological Diversity

5.6.1 Morphological Range Diversity Index (MRDI)

The variance seems to be a good choice to represent morphological diversity but unfortunately, most paleobiological data sets include both quantitative and qualitative data and there is no meaningful distance measurement between nominal data, such as the pattern of ribs. As a result, the range of forms is generally accepted as a measure of morphological diversity. For qualitative data, the MRDI is the percentage of occupied character states. For quantitative data, MRDI is the difference between the maximum and minimum of the morphologic characters with normalization so that the MRDI value always falls between 0 and 1. Van Vale (1974) discusses the multivariate generalization of the univariate range, suggesting two measures: the sum of the univariate range of morphological variables and the product of univariate ranges. The latter represents the volume of morphospace occupied, but this quantity vanishes or nearly vanishes if any of the ranges is zero or nearly zero (Foote 1993). I will use as a starting point the sum of the morphological range as a measure of

morphological diversity. Figure 5-7A shows the morphological diversity for each of the 20 successive zones using the four basic characters of shell geometry. Each bar in the graph represents a zone and each shaded portion represents the morphological diversity for a character with all characters having the same weight. Figure 5-8A adds ornamentation to the four shell geometry characters. In Chapter 4, I demonstrated that covariation is widespread in ammonites and covariation will introduce a certain amount of redundancy into the measures of morphologic diversity, as is evident in Table 5-2. The shaded boxes in Table 5-2 indicate pairs of characters with correlation coefficients greater than 0.5. the most significant correlation occurs among the three parameters describing rib density, form and furcation (coefficient > 0.8) which are therefore combined (averaged) into one parameter (RIB) for this study. The covariations among W, U, and WWWH demonstrated by Eqs (1) and (2) in section 4.2 are also evident here. As can be expected, there is a significant correlation between ornamentation and whorl shape, and a correlation between rib furcation and tuberculation. To eliminate the redundancy caused by covariations, a multiple linear regression was performed for each parameter using the rest of the parameters. The ratio of residual to total variance which indicates the degree of independence of each parameter with respect to the rest (Table 5-3) can then be used as a weight in the summing up of the MRDI values for each zone as shown in Fig.5-7B, 5-8B.

Table 5-2 MRDI correlation matrix (Shaded values have correlation coefficients > 0.5)

			WWW	VENTE	PRH	PFOR	FUR	TUBER
W								
U	0.63598							
WWWH	0.55924	0.693						
VENTER	0.06671	0.42196	0.25244					
PRHW	0.09646	0.20895	0.56585	0.15635				
PFORM	0.36518	0.32613	0.60169	0.1965	0.84187			
FURC	0.20316	0.38701	0.64185	0.25738	0.92244	0.84227		
TUBERC	0.44093	0.36640	0.49278	0.11998	0.3470	0.52166	0.53911	

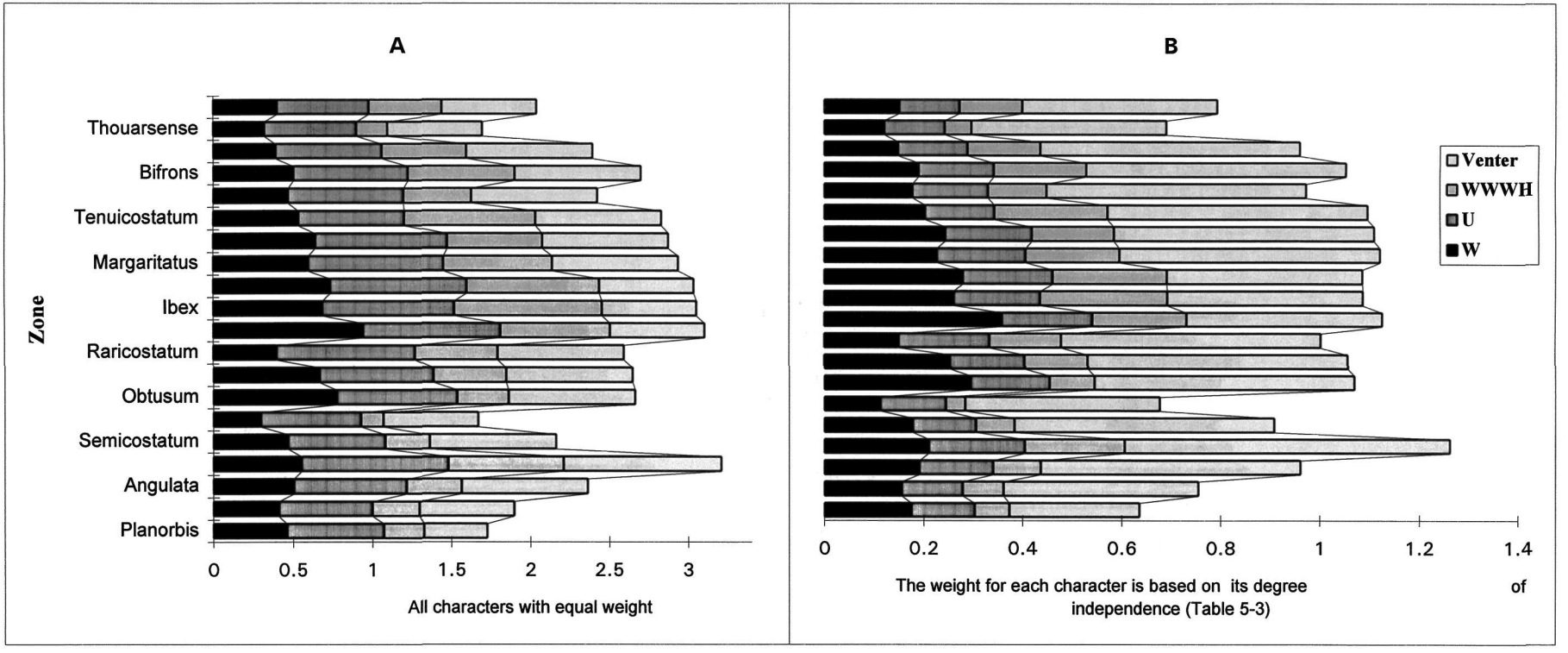


Figure 5-7 The Accumulation of Morphological Range Diversity Index for Basic Shell Geometry

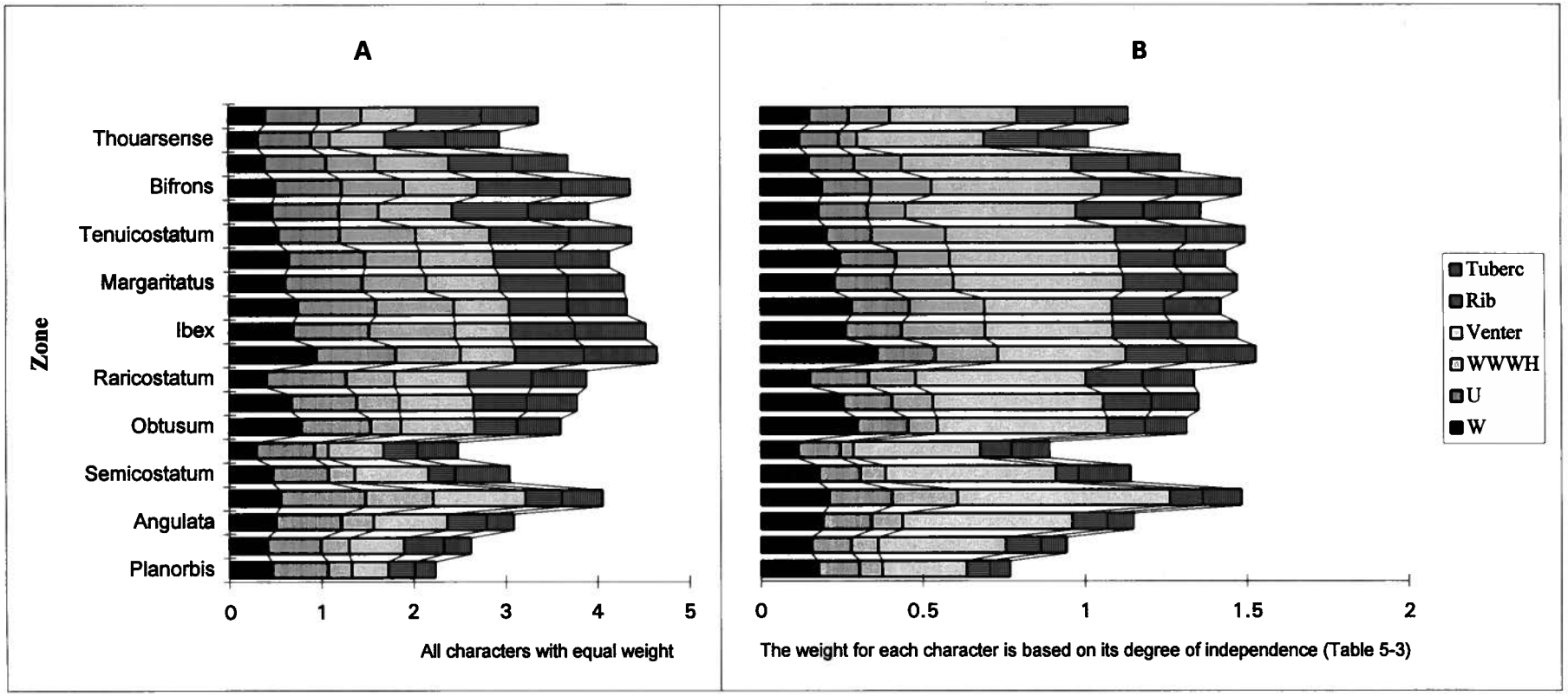


Figure 5-8 The Accumulation of Morphological Range Diversity Index for All Parameters

Table 5-3 Measure of independence--residuals from multiple linear regression

Sum of Squares	W	U	WWW	VENTER	RIB	TUBERC
Total	0.320	0.189	0.731	0.121	0.512	0.341
Residual	0.197	0.050	0.278	0.231	0.174	0.124
Ratio %	38.10	20.92	27.55	65.62	25.36	26.66

The diversity of basic shell geometry (Fig. 5-7B) shows a rapid rise to an early Sinemurian maximum through morphological radiation, followed by a pronounced fall in the middle Sinemurian. Thereafter morphological diversity remains high and is stable apart from two modest troughs, one in the Falciferum Zone (which, incidentally, correlates with the early Toarcian anoxic event in northwest Europe) and another in the Thouarsense Zone. Fig. 5-8B shows a similar general pattern except that the Pliensbachian and Toarcian ammonites have significantly higher ornamentation diversities than those of the Hettangian and Sinemurian.

5.6.2 Shannon Diversity Index (SDI)

The measure of information proposed by Shannon (1948) is the bit. Thus, if we have a device such as a switch, which has two stable positions, on or off, then it can store 1 bit of information

$$\log_2 2 = 1.0 \text{ bit}$$

There is a close relationship between information and uncertainty, so the more uncertain an answer is, then the more information is passed when the answer is given. If there are number of alternatives, then the amount of information is maximized when each is equally likely to occur. If we consider a system with N alternatives, and the probabilities of each alternative is given by

$$p_1, p_2, p_3, \cdots\cdots, p_n$$

then the information H is given by

$$H = -\Sigma p_i \log_2 p_i \qquad i = 1, 2, \ldots, n$$

H is the entropy of the system. If we consider the case of two alternatives, each being equally likely, then

$$p_1 = p_2 = 0.5$$

$$H = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1.0$$

if $p_1 = 1$ and $p_2 = 0$ (or vice versa), then H=0. Thus H is maximized when $p_1 = p_2 = \cdots = p_n$ and H is equal to $\log_2 n$, the ratio

$$-\Sigma p_i \log_2 p_i / \log_2 n \qquad i = 1, 2, \ldots, n$$

is known as the relative entropy (RE) (Shannon 1948), and can be used as an alternative to the morphological diversity range index. RE describes not only the range of the morph space occupied, but also the evenness of the occupation. For qualitative morphological characters (VENTER, FURC, TUBERC) the Shannon diversity index (SDI) is defined as

$$SDI = -\Sigma f_i \log_n f_i \qquad i = 1, 2, \ldots, n$$

n --- number of the states for the character

f_i --- frequency of the state i

For quantitative morphological characters (W, U, WWWH, DMAX), there are no natural division of states, therefore each character is divided into n evenly spaced intervals. If n is equal to one, the whole character range is considered as one state and the SDI is equal to zero. As n increases, the SDI gradually rises to approach the true diversity of the character. Experiment shows that the SDI starts to stabilize when n is bigger than one fifth of the sample size if the sample size is not too small (>40), and this value is subjectively adopted here. This is not critical since the current study is not meant to estimate the absolute morphological diversity of ammonite species that occurs in each zone, but to assess the relative changes. The change in diversity shown in Figs. 5-9 and 5-10 is in general agreement with the pattern observed in Figs. 5-7 and 5-8. There is a rapid rise to an early Sinemurian maximum through morphological radiation, followed by a pronounced fall in the middle Sinemurian with diversity remaining relatively high and stable thereafter. This temporal pattern is also reflected in Fig. 5-11 where the first ammonite superfamily in the Early Jurassic -- the Psiloceratacea.

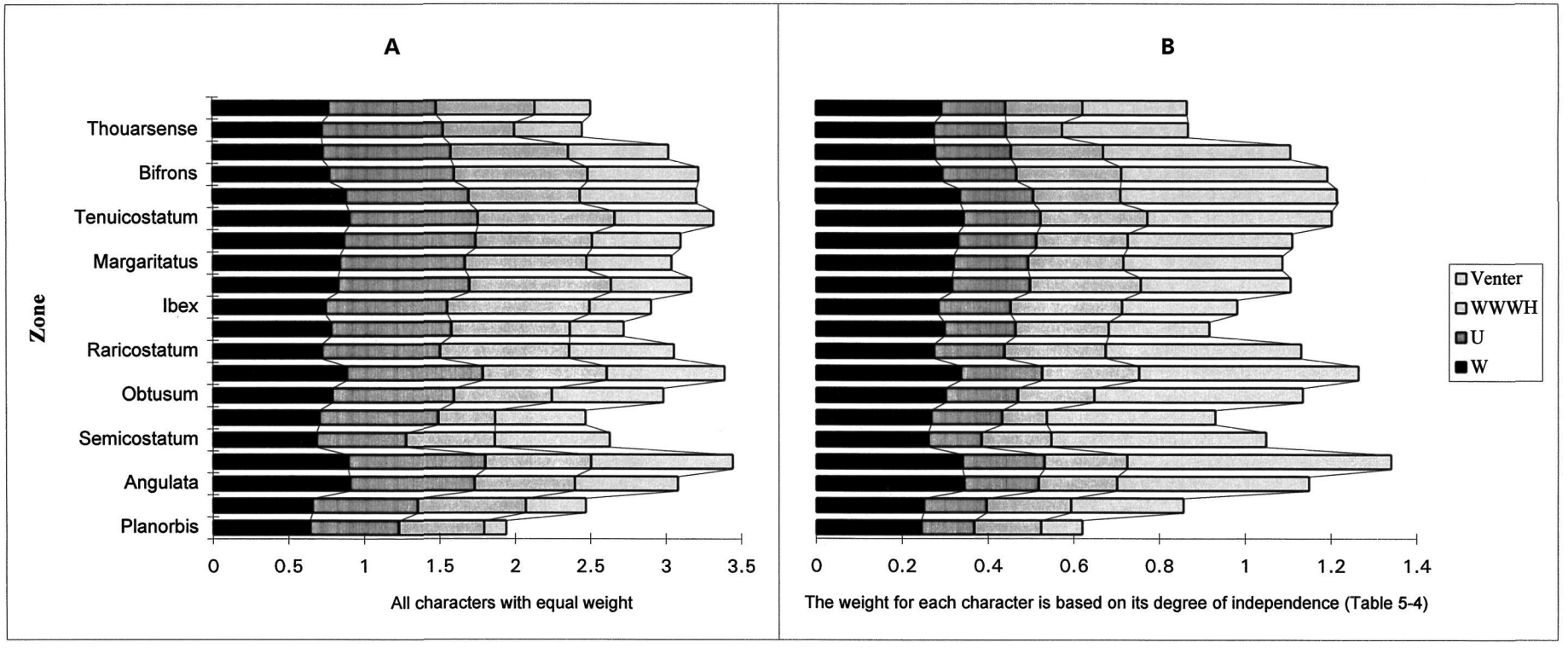


Figure 5-9 The Accumulation of Morphological Shannon Diversity Index for Basic Shell Geometry

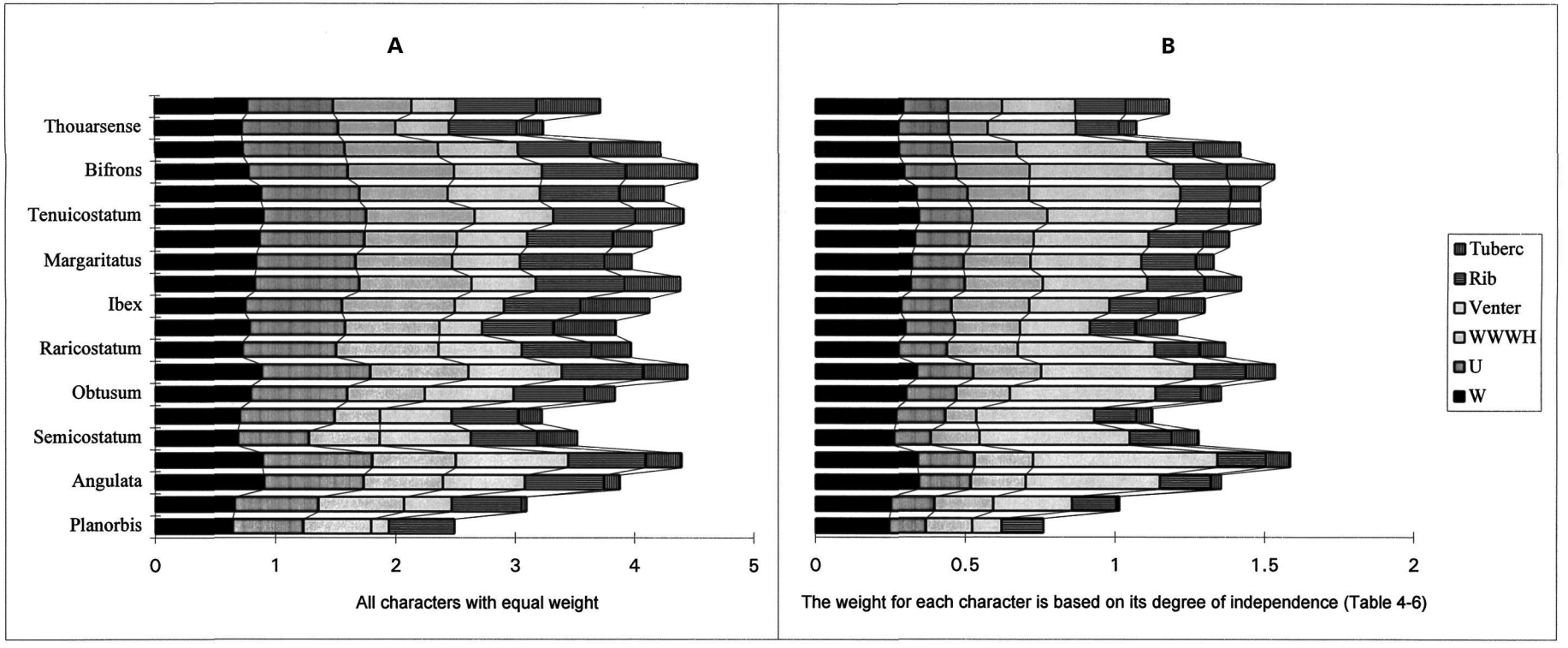


Figure 5-10 The Accumulation of Morphological Shannon Diversity Index for All Characters

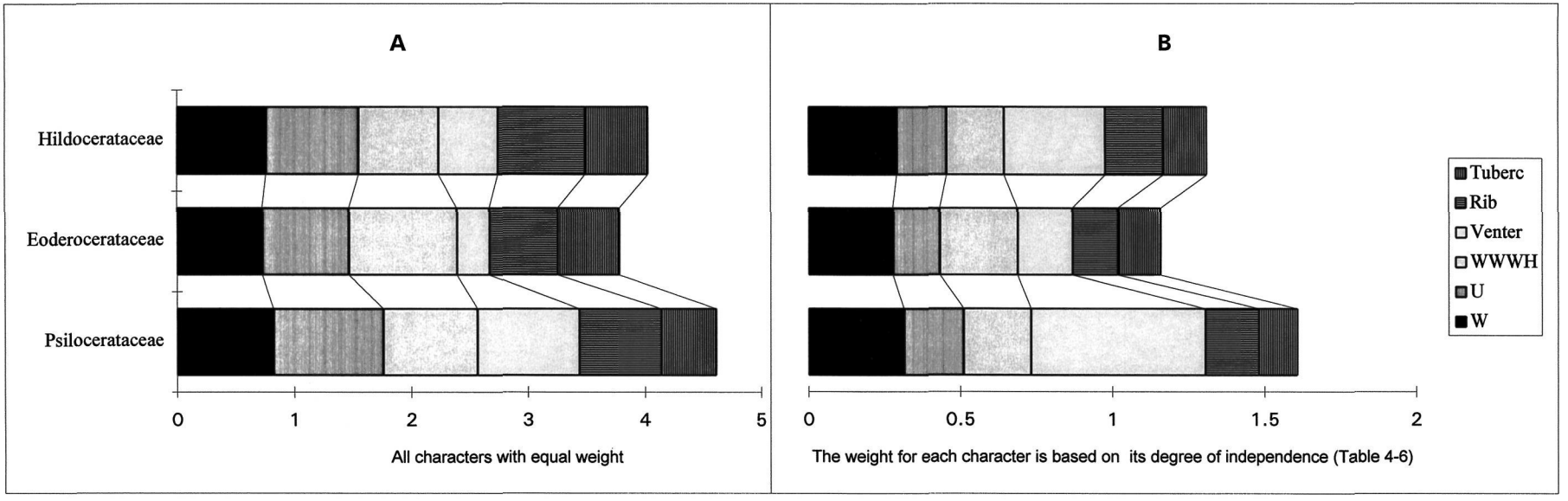


Figure 5-11 The Accumulation of Morphological Shannon Diversity Index by Superfamily

Table 5-4 Residuals from multiple linear regressions of SDI

Sum of Squares	W	U	WWW	VENTER	RIB	TUBERC
Total	0.108	0.093	0.309	0.397	0.056	0.346
Residual	0.029	0.051	0.131	0.286	0.014	0.137
Ratio %	21.16	35.41	29.77	41.87	20	28.36

shows the greatest morphological diversity, followed by the morphologically more restricted Eoderoceratáceae, and a diversity rebound with the Hildoceratáceae.

5.7 Discrepancies between Morphological and Taxonomic Diversity

It is evident by comparing Fig. 5-5 with Figs. 5-8 and 5-10 that initial morphological diversification in Lower Jurassic ammonites far exceeded the proliferation of lower-level taxa. Such a pattern suggests that morphological transitions were larger earlier in this clade's history, a point that is relevant to the ecological and evolutionary mechanisms of diversification (Stanley 1979; Foote 1992, 1993). A similar pattern has been discussed for echinoderms (Foote 1992; Sprinkle 1980), and stenolaemate bryozoans (Anstey and Pachut 1992). The most striking fact emerging from the comparison is that the morphological diversity of the Bucklandi Zone with 46 species is higher than that of the Margaritatus Zone with 177 species which demonstrates morphological diversity need not increase as taxonomic diversity increases. Such morphological stagnation indicates taxonomic diversification that is highly constrained morphologically (Foote 1993). Two patterns are seen during the decline of taxonomic diversity: (1) from the Spinatum Zone through the Tenuicostatum Zone to the Falciferum Zone, the number of species drops to 49 from a high level of 161, whereas the morphological diversity remains relatively stable suggesting that the extinct species are randomly distributed with respect to morphology; (2) from the Bucklandi Zone through the Semicostatum Zone to the Turneri Zone, morphological diversity decreases along with

taxonomic diversity which may imply selective extinction in certain regions of the morphospace, but could also be due to the very low level of species diversity in the Turneri Zone.

5.8 Other interesting observations

Table 5-5 shows that all families that radiate into numerous species (>13) in the first zone of their occurrence have durations of less than 4 zones, which may imply that evolution rates associated with these families are high. In other words, rapid origination is associated with short duration, although the time resolution of my data does not warrant a firm conclusion.

Fig. 5-12 shows the average shell diameter changes for Lower Jurassic ammonite superfamilies through 20 successive zones. The basic calculation unit is species and the maximum shell diameter achieved for the species is used to compute the average for the superfamily. The rationale for using maximum instead of average shell diameter for species is to eliminate juvenile and immature forms from the calculation whenever possible. The average maximum diameter curves for the three superfamilies are more or less parallel. Most noticeably, the diameter curves show a synchronized drop from the Obtusum Zone to the Oxynotum Zone, followed by a parallel rise in the Raricostatum Zone. Furthermore, the mean maximum diameters for all three superfamilies decrease from the Jamesoni Zone to the Ibex Zone. It is possible that the observed parallelism among the average diameters for the three superfamilies is a coincidence which may disappear at the family level. However, Fig. 5-13 presents a similar pattern with the diameter curves showing a pronounced trough in the Oxynotum Zone, and six out of eight families showing a significant decrease in shell diameter in the Ibex Zone. Interestingly, these two zones show high family origination rates, relatively low species diversity (Fig. 5-5A, D) and are times of regression (Hallam 1988). This may imply *r selection* whereby times of regression cause a deterioration of the environment, increasing the stress on organisms and promoting the origination of high-level

taxa which radiate at lower taxonomic levels during sea level rise. "The new taxa evolving at times of low sea-level stand are often smaller than their ancestors, and increase in size phyletically during the subsequent, environmentally less stressful times of high sea-level stand" (Hallam 1987). Another possible explanation for the observed correlation is that it is an artifact of preservation bias but my data do not contain information on facies and so cannot address this problem.

Table 5-5 Number of species in each zone for Lower Jurassic ammonite families

Zone\Family	Psi	Sch	Oxy	Cym	Ech	Ari	Ama	Coe	Dac	Eod	Lip	Phr	Pol	Hil	Phy
Levesquei														40	13
Thouarsense														23	5
Variabilis									10					14	33
Bifrons									43					38	7
Falcifer									12					37	
Tenuicostatum									30					72	
Spinatum			5				19		23					114	
Margaritatus			6	1			26		12	2	2	2		126	
Davoei			7	2				10	15	8	29	3	6	41	
Ibex			1	1				7	7	9	24	2	33	7	
Jamesoni			12					17		18		12	53	1	
Raricostatum		2	11		38			4		16		3			
Oxynotum		4	11		14	2		1		5					
Obtusum		2		1		18				5					
Turneri		1				17				4					
Semicostatum		2		1		28									
Bucklandi	7	11		1		27									
Angulata	8	14				8									
Liasicus	12	12				5									
Planorbis	30														

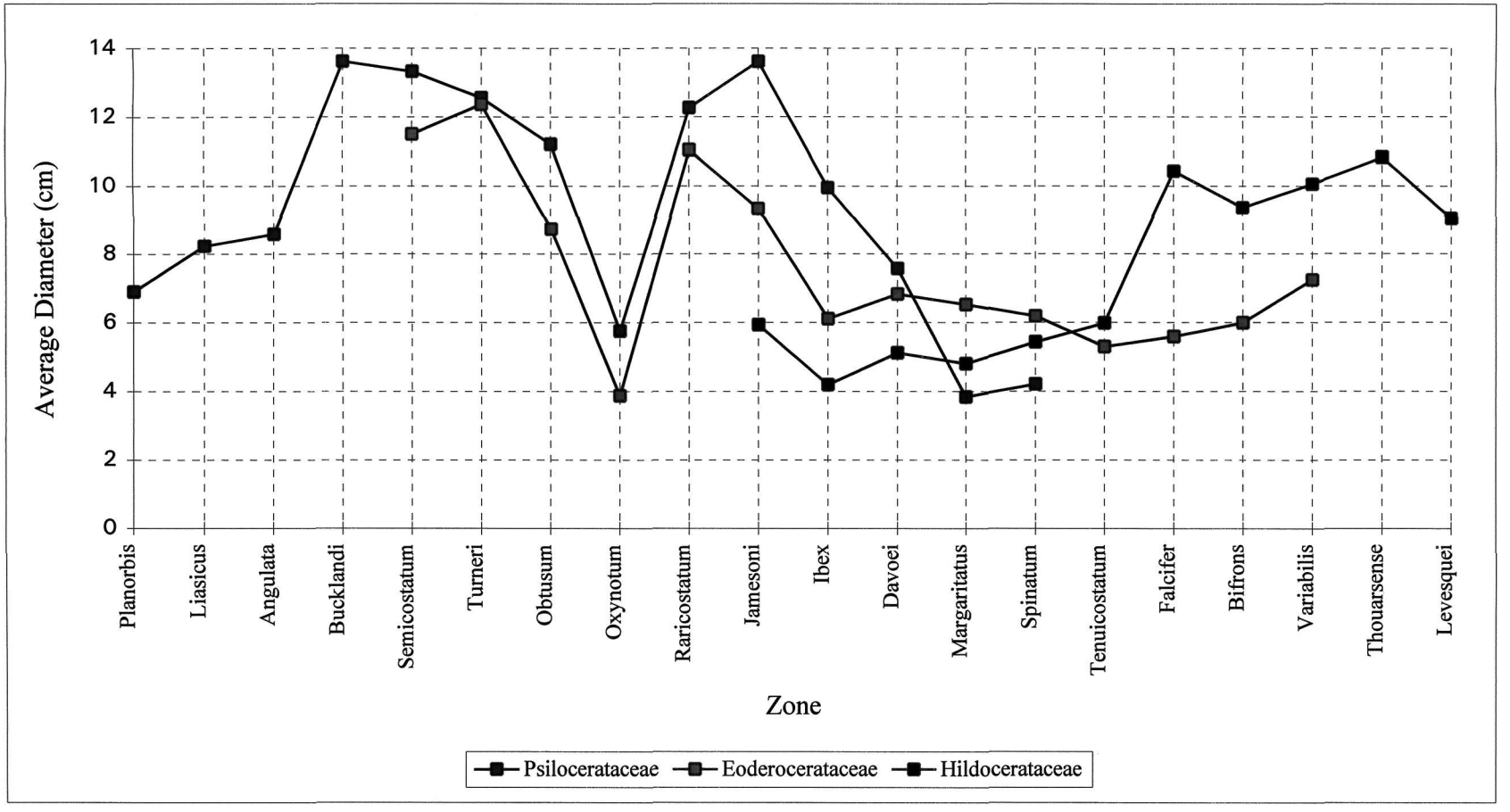


Figure 5-12 Average Shell Diameter Change for Lower Jurassic Ammonite Superfamilies

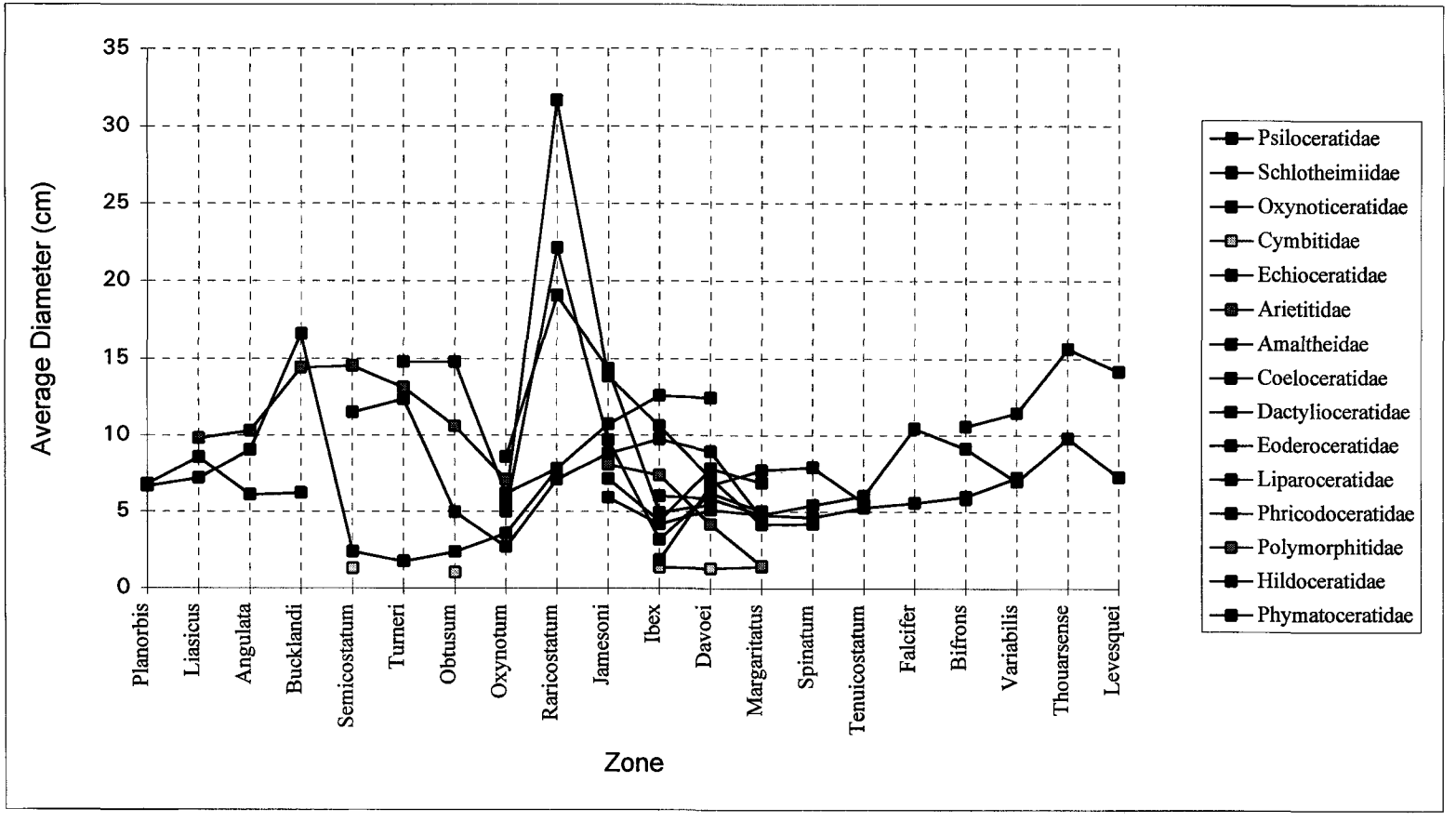


Figure 5-13 Average Shell Diameter Change for Lower Jurassic Ammonite Families

Chapter 6 An Aid to Identification

6.1 Introduction

Identifying ammonite species requires the recognition of associations of ammonite shell morphologies which can be demonstrated to maintain their superpositional relationships throughout their entire geographic distribution. Each species shows variation within the population and also during ontogeny, variation which must be determined before a species can be identified with confidence. Traditionally intraspecific variation is determined by examining the illustrations and descriptions of the specimens of the taxon under consideration which means scouring a voluminous literature that is in many different languages and often reaches back well into the last century (Smith 1986). In order to effectively utilize the literature, the attributes of a large number of specimens have to be memorized and evaluated in an objective manner. Direct comparison of two specimens is relatively easy but the problem increases rapidly as more specimens are involved. To a large extent the image database Ammon can overcome these problems. The basic idea is that given information supplied by the user, a search of Ammon is made and closely matching species are displayed on the screen for visual comparison with the specimen in question. If the candidate list is large, both the rough stratigraphic range and geographic locality of the specimen can be used to narrow the search. Searches can also be restricted to a certain diameter range to circumvent the problem of ontogenetic variation.

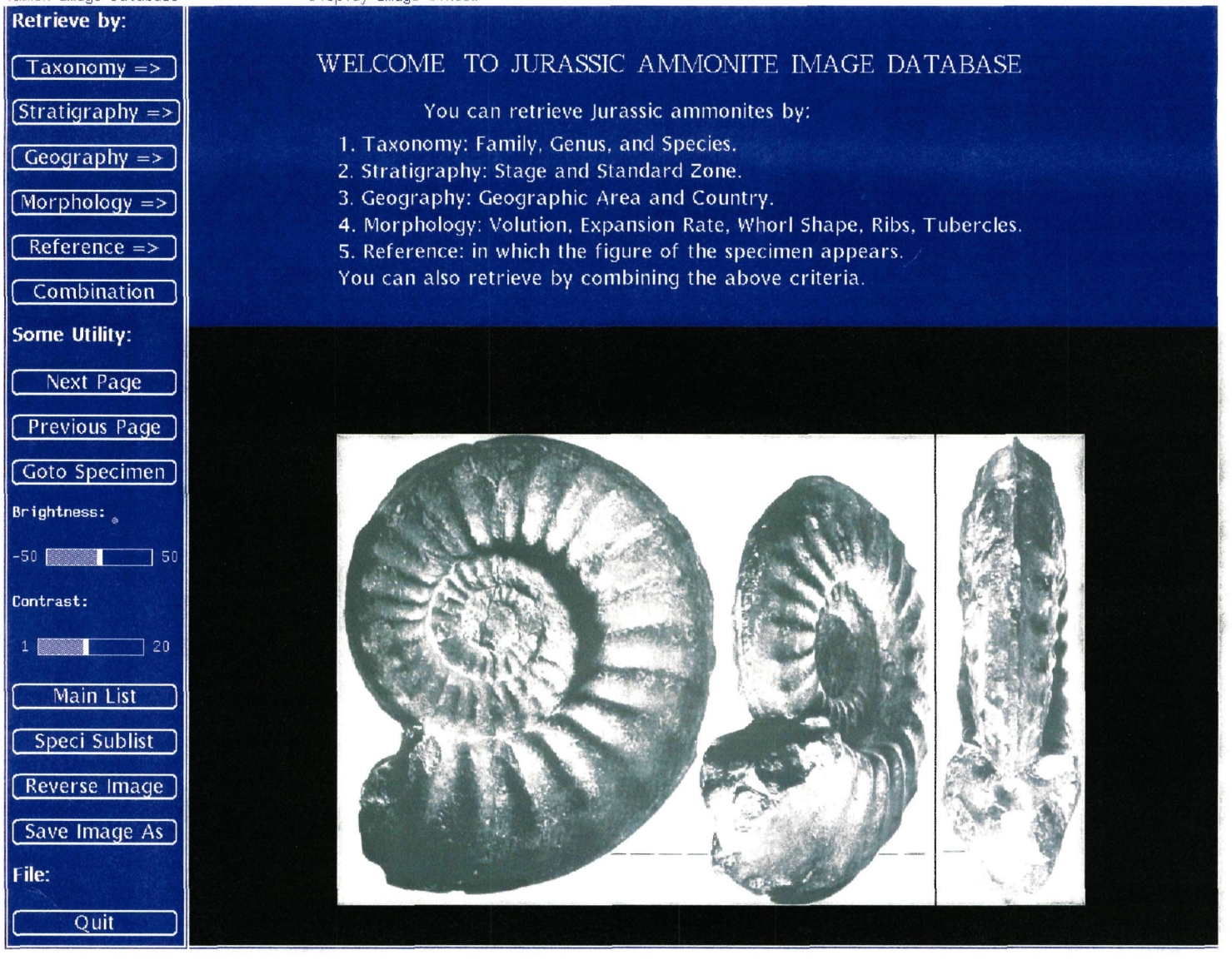


Figure 6-1 The start-up screen of the database Ammon

6.2 User Interface

Ammon has an easy to use, menu-driven user interface which is written in Sunview and the embedded SQL query language of Oracle RDBMS. Fig. 6-1 shows the start-up screen which consists of three windows: the upper-right window displays a welcome message and explains what user can do with the system, the lower-right is a graphics window showing the lateral and frontal views of a Lower Jurassic ammonite specimen and the left window hosts available command buttons, menu-buttons and sliders. Users can retrieve Jurassic (mainly Lower Jurassic) ammonites by Taxonomy, Stratigraphy, Geography, Morphology, Reference or various Combinations:

1. Taxonomy

When the **Taxonomy** is pressed with the right mouse button in the left window, a pull-right menu appears which shows the available ammonoid families in the database (Fig. 6-2). If, for example, the user selects the ammonite family **Dactylioceratidae** by highlighting it and releasing the mouse button, all specimens of the family in the database will be retrieved. Each item itself in the family menu has a pull-right menu such that if the user pulls right from the item **Dactylioceratidae,** a genus list for the family would appear (Fig. 6-2). Suppose that *Dactylioceras* is selected, then the screen shown in

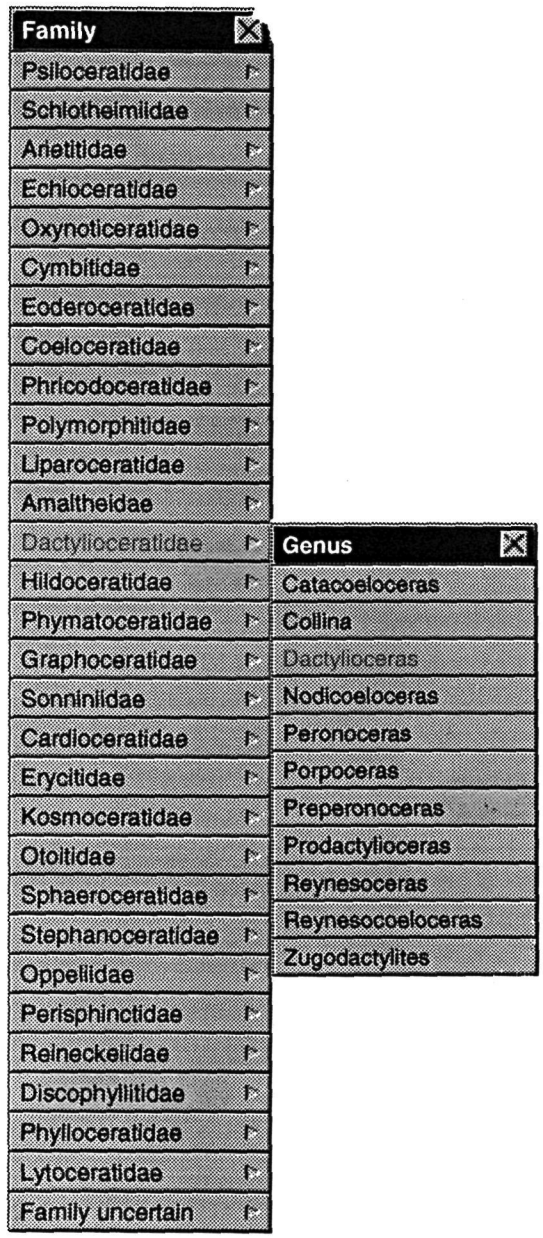


Figure 6-2 Family Menu (Left)
Genus submenu of Dactylioceratidae

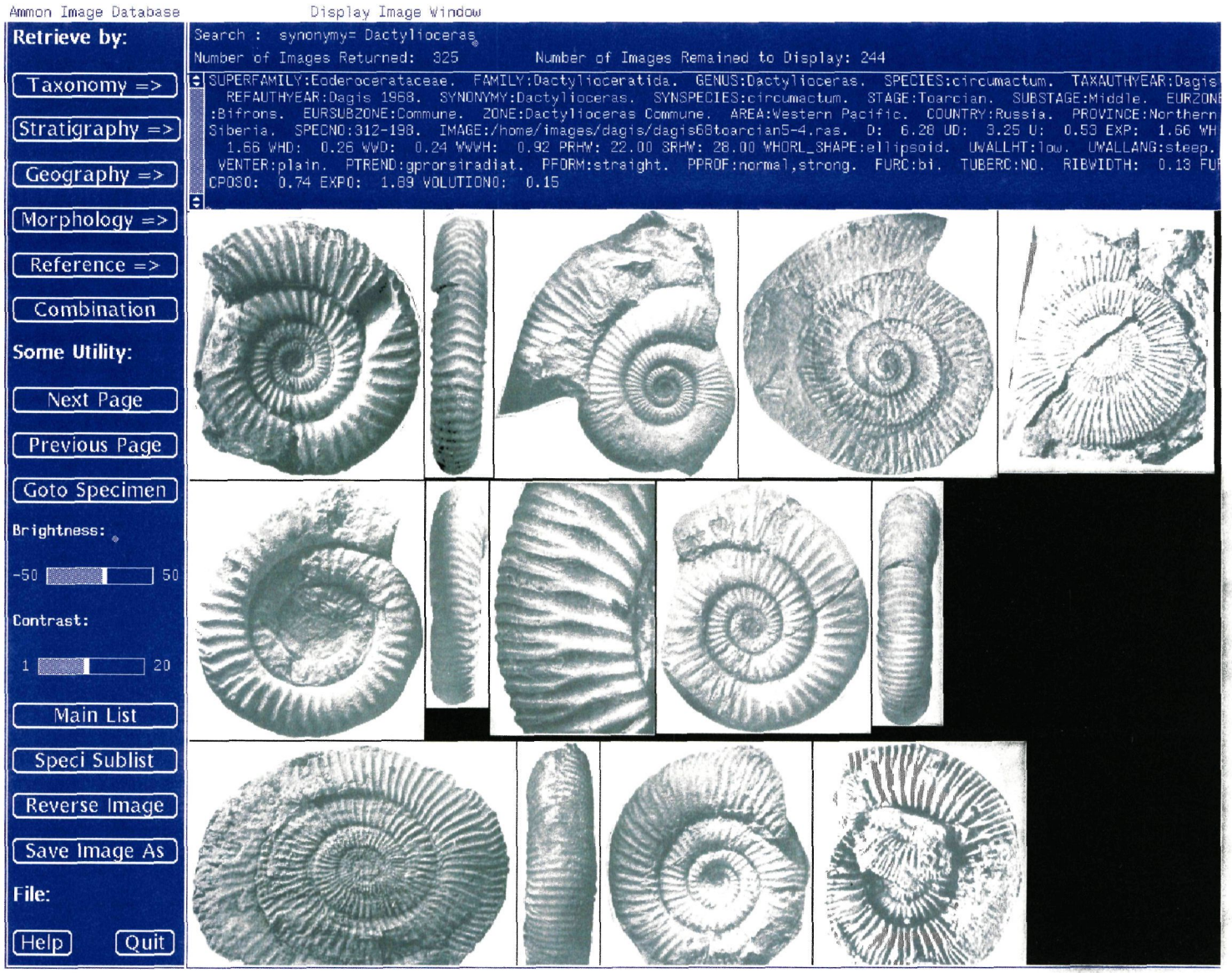


Figure 6-3 The retrieval results of Dactylioceras

Fig. 6-3 would be the query result. The top-right window records the query text and gives the number of images retrieved and the number remained to display. Images are displayed in order of size for efficient utilization of the screen space and the user can go to next screen of images by pressing the **Next Page** with the left mouse button and back to previous screen by pressing the **Previous Page** button. A user interested in a specific specimen can click on the image with the left mouse button which will display all information related to the specimen in the middle-right text window (Fig. 6-3). A user interested in the species to which the specimen belongs and wants to examine the intraspecific variation can mark the specimen with the middle mouse button and click on the **Speci Sublist** (Fig. 6-3). The user can return to the genus level by clicking on the **Main List** button. Exercises such as this enable the user to get familiar with a particular taxonomic group very quickly.

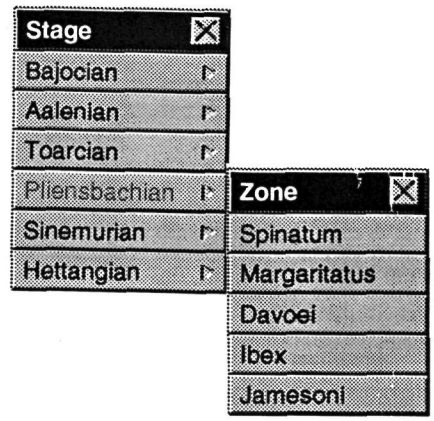


Figure 6-4 Stage Menu (Left) Zone Submenu of Pliensbachian

Figure 6-5 Country Menu

2. Stratigraphy

The **Stratigraphy** button has a two-level menu. The top level is the stage menu (Fig. 6-4) which lists the four Lower Jurassic stages and two of the Middle Jurassic stages. Each stage has a pull-right menu which shows the zones (northwest Europe scheme) in the stage. The user can scan the

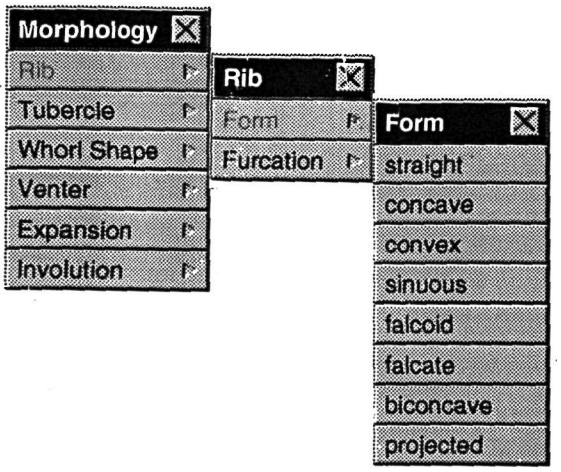


Figure 6-6 Morphology Menu (Left), Rib Submenu (Middle) and Rib Form Sub-submenu (Right)

database by zone and get some idea of how ammonite morphology changes through time.

3. Geography

The user can explore morphological and temporal changes of ammonite taxa in different parts of the world by using **Geography** as query criterion, he or she can retrieve ammonites by geographic region or country (Fig. 6-5). There are five geographic regions: Northwest Europe, Western Tethyan, Western Pacific, North America and South America each of which has thousands of specimens in the database and could easily overwhelm the user. Geographic regions are most commonly used with other criteria to narrow a search.

4. Morphology

Morphology is one of the two foundations for fossil identification. The user can also search the database using morphological characters such as the degree of involution, shell expansion rate, rib form & furcation pattern, tubercle pattern, whorl shape and ventral geometry (Fig. 6-6). If the images displayed in the graphics window are fuzzy or do not have

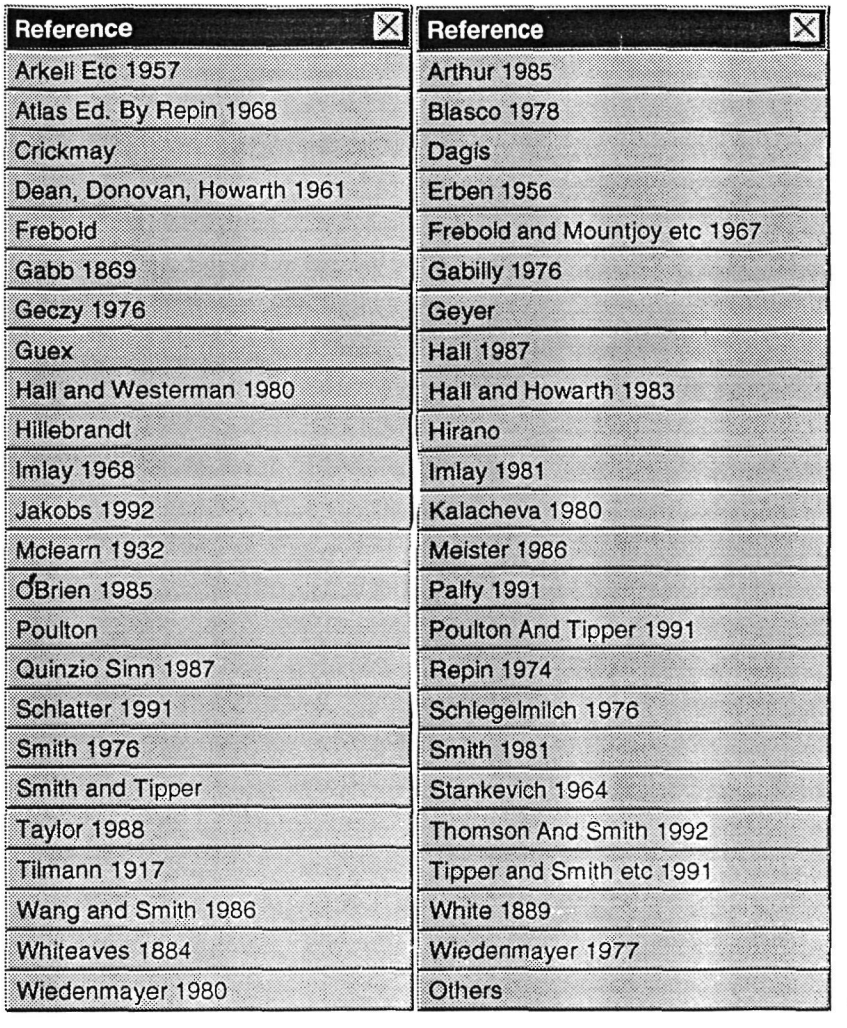


Figure 6-7 Reference Menu

the right brightness, the user can adjust them using the **Brightness** and **Contrast** sliders in the command window.

5. Reference

A user interested in a specific person's work can query the database by **Reference** (Fig. 6-7) which has a two-column author menu with authors arranged in alphabetic order and the list will grow as new references are added to Ammon.

6. Combination

The **Taxonomy**, **Stratigraphy**, **Geography**, **Morphology** and **Reference** menu system offers the user easy access to the database but at the same time reduces flexibility because the user can only search the database in one area of inquiring at a time and the file that lists all menu items needs to be updated whenever new character states are added. For any serious application, a combination of the above search criteria is needed as provided by the **Combination** button in the command window (Fig. 6-3). Suppose that the illustrations in the start-up screen are of interest and the user wants to know the species name and the locality where the specimen was collected. The user composes a query using the morphological features available from the image: a carinate venter, 13 ribs per half whorl at a shell diameter of 12 cm, the average trend of ribs is gently rursiradiate, no tubercles. Taking into account measurement error, the query is typed into the search field in the top-right text window as :

venter=carinate and prhw>10 and prhw<16 and d>10 and d<14 and ptrend=grursiradiate and tuberc=no

and the **Combination** button pressed to search the database. The response to the query (Fig. 6-8) shows the specimen that appears in the start-up screen, click on it with the left mouse button, all information related to this specimen is displayed in the middle-right window: the specimen is named as *Agassiceras scipionianum* and was collected from southwest Germany.

With the **Save Image As** button and the **File:** name field, the user can save the whole or any part of the graphics window (using mouse button to mark the region of interest) as a file for later reference or print out. The textual information displayed in the middle-right window can be saved as an ASCII file using the pull-right menu of the window, so that the information is available to other applications.

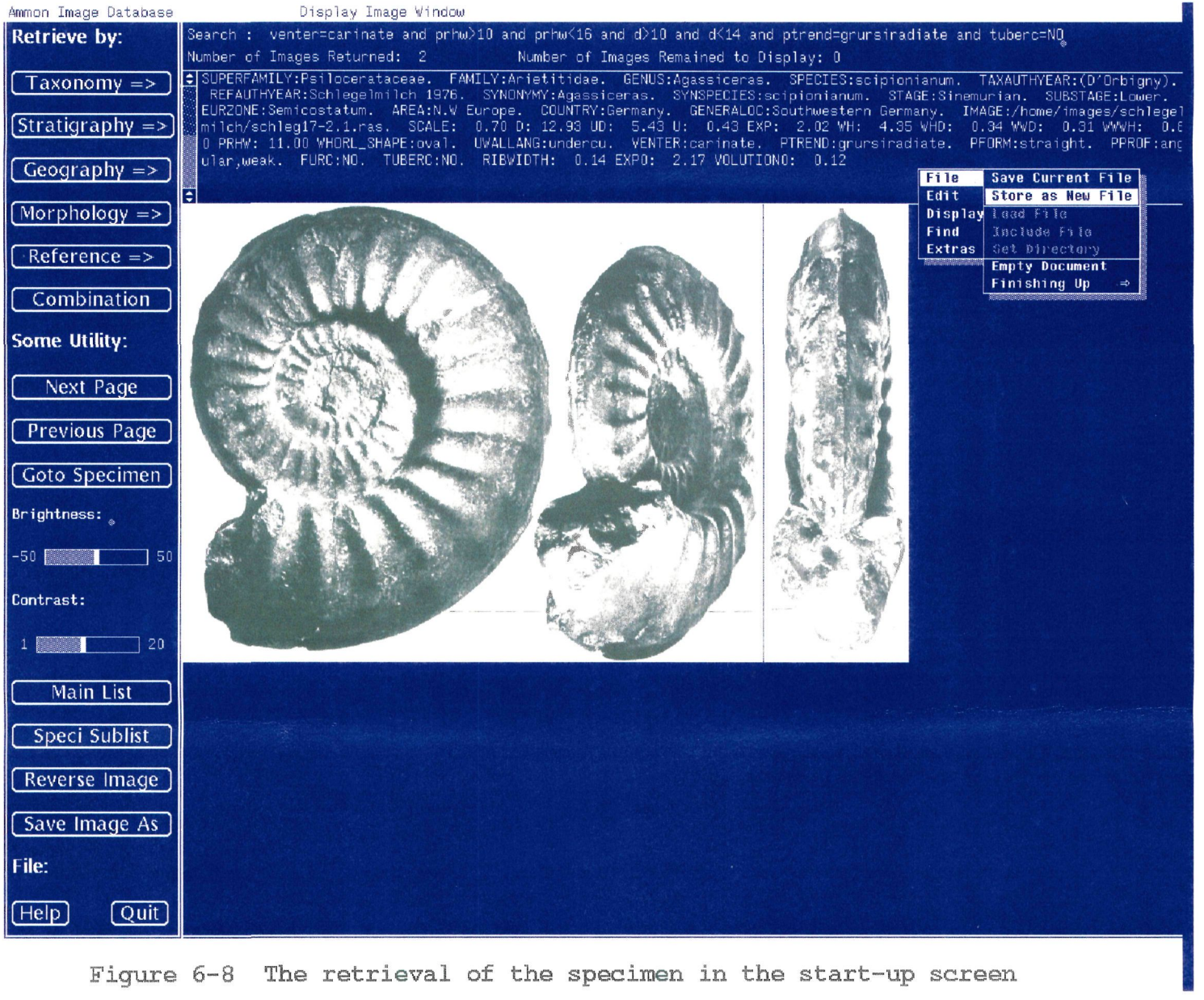


Figure 6-8 The retrieval of the specimen in the start-up screen

6.3 Examples

It would be interesting to see what kind of role that Ammon can play and how effective it can be in dealing with the problem of fossil identification. A colleague of mine, Genga Nadaraju of the UBC Paleontology Lab has just finished her Master's thesis entitled "Triassic-Jurassic biochronology of the eastern Iskut River map area, northwestern British Columbia" but the illustrations and taxonomic descriptions of her ammonite collections have not yet been entered into Ammon. Most of the ammonite specimens from Nadaraju's collection are poorly preserved and fragmentary thereby providing a challeging test of the system.

1. Example 1 -- Fig. 6-9 (Nadaraju 1993, Plate V, Fig. 1)

Measurements (From Nadaraju 1993, p.98):

SPECIMEN	D (cm)	UD (cm)	U	PRHW
C-201415a	≈ 8.53	5.00	0.58	33

Description: Evolute form with a wide umbilicus. Flanks slightly convex and ornamentation consisting of numerous simple ribs that trend prorsiradiately on the outer part of the flanks.

A query based on the above measurements and description can be composed:

U>0.55 and U<0.62 and UD>4 and PRHW>25 and PRHW<40 and PTREND like prorsiradiate and FURC=no and TUBERC=no. Each character in the query is given a range centering around the

Figure 6-9 Plate V, Fig. 1 of Nadaraju 1993

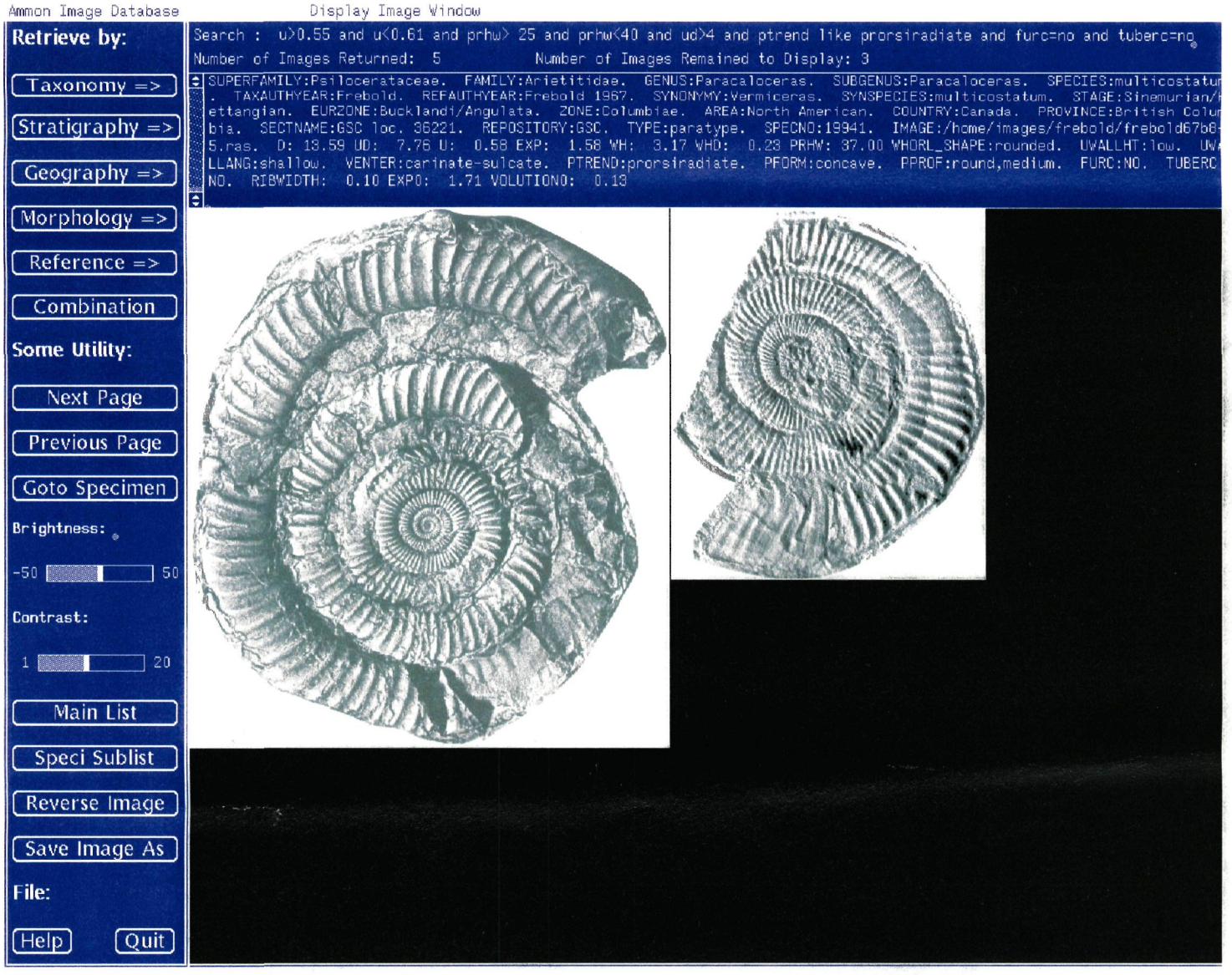


Figure 6-10 Query results of Example 1

measured value as well as taking into account the measurement errors and intraspecific variations. For example, "PTREND like prorsiradiate" would include specimens with either prorsiradiate ribs or gently prorsiradiate ribs. As result of the query, 5 specimens are retrieved from the database. Fig. 6-10 shows the first two specimens and by clicking on the left specimen with the left mouse button, all textual information related to it appears in the middle-right window: a *Paracaloceras multicostatum* from Upper Hettangian and Lower Sinemurian. *Paracaloceras* cf. *multicostatum* was the identification made by Nadaraju (1993). The other four candidates are : *Paltechioceras yakounense, Paltechioceras harbledownense, Echioceras aklavikense, Leptechioceras alvarezi*, all from Upper Sinemurian If it is known that the specimen is from Hettangian or Lower Sinemurian, then *Paracaloceras multicostatum* offers the best candidate.

Example 2 -- Fig. 6-11 (Nadaraju 1993, Plate VI, Fig. 1)

Measurements

SPECIMEN	D (cm)	U	W	PRHW
C-201631b	≈ 14	0.73	1.46	≈ 36

Description: Evolute, ellipsoidal shell with many whorls. Ornamentation consists of simple ribs that are dense. No keel observed due to the state of preservation.

A query based on the above measurements and description can be composed:

U>0.67 and W<1.5 and PRHW>30 and D>12 and FURC=no

which leads to retrieval of four specimens from the database and the first one of which (Fig. 6-12) is *Alsatites* cf. *proaries*, the identification made by Nadaraju (1993). The other candidates are : *Alsatites liasicus* from the Middle Hettangian and *Coroniceras longidomus* from the Lower Sinemurian.

Figure 6-11 Plate VI, Fig. 1 of Nadaraju 1993

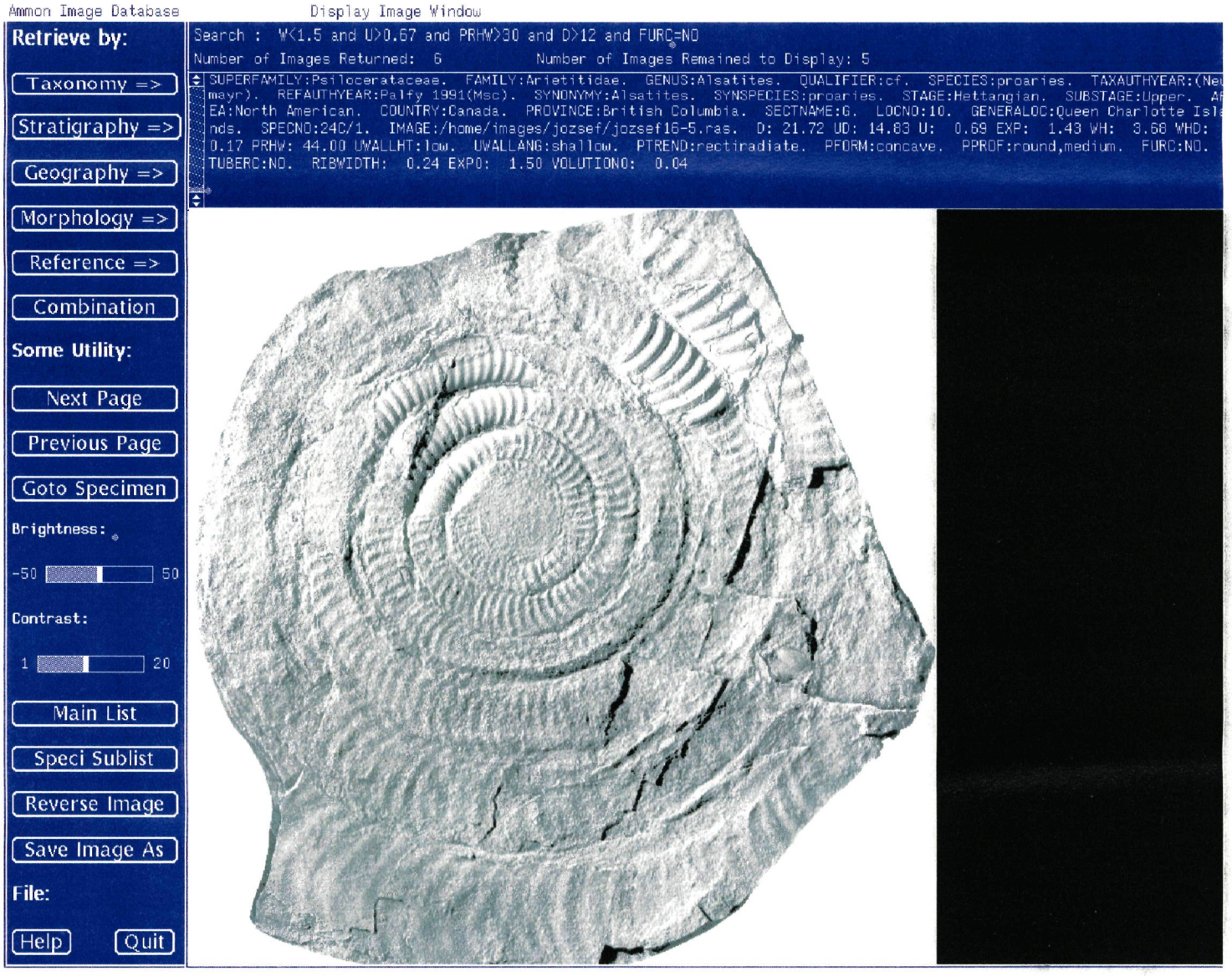


Figure 6-12 Query results of Example 2

Example 3 -- Fig. 6-13 (Nadaraju 1993, Plate VIII, Fig. 1)

Measurements

SPECIMEN	D (cm)	U	WWWH	PRHW
C-201453	≈ 4.2	0.42	1.00	13

Description: Midvolute form with an elliptical whorl section reaching its greatest width near the middle of the flank. Each whorl embraces nearly half of the preceding one. Rounded, plain venter. Ornamentation consists of straight ribs that arise at the umbilical shoulder becoming progressively sharper and thicker to mid-flank and then fading at the ventro-lateral shoulder.

A query can be formulated as:

U>0.37 and U<0.47 and WWWH>0.85 and WWWH<1.2 and PRHW>10

and PRHW<16 and D>3 and D<6 and VENTER=plain and TUBERC=no

Fig. 6-14 shows the query result. Most of the specimens have ribs that cross the venter which does not fit the above description. The only two species that have unornamented venters are *Badouxia canadensis* and *Franziceras coronoides*. The former has ribs that gradually fade whereas the latter has ribs that terminate suddenly at the ventro-lateral shoulder. Fig. 6-13 is probably a *Badouxia canadensis,* the identification made by Nadaraju (1993). Rib strength change across flank can be perceived easily by the human eye but it is difficult to describe in rigid morphological terms that can be used in a computerized database. In order to maximize the benefit and minimize the potential for errorious conclusions, it is clear that human and computer interaction is the best approach to identification.

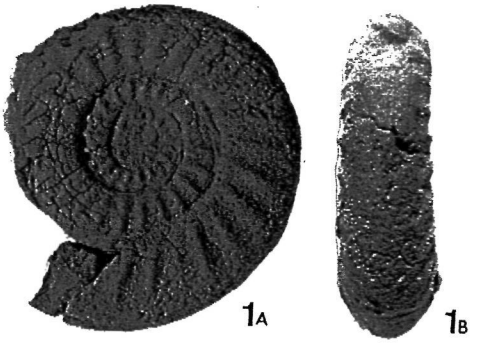


Fig. 6-13 Plate VIII, Fig. 1 of Nadaraju 1993

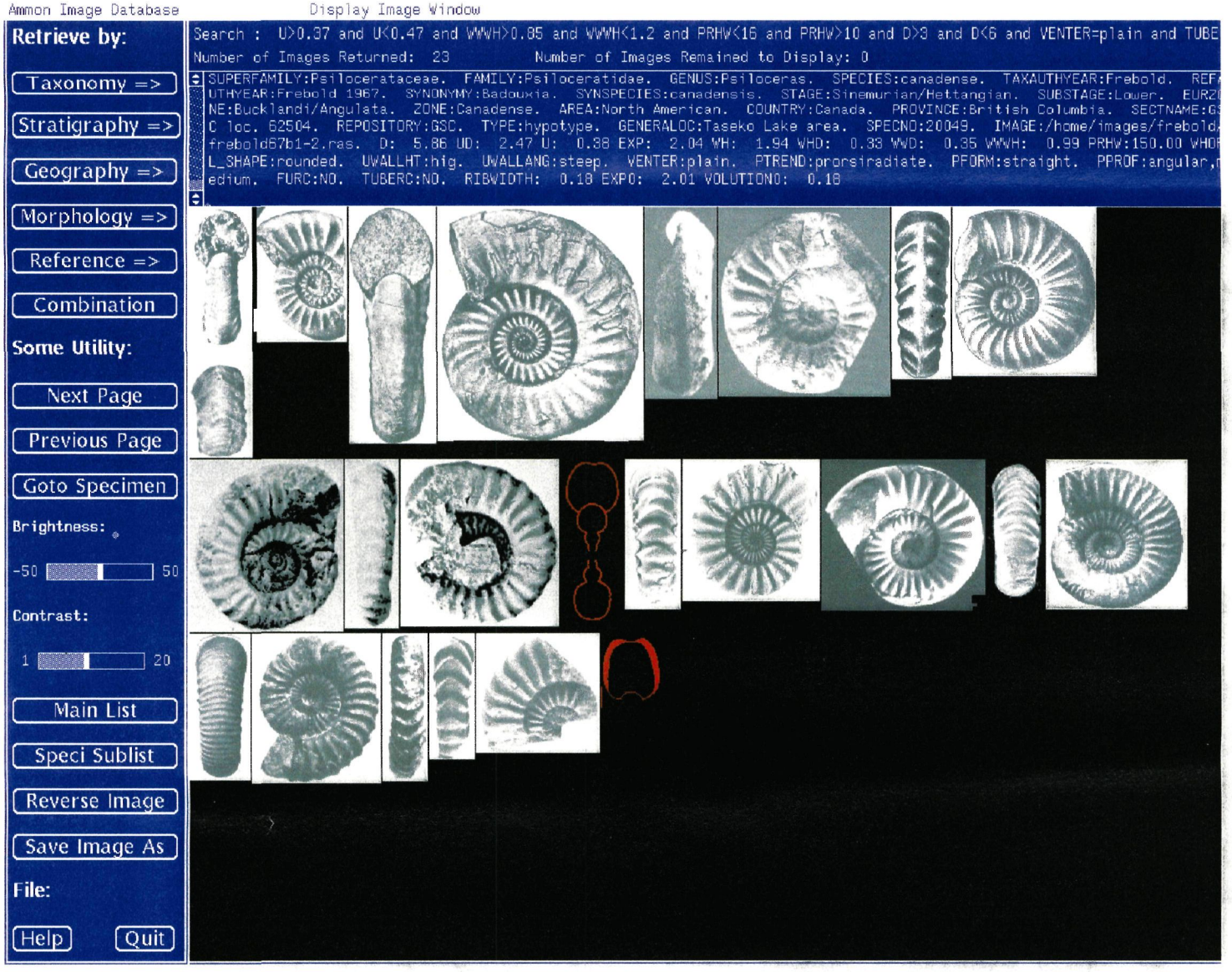


Figure 6-14 Query results of Example 3

In summary, Ammon can be very useful in assisting identification of Lower Jurassic ammonites. Instead of memorizing thousands of species, the user need only to know a dozen morphological descriptors and a few database operators such as AND, OR, NOT, LIKE, IN. A new user of Ammon may need to measure a few quantitative morphological parameters but after some practices the user should be able to build up a sense of relationship between quantitative and qualitative parameters. At this point identification becomes fun. A computerized ammonite image database cannot replace the human expert but does alleviate the burden of memorizing many species and associated data allowing the user to focus on more important questions

The following figures show the variations of basic shell geometry for Lower Jurassic ammonite genera which aims to define these genera more objectively; similar plots for species are not made due to space limitations.

Figure 6-15 Shaded boxs show the range of expansion rate for each genus, the figures in the bracket show number of specimens in each genus

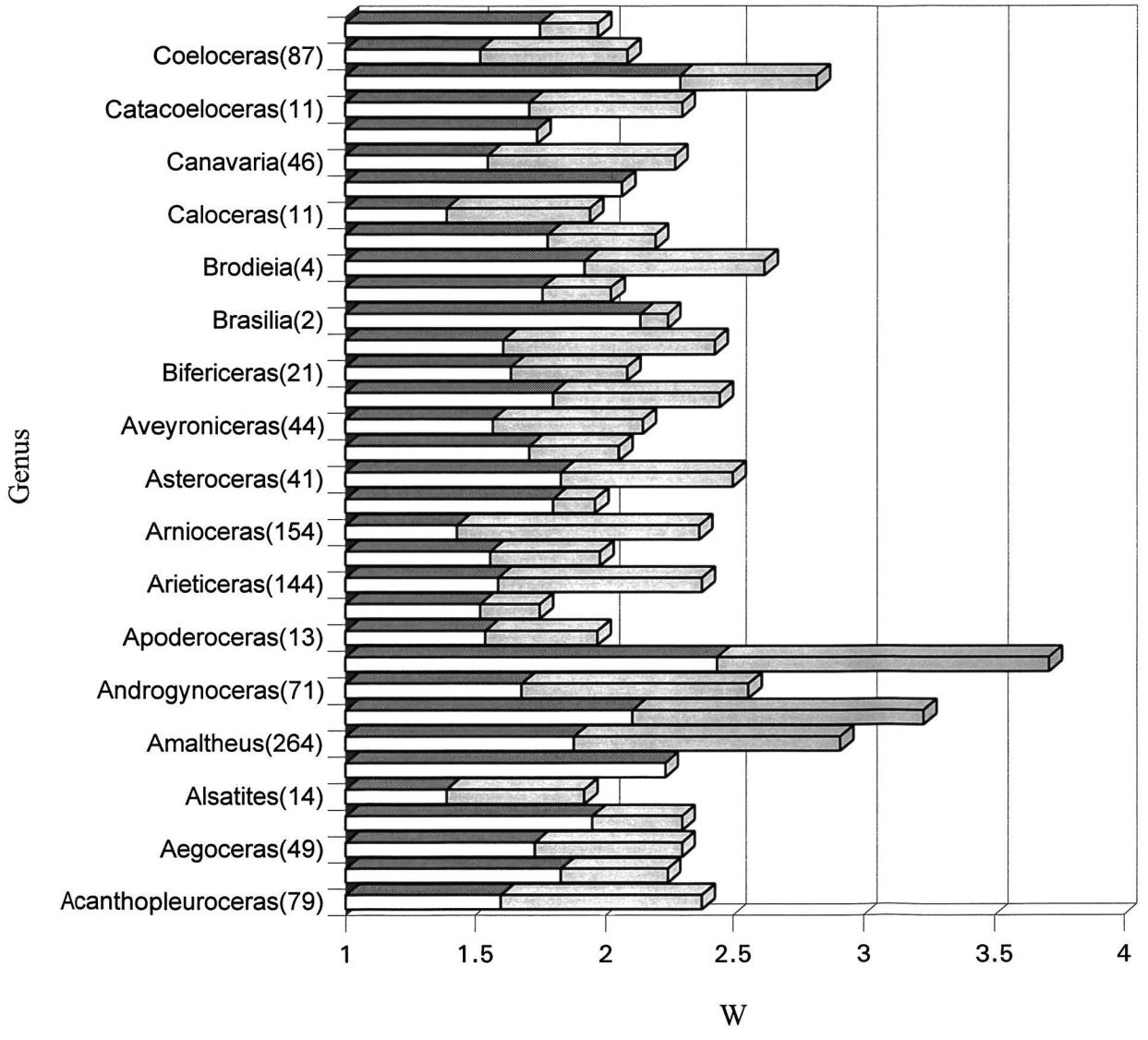


Figure 6-16 Shaded boxes show the range of expansion rate for each genus, the figures in the bracket show number of specimens in each genus

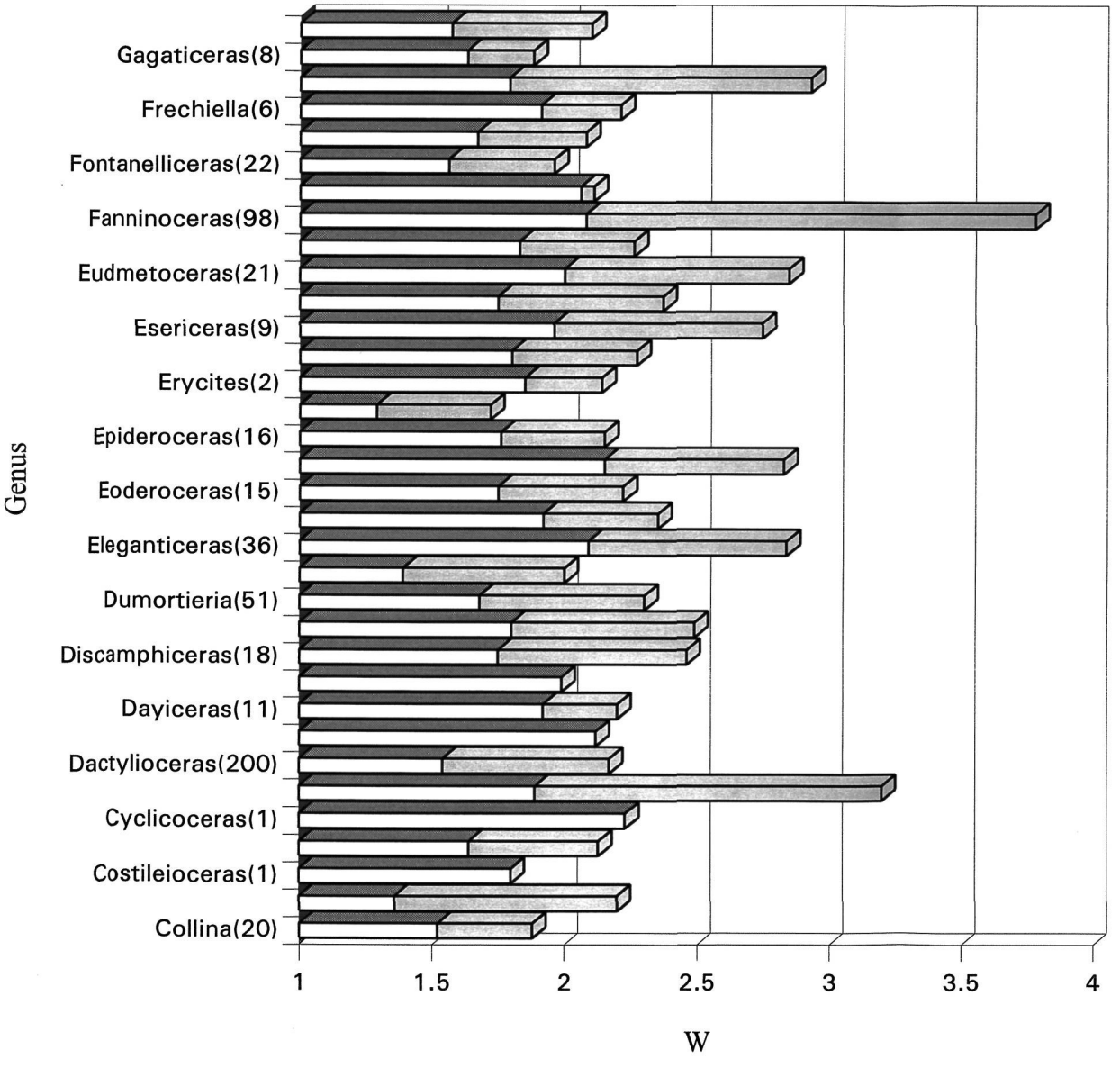


Figure 6-17 Shaded boxes show the range of expansion rate for each genus, the figures in the bracket show number of specimens in each genus

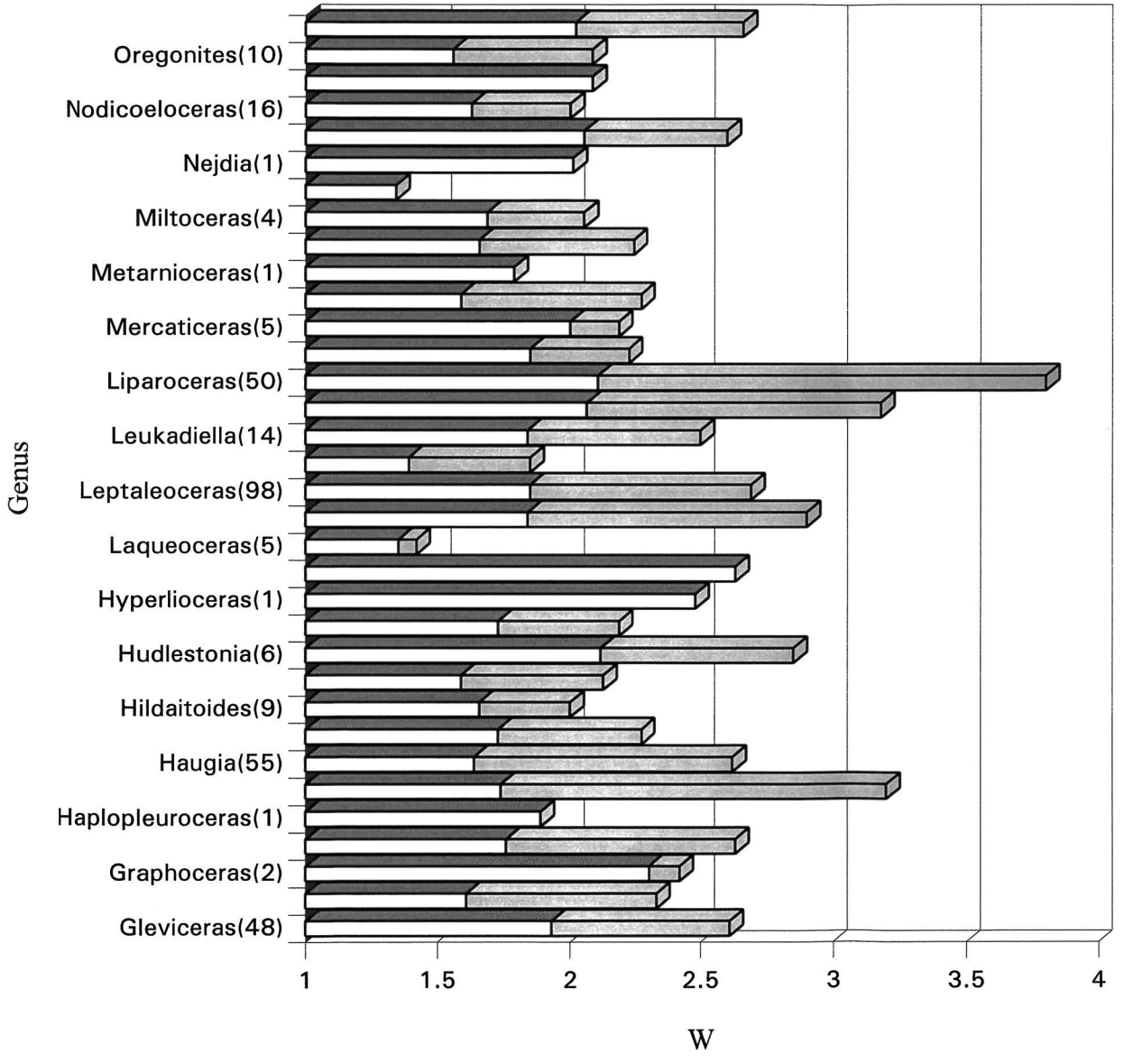


Figure 6-18 Shaded boxes show the range of expansion rate for each genus, the figures in the bracket show number of specimens in each genus

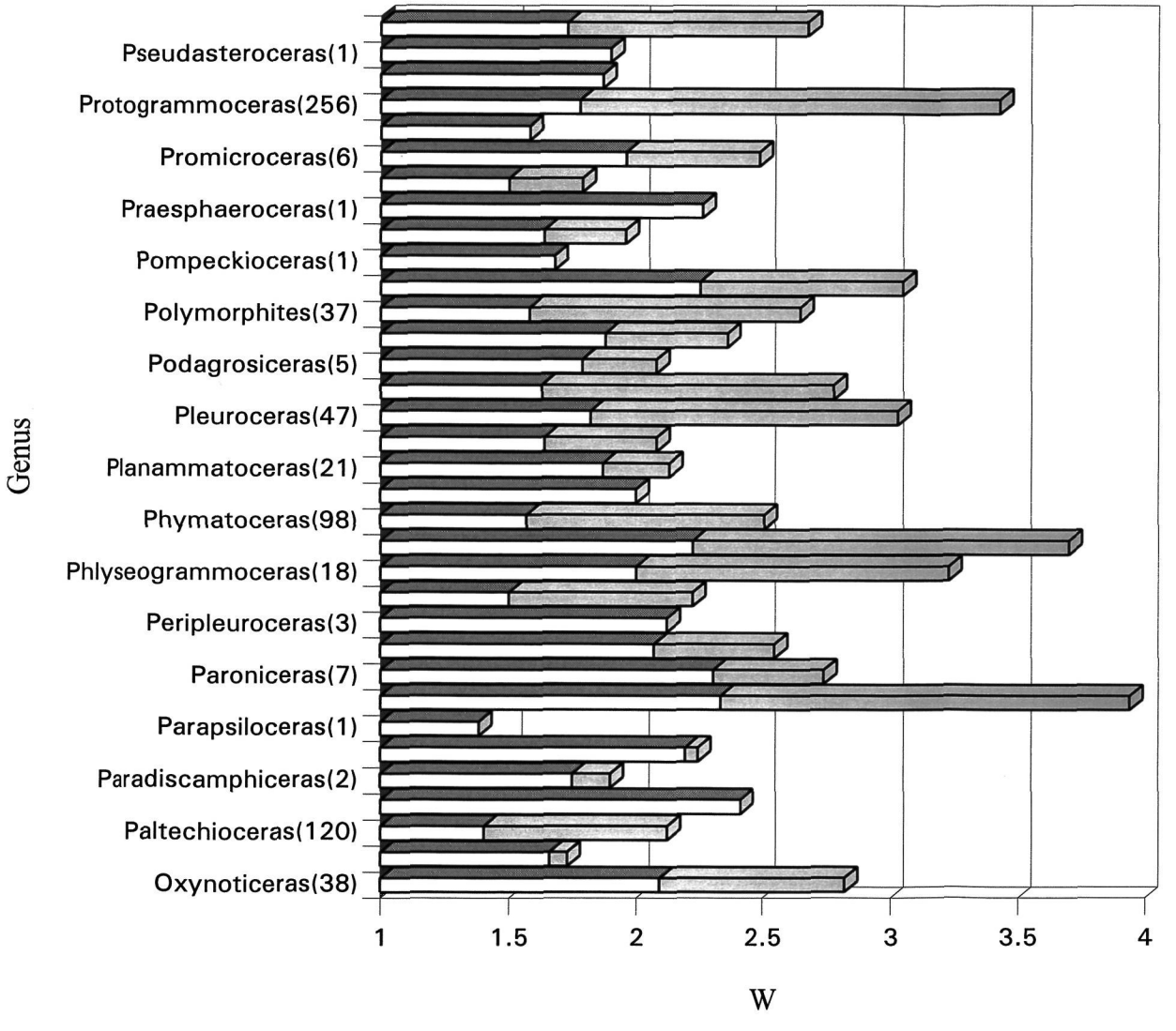


Figure 6-19 Shaded boxes show the range of expansion rate for each genus, the figures in the bracket show number of specimens in each genus

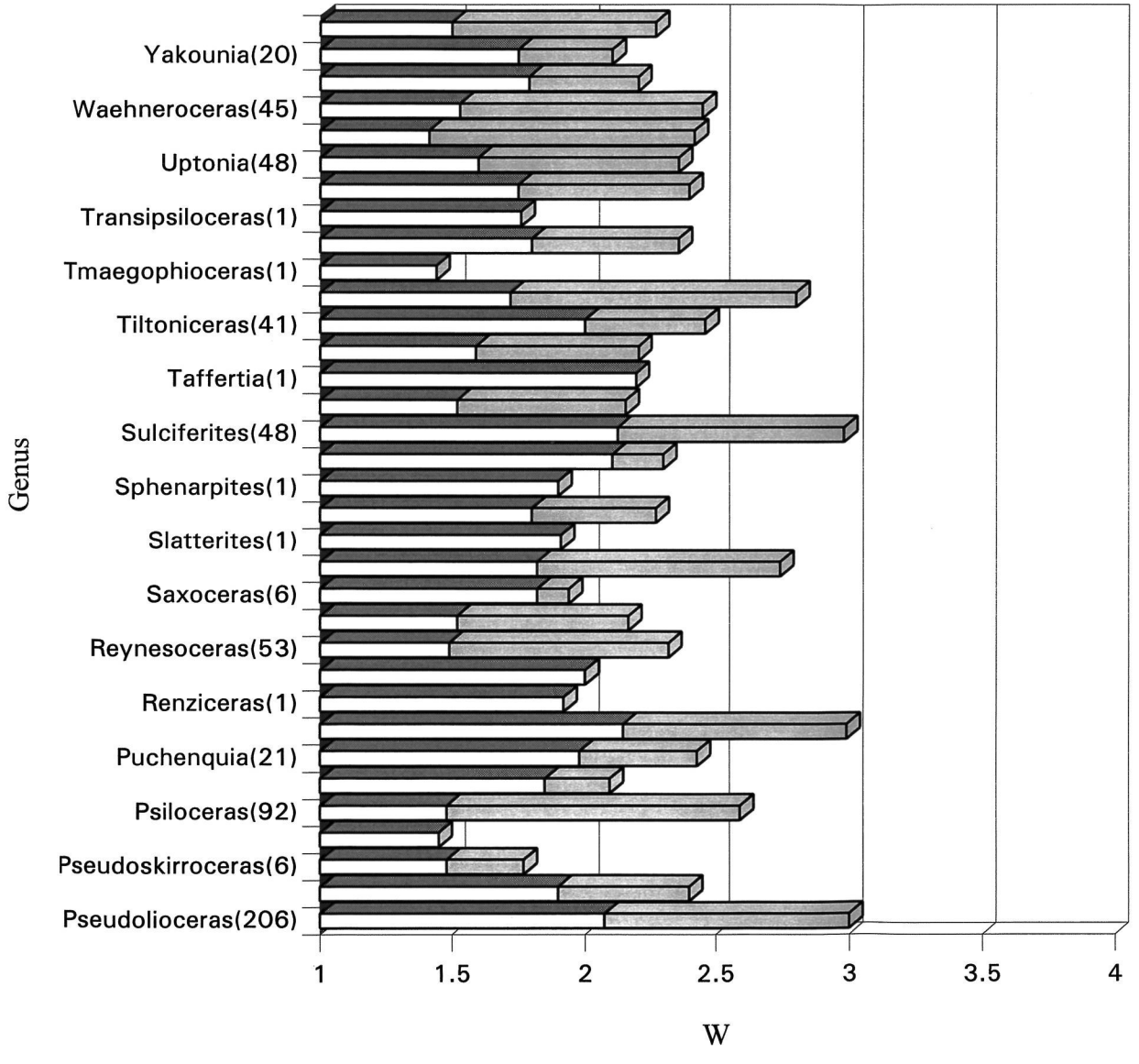


Figure 6-20 Shaded boxes show the range of whorl overlap (U) for each genus, the figures in the bracket show number of specimens in each genus

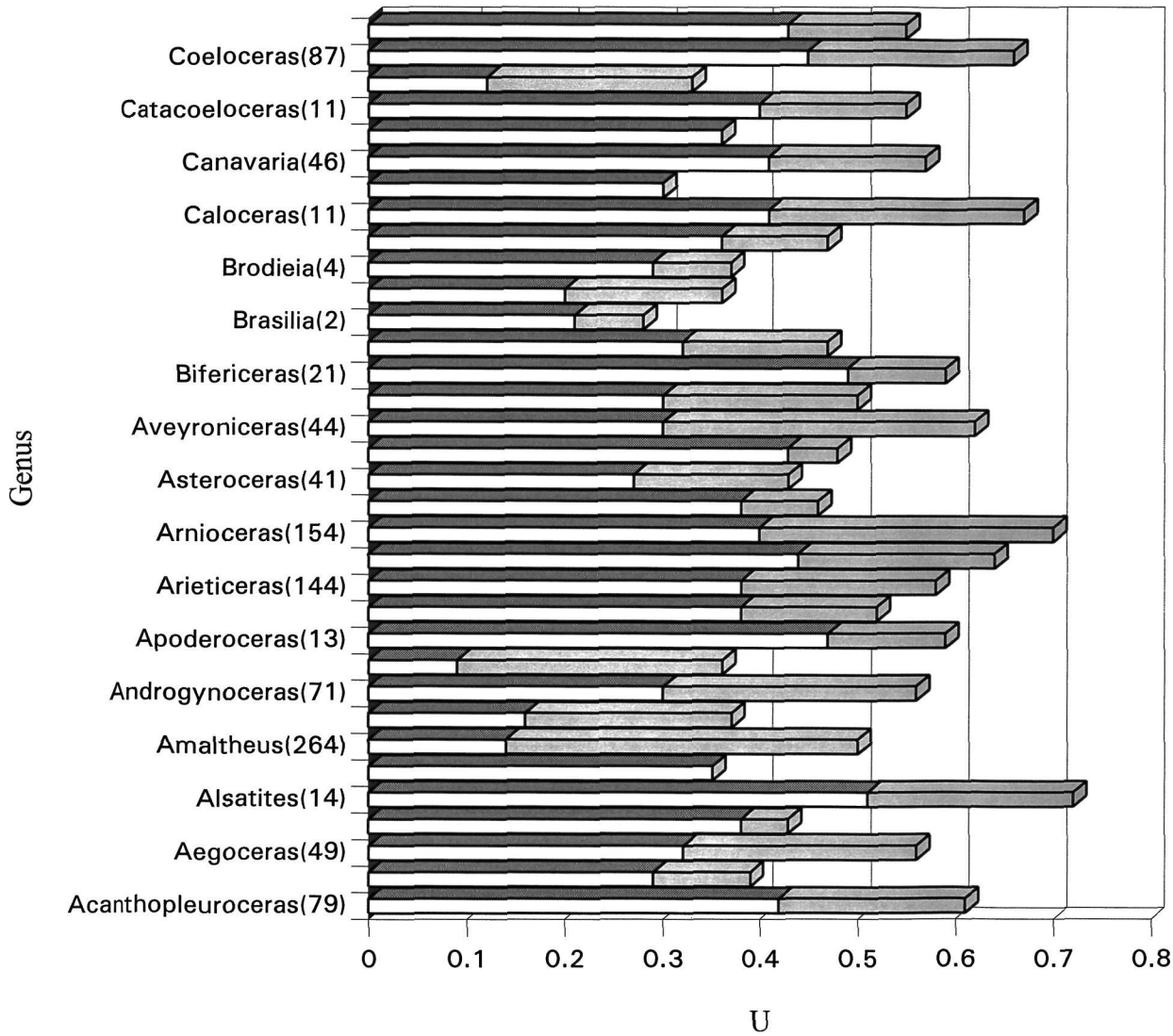


Figure 6-21 Shaded boxes show the range of whorl overlap (U) for each genus, the figures in the bracket show number of specimens in each genus

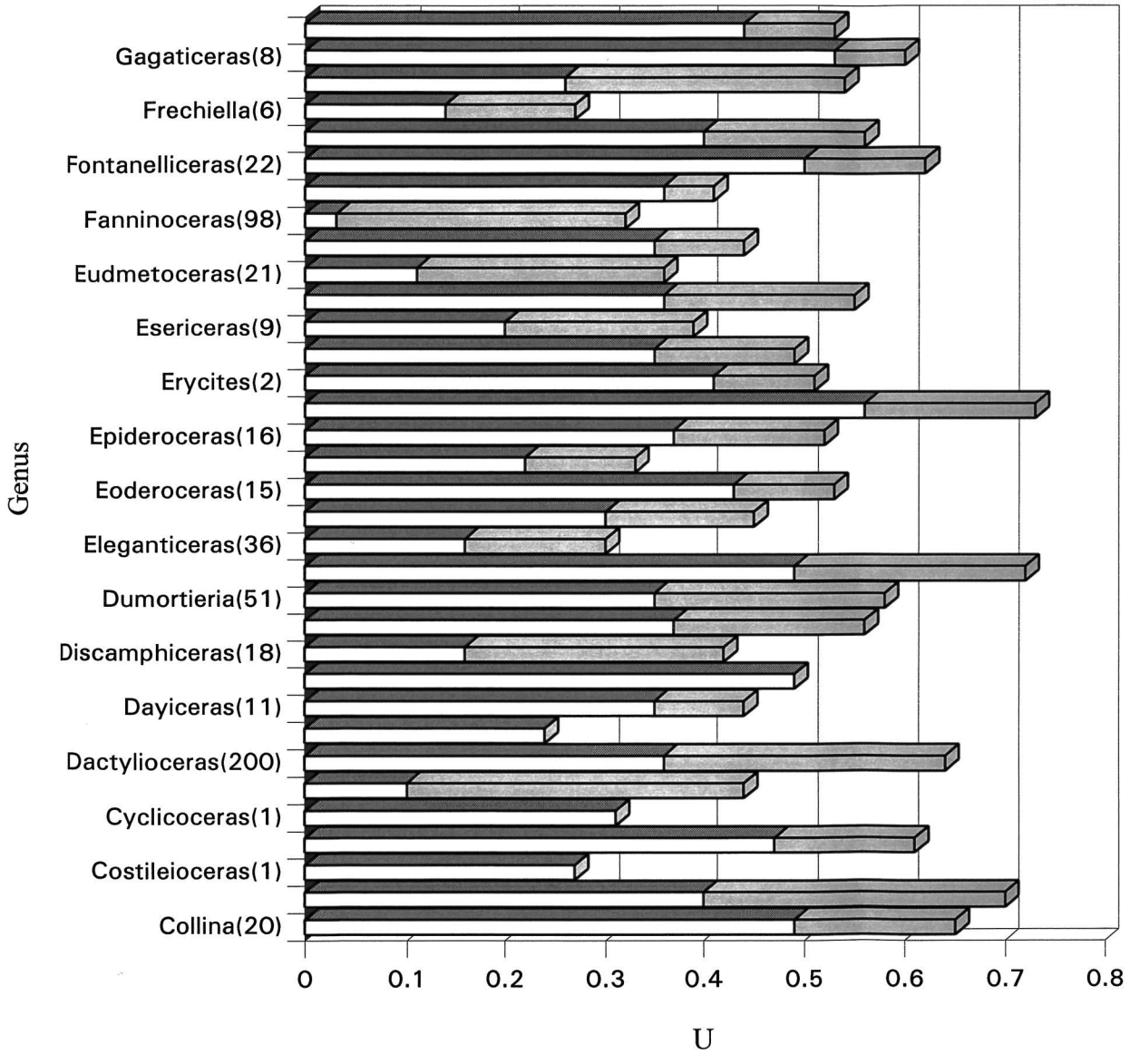


Figure 6-22 Shaded boxes show the range of whorl overlap (U) for each genus, the figures in the bracket show number of specimens in each genus

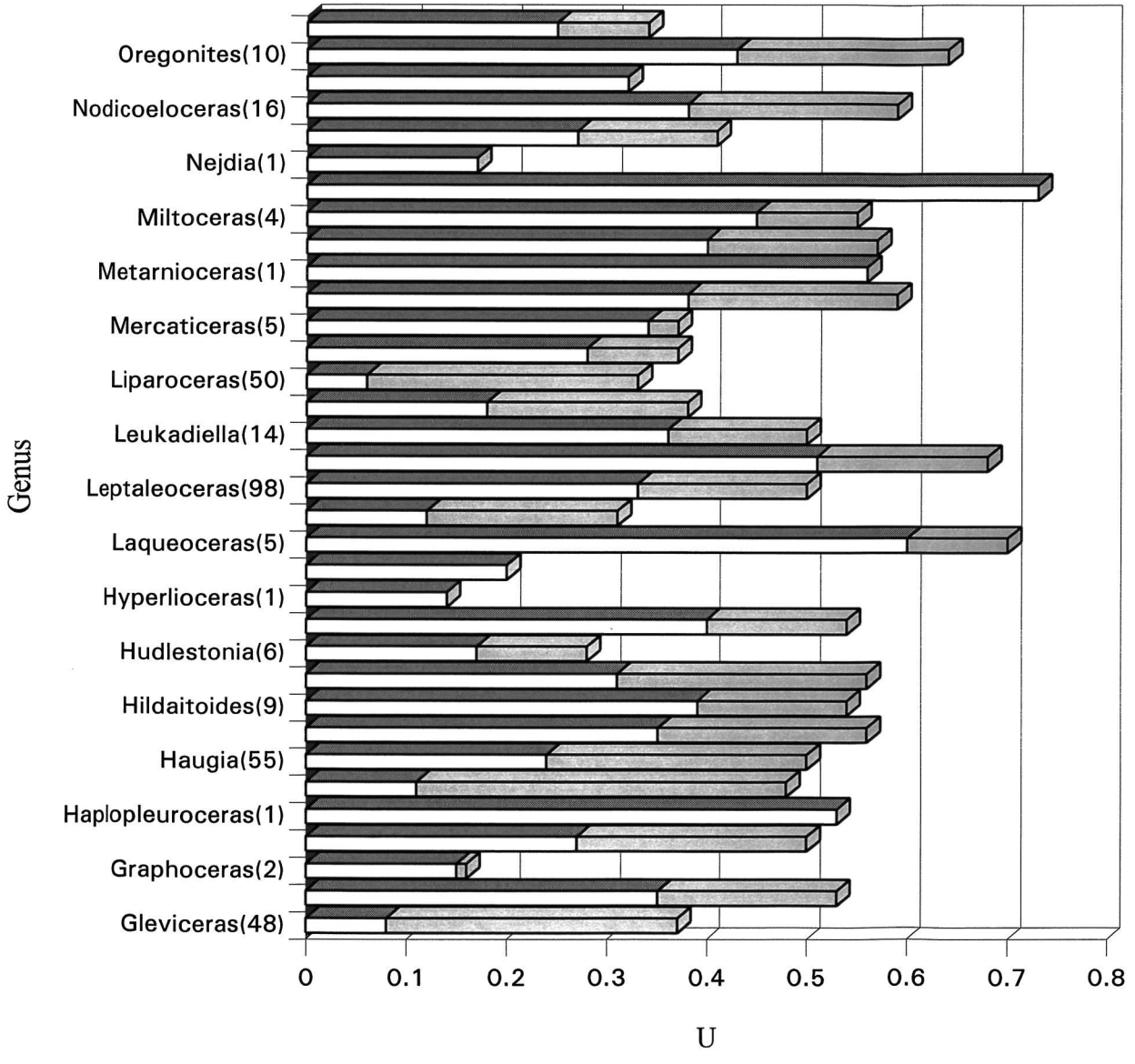


Figure 6-23 Shaded boxes show the range of whorl overlap (U) for each genus, the figures in the bracket show number of specimens in each genus

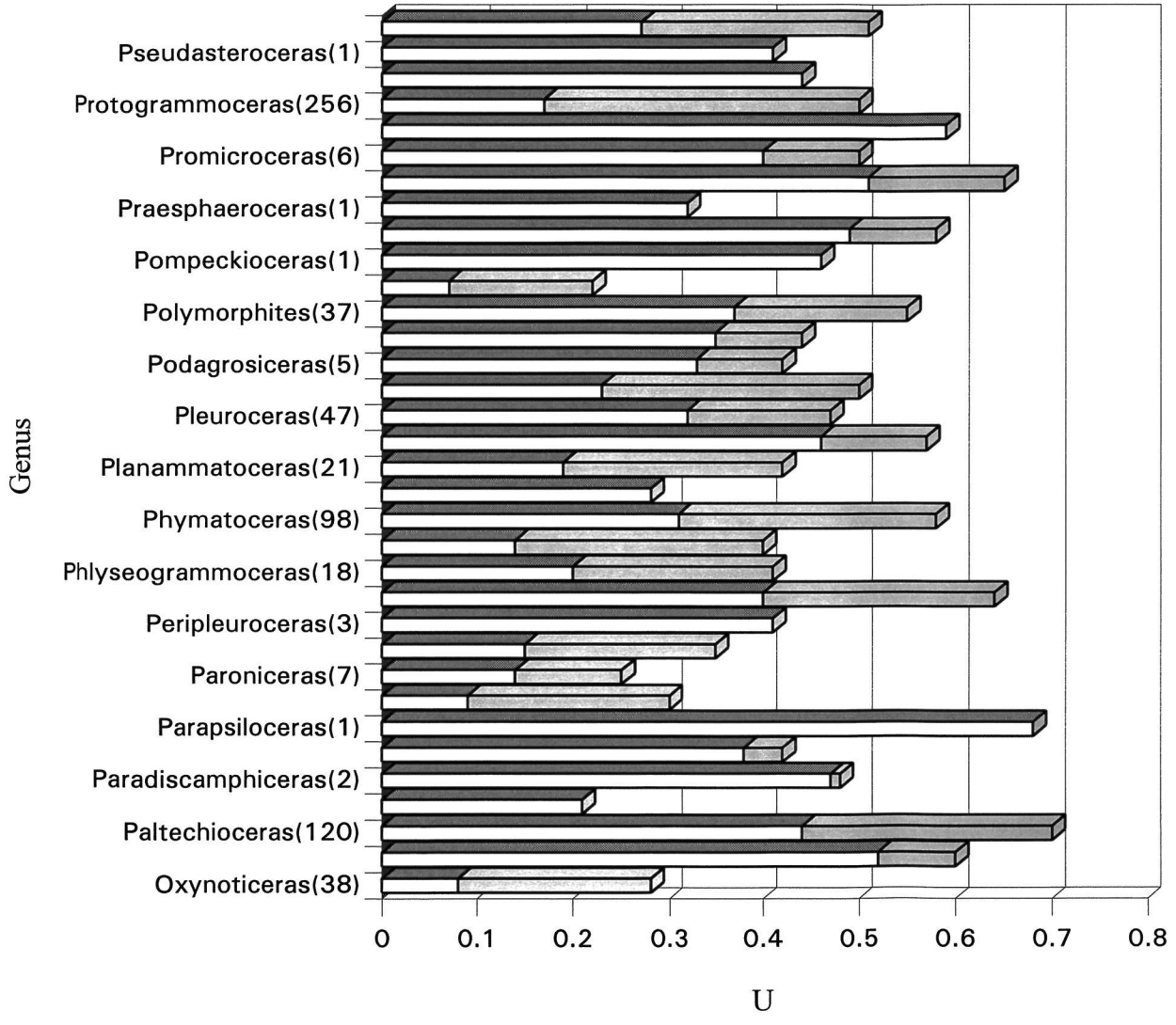


Figure 6-24 Shaded boxes show the range of whorl overlap (U) for each genus, the figures in the bracket show number of specimens in each genus

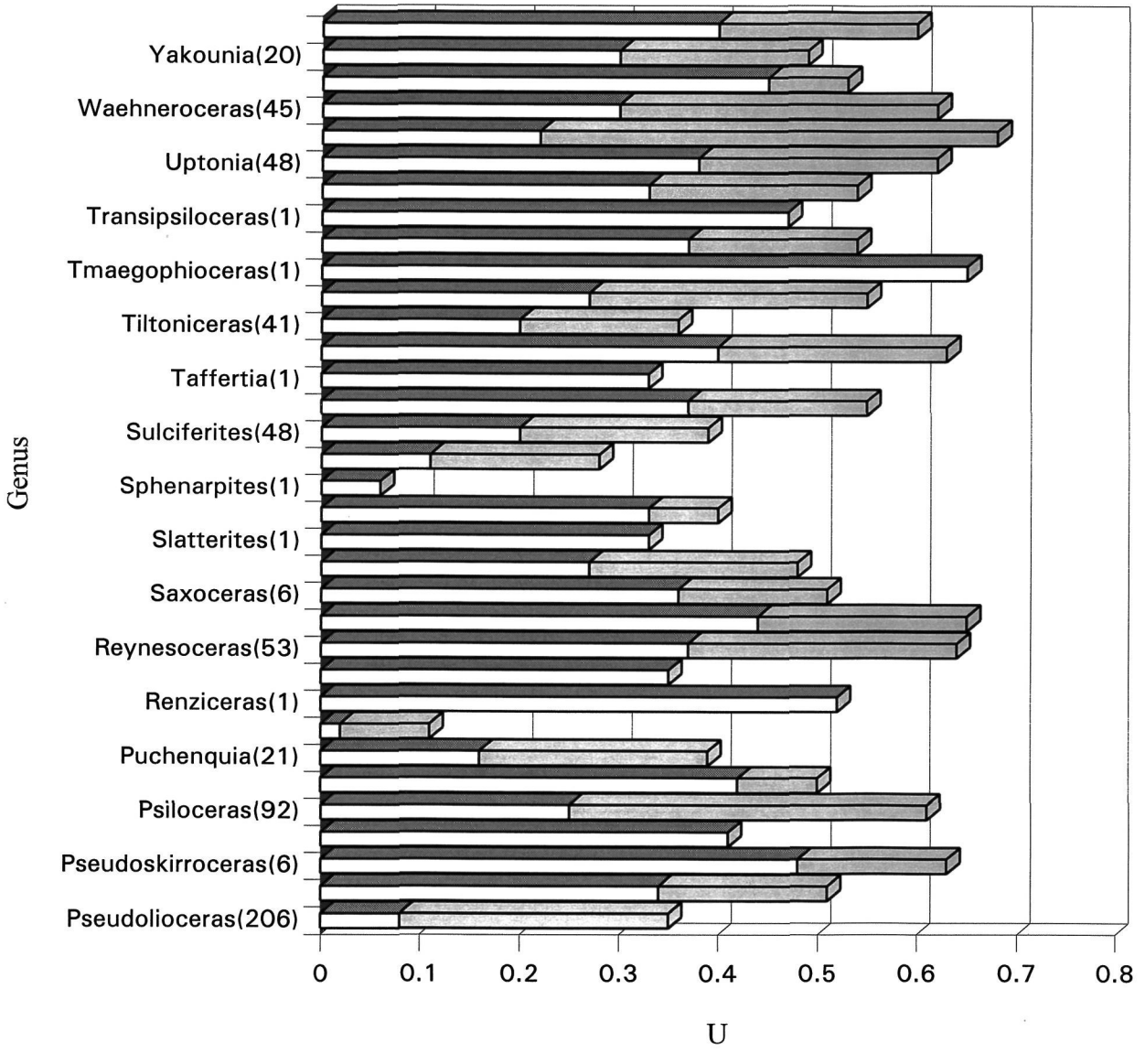


Figure 6-25 Shaded boxes show the range of WWWH for each genus, the figures in the bracket show number of specimens in each genus

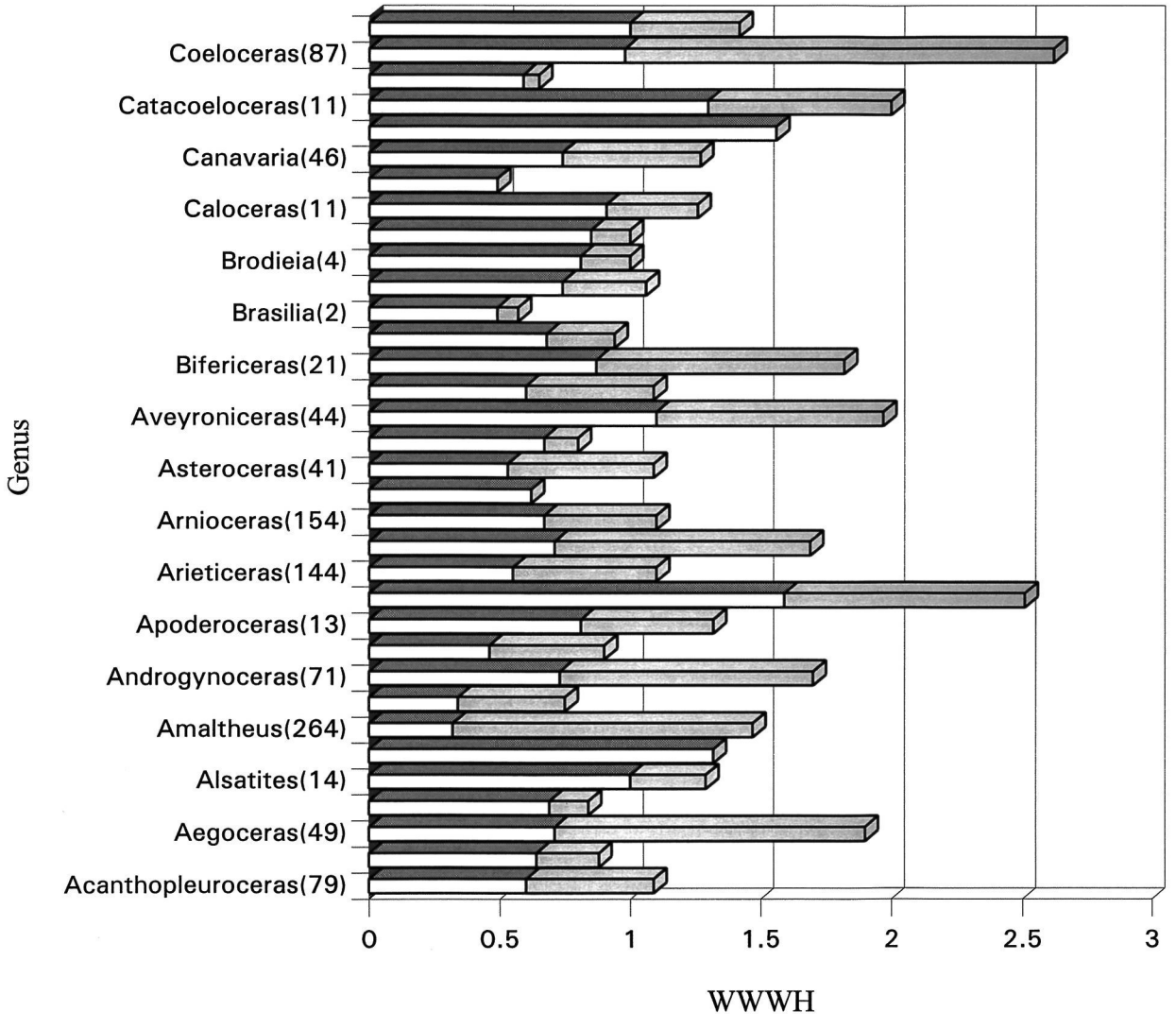


Figure 6-26 Shaded boxes show the range of WWWH for each genus, the figures in the bracket show number of specimens in each genus

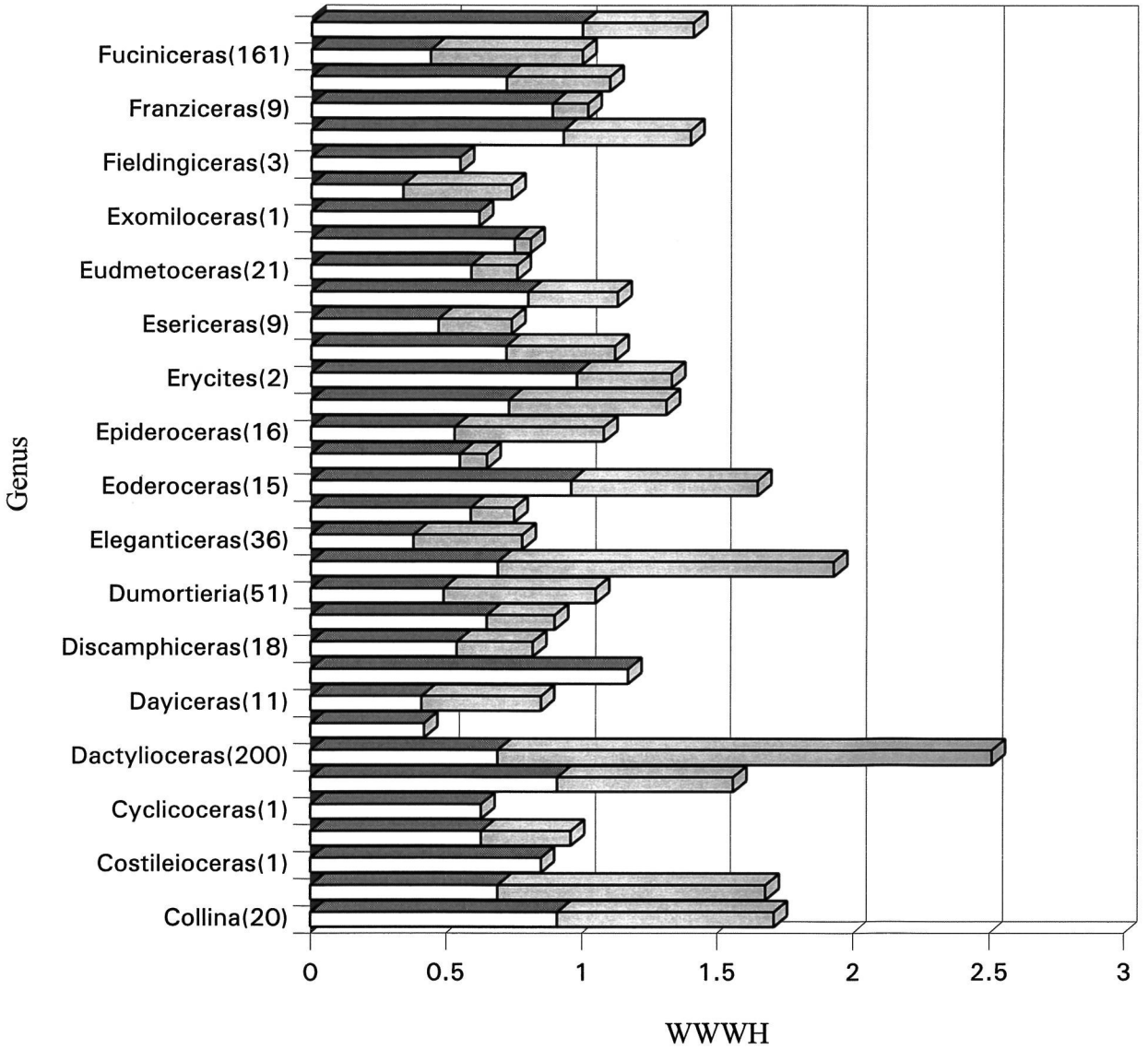


Figure 6-27 Shaded boxes show the range of WWWH for each genus, the figures in the bracket show number of specimens in each genus

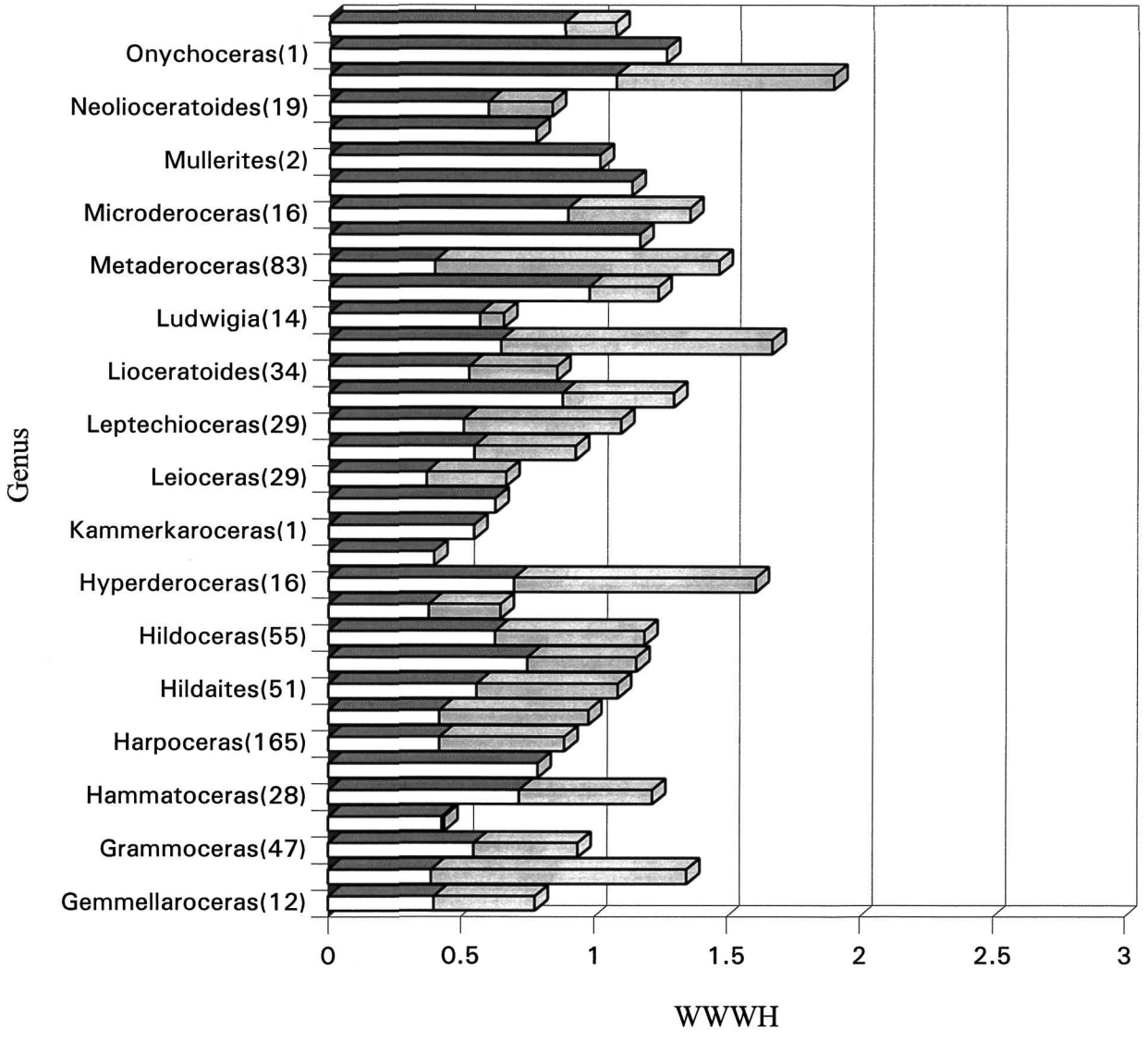


Figure 6-28 Shaded boxes show the range of WWWH for each genus, the figures in the bracket show number of specimens in each genus

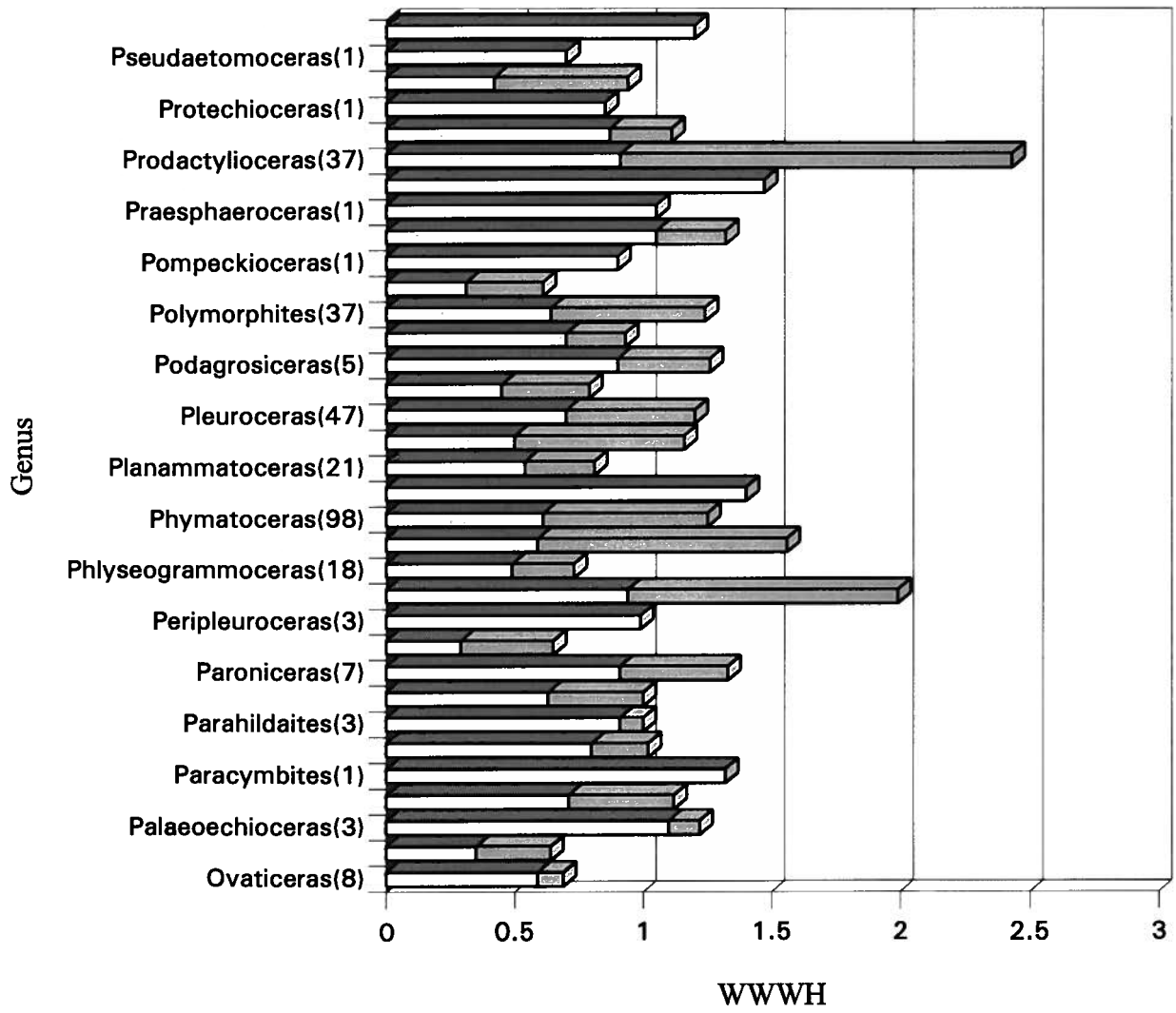
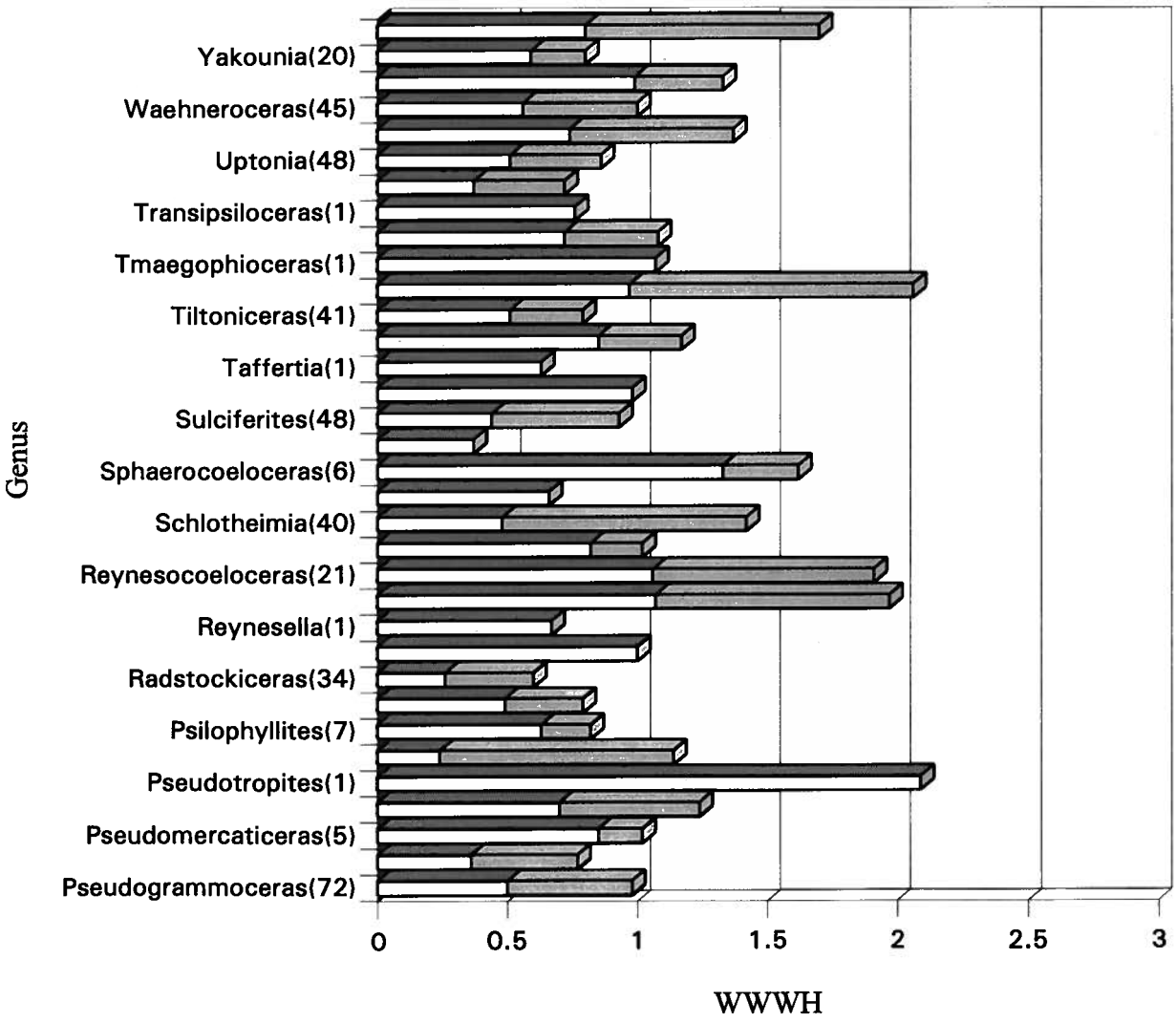


Figure 6-29 Shaded boxes show the range of WWWH for each genus, the figures in the bracket show number of specimens in each genus



Chapter 7 Summary and Conclusions

This thesis presents the first computerized image database for ammonites which allows automated measurement of morphological parameters from images. The database Ammon contains 7790 specimens representing 15 Lower Jurassic ammonite families, 179 genera and 1319 species. Each specimen has 102 descriptors covering taxonomy, quantitative morphology, qualitative morphology, stratigraphy, locality and general comments. Computer image analytical techniques are used to derive morphological parameters from images in the database and a graphics interface is implemented to give the user easy access.

Elliptical Fourier representation is used to quantify whorl shape for the first time which allows the construction of intermediate whorl shapes and the theoretical extension of morphologic trends beyond the forms available. Fourier representation of whorl shape enables a more objective documentation and comparison of whorl shape variations within and among taxa.

The species density diagrams for Lower Jurassic ammonites are plotted with respect to basic shell geometry. The distributions are strikingly similar to those for Mesozoic ammonoids in general, being largely concentrated on low W and high U regions and suggesting the same geometrical restraints. The data show strong correlations among basic geometric parameters (W, U, WWWH) which may be related to restrictions imposed by the shape and relative size of the whorl. This has been approached mathematically.

Preliminary study of morphological and stratigraphical data from Ammon suggests a possible correlation between basic shell geometry, morphological variability and taxa duration. Low within-population morphological variabilities and short durations are associated with taxa that inhabit the most densely populated region of low W and high U.

Away from this peak in faster expanding regions, there is a steady increase in both morphological variability and taxa longevity implying a decrease in selective pressure.

The inter-relationships of shell expansion rate, degree of involution, whorl shape, ribs, tubercles and venter have been studied using the Ammon database. A few covariations have been observed and documented quantitatively. The major covariation patterns are: (a) Faster expanding shells tend to be more involute, involute shells are more compressed than evolute shells, and compressed shells have finer ribs. (b) The frequency of shells with tubercles increases if ribs furcate. (c) Tubercles of compressed shells tend to be on the lower flank whereas tubercles of depressed shells tend to be on the upper flank. (d) Shells with sinuous ribs are more involute and more compressed than shells with straight ribs. (e) There is a positive correlation between tuberculation and rib strength. (f) The frequency of shells with a keels increases in shells that are compressed, or have a small fineness ratio. Some interpretations and speculations for these covariations are given.

The taxonomic diversity of Lower Jurassic ammonites peaks first at the family level in the Ibex Zone, followed by generic data one zone later, and finally species in the Margaritatus Zone implying a delay in radiation for lower levels of ammonite taxa. The late Sinemurian and early Pliensbachian have high family origination rates and relatively low species diversities; species reach their highest diversity level in the late Pliensbachian, which also has the highest family extinction rate and an almost zero origination rate. The data support the claim (Raup and Sepkoski 1986) that the Pliensbachian is a time of mass extinction. Furthermore, the data indicate that the extinction event happened in the Late Pliensbachian and was gradual for ammonites at the family level.

Two morphological diversity indexes are used to study changes through time. The range index describes the range of the morphospace occupied. The Shannon Index measures both the range and the evenness of the occupation. Both indexes indicate a similar pattern.

There is a rapid rise to an early Sinemurian maximum through morphological radiation especially in terms of the basic shell geometry, followed by a pronounced fall in the middle of Sinemurian with diversity remaining relatively high and stable thereafter. The data also show that Pliensbachian and Toarcian ammonites have significantly higher ornamentation diversities than earlier forms in the Hettangian and Sinemurian.

It is evident by comparing taxonomic and morphological diversity changes through time that initial morphological diversification in Lower Jurassic ammonites far exceeded the proliferation of lower-level taxa. The most striking fact emerging from the comparison is that the morphological diversity of the Bucklandi Zone with 46 species is higher than that of the Margaritatus Zone with 177 species. It demonstrates that morphological diversity need not increase as taxonomic diversity increases. "Such morphological stagnation indicates taxonomic diversification that is highly constrained morphologically" (Foote 1993).

There are two interesting observations although the resolution and type of my data does not warrant firm conclusions: (1) All Lower Jurassic ammonite families that radiate into numerous species in the first zone of their occurrence have durations of less than 4 zones; in other words, rapid origination is associated with short duration. (2) The mean maximum shell diameters of species within families and superfamilies show synchronized drops in the Oxynotum Zone and Ibex Zone. Interestingly, these two zones show high family origination rates and relatively low species diversities.

Finally a few specimens collected from the Iskut River map area are used to test Ammon as a tool for identification; the results are encouraging.

References

Ackerly, S. C. 1989. Kinematics of accretionary shell growth with examples from brachipods and molluscs. Paleobiology **15** : 147-164.

Anstey, R. L. and Delmet, D. A. 1973. Fourier analysis of zooecial shapes in fossil tubular bryozoans. Geological Society of America Bulletin **84**: 1753-1764.

Arkell, W. J., Kummel, B. and Wright, C. W. 1957. Mesozoic ammonoidea. *in* The Treatise on Invertebrate Paleontology (Moore, R. C., ed.), University of Kansas Press.

Arnold, A. J. and Fristrup, K. 1982. The theory of evolution by natural selection: a hierarchical expansion. Paleobiology **8(2)**: 113-129.

Anonymous 1990a. Introduction to SQL by Oracle Corporation.

-- 1990b. SQL*Loader User's Guide (Version 1.0) by Oracle Corporation.

-- 1990c. SQL Language Reference Manual (Version 6.0) by Oracle Corporation.

Arthur, A.J. 1985. Biostratigraphy of the Lower to Middle Jurassic sediments, Ashcroft area, south central British Columbia. B.Sc. thesis, University of British Columbia.

-- 1987. Mesozoic stratigraphy and paleontology of the west side of Harrison Lake, southwestern British Columbia. M.Sc. thesis, University of British Columbia.

Arthur, A., Smith, P. L., Monger, J. W. and Tipper, H. W. 1992: Mesozoic stratigraphy and Jurassic paleontology west of Harrison Lake, southwestern British Columbia. Geological Survey of Canada Bulletin **441**.

Batt, R. J. 1989. Ammonite shell morphotype distributions in the Western Interior Greenhorn Sea and some paleoecological implications. Palaios **4**: 32-42.

Bayer, U. 1985. Pattern recognition problems in geology and paleontology. Berlin: Springer-Verlag.

Bayer, U. and McGhee, G. R. 1984. Iterative evolution of Middle Jurassic ammonite faunas. Lethaia, **17**: 1-15.

-- 1985. Evolution in marginal epicontinental basins: the role of phylogenetic and ecological factors. *in* Sedimentary and Evolutionary Cycles (Bayer, U. and Seilacher, A., eds.) p. 164-215, Springer-Verlag, Berlin.

Beerbower, J. Jordan, D. 1969. Application of information theory to paleontologic problems: taxonomic diversity. Journal of Paleontology, **43(5)**: 1184-1198.

Billings, E. 1873. On the Mesozoic Fossils from British Columbia collected by Mr. James Richardson in 1872. Geological Survey of Canada, Report of Progress, 1872-1873, p. 71-75.

Blasco, G., Levy, R. and Nullo, F. 1978. Los amonites de la formacion Osta Arena (Liasico) y su posicion estratigrafica-Pampa de Agnia-(Provincia del Chubut): VII Congreso Geologico Argentino, Neuquen, Actas, **2**: 407-429.

Benson, R. H., Chapman, R. E. and Siegel, A. F. 1982. On the measurement of morphology and its change. Paleobiology **8**: 328-339.

Bisby, F. A. 1983. Information services in taxonomy. In Allkin, R. and Bisby, F. A. eds. Databases in Systematics.

Canfield, D. J. and Anstey, R. L. 1981. Harmonic analysis of Cephalopod suture patterns. Mathematical Geology **13**: 23-35.

Canny, J. 1986. A computaional approach to edge detection. IEEE Trans. on pattern analysis and machine intelligence **8**: 679-697.

Chamberlain, J. A. 1976. Flow patterns and drag coefficients of cephalopod shells. Paleontology **19(3)**: 539-563.

-- 1981. Hydromechanical design of fossil cephalopods In: House, M. R. and Senior, J. R. eds. The Ammonoidea. Systematics Association Special Volume **18**: 289-336.

Checa, A. 1987. Morphogenesis in ammonites - differences linked to growth pattern. Lethaia **20**: 141-148.

Checa, A. and Westermann, G. E. G. 1989. Segmental growth in planulate ammonites: inferences on coastal function. Lethaia **22**: 95-100.

Christopher, R. A. and Waters, J. A. 1974. Fourier series as a quantitative descriptor of miospore shape. Journal of Paleontology **48**: 697-709.

Clark, M. W. 1981. Quantitative shape analysis: a review. Mathematical Geology **13**: 303-320.

Cleynenbreugel, V. et al 1988. Knowledge-based road network extraction on SPOT satellite images. 4th International Conference on Pattern Recognition.

Conrad, M. A. and Beightol, D. S. 1988. Expert systems identify fossils and manage large paleontological databases. Geobyte, No. 2 : 42-46.

Crickmay, C.H. 1925: A note on two of Hyatt's Liassic ammonites. Proceedings of the California Academy of Sciences, fourth series, **14**: 77-81.

-- 1928: The stratigraphy of Parson Bay, British Columbia. University of California Publications, Bulletin of the Department of Geological Sciences , **18**: 51-70.

-- 1932 : A new Jurassic ammonite from the Coast Ranges of California. American Midland Naturalist, **13**:1-7, pl. 1-2.

-- 1932-1933: Some of Alpheus Hyatt's unfigured types from the Jurassic of California. United States Geological Survey, Professional Paper **175-B**: 51-64, pl. 14-18.

-- 1933: Mount Jura investigation. Bulletin of the Geological Society of America, **44**: 895-926, pl. 23-34 .

Dagis, A.A. 1968. Toarcian Ammonites (Dactylioceratidae) from northern Siberia. U.S.S.R. Academy of Sciences, Siberian Branch, Transactions of the Institute of Geology and Geophysics, no. 40, 108 p., 12 pl.

-- 1971. On the Toarcian-Pliensbachian boundary and zonation of the Lower Toarcian sediments in the northeastern U.S.S.R.. Geologiya i Geofizika, no. 5 (136), p. 128-132.

-- 1974. Toarcian ammonites (Hildoceratidae) from northern Siberia. U.S.S.R. Academy of Sciences, Siberian Branch, Transactions of the Institute of Geology and Geophysics, no. 99, 188 p., 19 pl.

-- 1976. Late Pliensbachian ammonites (Amaltheidae) of the North Siberia: Publishing House, Nauka, Siberian Branch, Novosibirsk, 79 p., 17 pl. (in Russian).

Davis, J. C. 1986. Statistics and data analysis in geology. New York: Wiley.

Dawson, G.M. 1880. Queen Charlotte Islands. Geological Survey of Canada, Report of Progress, 1878-1879.

Dean, W.T., Donovan, D.T. and Howarth, M.K. 1961. The liassic Ammonite zones and subzones of North-West european province. Bulletin of the British Museum (Natural History), **4**(10), p. 437-505, pl. 63-74.

Doguzhaeva, L. 1982. Rhythms of ammonoid shell secretion. Lethaia **15:** 385-394.

Dommergues, J. L. 1988. Can ribs and septa provide an alternative standard for age in ammonite ontogenetic studies ? Lethaia **21:** 243-256.

Donovan, D.T. 1964. Cephalopod phylogeny and classification. Biology Review **39**: 259-287

-- 1985. Ammonite shell form and transgression in the British Lower Jurassic *in* Sedimentary and Evolutionary Cycles (Bayer, U. and Seilacher, A., eds.) p. 48-57, Springer-Verlag. Berlin.

Donovan, D.T., Callomon, J.H. and Howarth, M.K. 1981. Classification of the Jurassic Ammonitina. *in* The Ammonoidea, (House, M.R. and Senior, J.R., eds.), Systematics Association, Special Publication 18, Academic Press, p. 101-155.

Ehrlich, R. 1983. Comments on the validity of Fourier descriptors in systematics: a reply to Bookstein. Systematic Zoology 32 : 202-206.

Erben, H.K. 1956: El Jurasico inferior de Mexico y sus amonitas. XX Congreso Geologico Internacional, 393 p.

Foote, M. 1989. Perimeter-based Fourier analysis: a new morphometric method applied to the trilobite cranidium. Journal of Paleontology **63(6)**: 880-885.

-- 1991. Morphologic patterns of diversification: examples from trilobites. Paleontology **34(2)**: 461-485.

-- 1992. Rarefaction analysis of morphological and taxonomic diversity. Palaeobiology **18(1)**: 1-16.

-- 1993. Discordance and concordance between morphological and taxonomic diversity. Paleobiology **19(2)**: 185-204.

Frebold, H. 1951: Lowermost Jurassic Fauna in Whitesail Lake map-area, British Columbia. in Contributions to the Palaeontology and Stratigraphy of the Jurassic System in Canada, Geological Survey of Canada Bulletin **18**: 18-21, 1 pl.

-- 1957: The Jurassic Fernie Group in the Canadian Rocky Mountains and Foothills. Geological Survey of Canada Memoir **287**:1-197, 44 pl.

-- 1958. Stratigraphy and correlation of the Jurassic in the Canadian Rocky Mountains and Alberta Foothills. *in* Jurassic and Carboniferous of Western Canada, American Association of Petroleum Geologists.

-- 1959: Marine Jurassic rocks in Nelson and Salmo areas southern British Columbia. Geological Survey of Canada, Bulletin **49**: 31 p., 5 pl.

-- 1960. The Jurassic faunas of the Canadian Arctic (Lowermost Jurassic and lowermost Middle Jurassic ammonites). Geological Survey of Canada, Bulletin **59**: 1-33, pl. 1-15.

-- 1964a: Lower Jurassic and Bajocian ammonoid faunas of northwestern British Columbia and southern Yukon. Geological Survey of Canada Bulletin **116**: 31 p. 8 pl.

-- 1964b: Illustrations of Canadian fossils Jurassic of Western and Arctic Canada. Geological Survey of Canada, Paper **63-4**: 1-107, pl. 1-51.

-- 1966: Upper Pliensbachian beds in the Fernie Group of Alberta. Geological Survey of Canada, Paper **66-27**: 1-7, 1 pl.

-- 1967a: Hettangian ammonite faunas of the Taseko lakes area British Columbia. Geological Survey of Canada, Bulletin **158**: 1-35, pl. 1-9.

-- 1967b: Position of the Lower Jurassic genus *Fanninoceras* McLearn and the age of the Maude Formation on Queen Charlotte Islands. Canadian Journal of Earth Sciences, **4**: 1145-1149.

-- 1970: Pliensbachian ammonoids from British Columbia and Southern Yukon. Canadian Journal of Earth Sciences, **7**: 435-452, pl. 1-4.

-- 1975. The Jurassic faunas of the Canadian Arctic (lower Jurassic ammonites, biostratigraphy and correlation) Geological Survey of Canada, Bulletin **243**, 24 pps., 5 pl.

-- 1976: The Toarcian and Lower Middle Bajocian beds and ammonites in the Fernie Group of southeastern British Columbia and parts of Alberta. Geological Survey of Canada Paper **75-39** : 20 p., 9 pl.

Frebold, H. Mountjoy, E.W. and Tempelman-Kluit, D. 1967. New occurrences of Jurassic rocks and fossils in central and northern Yukon. Geological Survey of Canada, Paper **67-12**

Frebold, H. and Tipper, H.W. 1969. Lower Jurassic rocks and fauna near Ashcroft, British Columbia and their relation to some granitic plutons (92-1). Geological Survey of Canada, Paper **69-23**: 20 p., 1 pl.

Frebold, H. and Tipper, H.W. 1970. Status of the Jurassic in the Canadian Cordillera of British Columbia, Alberta and Southern Yukon. Canadian Journal of Earth Sciences, **7**(1): 1-21.

Frebold, H., Tipper, H.W. and Coates, J.A. 1969. Toarcian and Bajocian rocks and guide ammonites from Southwestern British Columbia. Geological Survey of Canada,

Paper **67-10**: 55 p., 6 pl.

Gabb, W.M. 1869. Descriptions of some secondary fossils from the Pacific Coast States. American Journal of Conchology, **5**: 5-18, pl. 3 .

Gabilly, J. 1975. Évolution et systéque des Phymatoceratinae et des Grammoceratinae (Hildocerataceae Ammonitina) de la région de Thouars, stratotype du Toarcien. Mémoires de la Société Géologique de France, no. 124, Nouvelle Série, tome LIV, p. 1-196, pl. I-XXXVI.

-- 1976. Le Toarcien à Thouars et dans le centre-ouest de la France - biostratigraphie - evolution de la faune (Harpoceratinae, Hildoceratinae). Les Stratotypes Français, Publications du Comite Français de Stratigraphie, **3**: 1-217, pl. 1-29.

Géczy , B. 1976. Les ammonitines du Carixien de la Montagne du Bakony. Akadémiai Kiadó, Budapest. 223 p.

Geyer, O. F. 1974. Der Unterjura (Santiago-Formation) von Ekuador. Neues Jahrbuch für Geologie und Paläontologie, Monatshefte, **9**: 525-541.

Geyer, O. F. 1979. Ammoniten aus dem tiefen Unterjura von Nord-Peru. Paläontologische Zeitschrift, **53**: 198-213.

Gould, S. J. 1991. The disparity of the Burgess Shale arthropod fauna and the limits of cladistic analysis: why we must strive to quantify morphospace. Paleobiology **17(4):** 411-423.

Gregory, S. 1963. Statistical methods and the geographer. Longmans, Green and Co. Ltd.

Guex, J. 1980. Remarques preliminaires sur la distribution stratigraphique des ammonites hettangiennes du New York Canyon (Gabbs Valley Range, Nevada). Bulletin de Geologie Lausanne, **250**: 127-140.

-- 1981. Quelques cas de dimorphisme chez les ammonoides du Lias inferieur. Bulletin Soc. Vaud. Sc. Nat., **75:**239-248.

-- 1989. Note sur le genre *Franziceras* Buckman (Ammonoidea, Cephalopoda). Bulletin de Geologie Lausanne, **305**:347-354.

Hall, R.L. 1987. New Lower Jurassic ammonite fauna from the Fernie Formation, southern Canadian Rocky Mountains. Canadian Journal of Earth Science, **24**: 1688-1704, 5 pl.

Hall, R.L. and Howarth, M.K. 1983. *Protogrammoceras paltum* (Buckman), a late Pliensbachian (Jurassic) ammonite from Axel Heiberg Island, Canadian Arctic Archipelago. Canadian Journal of Earth Sciences, **20**: 1470-1475.

Hall, R.L. and Westermann, G.E.G. 1980. Lower Bajocian (Jurassic) cephalopoda faunas from Western Canada and proposed assemblage zones for the Lower Bajocian of North America. Palaeontographica Americana, v. **9**(52): 1-93, pl. 1-15.

Hallam, A. 1985. Jurassic molluscan migration and evolution in relation to sea level changes. *in* Sedimentary and Evolutionary Cycles (Bayer, U. and Seilacher, A., eds.) p. 4-5, Springer-Verlag, Berlin.

-- 1986. The Pliensbachian and Tithonian extinction events. Nature **319**: 765-767.

-- 1987. Radiations and extinctions in relation to environmental change in the marine Lower Jurassic of northwest Europe. Paleobiology **13(2):** 152-168.

Harland, W.B., Armstrong, R.L., Cox, A.V., Craig, L.E., Smith, A.G. and Smith, D.G. 1989. A geologic time scale, 1989. Cambridge University Press.

Hillebrandt, A. von. 1973a. Neue Ergebnisse über den Jura in Chile und Argentinien. Münstersche Forschungen zur Geologie und Paläontologie, Heft **31/32**: 167-199, pl. 1.

-- 1973b. Die Ammoniten-Gattungen *Bouleiceras* und *Frechiella* im Jura von Chile und Argentinien. Eclogae geologicae Helvetiae, **66/2**: 351-363, pl. I-III.

-- 1982. Faunas de amonites del Liasico inferior y medio (Hettangiano hasta Pliensbachiano) de America del sur (Excluyendo Argentina). Cuncas sedimentarias del Jurasico y Cretacico de America del sur, **2**: 499-538, pl. I-X.

-- 1987. Liassic ammonite zones of South America and correlations with other provinces with descriptions of new genera and species of ammonites. Biostratigrafía de los sistemas regionales del Jurásico y Cretácico en América del Sur, Tomo I: Jurásico anterior a los movimentos málmicos, **1**: 111-157, 14 pl.

Hillebrandt, A. von and Schmidt-Effing, R. 1981. Ammoniten aus dem Toarcium (Jura) von Chile (Südamerika) - Die Arten der Gattungen *Dactylioceras*, *Nodicoeloceras*, *Peronoceras* und *Collina*. Zitteliana, Abhandlungen der Bayerischen Staatssammlung für Paläontologie und historische Geologie, **6**, 74 p., 8 pl.

Hirano, H. 1971a. Biostratigraphic study of the Jurassic Toyora Group, Part I. Memoirs from the Faculty of Science, Kyushu University, Series D, Geology, **21**(1): 93-128, pl. 14-20.

-- 1971b. Biostratigraphic Study of the Jurassic Toyora Group, Part II. Transactions and Proceedings of the Palaeontological Society of Japan, N.S., no. 89, p. 1-14, pl. 1-4.

-- 1971c. Biostratigraphic Study of the Jurassic Toyora Group, Part III. Transactions and Proceedings of the Palaeontological Society of Japan, N.S., no. 90, p. 45-71, pl. 9-10.

Hohenegger, J. and Tatzreiter, F. 1992. Morphometric methods in determination of ammonite species, exemplified through *Balatonites* shells (Middle Triassic). Journal of Paleontology **66(5)**: 801-816.

Horn, B. K. P. 1986. Robert Vision. The MIT Press.

Howarth, M.K. 1992. The ammonite family Hildoceratidae in the Lower Jurassic of Britain. Monograph of the Palaeontographical Society, part 1, 1-106, pl. 1-16.

Huber, B. T. 1990. Digital-image processing will replace most hard-copy photography in palynology. Geotimes **35(2)**: 43-44.

Illert, C. and Pickover, C. A. 1992. Generating irregularly oscillating fossil seashells. IEEE Computer Graphics and Applications. No. **5**: 18-22.

Imlay, R.W. 1968. Lower Jurassic (Pliensbachian and Toarcian) Ammonities from Eastern Oregon and California. United States Geological Survey, Professional Paper **593-C**: 51. 9 pl.

Imlay, R.W. 1981. Early Jurassic Ammonites from Alaska. United States Geological Survey, Professional Paper **1148**: 49 p., 12 pl.

Jakobs, G. 1992. Toarcian (Lower Jurassic) ammonite biostratigraphy and ammonite fauna of North America. Unpublished Ph.D. thesis, University of British Columbia, 682 p.

Jones, D.L. and Moore, J.G. 1973. Lower Jurassic ammonite from the South-Central Sierra Nevada, California. Journal of Research, U.S.Geological Survey, **1**: 453-458.

Kaesler, R. L. and Waters, J. A. 1972. Fourier analysis of the ostracode margin : Geological Society of America Bulletin **83**: 1169-1178.

Kauffman, E. G. 1977. Evolutionary rates and biostratigraphy. In Kauffman, E. G. and Hazel, J. E. (eds.) Concepts and Methods of Biostratigraphy: 109-141.

Kennedy, W. J. and Cobban, W. A. 1976. Aspects of ammonite biology, biogeography and biostratigraphy. Special Papers in Palaeontology, Palaeontological Association, London, **17**: 1-94.

Klapper, G. and Foster, C. T. J. 1993. Shape analysis of Frasnian species of the Late Devonian conodont genus Palmatolepis. Journal of Paleontology 1993, supplement to No. 4.

Koch, G. S. J. and Link, R. F. 1971. Statistical analysis of geological data. New York : Wiley.

Krumbein, W. C. and Graybill, F. A. 1965. Introduction to statistical models in geology. New York: McGraw-Hill.

Kuhl, F. P. and Giardina, C. R. 1982. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing **18**: 236-258.

Kullmann, J., Korn, D., Kullmann, P. S. and Petersen, M. S. 1991. The status of GONIAT, a data-base for Paleozoic ammonoids. First Canadian Paleontology Conference, Vancouver. p. 48.

Landman, N. H. 1989. Iterative progenesis in Upper Cretaceous ammonites. Paleobiology **15(2)**: 95-117.

Landman, N. H., Rye, D. M. and Shelton, K. L. 1983. Early ontogeny of *Eutrephoceras* compared to Recent *Nautilus* and Mesozoic ammonites: evidence from shell morphology and light stable isotopes. Paleobiology **9(3)**: 269-279.

Lehmann, U. 1981. The ammonites -- their life and their world. Cambridge University Press.

Liang, B. 1988. A knowledge-based system for analysing sedimentary environment. Geological Science and Technology Information **7**: 125-130.

-- 1988. Artificial intelligence and its application to the Guizhou Coal Basin, China. Earth Science **9(3)**: 350-355.

-- 1990. An object-oriented pattern recognition algorithm. In Zhao, P. D. (ed.): Statistical analysis in mineral exploration. p. 120-123.

-- 1990. Expert system and its application in sedimentary environment analysis. In Zhao, P. D. (ed.): Statistical analysis in mineral exploration. p. 65-73.

Liang, B. and Smith, P. L. 1991. Measurements from ammonoid images using a computer. First Canadian Paleontology Conference, Vancouver, p. 50.

-- 1992. A Lower Jurassic ammonite image database: applications and implications. Second Canadian Paleontology Conference, Ottawa, p. 16-17.

Lohmann, G. P. 1983. Eigenshape analysis of microfossils : a general morphometric procedure for describing changes in shape. Mathematical Geology **15**: 659-672.

Lupher, R.L. 1941. Jurassic Stratigraphy in Central Oregon. Bulletin of the Geological Society of America, **52**: 219-270, 4 pl., 3 fig.

MacKenzie, J.D. 1916. Geology of Graham Island, British Columbia. Geological Survey of Canada, Memoir 88.

Marr, D. 1982. Vision. San Francisco: Freeman.

Meister, C. 1986. Les ammonites du Carixien des Causses, France. -- Mém. Suis. Paléont, **109**, 209 p.

-- 1989. Les ammonites du Domérien des Causses, France. -- Analyses paléontologiques et stratigraphiques. -- Cahiers Pal. (CNRS edit), 98 p., Paris 1989

McLearn, F.H. 1930. Notes on some Canadian Mesozoic faunas. Transactions of the Royal Society of Canada, series 3, **24**:1-7, 2pl.

-- 1932. Contributions to the stratigraphy and paleontology of Skidegate Inlet, Queen Charlotte Islands, British Columbia. Transactions of the Royal Society of Canada, series 3, **26**: 51-80, 10 pl.

Mutvei, H. and Reyment, R. A. 1973. Buoyancy control and siphuncle function in ammonoids. Palaeontology **16(3)**: 623-636.

O'Brien, J.A. 1987. Jurassic biostratigraphy and evolution of the Methow Trough, southwestern British Columbia. B.Sc. thesis, University of British Columbia.

Okamoto, T. 1988. Analysis of heteromorph ammonoids by differential geometry. Palaeontology **31**: 35-52.

Pálfy, J. 1991. Uppermost Hettangian to lowermost Pliensbachian (Lower Jurassic) biostratigraphy and ammonoid fauna of the Queen Charlotte Islands, British Columbia.- Unpublished M.Sc. thesis, University of British Columbia, Vancouver .

Poulton, T.P. 1991. Hettangian through Aalenian (Jurassic) guide fossils and biostratigraphy, northern Yukon and adjacent Northwest Territories. Geological Survey of Canada, Bulletin 410, 95 p., 18 pl.

Poulton, T.P. and Tipper, H.W. 1991. Aalenian ammonites and strata of western Canada. Geological Survey of Canada, Bulletin 411, 71 p., 7 pl.

Price, D. 1984. Computer-based storage and retrieval of paleontological data at the Sedgwick Museum, Cambridge, England. Paleontology **27**: 393-406.

Quinzio Sinn, L. A. 1987. Stratigraphische Untersuchungen im Unterjura des Südteils der Provinz Antofagasta in Nord-Chile. Berliner geowissenschaftliche Abhandlungen, Reihe A, **87**, 105 p.

Raup, D. M. 1966. Geometric analysis of shell coiling: general problems. Journal of Paleontology **40(5)**: 1178-1190.

-- 1967. Geometric analysis of shell coiling: coiling in Ammonoids. Journal of Paleontology **41(1)**: 43-65.

Raup, D. M. and Crick, R. E. 1981. Evolution of single characters in the Jurassic ammonite Kosmoceras. Paleobiology **7(2)**: 200-215.

-- 1982. *Kosmoceras*: evolutionary jumps and sedimentary breaks. Paleobiology **8(2)**: 90-100.

Raup, D. M. and Sepkoski, J. J. 1986. Periodic extinction of families and genera. Science **231**: 833-836.

Repin, Y.S. 1968. A new ammonoid genus from the Toarcian of the north-eastern U.S.S.R.. Palaeontological Journal, **3**: 139-142.

-- 1974. Biostratigraphy of the boreal Mesozoic, U.S.S.R. Academy of Sciences, Siberian Branch, **136**: 51-66, 165-174.

Rich, D. 1989. Image storage has geologic applications. Geotimes **34(9)**: 10-11.

Riedel, W. R. and Tway, L. E. 1990. Artificial intelligence applications in paleontology and stratigraphy. In: Agterberg, F. P. and Bonham-Carter, G. F. [ed.]: Statistical applications in the earth sciences. Geological Survey of Canada, Paper **89-9**: 383-388.

Rock, N. M. S. 1988. Numerical geology : a source guide, glossary, and selective bibliography to geological uses of computers and statistics. Berlin; New York : Springer-Verlag.

Rogers, M. J., Donovan, D. T. and Rogers, M. H. 1990. A deductive enquiry system for a palaeotological database of museum material. Palaeontology **33(3)**: 613-622.

Sanborn, A.F. 1960. Geology and paleontology of the southwest quarter of the Big Bend quadrangle. California Division of Mines, Special Report **63**: 3-26.

Saunders, W. B. 1983. Natural rates of growth and longevity of *Nautilus belauensis*. Paleobiology **9(3)**: 280-288.

Saunders, W. B. and Shapiro, E. A. 1986. Calculation and simulation of ammonoid hydrostatics. Paleobiology **12(1)**: 64-79.

Saunders, W. B. and Swan, A. R. H. 1984. Morphology and morphologic diversity of mid-Carboniferous (Namurian) ammonoids in time and space. Paleobiology **10(2)**: 195-228.

Savazzi, E. 1990. Theoretical morphology of shells aided by microcomputers. In: Hanley, J. T., Merriam, D. F. eds. Microcomputer applications in geology II: 229-240.

Schlegelmilch, R.S. 1976. Die Ammoniten des süddeutschen Lias. Gustav, Fischer, Verlag, Stuttgart, 212 p., 48 pl.

Schlatter, R. 1991. Biostratigraphie und Ammonitenfauna des Ober-Lotharingium und Unter-Pliensbachium im Klettgau (Kanton Schaffhausen, Schweiz) und angrenzender Gebiete, Schweizerische Palaontologische Abhandlungen, **113**, 133 p., 21 pl.

Sepkoski, J. J. 1993. Ten years in the library: new data confirm paleontological patterns. Paleobiology **19(1)**: 43-51.

Sey, I.I. and Kalacheva, E.D. 1980. Biostratigraphy of the Lower and Middle Jurassic deposits of the Far East. Ministry of Geology, U.S.S.R., Geological Institute, Nedra, 187 p., 34 pl (in Russian).

Smith, P.L. 1976. Biostratigraphy of the Snowshoe Formation (Jurassic) in the Izee area Grant County, Oregon. M.Sc. thesis, Portland State University, 213 p., 7 pl.

-- 1980. Biostratigraphy and ammonoid fauna of the Lower Jurassic (Sinemurian, Pliensbachian and lowest Toarcian) of eastern Oregon and western Nevada. Ph.D. hesis, McMaster University.

-- 1983: The Pliensbachian ammonite *Dayiceras dayiceroides* and Early Jurassic paleogeography. Canadian Journal of Earth Sciences **20:** 86-91.

-- 1986. The implications of data base management systems to paleontology: a discussion of Jurassic ammonoid data. Journal of Paleontology, **60**: 327-340.

Smith, P.L., Thomson, R.C. and Tipper, H.W. 1984. Lower and Middle Jurassic sediments and volcanics of the Spatsizi map area, British Columbia. *in* Current Research, Part A, Geological Survey of Canada, Paper **84-1A**: 117-120.

Smith, P.L. and Tipper H.W. 1986. Plate tectonics and paleobiogeography: Early Jurassic (Pliensbachian) endemism and diversity. Palaios **1**: 399--412.

Smith, P.L. and H.W. Tipper 1988. Biochronology, stratigraphy and tectonic setting of the Pliensbachian of Canada and the United States. Second International Symposium on Jurassic Stratigraphy, Lisbon, **1**:119-138.

Smith, P.L., Tipper, H.W. Taylor, D.G. and Guex, J. 1988. An ammonite zonation for the Lower Jurassic of Canada and the United States: the Pliensbachian. Canadian Journal of Earth Sciences, **25**:1503-1523.

Smith, P.L. and Westermann, G.E.G. 1990. Paleobiogeography of the Ancient Pacific. Science, **249**, p. 680.

Spicer, R. A. Computerized palaeobotanical data bases: the way forward? In: Spicer, R. A. and Thomas, B. A. eds. Systematics and Taxonomic Approaches in Palaeobotany.

Stankevich, E.S. 1964, Ammonites in the Jurassic sandy-clay deposits of the northwestern Caucasus: Moskow-Leningrad, Nauka, 99 p. (in Russian).

Stanley, G.D. Jr. and Yancey, T.E. 1990. Paleobiogeography of the Ancient Pacific. Science, **249**, p. 680-681.

Stanley, G.D. Jr. and Yang, X. N. 1987. Approximate evolutionary stasis for bivalve morphology over millions of years: a multivariate, multilineage study. Paleobiology **13(2)**: 113-139.

Stansfield, S. A. 1986. ANGY: a rule-based expert system for automatic segmentation of coronary vessels from digital subtracted angiograms. IEEE Trans. Pattern Anal. Machine Intell. **8**: 188-199.

Sutherland Brown, A. 1968. Geology of the Queen Charlotte Islands, British Columbia. British Columbia Department of Mines and Petroleum Resources, Bulletin **54**.

Swan, A. R. H. and Saunders, W. B. 1987. Function and shape in late Paleozoic (mid-Carboniferous) ammonoids. Paleobiology **13(3)**: 297-311.

Taylor, D.G. 1988a. *Paradiscamphiceras*, a new Lower Liassic ammonite genus. Bull. Soc. vaud. Sc. nat. **79**:117-122.

-- 1988b. Middle Jurassic (late Aalenian and early Bajocian) ammonite biochronology of the Snowshoe Formation, Oregon. Oregon Geology **50**:123-138.

-- 1990. Two species of *Paracaloceras* from the Canadense Zone (Hettangian-Sinemurian) in Nevada (U.S.A.). Bulletin de Geologie Lausanne, **309**: 211-219.

Taylor, D.G., Callomon, J.H., Hall, R., Smith, P.L., Tipper, H.W. and Westermann, G.E.G. 1984. Jurassic ammonite Biogeography of western North America: the tectonic implications. Geological Society of Canada Special Paper **27**: 121-141.

Thomson, R.C. 1985. Lower to Middle Jurassic (Pliensbachian to Bajocian) stratigraphy and Pliensbachian ammonite fauna of the northern Spatsizi area, north central British Columbia. M.Sc. thesis, University of British Columbia.

Thomson, R.C., Smith, P.L. and Tipper, H.W. 1986. Lower to Middle Jurassic (Pliensbachian to Bajocian) stratigraphy of the northern Spatsizi area, north-central British Columbia. Canadian Journal of Earth Sciences, **23**, no. 12, p. 1963-1973.

Thomson, R.C and P.L. Smith 1992. Pliensbachian (Lower Jurassic) biostratigraphy and ammonite fauna of the Spatsizi area, north-central British Columbia. Geological Survey of Canada, Bulletin **437**.

Tillman, N. 1917. Die Fauna des unteren und mittleren Lias in Nord und Mittel-Peru. N. Jahrb. Min. **41**: 628-712, pl. 21-26.

Tipper, H.W. and Richards, T.A. 1976. Jurassic stratigraphy and history of north-central British Columbia. Geological Survey of Canada, Bulletin **270**, 73 p.

Tipper, H.W., Smith, P.L., Cameron, B.E.B., Carter, E.S., Jakobs, G.K. and Johns, M.J. 1991. Biostratigraphy of the Lower Jurassic formations of the Queen Charlotte Islands, British Columbia. In: Woodsworth, G.J. [ed.]: Evolution and Hydrocarbon potential of the Queen Charlotte basin, British Columbia. Geological Survey of Canada, Paper **90-10**: 203-235.

Torre, V. and Poggio, T. A. 1986. On edge detection. IEEE Trans. on pattern analysis and machine intelligence **2**: 147-162.

Tozer, E.T. 1982. Late Triassic (upper Norian) and earliest Jurassic (Hettangian) rocks and ammonoid faunas, Halfway River and Pine Pass map areas, British Columbia. Current Research, Part A, Geological Survey of Canada, Paper **82-1A**: 385-391.

Wang, Y. and Smith, P. L. 1986. Sinemurian (Early Jurassic) ammonite fauna from the Guangdong region of southern China. Journal of Paleontology, **60**: 1075-1085.

Ward, P. D. 1981. Shell sculpture as a defensive adaptation in ammonoids. Paleobiology **7(1)**: 96-100.

-- 1987. The natural history of Nautilus. Allen and Unwin Inc.

Ward, P. D. and Signor, P. W. 1983. Evolutionary tempo in Jurassic and Cretaceous ammonites. Paleobiology **9(2)**: 183-198.

Warren, P.S. 1931. A Lower Jurassic fauna from Fernie, B.C. Transactions of the Royal Society of Canada, series 3, **25**: 105-111, pl. 1.

Westermann, G. E. G. 1966. Covariation and taxonomy of the Jurassic ammonite *Sonninia adicra* (Waagen). N. Jb. Geol. Paläont. Abh. **124(3)**: 289-312

-- 1971. Form, structure and function of shell and siphuncle in coiled Mesozoic Ammonoids. Royal Ontario Museum Publications in Life Sciences.

-- 1982. The connecting rings of *Nautilus* and Mesozoic ammonoids: implications for ammonoid bathymetry. Lethaia **15**: 373-383.

-- 1993. Global bio-events in mid-Jurassic ammonites controlled by seaways. In House, M. R. ed. The Ammonoidea -- Environment, Ecology, and Evolutionary Change.

White, C.A. 1889. On vertebrate fossils from the Pacific Coast. United States Geological Survey Bulletin, pp. 71-98, pl. 13.

Whiteaves, J.F. 1876. On some invertebrates from the coal-bearing rocks of the Queen Charlotte Islands. Geological Survey of Canada, Mesozoic Fossils, **1**: 1-92 .

Whiteaves, J.F. 1884. Mesozoic Fossils: Part III-On the fossils of the coal-bearing deposits of the Queen Charlotte Islands collected by Dr. G.M. Dawson in 1878. Geological and Natural History Survey of Canada-Mesozoic Fossils, **1**: 1-261, 14 pl.

Wiedenmayer, F. (1977): Die Ammoniten des Besazio-Kalks (Pliensbachian, Sudtessin). Schweizerische palaontologische Abhandlung, **98**: 1-169, 19 pl.

-- 1980. Die Ammoniten der mediterranen Provinz im Pliensbachian und unteren Toarcian aufgrund neuer Untersuchungen im Generoso-Becken (Lombardischen Alpen): Denkshriften Schweizerischen Naturforschenden Gessellschaft, **93**, 34 pl.

Willard, B. 1963. Ontogeny of the Jurassic ammonite Arietites from Peru. Pennsylvania Academy of Science, **37**: 211-215

Younker, J. L. and Ehrlich, R. 1977. Fourier biometrics: harmonic amplitude as multivariate shape descriptors. Syst. Zoology **26**: 336-346.

Various morphological parameter value combinations and their abundance in the database Ammon

The number in table represents the number of occurrences for each pair , light shaded cells show areas of few occurrence, dark shaded cells show areas of particular interest.

1. Expansion Rate--W:

(1) Degree of involution -- Volution0 (V)

W/V	<0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	>0.9
<1.5	24	14	4	1	1	2	1	0	0	0
1.5-1.7	139	114	84	37	18	10	2	0	1	0
1.7-1.9	155	207	207	160	85	49	25	10	2	0
1.9-2.1	96	139	185	180	134	113	57	25	12	7
2.1-2.3	45	64	87	84	118	90	78	35	26	11
2.3-2.5	20	19	40	41	56	65	72	53	27	6
2.5-2.7	6	11	19	30	34	42	22	27	19	8
2.7-3.0	2	4	8	13	14	18	17	23	14	5
3.0-3.3	0	0	1	3	4	1	5	8	11	9
>3.3	0	0	1	2	2	2	3	7	6	1

(2) WWWH

W\WWWH	<0.6	0.6-0.8	0.8-1.0	1.0-1.2	>1.2
<1.5	0	1	5	18	10
1.5-1.7	5	34	70	67	101
1.7-1.9	23	163	167	123	139
1.9-2.1	64	207	163	86	79
2.1-2.3	75	139	81	32	36
2.3-2.5	66	82	37	16	7
2.5-2.7	34	54	7	9	5
2.7-3.0	23	22	5	5	4
3.0-3.3	5	6	4	4	4
>3.3	5	4	7	0	0

(3) Fineness Ratio--WWD

(2) Fineness Ratio

V\WWD	<0.2	0.2-0.25	0.25-0.3	0.3-0.35	0.35-0.4	0.4-0.45	0.45-0.5	0.5-0.55	>0.55
<0.1	15	60	76	48	26	16	5	3	2
0.1-0.2	13	54	98	67	37	21	6	7	3
0.2-0.3	8	87	82	81	49	28	4	4	8
0.3-0.4	8	60	80	67	34	17	11	9	10
0.4-0.5	3	38	78	51	21	7	8	5	8
0.5-0.6	8	21	67	47	18	4	4	1	12
0.6-0.7	4	33	51	35	12	4	4	5	6
0.7-0.8	3	19	28	18	8	2	1	6	14
0.8-0.9	3	10	23	12	6	1	1	3	5
>0.9	3	20	2	2	0	1	0	1	1

(3) Primary Rib Density

V\PRHW	<10	10-20	20-30	30-40	40-50	50-60	>60
<0.1	10	170	185	59	12	13	14
0.1-0.2	14	261	185	54	17	5	7
0.2-0.3	20	298	201	59	18	6	0
0.3-0.4	27	254	127	60	19	3	3
0.4-0.5	23	221	106	55	13	8	4
0.5-0.6	19	194	74	47	15	6	4
0.6-0.7	14	139	57	24	8	5	4
0.7-0.8	9	87	39	11	6	2	4
0.8-0.9	5	43	23	6	1	5	7
>0.9	0	8	0	1	0	2	1

(4) Rib Width

V\RIBWIDTH	<0.1	0.1-0.15	0.15-0.2	0.2-0.25	>0.25
<0.1	38	112	132	108	79
0.1-0.2	39	127	170	136	72
0.2-0.3	42	192	171	122	78
0.3-0.4	48	160	150	89	45
0.4-0.5	75	143	125	61	26
0.5-0.6	83	131	82	45	19
0.6-0.7	73	95	62	15	5
0.7-0.8	51	69	21	14	3
0.8-0.9	51	24	11	3	0
>0.9	10	2	0	0	0

(5) Primary Rib Form

V\PFORM	straight	concave	convex	projected	sinuous	falcoid	falcate
<0.1	335	36	0	8	72	11	4
0.1-0.2	378	17	1	15	87	33	8
0.2-0.3	315	19	2	27	160	72	11
0.3-0.4	190	32	1	29	162	68	11
0.4-0.5	98	22	0	30	148	113	14
0.5-0.6	59	13	2	36	139	103	5
0.6-0.7	36	9	0	17	94	87	6
0.7-0.8	34	1	0	7	48	60	1
0.8-0.9	18	0	0	8	23	36	1
>0.9	4	0	0	1	4	4	0

(6) Primary Rib Trend

V\PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
<0.1	14	38	289	84	37
0.1-0.2	19	50	354	80	23
0.2-0.3	16	53	416	79	28
0.3-0.4	10	33	346	59	27
0.4-0.5	4	19	309	56	18
0.5-0.6	1	9	257	61	12
0.6-0.7	2	5	177	43	5
0.7-0.8	0	4	117	20	1
0.8-0.9	0	0	57	15	1
>0.9	0	0	10	0	0

(7) Primary Rib Profile

V\PPROF	non	weak	round +weak	round	angular	round +strong	strong	angular +strong
<0.1	11	8	2	89	62	38	166	7
0.1-0.2	14	12	10	168	59	50	157	7
0.2-0.3	10	24	21	184	80	72	125	3
0.3-0.4	37	36	28	189	60	38	68	3
0.4-0.5	26	31	25	194	31	28	47	1
0.5-0.6	18	39	38	180	15	8	14	0
0.6-0.7	21	36	36	126	3	4	6	0
0.7-0.8	26	24	29	57	3	4	7	0
0.8-0.9	25	22	17	30	0	2	0	0
>0.9	29	9	1	4	0	0	0	0

(8) Rib Furcation

V\FURC	non	bundled	bi	intercalatory	tri	multi	loop
<0.1	376	3	80	1	15	2	6
0.1-0.2	423	12	99	3	18	4	2
0.2-0.3	459	31	97	5	15	8	2
0.3-0.4	431	18	66	2	16	7	0
0.4-0.5	356	19	68	5	5	6	0
0.5-0.6	310	22	42	3	6	1	0
0.6-0.7	224	19	29	3	3	1	1
0.7-0.8	152	3	16	3	6	0	5
0.8-0.9	102	0	6	4	0	1	5
>0.9	44	0	0	0	0	0	1

(9) Tuberculation

V\TUBERC	non	uni	bi	bullate	clavate
<0.1	316	145	23	3	0
0.1-0.2	387	154	25	6	0
0.2-0.3	466	139	17	13	1
0.3-0.4	458	68	17	7	1
0.4-0.5	400	46	10	10	0
0.5-0.6	346	34	11	1	0
0.6-0.7	256	20	4	2	0
0.7-0.8	164	6	17	1	0
0.8-0.9	110	0	8	0	0
>0.9	45	0	2	0	0

(10) Furcation Position

V\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.1	376	8	6	30	57
0.1-0.2	423	14	16	46	58
0.2-0.3	459	36	31	44	42
0.3-0.4	431	28	36	33	11
0.4-0.5	355	28	51	15	4
0.5-0.6	310	20	45	4	3
0.6-0.7	224	16	30	7	1
0.7-0.8	151	4	16	3	4
0.8-0.9	102	0	6	3	1
>0.9	44	0	0	0	0

(11) Unituberculate Position

V\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.1	338	4	3	34	108
0.1-0.2	412	10	9	30	111
0.2-0.3	482	24	15	36	79
0.3-0.4	474	20	12	19	26
0.4-0.5	409	21	11	17	8
0.5-0.6	357	21	8	2	4
0.6-0.7	260	14	7	1	0
0.7-0.8	181	2	3	0	2
0.8-0.9	118	0	0	0	0
>0.9	47	0	0	0	0

(12) Venter

V\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
<0.1	184	131	85	9	5
0.1-0.2	204	174	93	10	1
0.2-0.3	176	239	117	12	6
0.3-0.4	128	248	102	17	0
0.4-0.5	84	239	77	9	2
0.5-0.6	57	247	28	2	1
0.6-0.7	38	187	13	6	2
0.7-0.8	39	112	2	3	0
0.8-0.9	26	55	2	1	3
>0.9	9	4	0	0	0

(13) Whorl Shape

V\WHORL SHAPE	coro	trape	tri	lanceo	quadrate	round	rect	ellips	ogival	oval
<0.1	10	0	1	0	42	76	57	112	25	16
0.1-0.2	14	2	0	1	49	71	95	132	36	18
0.2-0.3	10	1	1	0	40	66	154	142	48	28
0.3-0.4	5	5	4	0	29	52	128	107	87	30
0.4-0.5	0	5	7	2	12	26	99	74	90	30
0.5-0.6	2	5	11	7	5	10	71	57	87	42
0.6-0.7	0	1	15	10	2	6	35	36	76	44
0.7-0.8	0	3	33	23	2	8	5	16	29	34
0.8-0.9	0	2	33	8	1	2	1	8	18	22
>0.9	0	0	30	1	1	1	0	0	6	0

(14) Shell Diameter

V\D	<3	3-6	6-9	9-12	12-15	>15
<0.1	63	211	114	61	14	24
0.1-0.2	101	252	112	58	18	31
0.2-0.3	146	270	136	39	19	26
0.3-0.4	160	243	80	38	7	23
0.4-0.5	143	201	70	27	14	11
0.5-0.6	111	171	64	24	6	16
0.6-0.7	56	135	48	18	11	14
0.7-0.8	46	79	39	12	5	7
0.8-0.9	12	43	27	18	6	12
>0.9	2	12	17	6	7	3

3. WWWH

(1) Fineness Ratio

WWWH\WWD	<0.2	0.2-0.25	0.25-0.3	0.3-0.35	0.35-0.4	0.4-0.45	0.45-0.5	0.5-0.55	>0.55
<0.6	36	130	87	22	0	1	1	0	0
0.6-0.8	29	189	286	127	31	6	1	0	0
0.8-1.0	7	84	173	161	61	11	5	7	3
1.0-1.2	2	31	67	118	71	26	7	8	8
>1.2	1	5	21	53	81	79	38	39	62

(2) Primary Rib Density

WWWH\PRHW	<10	10-20	20-30	30-40	40-50	50-60	>60
<0.6	4	112	67	51	11	9	10
0.6-0.8	23	348	206	85	25	12	8
0.8-1.0	23	327	156	34	7	5	5
1.0-1.2	31	217	96	28	10	2	8
>1.2	53	177	88	38	17	4	8

(3) Rib Width

WWWH\RIBWIDTH	<0.1	0.1-0.15	0.15-0.2	0.2-0.25	>0.25
<0.6	81	91	40	12	2
0.6-0.8	96	223	201	100	26
0.8-1.0	45	111	155	140	62

1.0-1.2	23	63	97	82	79
>1.2	21	74	77	92	96

(4) Primary Rib Form

WWWH\PFORM	straight	concave	convex	projected	sinuous	falcoid	falcate
<0.6	28	4	0	19	82	87	5
0.6-0.8	164	16	2	56	257	132	20
0.8-1.0	288	38	1	24	121	29	8
1.0-1.2	257	26	1	8	37	6	7
>1.2	319	13	2	6	16	2	0

(5) Primary Rib Trend

WWWH \PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
<0.6	2	11	163	35	2
0.6-0.8	11	43	468	99	9
0.8-1.0	16	49	343	62	34
1.0-1.2	11	27	230	49	20
>1.2	7	11	217	72	49

(6) Primary Rib Profile

WWWH\PPROF	non	weak	round +weak	round	angular	round +strong	strong	angular +strong
<0.6	43	43	32	108	4	5	3	0
0.6-0.8	37	57	68	306	31	52	49	0
0.8-1.0	17	21	16	154	73	47	110	6
1.0-1.2	6	11	5	76	46	34	115	2
>1.2	17	2	0	67	33	22	176	3

(7) Rib Furcation

WWWH\FURC	non	bundled	bi	intercalatory	tri	multi	loop
<0.6	280	8	13	3	10	3	0
0.6-0.8	595	55	84	6	5	4	2
0.8-1.0	470	12	84	3	4	3	7
1.0-1.2	307	10	65	3	14	2	4
>1.2	214	3	131	0	41	12	4

(8) Tuberculation

WWWH\TUBERC	non	uni	bi	bullate	clavate
<0.6	354	21	0	1	0
0.6-0.8	650	111	18	11	0
0.8-1.0	431	127	45	8	0
1.0-1.2	275	106	34	8	0
>1.2	197	193	30	2	3

(9) Furcation Position

WWWH\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.6	279	6	18	4	7
0.6-0.8	595	56	70	11	13
0.8-1.0	469	19	26	30	30
1.0-1.2	307	17	12	20	43
>1.2	214	4	14	76	94

(10) Unituberculate Position

WWWH\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.6	354	7	5	3	7
0.6-0.8	668	48	16	8	50
0.8-1.0	476	17	9	23	86
1.0-1.2	309	12	7	17	78
>1.2	225	3	11	81	105

(11) Venter

WWWH\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
<0.6	46	261	13	8	0
0.6-0.8	159	479	114	16	2
0.8-1.0	235	225	117	17	6
1.0-1.2	214	108	78	12	7
>1.2	332	46	34	3	7

(12) Whorl Shape

WWWH\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogival	oval
<0.6	0	0	77	10	0	0	69	42	116	49
0.6-0.8	0	4	14	12	0	0	220	226	183	92
0.8-1.0	2	9	1	1	140	167	40	74	53	45
1.0-1.2	7	3	0	0	82	169	35	59	15	22
>1.2	51	6	2	0	0	83	91	164	1	3

(13) Shell Diameter

WWWH\D	<3	3-6	6-9	9-12	12-15	>15
<0.6	39	121	94	51	26	44
0.6-0.8	128	311	170	91	39	51
0.8-1.0	125	252	128	52	17	37
1.0-1.2	93	178	80	29	12	31
>1.2	139	188	67	11	6	14

4. Fineness Ratio

(1) Primary Rib Density

WWD\PRHW	<10	10-20	20-30	30-40	40-50	50-60	>60
<0.2	0	14	29	13	3	2	0
0.2-0.25	4	164	115	55	11	11	11
0.25-0.3	24	316	161	60	17	9	10
0.3-0.35	22	265	101	34	16	0	6
0.35-0.4	18	136	47	15	4	1	3
0.4-0.45	19	50	28	14	3	0	0
0.45-0.5	8	23	12	3	2	0	0
0.5-0.55	8	31	7	1	2	0	0
>0.55	20	36	7	0	0	1	0

(2) Rib Width

WWD\RIBWIDTH	<0.1	0.1-0.15	0.15-0.2	0.2-0.25	>0.25
<0.2	11	21	21	6	2
0.2-0.25	64	122	88	65	34
0.25-0.3	88	165	183	110	49
0.3-0.35	42	120	129	103	55
0.35-0.4	17	47	60	56	46
0.4-0.45	8	23	31	23	28
0.45-0.5	4	10	10	12	12
0.5-0.55	8	11	10	9	11
>0.55	11	10	8	19	16

(3) Primary Rib Form

WWD\PFORM	straight	concave	convex	projected	sinuous	falcoid	falcate
<0.2	27	4	0	5	15	7	1
0.2-0.25	156	15	0	22	109	63	9
0.25-0.3	222	19	3	44	200	90	15
0.3-0.35	233	26	0	22	110	47	11
0.35-0.4	153	15	0	8	34	15	4
0.4-0.45	88	6	1	1	13	4	0
0.45-0.5	39	1	1	1	5	0	0
0.5-0.55	37	1	1	1	7	1	0
>0.55	49	2	0	5	2	0	0

(4) Primary Rib Trend

WWD\PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
<0.2	0	10	33	15	0
0.2-0.25	10	38	257	48	12
0.25-0.3	13	34	419	89	20
0.3-0.35	9	29	311	68	24
0.35-0.4	6	12	156	26	26
0.4-0.45	3	4	70	23	11
0.45-0.5	0	4	30	11	3
0.5-0.55	0	0	33	8	5
>0.55	1	1	38	9	9

(5) Primary Rib Profile

WWD\ PPROF	non	weak	round +weak	round	angular	round+strong	strong	angular +strong
<0.2	10	6	2	24	5	2	4	0
0.2-0.25	37	43	32	154	20	27	44	2
0.25-0.3	20	41	45	244	47	49	89	2
0.3-0.35	22	24	23	143	56	39	98	2
0.35-0.4	12	9	10	49	31	16	71	0
0.4-0.45	7	2	2	23	10	5	53	0
0.45-0.5	2	4	1	8	1	2	26	1
0.5-0.55	2	1	0	21	3	2	19	0
>0.55	9	2	0	11	1	3	34	1

(6) Furcation

WWD\FURC	non	bundled	bi	intercalatory	tri	multi	loop
<0.2	66	1	3	0	3	0	0
0.2-0.25	338	27	41	2	6	4	1
0.25-0.3	473	33	95	2	12	5	3
0.3-0.35	378	12	72	3	10	1	0
0.35-0.4	165	8	54	4	9	0	1
0.4-0.45	66	3	39	0	11	1	1
0.45-0.5	22	2	21	1	4	2	0
0.5-0.55	18	0	21	0	5	3	4
>0.55	28	1	18	0	12	8	6

(7) Tuberculation

WWD\TUBERC	non	uni	bi	bullate	clavate
<0.2	71	10	0	0	0
0.2-0.25	344	83	11	6	0
0.25-0.3	490	122	13	13	0
0.3-0.35	369	89	21	4	0
0.35-0.4	153	78	12	2	1
0.4-0.45	57	52	12	2	0
0.45-0.5	21	25	5	0	1
0.5-0.55	22	20	12	0	0
>0.55	20	29	23	0	1

(8) Furcation Position

WWD\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.2	66	1	2	0	4
0.2-0.25	338	27	16	15	20
0.25-0.3	473	30	58	22	38
0.3-0.35	377	18	27	19	34
0.35-0.4	164	9	16	17	30
0.4-0.45	66	6	8	14	26
0.45-0.5	22	2	2	14	11
0.5-0.55	18	1	6	13	9
>0.55	28	1	3	23	12

(9) Unituberculate Position

WWD\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.2	71	1	0	0	9
0.2-0.25	355	24	5	13	47
0.25-0.3	503	29	15	14	77
0.3-0.35	389	12	8	15	59
0.35-0.4	165	7	6	21	47
0.4-0.45	69	6	5	13	30
0.45-0.5	25	1	1	14	11
0.5-0.55	34	0	2	11	7
>0.55	43	0	5	18	7

(10) Venter

WWD\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
<0.2	15	46	14	0	0
0.2-0.25	114	238	53	7	2
0.25-0.3	182	337	88	7	2
0.3-0.35	172	202	75	10	6
0.35-0.4	132	64	37	6	3
0.4-0.45	89	14	15	2	2
0.45-0.5	40	4	3	1	4
0.5-0.55	43	4	0	3	2
>0.55	62	8	1	1	1

(11) Whorl Shape

WWD\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogi-val	oval
<0.2	0	0	13	0	0	3	14	27	18	3
0.2-0.25	1	0	48	4	17	31	116	104	71	30
0.25-0.3	4	4	19	8	27	52	129	145	166	53
0.3-0.35	3	5	10	7	56	74	86	96	58	43
0.35-0.4	14	1	0	2	44	58	23	51	10	22
0.4-0.45	12	2	0	0	23	27	11	27	3	9
0.45-0.5	6	0	0	0	6	5	4	25	0	5
0.5-0.55	4	1	1	0	4	6	3	28	0	2
>0.55	9	7	0	0	4	8	2	40	0	1

(12) Shell Diameter

WWD\D	<3	3-6	6-9	9-12	12-15	>15
<0.2	1	22	15	15	11	17
0.2-0.25	40	140	119	73	29	43
0.25-0.3	100	295	144	57	16	26
0.3-0.35	121	219	91	23	11	18
0.35-0.4	83	112	34	6	1	10
0.4-0.45	41	61	12	2	4	3
0.45-0.5	20	23	6	3	0	0
0.5-0.55	26	18	6	0	1	3
>0.55	39	19	9	3	3	0

5. Primary Rib Density

(1) Rib Width

PRHW\RIBWIDTH	<0.1	0.1-0.15	0.15-0.2	0.2-0.25	>0.25
<10	3	16	29	29	78
10-20	108	411	571	502	307
20-30	105	475	397	167	32
30-40	148	209	69	10	0
40-50	76	54	3	2	0
50-60	54	8	1	0	0
>60	56	3	0	0	0

(2) Primary Rib Form

PRHW\PFORM	straight	concave	convex	projected	sinuous	falcoid	falcate
<10	114	2	1	8	30	1	0
10-20	947	105	5	121	495	172	33
20-30	572	59	0	41	300	186	12
30-40	167	11	0	13	92	144	7
40-50	44	5	0	0	42	40	4
50-60	17	0	0	0	15	29	1
>60	20	0	0	0	14	22	3

(3) Primary Rib Trend

PRHW\PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
<10	5	10	112	17	6
10-20	52	147	1302	212	83
20-30	28	71	787	215	37
30-40	1	18	302	80	22
40-50	0	5	89	27	8
50-60	0	1	39	9	10
>60	0	0	29	16	14

(4) Primary Rib Profile

PRHW\PPROF	weak	round+weak	round	angular	round+strong	strong	angular+strong
<10	9	7	45	10	14	48	2
10-20	101	90	673	175	152	363	9
20-30	61	69	373	151	78	228	7
30-40	28	32	148	28	19	87	4
40-50	9	7	49	12	4	18	4
50-60	5	6	19	7	1	11	1
>60	2	0	24	3	0	8	1

(5) Rib Furcation

PRHW\FURC	non	bundled	bi	intercalatory	tri	multi	loop
<10	120	17	18	0	8	6	0
10-20	1528	99	286	18	47	19	13
20-30	1002	5	194	9	23	5	7
30-40	373	1	70	2	10	2	2
40-50	115	3	20	0	5	0	0
50-60	57	1	6	1	3	0	0
>60	60	1	3	0	0	0	2

(6) Tuberculation

PRHW\TUBERC	non	uni	bi	bullate	clavate
<10	76	62	27	2	2
10-20	1464	411	99	35	1
20-30	1003	204	28	10	0
30-40	384	74	1	1	0
40-50	120	22	0	1	0
50-60	54	12	2	0	0
>60	52	11	3	0	0

(7) Furcation Position

PRHW\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<10	120	16	19	7	7
10-20	1527	128	181	80	71
20-30	1002	6	36	89	99
30-40	372	3	9	31	42
40-50	115	3	2	8	15
50-60	57	1	1	0	8
>60	60	1	0	2	2

(8) Unituberculate Position

PRHW\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<10	102	17	15	17	18
10-20	1560	99	47	77	227
20-30	1031	3	5	49	157
30-40	385	1	4	14	56
40-50	120	3	0	6	14
50-60	56	0	0	6	6
>60	55	0	0	7	4

(9) Venter

PRHW\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
<10	65	72	11	2	2
10-20	524	888	256	32	11
20-30	296	484	239	44	9
30-40	115	203	81	7	1
40-50	40	54	29	1	0
50-60	28	20	9	0	0
>60	31	22	5	0	0

(10) Whorl Shape

PRHW\ WHORL_SHAPE	coro	trape	tri	lanceo	quad-rate	round	rect	ellips	ogival	oval
<10	6	11	1	0	24	21	13	40	19	9
10-20	27	8	36	35	132	212	230	394	279	148
20-30	16	2	24	9	49	116	225	272	126	81
30-40	6	0	11	0	16	25	138	80	55	25
40-50	1	0	3	0	4	9	53	20	13	6
50-60	1	0	8	0	2	5	15	11	6	0
>60	0	0	5	0	5	10	8	9	8	5

(11) Shell Diameter

PRHW\D	<3	3-6	6-9	9-12	12-15	>15
<10	79	66	19	4	0	1
10-20	540	893	301	129	45	90
20-30	220	584	234	109	41	55
30-40	67	183	133	42	18	17
40-50	10	46	58	15	6	7
50-60	2	17	29	14	3	3
>60	1	6	22	16	8	13

6. Rib Width

(1) Primary Rib Form

RIBWIDTH \PFORM	straight	concave	convex	projected	sinuous	falcoid	falcate
<0.1	148	16	0	30	141	188	11
0.1-0.15	459	45	0	50	382	222	11
0.15-0.2	525	57	1	50	286	134	18
0.2-0.25	433	46	4	37	131	39	19
>0.25	320	17	1	16	48	13	1

(2) Primary Rib Trend

RIBWIDTH \PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
<0.1	4	13	347	102	30
0.1-0.15	20	73	791	195	47
0.15-0.2	18	63	760	146	55
0.2-0.25	33	63	469	89	32
>0.25	11	40	296	47	17

(3) Primary Rib Profile

RIBWIDTH\PPROF	weak	round +weak	round	angular	round+ strong	strong	angular+ strong
<0.1	74	49	185	37	6	56	9
0.1-0.15	81	93	392	159	39	176	8
0.15-0.2	39	52	397	119	74	197	5
0.2-0.25	16	13	234	49	77	190	3
>0.25	3	4	124	20	73	151	3

(4) Rib Furcation

RIBWIDTH\FURC	non	bundled	bi	intercalatory	tri	multi	loop
<0.1	393	21	95	5	18	5	15
0.1-0.15	837	60	217	16	39	8	3
0.15-0.2	844	29	171	5	15	7	3
0.2-0.25	593	11	82	4	16	5	2
>0.25	370	6	31	0	8	6	1

(5) Tuberculation

RIBWIDTH\TUBERC	non	uni	bi	bullate	clavate
<0.1	454	65	29	4	0
0.1-0.15	924	216	21	19	0
0.15-0.2	824	207	30	12	1
0.2-0.25	507	167	30	9	0
>0.25	261	111	45	3	2

(6) Furcation Position

RIBWIDTH\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.1	392	23	53	39	28
0.1-0.15	837	82	90	78	83
0.15-0.2	844	41	67	59	57
0.2-0.25	593	8	25	26	55
>0.25	369	4	12	14	21

(7) Unituberculate Position

RIBWIDTH\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
<0.1	483	13	10	20	26
0.1-0.15	943	60	21	50	106
0.15-0.2	853	34	19	40	128
0.2-0.25	537	7	12	27	130
>0.25	305	3	8	32	74

(8) Venter

RIBWIDTH\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
<0.1	138	243	60	19	2
0.1-0.15	243	542	193	26	5
0.15-0.2	246	462	188	19	7
0.2-0.25	225	271	99	9	6
>0.25	170	129	53	3	3

(9) Whorl Shape

RIBWIDTH\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogi-val	oval
<0.1	2	3	42	1	15	27	117	55	87	53
0.1-0.15	10	3	32	17	40	63	230	222	164	86
0.15-0.2	14	11	7	20	56	86	166	241	134	58
0.2-0.25	19	3	3	5	62	76	99	148	78	34
>0.25	8	0	0	1	49	94	33	92	21	22

(10) Shell Diameter

RIBWIDTH\D	<3	3-6	6-9	9-12	12-15	>15
<0.1	57	191	162	68	32	38
0.1-0.15	205	561	234	99	36	39
0.15-0.2	254	492	184	80	20	42
0.2-0.25	220	309	103	41	12	25
>0.25	149	181	61	16	6	9

7. Primary Rib Form

(1) Primary Rib Trend

PFORM\PTREND	rursiradiate	grursiradiate	rectiradiate	gprorsiradiate	prorsiradiate
straight	56	126	1229	336	101
concave	3	7	74	42	48
convex	0	2	2	0	1
projected	11	18	100	26	8
sinuous	11	54	784	89	21
falcoid	0	33	458	73	1
falcate	5	12	24	13	0

(2) Rib Profile

PFORM\ PPROF	weak	round+ weak	round	angular	round+ strong	strong	angular+ strong
straight	66	23	422	239	91	637	22
concave	7	3	33	52	4	27	2
convex	0	0	4	0	0	0	0
projected	7	2	92	14	11	15	1
sinuous	81	81	480	62	90	60	2
falcoid	46	100	284	19	54	12	0
falcate	2	3	17	0	19	12	1

(3) Rib Furcation

PFORM\FURC	non	bundled	bi	intercalatory	tri	multi	loop
straight	1362	12	408	9	78	21	9
concave	163	1	11	3	4	0	0
convex	5	0	1	0	0	0	0
projected	162	3	4	4	4	6	0
sinuous	744	101	120	9	7	5	5
falcoid	540	10	39	5	3	0	0
falcate	56	0	5	0	0	0	0

(4) Tuberculation

PFORM\TUBERC	non	uni	bi	bullate	clavate
straight	1160	590	125	22	2
concave	163	13	1	4	1
convex	5	1	0	0	0
projected	163	13	7	0	0
sinuous	812	146	13	20	0
falcoid	589	7	0	1	0
falcate	60	1	0	0	0

(5) Furcation Position

PFORM\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
straight	1361	34	74	194	226
concave	163	1	7	3	6
convex	5	0	1	0	0
projected	162	4	4	3	7
sinuous	743	105	112	15	5
falcoid	540	12	40	0	0
falcate	56	1	3	1	0

(6) Unituberculate Position

PFORM\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
straight	1285	24	31	152	407
concave	164	1	0	1	16
convex	5	0	0	0	1
projected	170	0	0	3	10
sinuous	821	86	37	14	33
falcoid	589	5	2	0	1
falcate	60	1	0	0	0

(7) Venter

PFORM\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
straight	824	445	200	55	8
concave	60	31	40	12	5
convex	4	2	0	0	0
projected	34	109	7	4	0
sinuous	77	627	167	5	8
falcoid	24	397	154	0	1
falcate	0	36	23	0	0

(8) Whorl Shape

PFORM\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogival	oval
straight	51	18	11	0	146	283	155	451	94	88
concave	0	0	2	0	10	40	4	44	4	11
convex	0	0	0	0	0	1	1	2	1	0
projected	1	0	16	0	5	4	15	20	58	10
sinuous	1	2	28	4	43	16	227	198	227	82
falcoid	0	0	26	39	9	2	211	45	95	61
falcate	0	0	0	1	4	0	31	2	6	1

(9) Shell Diameter

PFORM\D	<3	3-6	6-9	9-12	12-15	>15
straight	468	814	357	129	48	76
concave	26	66	38	23	4	21
convex	3	3	0	0	0	0
projected	41	74	33	20	10	5
sinuous	227	472	170	76	18	28
falcoid	112	270	130	48	21	15
falcate	6	22	15	7	3	8

8. Primary Rib Trend

(1) Rib Profile

PTREND\ PPROF	weak	round+ weak	round	angular	round+ strong	strong	angular+ strong
rursiradiate	3	1	22	15	19	16	1
grursiradiate	9	5	85	40	40	45	4
rectiradiate	144	173	981	244	182	497	19
gprorsiradiate	27	23	171	52	19	147	4
prorsiradiate	5	1	50	31	4	47	0

(2) Rib Furcation

PTREND\FURC	non	bundled	bi	intercalatory	tri	multi	loop
rursiradiate	78	1	4	0	2	1	0
grursiradiate	206	4	29	1	9	2	1
rectiradiate	2071	111	376	22	64	19	12
gprorsiradiate	409	9	136	5	12	8	2
prorsiradiate	146	0	29	0	5	0	1

(3) Tuberculation

PTREND\TUBERC	non	uni	bi	bullate	clavate
rursiradiate	64	16	4	2	0
grursiradiate	182	46	19	4	1
rectiradiate	2021	515	102	35	2
gprorsiradiate	423	134	18	6	0
prorsiradiate	132	49	0	0	0

(4) Furcation Position

PTREND\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
rursiradiate	78	3	2	1	2
grursiradiate	206	16	8	1	19
rectiradiate	2069	112	167	145	157
gprorsiradiate	409	19	43	56	50
prorsiradiate	146	1	8	12	13

(5) Unituberculate Position

PTREND\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
rursiradiate	68	2	2	2	12
grursiradiate	200	11	2	7	32
rectiradiate	2120	90	56	112	297
gprorsiradiate	441	12	9	30	89
prorsiradiate	132	0	0	17	32

(6) Venter

PTREND\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
rursiradiate	7	35	29	0	2
grursiradiate	22	117	80	0	4
rectiradiate	623	1235	396	61	17
gprorsiradiate	222	198	66	7	0
prorsiradiate	126	23	9	2	0

(7) Whorl Shape

PTREND\WHORL_SHAPE	coro	trape	tri	lanceo	quad-rate	round	rect	ellips	ogival	oval
rursiradiate	0	0	0	0	9	4	20	15	9	8
grursiradiate	0	2	2	0	24	15	55	34	34	11
rectiradiate	27	11	59	41	163	207	473	520	364	187
gprorsiradiate	16	5	11	0	18	76	85	119	58	34
prorsiradiate	10	0	1	0	3	37	4	65	6	8

(8) Shell Diameter

PTREND\D	<3	3-6	6-9	9-12	12-15	>15
rursiradiate	8	33	22	12	5	6
grursiradiate	29	114	56	28	10	13
rectiradiate	656	1179	479	187	57	109
gprorsiradiate	114	265	120	48	14	18
prorsiradiate	36	76	42	20	5	2

9. Primary Rib Profile

(1) Rib Furcation

PPROF\FURC	non	bundled	bi	intercalatory	tri	multi	loop
non	230	0	0	0	0	0	0
weak	215	8	23	4	1	0	1
round+weak	184	6	17	3	2	0	0
round	1081	79	133	5	21	10	6
angular	336	5	34	0	6	4	1
round+strong	231	5	29	3	1	0	0
strong	440	6	253	3	44	14	10
angular+strong	15	0	13	0	0	0	0

(2) Tuberculation

PPROF\TUBERC	non	uni	bi	bullate	clavate
non	225	3	1	1	0
weak	232	16	3	1	0
round+weak	195	7	7	3	0
round	1046	220	54	15	0
angular	312	62	3	9	0
round+strong	237	25	5	2	0
strong	386	321	55	7	1
angular+strong	21	6	0	0	1

(3) Furcation Position

PPROF\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
non	230	0	0	0	0
weak	215	8	23	2	1
round+weak	184	12	13	1	0
round	1080	86	95	30	34
angular	336	7	8	10	25
round+strong	231	12	12	5	6
strong	439	8	19	141	151
angular+strong	15	1	0	6	6

(4) Unituberculate Position

PPROF\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
non	226	0	0	3	1
weak	235	6	4	0	7
round+weak	202	9	1	0	0

round	1099	72	28	30	106
angular	315	1	3	10	57
round+strong	241	7	3	8	10
strong	439	7	13	100	211
angular+strong	21	0	1	3	3

(5) Venter

PPROF\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
non	85	50	5	3	0
weak	59	124	15	0	3
round+weak	16	161	24	0	2
round	235	787	174	6	4
angular	82	114	96	43	2
round+strong	29	136	77	0	6
strong	444	130	71	3	2
angular+strong	9	6	5	3	1

(6) Whorl Shape

PPROF\WHORL_SHAPE	coro	trape	tri	lanceo	quad-rate	round	rect	ellips	ogival	oval
non	1	2	46	7	0	28	13	33	28	10
weak	0	0	22	5	2	14	29	47	46	31
round+weak	0	6	11	2	1	2	41	46	53	32
round	4	6	27	29	49	68	251	259	265	96
angular	13	0	3	0	20	53	68	64	29	25
round+strong	0	1	0	5	34	15	81	59	15	14
strong	31	6	0	0	78	110	74	171	44	20
angular+strong	0	0	0	0	6	2	5	5	1	1

(7) Shell Diameter

PPROF\D	<3	3-6	6-9	9-12	12-15	>15
non	53	82	37	21	11	25
weak	64	110	35	20	11	12
round+weak	46	98	39	16	5	8
round	286	550	274	128	42	53
angular	76	194	62	24	7	21
round+strong	42	118	61	25	11	12
strong	171	355	160	46	14	16
angular+strong	2	13	5	3	1	4

10. Rib Furcation

(1) Tuberculation

FURC\TUBERC	non	uni	bi	bullate	clavate
non	3017	386	113	25	2
bundled	31	85	1	10	0
bi	326	243	14	14	0
intercalatory	26	1	3	0	0
tri	18	67	10	1	0
multi	7	18	7	0	0
loop	2	7	14	0	1

(2) Unituberculate Position

FURC\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
non	3130	11	2	67	333
bundled	32	68	27	0	0
bi	336	37	27	79	118
intercalatory	29	0	0	1	0
tri	28	6	10	21	31
multi	14	1	5	12	0
loop	16	0	0	1	7

(3) Venter

FURC\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
non	761	1542	542	72	22
bundled	3	77	44	0	0
bi	333	145	47	12	1
intercalatory	4	20	0	2	0
tri	64	23	0	2	0
multi	18	12	0	1	0
loop	17	2	3	0	1

(4) Whorl Shape

FURC\WHORL_SHAPE	coro	trape	tri	lanceo	quad-rate	round	rect	ellips	ogival	oval
non	16	15	128	51	154	358	592	652	423	222
bundled	0	0	5	0	11	2	31	27	42	2
bi	31	4	3	0	50	61	55	153	58	41
intercalatory	0	0	0	0	2	2	6	4	2	7
tri	3	2	0	0	9	6	9	27	16	11
multi	8	0	0	0	1	0	3	7	5	3
loop	0	2	0	0	6	4	4	2	0	0

(5) Shell Diameter

FURC\D	<3	3-6	6-9	9-12	12-15	>15
non	802	1501	665	285	106	174
bundled	8	46	32	19	7	15
bi	147	279	111	35	10	9
intercalatory	4	7	7	4	0	8
tri	13	39	28	8	3	5
multi	9	10	4	4	5	0
loop	0	13	4	0	3	4

11. Tuberculation

(1) Furcation Position

TUBERC\FURCPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
non	3017	57	163	110	61
uni	384	82	70	97	169
bi	113	5	6	8	14
bullate	25	14	9	2	0
clavate	2	0	0	0	0

(2) Venter

TUBERC\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
non	699	1647	557	91	19
uni	448	219	50	2	0
bi	112	31	7	0	1
bullate	1	10	34	0	2
clavate	0	0	1	0	2

(3) Whorl Shape

TUBERC\WHORL_SHAPE	coro	trape	tri	lanceo	quad-rate	round	rect	ellips	ogival	oval
non	6	6	135	54	106	355	607	687	476	270
uni	55	5	5	0	96	76	99	183	81	29
bi	2	15	0	0	32	21	16	32	3	10
bullate	0	0	1	0	4	8	9	14	3	0
clavate	0	0	0	0	3	0	0	0	0	0

(4) Shell Diameter

TUBERC\D	<3	3-6	6-9	9-12	12-15	>15
non	810	1528	665	273	111	174
uni	168	353	159	77	25	37
bi	26	54	50	18	6	14
bullate	5	15	5	11	4	10
clavate	2	1	0	0	0	0

12. Furcation Position

(1) Unituberculate Position

FURCPOS\UNITUBPOS	non	<0.25	0.25-0.5	0.5-0.75	>0.75
non	3130	11	2	67	331
<0.25	59	98	0	0	1
0.25-0.5	168	14	62	2	2
0.5-0.75	118	0	7	89	3
>0.75	75	0	0	23	146

(2) Venter

FURCPOS\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
non	761	1541	542	71	22
<0.25	3	89	55	1	0
0.25-0.5	24	150	36	13	1
0.5-0.75	176	13	1	2	0
>0.75	218	12	0	0	0

(3) Whorl Shape

FURCPOS\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogi- val	oval
non	16	15	128	51	154	358	591	652	422	222
<0.25	0	1	5	0	17	3	35	29	41	10
0.25-0.5	2	2	3	0	12	7	36	38	54	37
0.5-0.75	13	0	0	0	9	28	15	82	12	4
.0.75	27	3	0	0	34	31	13	68	16	10

(4) Shell Diameter

FURCPOS\D	<3	3-6	6-9	9-12	12-15	>15
non	801	1501	665	285	106	173
<0.25	10	52	43	29	7	16
0.25-0.5	46	118	43	15	10	13
0.5-0.75	71	92	34	11	4	3
>0.75	50	115	58	12	5	4

13. Unituberculate Position

(1) Venter

UNITUBPOS\VENTER	plain	carinate	carinate-sulcate	sulcate	tricarinate
non	811	1678	561	91	19
<0.25	1	84	34	1	1
0.25-0.5	14	33	19	0	0
0.5-0.75	141	20	5	0	0
>0.75	293	92	30	1	4

(2) Whorl Shape

UNITUBPOS\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	roun d	rect	ellips	ogi- val	oval
non	8	21	135	54	135	376	622	719	479	280
<0.25	0	4	4	0	14	3	25	19	41	7
0.25-0.5	4	1	1	0	8	2	9	13	9	10
0.5-0.75	16	0	0	0	13	26	16	65	6	5
>0.75	35	0	1	0	71	53	59	100	28	7

(3) Shell Diameter

UNITUBPOS\D	<3	3-6	6-9	9-12	12-15	>15
non	836	1580	715	290	116	188
<0.25	4	37	36	27	6	14
0.25-0.5	14	35	10	5	4	3
0.5-0.75	53	81	25	12	8	5
>0.75	104	218	93	45	12	25

14. Venter

(1) Whorl Shape

VENTER\ WHORL_SHAPE	coro	trape	tri	lanceo	quad -rate	round	rect	ellips	ogi-val	oval
plain	63	13	26	0	92	279	79	398	55	71
carinate	0	11	50	48	60	96	395	365	445	179
carinate-sulcate	0	2	0	0	75	56	229	106	35	19
sulcate	0	0	0	0	1	17	8	27	8	24
tricarinate	0	0	0	0	10	3	2	3	0	5

(2) Shell Diameter

VENTER\D	<3	3-6	6-9	9-12	12-15	>15
plain	326	512	247	88	28	58
carinate	381	854	366	156	64	85
carinate-sulcate	77	264	141	70	34	63
sulcate	36	35	10	4	1	5
tricarinate	2	10	2	5	0	5

15. Whorl Shape

(1) Shell Diameter

WHORL_SHAPE\D	<3	3-6	6-9	9-12	12-15	>15
coro	11	38	7	2	1	4
trape	7	7	7	1	1	3
tri	19	43	35	19	12	13
lanceo	14	26	8	4	2	0
quadrate	49	98	49	14	10	21
round	114	210	75	30	12	19
rect	113	280	193	77	31	37
ellips	228	363	183	68	22	52
ogival	98	277	88	55	21	23
oval	64	128	53	26	10	28

Appendix 2 Source code for the graphics interface of Ammon

There are 13 files in this program: ammon.h, retrieve.h, retrieve.c, amenu.h, amenu.c, display.c, query.c, dbms.c, misc.c, utility.h, utility.c, win.h, win.c:

1. ammon.h:

```
/* Header file for Ammon graphics interface
 * and interactive image measurement module.
 *
 * Author: Bo Liang
 * Department of Geological Sciences
 * The University of British Columbia
 * Date: September 1992
 * Modifications: April 1994
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <suntool/tty.h>
#include <suntool/alert.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/canvas.h>
#include <suntool/scrollbar.h>
#include <suntool/textsw.h>

/* the maximum number of specimens allowed for each query return */
#define MAX_SP          1000

/* the maximum number of images allowed for each query return, since one specimen can
   have multiple images, MAX_NUM+1000 gives extra room to accomondate images
*/
#define MAX_IMAGE       MAX_SP+1000

/* define the array index for descriptor_name */

#define IMAGE           35
#define SCALE           36
#define WHORL_SHAPE     52
#define ONTOGENY        61
#define APPROX          68

/* active descriptors in Ammon */
```

```
char*  descriptor_name[]={"SUPERFAMILY","FAMILY","SUBFAMILY","GENUS",
                          "SUBGENUS", "QUALIFIER", "SPECIES","SUBSPECIES",
                          "TAXAUTHYEAR", "REFAUTHYEAR", "SYNONYMY",
                          "SYNSPECIES","STAGE","SUBSTAGE","EURZONE",
                          "EURSUBZONE","ZONE","SUBZONE", "HORIZON",
                          "FORMATION", "MEMBER","SITU","ASSOCSPEC","AREA",
                          "COUNTRY","PROVINCE","SECTNAME","SECTNO","LOCNO",
                          "OTHERNO","REPOSITORY","TYPE", "GENERALOC",
                          "SPECNO", "REMARKS1", "IMAGE","SCALE", "DMAX", "D",
                          "UD","U", "EXP","WH","WHD", "WW","WWD","WWWH",
                          "PRHW", "SRHW","THW","BISPACE","CHW","WHORL_SHAPE",
                          "UWALLHT", "UWALLANG","VENTER","PTREND","PFORM",
                          "PPROF","FURC","TUBERC","ONTOGENY","MAXWH",
                          "RIBWIDTH","FURCPOS0","UNITUBPOS0","EXP0",
                          "VOLUTION0", "APPROX"};

#define NUM_OF_DESCRIPTOR    sizeof descriptor_name/sizeof descriptor_name[0]

/* the field length of the above descriptors */

short  len[]={19,17,22,20,20,15,30,20,30,40,22,30,25,40,30,30,30,20,25,30,20,2,
              6,20,20,30,50,25,20,14,40,14,80,30,200,50,4,6,6,6,5,5,6,5,6,5,5,
              4,4,4,5,4,30,4,8,16,14,30,16,12,10,4,3,3,5,3,4,3,40};

/* The following data arrays are used to initialize the menu */

/* ammanite family list in Ammon, new families may be added */

static char     *family[]={"Psiloceratidae","Schlotheimiidae","Arietitidae",
                "Echioceratidae","Oxynoticeratidae","Cymbitidae",
                "Eoderoceratidae","Coeloceratidae","Phricodoceratidae",
                "Polymorphitidae","Liparoceratidae","Amaltheidae",
                "Dactylioceratidae","Hildoceratidae","Phymatoceratidae",
                "Graphoceratidae","Sonniniidae","Cardioceratidae",
                "Erycitidae","Kosmoceratidae","Otoitidae",
                "Sphaeroceratidae","Stephanoceratidae","Oppeliidae",
                "Perisphinctidae","Reineckeiidae","Discophyllitidae",
                "Phylloceratidae","Lytoceratidae","Family uncertain"};

/* genus list in Ammon, if add new genus, make sure you update
   num_of_genus_in_family which can be tedious, better way to do it ? */


static char     *genus[]={"Alsatitoides","Badouxia","Caloceras",
                "Discamphiceras","Euphyllites","Laqueoceras","Mullerites",
```

"Paradiscamphiceras","Parapsiloceras","Psiloceras",
"Psilophyllites","Saxoceras","Transipsiloceras",
"Angulaticeras","Kammerkaroceras","Schlotheimia",
"Sulciferites","Waehneroceras","Aegasteroceras",
"Agassiceras","Alsatites","Arietites","Arnioceras",
"Asteroceras","Caenisites","Canavarites","Coroniceras",
"Defossiceras","Eparietites","Epophioceras","Euagassiceras",
"Metarnioceras","Pompeckioceras","Pseudaetomoceras",
"Pseudasteroceras","Pseudotropites","Sunrisites",
"Tmaegoceras","Vermiceras","Dubariceras","Echioceras",
"Gagaticeras","Leptechioceras","Palaeoechioceras",
"Paltechioceras","Protechioceras","Tmaegophioceras",
"Cheltonia","Fanninoceras","Gleviceras","Oxynoticeras",
"Paracymbites","Paroxynoticeras","Radstockiceras",
"Slatterites","Cymbites","Bifericeras","Crucilobiceras",
"Eoderoceras","Exomiltoceras","Metaderoceras",
"Microderoceras","Miltoceras","Promicroceras",
"Pseudoskirroceras","Xipheroceras","Apoderoceras",
"Coeloceras","Hyperderoceras","Pimelites","Praesphaeroceras",
"Tetraspidoceras","Epideroceras","Phricodoceras",
"Acanthopleuroceras","Dayiceras","Eoamaltheus",
"Gemmellaroceras","Parinodiceras","Peripleuroceras",
"Platypleuroceras","Polymorphites","Tropidoceras","Uptonia",
"Aegoceras","Androgynoceras","Liparoceras","Amaltheus",
"Amauroceras","Pleuroceras","Catacoeloceras","Collina",
"Dactylioceras","Nodicoeloceras","Peronoceras","Porpoceras",
"Preperonoceras","Prodactylioceras","Reynesoceras",
"Reynesocoeloceras","Zugodactylites","Arctomercaticeras",
"Arieticeras","Atacamiceras","Bouleiceras","Canavarella",
"Canavaria","Cylicoceras","Dumortieria","Eleganticeras",
"Fieldingiceras","Fontanelliceras","Frechiella",
"Fuciniceras","Grammoceras","Harpoceras","Hildaites",
"Hildoceras","Hildaitoides","Hudlestonia","Leioceras",
"Leptaleoceras","Leukadiella","Lioceratoides","Mercaticeras",
"Neolioceratoides","Oregonites","Onychoceras","Ovaticeras",
"Parahildaites","Paroniceras","Phlyseogrammoceras",
"Pleydellia","Podagrosites","Polyplectus","Protogrammoceras",
"Pseudogrammoceras","Pseudolioceras","Renziceras",
"Sphenarpites","Staufenia","Taffertia","Tiltoniceras",
"Tmetoceras","Bredyia","Brodieia","Erycites","Erycitoides",
"Esericeras","Eudmetoceras","Hammatoceras","Haplopleuroceras",
"Haugia","Nejdia","Phymatoceras","Planammatoceras",
"Podagrosiceras","Pseudomercaticeras","Puchenquia",
"Sphaerocoeloceras","Yakounia","Brasilia","Costileioceras",
"Darellia","Graphoceras","Hyperlioceras","Ludwigia",
"Reynesella","Dorsetensia","Euhoploceras","Fissilobiceras",

```
                "Fontannesia","Guhsania","Shirbuirnia","Sonninia",
                "Witchellia","Zurcheria","Arcticoceras","Arctocephalites",
                "Cadoceras","Cardioceras","Paracadoceras","Pseudocadoceras",
                "Podagrosiceras","Kepplerites","Kosmoceras","Docidoceras",
                "Emileia","Pseudotoites","Chondroceras","Eocephalites",
                "Eurycephalites","Iniskinites","Lilloettia","Loucheuxia",
                "Megasphaeroceras","Stehnocephalites","Xenocephalites",
                "Stephanoceras","Stemmatoceras","Oecotraustes","Oxycerites",
                "Prohecticoceras","Choffatia","Neuqueniceras","Rehmannia",
                "Harpophylloceras","Juraphyllites","Tragophylloceras",
                "Adabofoloceras","Calaiceras","Calliphylloceras",
                "Fergusonites","Holcophylloceras","Nevadaphyllites",
                "Partschiceras","Phylloceras","Procliviceras","Sowerbyceras",
                "Zetoceras","Alocolytoceras","Audaxlytoceras","Derolytoceras",
                "Holcolytoceras","Kallilytoceras","Lytoceras","Pleurolytoceras",
                "Pterolytoceras","Asaphoceras","Diplesioceras"};

short           num_of_genus_in_family[]={13,5,21,8,8,1,10,6,2,10,3,3,11,43,17,7,9,6,1,
                                          2,3,9,2,3,1,2,3,11,8,2};

/* stage list for Lower Jurassic and part of Middle Jurassic */

static char     *stage[]={"Hettangian","Sinemurian","Pliensbachian",
                "Toarcian","Aalenian","Bajocian"};

/* European zones in the above stages */

static char     *zone[]={"Planorbis","Liasicus","Angulata","Bucklandi",
                "Semicostatum","Turneri","Obtusum","Oxynotum",
                "Raricostatum","Jamesoni","Ibex","Davoei","Margaritatus",
                "Spinatum","Tenuicostatum","Falcifer","Bifrons",
                "Variabilis","Thouarsense","Levesquei","Opalinum",
                "Murchisonae","Concavum","Sowerbyi","Sauzei",
                "Humphriesianum"};

short           num_of_zone_in_stage[]={3,6,5,6,3,3};

/* large geographic regions in the Ammon */

static char     *area[]={"Western Pacific","Western Tethyan",
                "North American","South American","N.W Europe"};

/* all countries represented in the Ammon */

static char     *country[]={"Argentina","Austria","Britain","Canada",
                "Chile","China","Ecuador","France","Germany","Greece",
```

```
                "Hungary","Iran","Ireland","Italy","Japan","Malagasay",
                "Mexico","Peru","Portugal","Russia","Saudi Arabia",
                "Switzerland","United States","Turkey"};

/* rib form classification based on treatise */

static char     *ribForm[]={"straight","concave","convex","sinuous",
                "falcoid","falcate","biconcave","projected"};

/* rib furcation terms following Smith 1986 */

static char     *ribFurc[]={"intercalatory","bi","tri","multi","poly",
                "loop2","loop3","loopn","bundled"};

/* tubercle classification following Smith 1986 */

static char     *tubercle[]={"uni","bi","tri","multi","bullate","clavate"};

/* whorl shape classification following Smith 1986 */

static char     *whorlShape[]={"oval","ogival","ellipsoid","rounded",
                "lanceolate","rectangular","quadrate","triangular",
                "trapezoid","coronate","wellipsoid","wrectangular",
                "wtriangular"};

/* umbilical wall height */

static char     *wallHeight[]={"high","low"};

/* umbilical wall slope */

static char     *wallAng[]={"shallow","steep","vertical","undercut"};

/* venter shape following Smith 1986 */

static char     *venter[]={"plain","carinate","sulcate","carinate-sulcate",
                "bicarinate"};

/* source for the data in Ammon, should be reorganized in a more structured way, such as a
two level menu system */

static char     *reference[]={"Arkell Etc 1957","Arthur 1985",
                "Atlas Ed. By Repin 1968","Blasco 1978","Crickmay","Dagis",
                "Dean,Donovan,Howarth 1961","Erben 1956","Frebold",
                "Frebold and Mountjoy etc 1967","Gabb 1869","Gabilly 1976",
                "Geczy 1976","Geyer","Guex","Hall 1987",
```

```
                "Hall And Westerman 1980","Hall and Howarth 1983",
                "Hillebrandt","Hirano","Imlay 1968","Imlay 1981",
                "Jakobs 1992","Kalacheva 1980","Mclearn 1932","Meister 1986",
                "O'brien 1985","Palfy 1991","Poulton",
                "Poulton And Tipper 1991","Quinzio Sinn 1987","Repin 1974",
                "Schlatter 1991","Schlegelmilch 1976","Smith 1976","Smith 1981",
                "Smith And Tipper","Stankevich 1964","Taylor 1988",
                "Thomson And Smith 1992","Tilmann 1917",
                "Tipper and Smith etc 1991","Wang And Smith 1986",
                "White 1889","Whiteaves 1884","Wiedenmayer 1977",
                "Wiedenmayer 1980","Others"};

Frame           command_frame;      /* root frame to host menus and utilities*/
Frame           display_window;     /* window for image display */
Canvas          image_canvas; /* area in the display_window for image display */
Pixwin          *pw;

/* text window for diplaying the textual information of the selected specimen */
Textsw          text_win;

Panel           command_win;        /* left window hosting all menus and utilities */
Panel           info_panel; /* top window hosting query field and other textual information */

/* define sizes of the above windows, its location can be infered */

struct size     {
                int     width;
                int     height;
                } screen_rect = {1152, 900};

struct size     command_rect  = {175, 900};
struct size     info_rect = {980, 300};

struct point    {
                int     x;
                int     y;
                } between[2]={{0,0},{0,0}};

struct point    control_point[100];

Panel_item      condition_text, show_num, wait_num;
Panel_item      bright, contrast;
Pixfont         *font;
unsigned char   red [256], green [256], blue [256];
colormap_t      colormap = { RMT_EQUAL_RGB, 256, red, green, blue };
```

```
/* use the shell icon as the system icon */
static short    icon_data[] = {
                    #include <images/cmdtool.icon>
                    };
static          mpr_static(ammon_icon, 64, 64, 1, icon_data);
static char     interface_image = "/home/liang/interface.ras";

char    *path;  /* record which menu has been selected */
char    *image_name[MAX_IMAGE];
char    *query_statement, *search_condition[12];
int     num_of_Items; /* number of specimens retrieved */
int     choice; /* record which menu item has been selected */
int     cs;     /* base brightness of image    */
int     start_status=0;/* info panel is not initialized yet */
int     In_sublist = 0; /* specimen is not in the sub (species) list
                           instead it is in the main (genus) list */
int     display_from, display_to; /* display status */
int     from_menu; /* whether query is issued from menu */
int     num_control_point;
int     rel[MAX_IMAGE]; /* index conversion between image_name and descriptor_value */
int     displayed[MAX_IMAGE]; /* image hasn't been displayed (0);forward displayed (1);
                                 backward displayed (-1) */
int     seq[MAX_IMAGE]; /* order images by height */

double          image_scale;

/* global functions */

void            create_menu_button();
void            create_command_button(Frame frame_local);
void            init_menu();
void            init_info_panel(Frame frame_local);
void            menu_handler(); /* action upon menu item selection */
void            query_from_menu();
void            query_from_text();
void            execute_query();
void            sort_image_list();
void            build_full_name();
void            adjust_display_window();
void            display_images();
void            query_to_plain(char *buf)
void            stop_op();
```

2. retrieve.h

```
/* specific data declarations for retrieve.c, not shared by morph.c */
```

```
/* define the array index for descriptor_name */

#define SYNSPECIES          11

/* image is in the main (genus) list instead of species sublist */

#define MAIN_LIST           2

/* after the first query, the display window splits into three subwindows */
  from top to bottom, they are info panel, text subwindow and image canvas */
  all three windows have the same width and their heights add up to screen height */

struct size     {
        int     width;
        int     height;
                } text_rect = {980, 128};

struct size     canvas_rect = {980, 720};

int     Sublist[MAX_IMAGE]; /* whether the image is in species sublist */
int     break_point_in_genus; /* point where program get into the species sublist */


/* functions not shared with morph module */

void    main_list();
void    goto_specimen();
void    find_species();
void    save_as();
```

3. retrieve.c

```
/* main program of the Ammon user interface */

#include <ammon.h>

main(argc, argv)
int argc;
char **argv;
{
```

```
int             i, j;
Scrollbar       vertical_sb;

/* create a frame to host the left command window */

command_frame =
        window_create(NULL, FRAME,
                FRAME_LABEL, "Ammon Image Database",
                FRAME_SHOW_LABEL, TRUE,
                WIN_WIDTH, command_rect.width,
                WIN_HEIGHT, command_rect.height,
                WIN_X, 0,
                WIN_Y, 0,
                FRAME_ICON, icon_create(ICON_IMAGE, &ammon_icon, 0),
                0);

/* create the command window to host the menu and command buttons */

command_win = window_create(command_frame, PANEL,
                PANEL_LABEL_BOLD, TRUE,
                PANEL_SHOW_MENU, TRUE,
                0);

/* load font to label menu and command buttons */

font=pf_open("/apps/sundesk/fonts/sunview/lucida-19.v");

create_menu_button();

create_command_button();

init_menu();

/* create a frame to host the  info and canvas window at the right side of
   the command window */

display_window =
        window_create(NULL, FRAME,
                FRAME_LABEL, " Welcome Window",
                FRAME_SHOW_LABEL, TRUE,
                WIN_X, command_rect.width,
                WIN_Y, 0,
                WIN_SHOW,TRUE,
                WIN_WIDTH, screen_rect.width-command_rect.width,
                WIN_HEIGHT,screen_rect.height,
                0);
```

```
/* create the info panel giving an introduction to the system */

init_info_panel(display_window);

vertical_sb = scrollbar_create(SCROLL_LINE_HEIGHT, 5, (char *)0);
 /* 5 is the width of the scrollbar */

image_canvas=window_create(display_window, CANVAS,
        WIN_VERTICAL_SCROLLBAR, vertical_sb,
        CANVAS_RETAINED, FALSE,
        CANVAS_AUTO_SHRINK, FALSE,
        WIN_CONSUME_PICK_EVENT, MS_LEFT,
        WIN_EVENT_PROC, canvas_handler,
        WIN_X, 0,
        WIN_Y, info_rect.height+5,  /* canvas under info panel */
        WIN_WIDTH, screen_rect.width-command_rect.width,
        WIN_HEIGHT, screen_rect.height-info_rect.height-5,
        0);

/* make the image_canvas a color canvas with 253 colors */

pw=(Pixwin *) window_get(image_canvas, WIN_PIXWIN);

pw_putcolormap(pw, 0, 253, red, green, blue);
/* reserve 3 color entries for bg and fg */

window_set(image_canvas,
        CANVAS_RETAINED, TRUE,
        0);

/* load the interface image into the canvas */

load_interface_image();

/* allocate memory for retrieval results of database query */

for (i=0; i<NUM_OF_DESCRIPTOR; i++)
        for (j=0; j<MAX_SP; j++)
                descriptor_value[i][j] = (char *)malloc(len[i]+2);

for (i=0; i<MAX_IMAGE; i++)
        image_name[i]=(char *)malloc(len[IMAGE]+2);

signal(SIGURG, stop_op);
```

```
        window_main_loop(command_frame);

        exit(0);

        }

/* load the interface image into the image_canvas */

static void
load_interface_image()
        {
        Pixrect *image;
        FILE   *fp;
        register int      w, h;

        clear_canvas();

        /* don't move the interface image file to other location */

        if ((fp=fopen(interface_image,"r")) == NULL)        {
                printf("Can't open %s\n",interface_image);
                return;                                     }

        image=pr_load(fp, colormap);
        w=image->pr_size.x;
        h=image->pr_size.y;
        red[252]=255; /* color entry 252 gives pure red */
        green[252]=blue[252]=0;
        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
        cs=red[1];      /* as an indicator of the brightness */

        /* display the image in the middle of the canvas */

        if (h<screen_rect.height-info_rect.height)
                pw_rop(pw,(screen_rect.width-command_rect.width-w)/2,
                        (screen_rect.height-info_rect.height-h)/2,w,h,PIX_SRC,image,0,0);
        else
                pw_rop(pw,(screen_rect.width-command_rect.width-w)/2,0,
                        w,h,PIX_SRC,image,0,0);
        pr_close(image);
        fclose(fp);
}
```

4. amenu.h

```
/* Header file for creating the menu */

#include <suntool/walkmenu.h>

#define MENU_START_ROW      0       /* row where menu items start */
#define MENU_INCREMENT      2       /* increment in terms of number of rows */

/* trace which family has been selected, taxonomy starts with 'a' */

static char     *family_path[]={"aa","ab","ac","ad","ae","af","ag","ah",
                "ai","aj","ak","al","am","an","ao","ap","aq","ar","as",
                "at","au","av","aw","ax","ay","az","aA","aB","aC","aD"};

/* trace which stage has been selected, stratigraphy starts with 'b' */

static char     *stage_path[]={"ba","bb","bc","bd","be","bf"};

/* five menu buttons */

static Panel_item       button_Taxonomy,button_Stratigraphy,button_Geography;
static Panel_item       button_Morphology,button_Reference;

/* five root-level menus associated with the buttons */

static Menu             taxonomy_menu, stratigraphy_menu, geography_menu;
static Menu             morphology_menu, reference_menu;

/* two geography submenus */

static Menu             area_menu, country_menu;

/* nine morphology submenus */

static Menu             rib_menu, ribform_menu, ribfurc_menu;
static Menu             tubercle_menu, whorlshape_menu, wall_menu;
static Menu             wallheight_menu, wallang_menu, venter_menu;

/* family menus under taxonomy, number of submenus is determined by the  size of the family
array */

#define num_of_family       (sizeof family/sizeof family[0])
static Menu             family_menu[num_of_family];
```

```
/* stage menus under stratigraphy, number of submenus is determined by
   the  size of the stage array */

#define num_of_stage (sizeof stage/sizeof stage[0])
static Menu             stage_menu[num_of_stage];

/ * The followings are menu related functions */

static void             find_path();
```

5. amenu.c

```
#include <ammon.h>
#include <amenu.h>

/* create the menu buttons for five root level menus */

extern void
create_menu_button()
        {
        panel_create_item(command_win,PANEL_MESSAGE,
                PANEL_LABEL_FONT, font,
                PANEL_ITEM_Y, ATTR_ROW(MENU_START_ROW),
                PANEL_LABEL_STRING,"Retrieve by:",
                0);

        button_Taxonomy = panel_create_item(command_win,PANEL_BUTTON,
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_ITEM_Y,
ATTR_ROW(MENU_START_ROW+MENU_INCREMENT),
                PANEL_EVENT_PROC, menu_handler,
                PANEL_LABEL_IMAGE,
                panel_button_image(command_win,"Taxonomy =>",13,font),
                0);

        button_Stratigraphy = panel_create_item(command_win,PANEL_BUTTON,
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_ITEM_Y,
ATTR_ROW(MENU_START_ROW+2*MENU_INCREMENT),
                PANEL_LABEL_FONT, font,
                PANEL_EVENT_PROC, menu_handler,
                PANEL_LABEL_IMAGE,
                panel_button_image(command_win,"Stratigraphy =>",13,font),
                0);
```

```
        button_Geography = panel_create_item(command_win,PANEL_BUTTON,
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_ITEM_Y,
ATTR_ROW(MENU_START_ROW+3*MENU_INCREMENT),
                PANEL_EVENT_PROC, menu_handler,
                PANEL_LABEL_IMAGE,
                panel_button_image(command_win,"Geography =>",13,font),
                0);

        button_Morphology = panel_create_item(command_win,PANEL_BUTTON,
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_ITEM_Y,
ATTR_ROW(MENU_START_ROW+4*MENU_INCREMENT),
                PANEL_EVENT_PROC, menu_handler,
                PANEL_LABEL_IMAGE,
                panel_button_image(command_win,"Morphology =>",13,font),
                0);

        button_Reference = panel_create_item(command_win,PANEL_BUTTON,
                PANEL_ITEM_Y,
ATTR_ROW(MENU_START_ROW+5*MENU_INCREMENT),
                PANEL_EVENT_PROC, menu_handler,
                PANEL_LABEL_IMAGE,
                panel_button_image(command_win,"Reference =>",13,font),
                0);

        }

extern void
init_menu()
        {
        register int      i, j, from, to;

    /* Create Taxonomy Menu */

        from=to=0;
        for (i=0;i<num_of_family;i++) {
                family_menu[i]=menu_create(MENU_GEN_PROC,find_path,
                        MENU_CLIENT_DATA,family_path[i],
                        MENU_FONT,font,
                        MENU_STRING_ITEM,"Genera Menu",
                        0,0);
                to += num_of_genus_in_family[i];
                for (j=from; j<to; j++)
                        menu_set(family_menu[i],
                                MENU_STRING_ITEM,genus[j],
```

```
                    j-from+1,0);
        from += num_of_genus_in_family[i];     }

    taxonomy_menu=menu_create(MENU_FONT,font,
                MENU_STRING_ITEM,"Family Menu",
                0,0);

    /* attach family menu to taxonomy menu */

    for (i=0;i<num_of_family;i++)
        menu_set(taxonomy_menu,MENU_PULLRIGHT_ITEM,
                family[i],family_menu[i],
                0);

/* create Stratigraphy Menu */

    from=to=0;
    for (i=0;i<num_of_stage;i++) {

    /* create a menu for each stage */

        stage_menu[i]=menu_create(MENU_GEN_PROC,find_path,
                    MENU_CLIENT_DATA,stage_path[i],
                    MENU_FONT,font,
                    MENU_STRING_ITEM,"Zone Menu",
                    0,0);
        to += num_of_zone_in_stage[i];
        for (j=from; j<to; j++)
            menu_set(stage_menu[i],
                    MENU_STRING_ITEM,zone[j],
                    j-from+1,0);
        from += num_of_zone_in_stage[i];     }

    stratigraphy_menu=menu_create(MENU_FONT,font,
                    MENU_STRING_ITEM,"Stage Menu",
                    0,0);

    /* attach stage menu to stratigraphy menu */

    for (i=0;i<num_of_stage;i++)
        menu_set(stratigraphy_menu,MENU_PULLRIGHT_ITEM,
                stage[i],stage_menu[i],
                0);

    /* create Geography Menu */
```

```
/* create area submenu */

area_menu=menu_create(MENU_GEN_PROC,find_path,
                MENU_CLIENT_DATA,"ca",
                MENU_FONT,font,
                MENU_STRING_ITEM,"Area Menu",
                0,0);

for (i=0; area[i] != NULL; i++)
        menu_set(area_menu,
                MENU_STRING_ITEM, area[i],
                i+1, 0);

/* create country submenu */

country_menu=menu_create(MENU_GEN_PROC,find_path,
        MENU_CLIENT_DATA,"cb",
        MENU_FONT,font,
        MENU_STRING_ITEM,"Country Menu",
        0,0);

for (i=0; country[i] != NULL; i++)
        menu_set(country_menu,
                MENU_STRING_ITEM, country[i],
                i+1, 0);

geography_menu=menu_create(MENU_FONT,font,
        MENU_STRING_ITEM,"Geography Menu",
        0,0);

/* attach area and country submenus to geography menu */

menu_set(geography_menu,
        MENU_PULLRIGHT_ITEM,"Area",area_menu,
        MENU_PULLRIGHT_ITEM,"Country",country_menu,
        0);

/* create Morphology Menu */

/* create rib form submenu, 'daa' records the path upon menu event */

ribform_menu=menu_create(MENU_GEN_PROC,find_path,
                MENU_CLIENT_DATA,"daa",
                MENU_FONT,font,
                MENU_STRING_ITEM,"Rib Form",
                0,0);
```

```
for (i=0; ribForm != NULL; i++)
        menu_set(ribform_menu,
        MENU_STRING_ITEM, ribForm[i],
        i+1, 0);

/* create rib furcation submenu, 'dab' records the path upon menu event */

ribfurc_menu=menu_create(MENU_GEN_PROC,find_path,
        MENU_CLIENT_DATA,"dab",
        MENU_FONT,font,
        MENU_STRING_ITEM,"Rib Furcation",
        0,0);

for (i=0; ribFurc[i] != NULL; i++)
        menu_set(ribfurc_menu,
                MENU_STRING_ITEM, ribFurc[i],
                i+1, 0);

rib_menu=menu_create(MENU_FONT,font,
                MENU_STRING_ITEM,"Rib Menu",
                0,0);

/* attach rib form and rib furcation submenus to rib submenu */

menu_set(rib_menu,
        MENU_PULLRIGHT_ITEM,"Rib Form",ribform_menu,
        MENU_PULLRIGHT_ITEM,"Rib Furcation",ribfurc_menu,
        0);

/* create tubercle submenu, 'db' records the path upon menu event */

tubercle_menu=menu_create(MENU_GEN_PROC,find_path,
                MENU_CLIENT_DATA,"db",
                MENU_FONT,font,
                MENU_STRING_ITEM,"Tubercle Form",
                0,0);

for (i=0; tubercle[i] != NULL; i++)
        menu_set(tubercle_menu,
                MENU_STRING_ITEM,tubercle[i],
                i+1,0);

/* create whorl shape submenu, 'dc' records the path upon menu event */

whorlshape_menu=menu_create(MENU_GEN_PROC,find_path,
```

```
                MENU_CLIENT_DATA,"dc",
                MENU_FONT,font,
                MENU_STRING_ITEM,"Whorl Shape",
                0,0);

for (i=0; whorlShape[i] != NULL; i++)
        menu_set(whorlshape_menu,
                MENU_STRING_ITEM,whorlShape[i],
                i+1,0);

/* create umbilical wall height submenu, 'dda' records the path upon menu event */

wallheight_menu=menu_create(MENU_GEN_PROC,find_path,
        MENU_CLIENT_DATA,"dda",
        MENU_FONT,font,
        MENU_STRING_ITEM,"Wall Height",
        0,0);

for (i=0; wallHeight[i] != NULL; i++)
        menu_set(wallheight_menu,
                MENU_STRING_ITEM,wallHeight[i],
                i+1,0);

/* create umbilical wall slope submenu, 'ddb' records the path upon menu event */

wallang_menu=menu_create(MENU_GEN_PROC,find_path,
        MENU_CLIENT_DATA,"ddb",
        MENU_FONT,font,
        MENU_STRING_ITEM,"Wall Slope",
        0,0);

for (i=0; wallAng[i] != NULL; i++)
        menu_set(wallang_menu,
                MENU_STRING_ITEM,wallAng[i],
                i+1,0);

wall_menu=menu_create(MENU_FONT,font,
        MENU_STRING_ITEM,"Umbilical Wall",
        0,0);

/* attach umbilical height and slope submenus to wall submenu */

menu_set(wall_menu,
        MENU_PULLRIGHT_ITEM,"Wall Height",wallheight_menu,
        MENU_PULLRIGHT_ITEM,"Wall Slope",wallang_menu,
        0);
```

```
        /* create venter shape submenu, 'de' records the path upon menu event */

        venter_menu=menu_create(MENU_GEN_PROC,find_path,
                MENU_CLIENT_DATA,"de",
                MENU_FONT,font,
                MENU_STRING_ITEM,"Venter Shape",
                0,0);

        for (i=0; venter[i] != NULL; i++)
                menu_set(venter_menu,
                        MENU_STRING_ITEM,venter[i],
                        i+1,0);

        /* attach rib, tubercle, whorl shape, umbilical wall and venter submenus to morphology
menu */

        morphology_menu=menu_create(MENU_FONT,font,
                                MENU_STRING_ITEM,"Morphology Menu",
                                0,0);
        menu_set(morphology_menu,
                MENU_PULLRIGHT_ITEM,"Rib",rib_menu,
                MENU_PULLRIGHT_ITEM,"Tubercle",tubercle_menu,
                MENU_PULLRIGHT_ITEM,"Whorl Shape",whorlshape_menu,
                MENU_PULLRIGHT_ITEM,"Umbilical Wall",wall_menu,
                MENU_PULLRIGHT_ITEM,"Venter",venter_menu, 0);

        /* Create Reference Menu, 'e' records the path upon menu event */

        reference_menu=menu_create(MENU_GEN_PROC,find_path,
                        MENU_CLIENT_DATA,"e",
                        MENU_FONT,font,
                        MENU_STRING_ITEM,"Reference Menu",
                        0,0);
        for (i=0; reference[i] != NULL; i++)
                menu_set(reference_menu,
                        MENU_STRING_ITEM,reference[i],
                        i+1,0);

        }

/* find_path determine which submenu has been selected and record the path */

static void
find_path(menu, operation)
        Menu                    menu;
```

```
        Menu_generate        operation;
        {
        switch (operation) {
                case MENU_NOTIFY:
                        path=menu_get(menu,MENU_CLIENT_DATA);
                        break;
                case MENU_DISPLAY:
                case MENU_DISPLAY_DONE:
                case MENU_NOTIFY_DONE:
                        break; }
        }

/* menu_handler determine which menu item has been selected and store the value
   in choice (int) */

extern void
menu_handler(item , event)
        Panel_item      item;
        Event           *event;
        {

        if (event_action(event) == MS_RIGHT && event_is_down(event)
        && event_is_button(event))  {
                path="";
                if (item==button_Taxonomy) {
                        panel_begin_preview(item, event);
                        choice = (int)menu_show(taxonomy_menu, command_win,event,0);
                                        }
        else if (item==button_Stratigraphy)   {
                panel_begin_preview(item, event);
                choice = (int)menu_show(stratigraphy_menu, command_win,event,0);
                                        }
        else if (item==button_Geography)      {
                panel_begin_preview(item, event);
                choice = (int)menu_show(geography_menu, command_win,event,0);
                                        }
        else if (item==button_Morphology)   {
                panel_begin_preview(item, event);
                choice = (int)menu_show(morphology_menu, command_win,event,0);
                                        }
        else if (item==button_Reference)      {
                panel_begin_preview(item, event);
                choice = (int)menu_show(reference_menu, command_win,event,0);
                                        }
        panel_accept_preview(item, event);
        panel_update_preview(item, event);
```

```
        if (*path)      {
                query_from_menu();
                num_of_Items=0;
                execute_query();
                if (num_of_Items)       {
                        from_menu=1;
                        sort_image_list();
                        build_full_name();
                        display_images();       }
                        }
        }
        else panel_default_handle_event(item, event);
        }
```

6. display.c

```
#include <ammon.h>

static int      x, y;
static int      width[MAX_IMAGE], height[MAX_IMAGE];
static int      current_page[200]; /* conversion between index on the page
                                        and general image list */
static int      num_in_page; /* number of images on the page */


/* initialize and display the first screen of the images */

extern void
display_images()
        {
        register int    i, j, k;
        char            tmp[10];
        FILE            *fp;
        Pixrect         *image;
        struct rasterfile       rh;

        /* read width and height for all retrieved images */

        for (i=0; i<num_of_Items; i++)      {
                if ((fp=fopen(image_name[i],"r")) == NULL)
                        goto next;
                pr_load_header(fp, &rh);
                width[i]=rh.ras_width;
                height[i]=rh.ras_height;
                fclose(fp);
```

```
next:   ;
                                }

if (!start_status)      /* if this is the first query issued, rearrange the windows */
        adjust_display_window();
if (from_menu)          /* if query is issued from menu, display the query
                           in the text field */
        {
        buf=&query_statement[42];
        query_to_plain(buf);
        i=strlen(buf);
        buf[i-2]=0;
        tmp[0]=0;
        sprintf(tmp,"%s%s",buf,search_condition[0]);
        panel_set(condition_text,
        PANEL_VALUE, tmp,
        0);
        }

/* display the number of images retrieved */

sprintf(tmp,"%d",num_of_Items);
panel_set(show_num,
        PANEL_VALUE, tmp,
        0);

/* reorder the image based on height so images are nicely tiled */

for (i=0; i<num_of_Items; i++)
        seq[i]=i;
for (i=0; i<num_of_Items-1; i++)
        for (j=i+1; j<num_of_Items; j++)
                if (height[seq[j]]>height[seq[i]])      {
                        k=seq[i];
                        seq[i]=seq[j];
                        seq[j]=k;       }

clear_canvas();         /* make ready for display */

/* if the height of the largest image > canvas height, increase the canvas height */
if (height[seq[1]]>screen_rect.height-info_rect.height-20)
        (void)window_set(image_canvas,
                        CANVAS_HEIGHT, height[seq[1]]+20,
                        0);

cs=x=y=display_to=0;
```

```
        display_from=1;
        num_in_page=0;

        display_to = num_of_Items;
        for (i=0; i<num_of_Items; i++)
                if (displayed[seq[i]] != 1)
                        if (!display_specimen(i, 1)) { /* display images of this specimen,
                                                          1 - forward */
                                break;  /* end of the page */
                                display_to = i; }

        modify_colormap(); /* adjust brightness based on all images of the page */

        /* show the number of images still waiting to display */

        sprintf(tmp,"%d",num_of_Items-num_in_page);
        panel_set(wait_num,
                PANEL_VALUE, tmp,
                0);
}

/* turn to the next page */

extern void
next_page()
        {
        int             i, k;
        char            tmp[10];

        /* haven't reached the end of the image list */

        if (display_to<num_of_Items && display_from<num_of_Items-1)  {

                clear_canvas();

        /* record where it starts */

                if (display_from>display_to) /* following previous page operation */
                        k=display_from=display_to=display_from+1;
                else
                        k=display_from=display_to;

                num_in_page=cs=x=y=j=0;             /* start of a new page */

                display_to = num_of_Items;
                for (i=k; i<num_of_Items; i++)
```

```
                if (displayed[seq[i]] != 1) /* hasn't been forward displayed */
                        if (!display_specimen(i, 1)) { /* display images of this specimen
                                                        1 - forward */
                                display_to = i;
                                break;  /* end of the page */
                                                }

        modify_colormap(); /* adjust brightness based on all images of the page */

        /* show the number of images still waiting to display */

        if (display_to < num_of_Items)      {
                if (display_to >= display_from)
                        k = num_of_Items-display_to-1;
                else if (display_to < display_from)
                        k=display_to-1;         }
        else
                k = 0;

        sprintf(tmp,"%d",k);
        panel_set(wait_num,
                PANEL_VALUE, tmp,
                0);
                                                }
    else
        panel_set(wait_num,
                PANEL_VALUE, "No!",
                0);
    }

/* turn the page backwards */

static void
previous_page()
    {
    int         i, k;
    char        tmp[10];

    /* haven't reached the begining of the image list */

    if (display_to && display_from)     {

        clear_canvas();

    /* record where it starts */
```

```
            if (display_to<display_from)
                    k=display_from=display_to;
            else
                    k=display_from=display_to=display_from-1;

            num_in_page=cs=x=y=j=0; /* start of a new page */

            display_to = 0;
            for (i=k; i>0; i--)
                    if (displayed[seq[i]] != -1) /* hasn't been displayed backwards */
                            if (!display_specimen(i, -1)) { /* -1 -> display backwards */
                                    display_to = i;
                                    break;  /* end of the page */
                                                    }

            modify_colormap(); /* adjust brightness based on all images of the page */

            /* show the number of images still waiting to display */

            if (display_to)
                    k = display_to;
            else
                    k = 0;
            sprintf(tmp,"%d",k);
            panel_set(wait_num,
                    PANEL_VALUE, tmp,
                    0);
                                    }
    else
            panel_set(wait_num,
                    PANEL_VALUE, "No!",
                    0);
    }


/* find all images of the speciemn sp and display them */

static int
display_specimen(sp, direction)
    int     sp, direction;
    {
    register int    i, j, j1, j2, k, order[10];
    FILE            *fp;
    Pixrect         *image;
```

```
/* images from the same specimen are sequential */

/* find the lower limit */

i=seq[sp];
while (rel[--i]==rel[seq[sp]]) ;
j1=i+1;

/* find the upper limit */

i=seq[sp];
while (rel[++i]==rel[seq[sp]]) ;
j2=i-1;

/* reorder the images of the specimen by height */

if (j2>j1)      {
        for (i=j1; i<=j2; i++)
                order[i-j1]=i;
        for (i=0; i<j2-j1; i++)
                for (j=i+1; j<=j2-j1; j++)
                        if (height[order[j]]>height[order[i]]) {
                                k=order[i];
                                order[i]=order[j];
                                order[j]=k;     }
                }
else
        order[0]=seq[sp];

/* display the images in order */

k=0;    /* the current height reached by tiled images */

for (i=0; i<j2-j1+1; i++)
        if (displayed[order[i]] != 1)    /* hasn't been displayed */     {
                if ((fp=fopen(image_name[order[i]],"r")) == NULL )          {
                        printf("Can't open image file %s\n",image_name[order[i]]);
                        goto next_image;        }
                image=pr_load(fp, colormap);
                if (num_in_page && x+width[order[i]]>wid) {
                        x=0;
                        y += k;
                        k = height[order[i]];  }
                if (y+height[order[i]]<screen_rect.height-info_rect.height || !y ) {
                        if (j<height[order[i]])
                                j=height[order[i]];
```

```
                        cs += red[1];

        pw_rop(pw,x,y,width[order[i]],height[order[i]],PIX_SRC,image,0,0);
                        x += width[order[i]];
                        pr_close(image);
                        fclose(fp);
                        displayed[order[i]] = direction;
                        current_page[num_in_page++]=order[i];
                                                        }
                else    {
                        display_to=sp;
                        pr_close(image);
                        fclose(fp);
                        return 0;  /* end of the page */ }
        next_image:     ;
                                                                }
        return 1;       /* more room on the page */

        }

/* cs - the base of brightness, adjust brightness by modifying colormap  */

static void
modify_colormap()
        {
        register int i;

        if (num_in_page)
                cs /= num_in_page;
        for (i=1; i<252; i++)
                red[i]=blue[i]=green[i]=(cs+i-1>255) ? 255:cs+i-1;
        green[252]=blue[252]=0;
        panel_set_value(bright,0);
        panel_set_value(contrast,10);
        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
        }

/* locate the selected image in the image list *

extern int
image_index(xo, yo)
        int     xo,yo;
        {
        register int    i, k, x1, y1, j=-1;
```

```
        x1=y1=0;
        k=height[current_page[0]];

        for (i=0; i<num_in_page; i++){
                if (x+width[current_page[i]]>wid)    {
                        y1 += k;
                        k = height[current_page[i]];
                        x1 = 0;
                                                }

                /* (xo, yo) falls within the range of image j */

                if (xo>x1 && xo<x1+width[current_page[i]] && yo>y1 &&
                        yo<y1+height[current_page[i]])      {
                        j=current_page[i];
                        break;
                }

                x += width[current_page[i]];
                if (k<height[current_page[i]])
                        k=height[current_page[i]];

        return j;       /* return the index number of the selected image */

        }

/* display all information of the selected specimen */

extern void
show_descriptors(index_of_image)
        int     index_of_image;
        {
        register int    i, k;
        float           f;
        char            tmp[200];

        textsw_normalize_view(text_win, (Textsw_index) window_get(text_win,
                                TEXTSW_INSERTION_POINT));

        /* display image name, quantitative descriptors and qualitative descriptors */

        for (i=0; i<NUM_OF_DESCRIPTOR; i++)

                if (i==IMAGE)       {
                        sprintf(tmp,"%s:%s.  ",descriptor_name[i],
                                image_name[index_of_image]);
```

```
                textsw_insert(text_win, tmp, strlen(tmp));
                        }

        else if ((i>IMAGE && i<WHORL_SHAPE || i>ONTOGENY &&
                i<APPROX) && (f=atof(descriptor_value[i][rel[index_of_image]]))>0)
        /* quantitative parameters */
{

        /* data exists for this descriptor */

                sprintf(tmp,"%s:%5.2f ",descriptor_name[i],f);
                textsw_insert(text_win, tmp, strlen(tmp));
                        }

        else if (descriptor_value[i][rel[index_of_image]][0] != ' ')    {

        /* data exists for this descriptor */

                k=len[i]-2;
                while (descriptor_value[i][rel[index_of_image]][k--] == ' ') ;
                descriptor_value[i][rel[index_of_image]][k+2]='\0';
                sprintf(tmp,"%s:%s.  ",descriptor_name[i],
                        descriptor_value[i][rel[index_of_image]]);
                textsw_insert(text_win, tmp, strlen(tmp));              }

textsw_insert(text_win, "\n", 1);

}
```

7. query.c

```
#include <ammon.h>

/* construct a query statement from a selected menu item */

extern void
query_from_menu()
        {
        register int i, j, depth, entry;
        char tmp[150];

        depth = 1;
        while (path[depth] != '\0')
                depth++;                /* the depth of the menu item */
```

```
j='A'-('z'-'a'+1); /* upper case conversion */

switch (path[0]) {
        case 'a':       /* a menu item in taxonomy */
                if (depth==2 && !choice) {   /* select a family */
                        query_statement="select * from
                                        OPS$liang.ammon_image
                                        where family= :";
                        if (path[1]>'a'-1)      /* lower case */
                                search_condition[0]=family[path[1]-'a'];
                        else    /* upper case */
                                search_condition[0]=family[path[1]-j];
                                        }
                else if (depth==2 && choice) {  /* select a genus */
                        query_statement="select * from
                                        OPS$liang.ammon_image
                                        where synonymy= :";
                        if (path[1]>'a'-1)      /* lower case */
                                depth=path[1]-'a';
                else
                        depth=path[1]-j;        /* upper case */
                        entry=0;
                        for (i=0; i<depth; i++)
                                entry += num_of_genus_in_family[i];
                        entry += choice-1; /* genus at entry is selected */
                        search_condition[0]=genus[entry];
                                        }
                break;
        case 'b':       /* a menu item in stratigraphy */
                        if (depth==2 && !choice)    { /* select a stage */
                                query_statement="select * from
                                                OPS$liang.ammon_image
                                                where stage= :";
                                search_condition[0]=stage[path[1]-'a'];
                                                }
                        else if (depth==2 && choice) {      /* select a zone */
                                query_statement="select * from
                                                OPS$liang.ammon_image
                                                where eurzone= :";
                                depth=path[1]-'a';      /* lower case */
                                entry=0;
                                for (i=0; i<depth; i++)
                                        entry += num_of_zone_in_stage[i];
                                entry += choice-1; /zone at entry is selected */
                                search_condition[0]=zone[entry];
                        }
```

```
            break;
        case 'c':       /* a menu item in geography */
            if (depth==2 && choice) {
                if (path[1]-'a') {
                    query_statement="select * from
                                OPS$liang.ammon_image
                                where country= :";
                    search_condition[0]=country[choice-1];
                        }
                else    {
                    query_statement="select * from
                                OPS$liang.ammon_image
                                where area= :";
                    search_condition[0]=area[choice-1];
                    }
                        }
        break;

        case 'd':       /* a menu item in morphology */
            if (depth>=2)    {
                switch (path[1])    {
                    case 'a':       /* rib */
                        if (path[2]-'a') {
                            query_statement="select * from
                            OPS$liang.ammon_image
                            where furc like :";
                            search_condition[0]=
                            (char *)strdup(ribFurc[choice-1]);
}
                        else if (path[2]-'a'-1)   {
                            query_statement="select * from
                            OPS$liang.ammon_image
                            where pform like :";
                            search_condition[0]=
                            (char *)strdup(ribForm[choice-1]);
                                }
                        break;
                    case 'b':
                        query_statement="select * from
                                OPS$liang.ammon_image
                                where tuberc like :";
                        search_condition[0]=
                        (char *)strdup(tubercle[choice-1]);
                    break;
            case 'c':
                        query_statement="select * from
```

```
                                        OPS$liang.ammon_image
                                        where whorl_shape like :";
                                        search_condition[0]=
                                        (char *)strdup(whorlShape[choice-1]);
                        break;
                case 'd':
                                if (path[2]-'a') {
                                        query_statement="select * from
                                        OPS$liang.ammon_image
                                        where uwallang like :";
                                        search_condition[0]=
                                        (char *)strdup(wallAng[choice-1]);
                                                }
                                else if (path[2]-'a'-1)  {
                                        query_statement="select * from
                                        OPS$liang.ammon_image
                                        where uwallht like :";
                                        search_condition[0]=
                                        (char *)strdup(wallHeight[choice-1]);
                        }
                break;
                case 'e':
                                query_statement="select * from
                                        OPS$liang.ammon_image
                                        where venter like :";
                                        search_condition[0]=
                                        (char *)strdup(venter[choice-1]);
                                break;
                        }
                tmp[0]=0;

/* bd is the bind variable in Oracle dynamic SQL statement */

                sprintf(tmp,"%s bd",query_statement);
                query_statement=(char *)strdup(tmp);

/* SQL 'like' operator needs % to indicate containing */

                sprintf(tmp,"%s%s%s","%",search_condition[0],"%");
                search_condition[0]=(char *)strdup(tmp);
                }
        break;
        case 'e':       /* a menu item in reference */
        if (choice)     {
                query_statement="select * from
                        OPS$liang.ammon_image
```

```
                                        where refauthyear like :";
                                search_condition[0]=(char *)strdup(reference[choice-1]);
                                i=strlen(search_condition[0]);
                                search_condition[0][i]='%';
                                search_condition[0][i+1]='\0';
                                tmp[0]=0;
                                sprintf(tmp,"%s bd",query_statement);
                                query_statement=(char *)strdup(tmp);
                                        }
                        break;
                }
        if (!strstr(query_statement, "like :")) {
                tmp[0]=0;
                sprintf(tmp,"%s bd",query_statement);
                query_statement=(char *)strdup(tmp);
                                                }

}

/* construct a query statement from the text field  */

extern void
query_from_text()
{
        int register    i, j, k, ind, nl=0;
        char            *buf, *replace(), tmp[200],tmp1[200];

        if (!start_status)      {
                adjust_display_window();
                return; }

        In_sublist=0;

        /* construct the query statement from condition field */

        buf=panel_get_value(condition_text);
        buf=substitute(buf,"= :","=");
        buf=substitute(buf,"like :","like");
        buf=substitute(buf,"> :",">");
        buf=substitute(buf,"< :","<");

        /* replace the bind variable value with bd1, bd2,..... */

        sprintf(tmp1,"%s",replace(buf));

        sprintf(tmp,"select * from OPS$liang.ammon_image where %s",tmp1);
```

```
        query_statement=(char *)strdup(tmp);

        sprintf(tmp,"%s",buf);
        i=0;
        if (buf=(char *)strtok(tmp,":"))        {
                if (strstr(buf,"like"))
                        nl=1;
                else
                        nl=0;
                while (buf=(char *)strtok(NULL,":"))        {
                        j=k=0;
                        if (nl)
                                tmp1[j++]='%';
                        else
                                tmp1[j]=0;
                        ind=1;
                        while (ind || (buf[k] != ' ' &&  buf[k] != ')' && buf[k] !='\0'))
                                if (buf[k] != ' '){
                                        tmp1[j++]=buf[k++];
                                        ind=0;  }
                                else
                                        k++;
                        if (nl)
                                tmp1[j++]='%';
                        tmp1[j]=0;
                        if (strstr(buf,"like"))
                                nl=1;
                        else
                                nl=0;
                        search_condition[i++]=(char *)strdup(tmp1);
                                                        }
                execute_query();
        if (num_of_Items)       {
                from_menu=0;
                sort_image_list();
                build_full_name();
                display_images();       }
        else    {
                panel_set_value(show_num,"None !");
                panel_set_value(wait_num,"0");      }     }
        else
                panel_set_value(condition_text,"Invalid Search Statement !");
}

/* replace ': bind variable value' in s with bd1, bd2,.. */
```

```
extern char *  replace(s)
        char    *s;
        {
        register int    i, j, k, ind;
        char            tmp[200], tp[10], *buf;

        i=strlen(s);
        j=1;
        tmp[0]=0;
        while (buf=(char *)strstr(s,":"))       {
                strncat(tmp,s,i-strlen(buf));
                sprintf(tp," : bd%d",j++);
                strcat(tmp,tp);
                k=ind=1;

        /* skip bind variable value */

                while (ind || (buf[k] !=' ' && buf[k] !='\0' && buf[k] !=')')) {
                        if (buf[k] !=' ')
                                ind=0;
                        k++;    }
                s=&buf[k];
                i=strlen(s);                    }

        strcat(tmp,s);
        s=(char *)strdup(tmp);
        return s;

        }
```

8. dbms.pc

```
/* all database related operations are in this file which need to be compiled by Oracle
precompiler before proceeding with cc and combine with other source code */

#include <stdio.h>
#include <ammon.h>

EXEC SQL BEGIN DECLARE SECTION;
        extern char     *query_statement;
        extern char     *search_condition[12];
        char            oracleid = '/';
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;
```

```
EXEC SQL INCLUDE SQLDA;

/* Declare the bind and select descriptors. */
SQLDA           *bind_dp;
SQLDA           *select_dp;

extern SQLDA    *sqlald();

extern void     sqlprc();

extern void     sqlnul();
int             null_ok;


extern void
execute_query()
        {
        register int    i,j,k;
        int             precision, scale;
        float           tmp;
        double          atof();

        /* Automatic log on to ORACLE */

        EXEC SQL CONNECT :oracleid;

        /* Set up ORACLE error handling. */

        EXEC SQL WHENEVER SQLERROR GOTO sqlerror;


        bind_dp = sqlald (12, 30, 0);
        bind_dp->N = 12; /* the maximum number of binding variables */

        select_dp = sqlald (NUM_OF_DESCRIPTOR, 15, 0);
        select_dp->N = NUM_OF_DESCRIPTOR;


        /* Prepare the statement and declare a cusor. */

        EXEC SQL PREPARE S FROM :query_statement;
        EXEC SQL DECLARE C CURSOR FOR S;

        EXEC SQL DESCRIBE BIND VARIABLES FOR S INTO bind_dp;

        /* If F is negative, there were more bind variables than originally
```

```
    allocated by sqlald(). */

    if (bind_dp->F <0)
    {
    printf("\nToo many bind variables for descriptor.");
    goto sqlerror;
    }

    /* Set the maximum number of array elements in the descriptor to the number found.
*/
    bind_dp->N = bind_dp->F;

    for (i=0; i<bind_dp->F; i++)
    {
            bind_dp->L[i] = strlen(search_condition[i]);
            bind_dp->V[i] = search_condition[i];
            bind_dp->I[i] = 0;
            bind_dp->T[i]=1;
    }

    /* Open the cursor and define the active set. */

    EXEC SQL OPEN C USING DESCRIPTOR bind_dp;

    EXEC SQL DESCRIBE SELECT LIST FOR S INTO select_dp;

    /* If F is negative, there were more select-list items than originally
       allocated by sqlald(). */

    if (select_dp->F <0)
    {
            printf("\nToo many select-list items for : %d", -(select_dp->F));
            goto sqlerror;
    }

    /* Set the maximum number of array elements in the descriptor to
       the number found. */

    select_dp->N = select_dp->F;

    /* Allocate storage for each select-list item. */
    printf("\n\n");
    for (i=0; i<select_dp->F; i++)
    {
    /* Turn off high-order bit of datatype. */
    sqlnul (&(select_dp->T[i]), &(select_dp->T[i]), &null_ok);
```

```
        switch (select_dp->T[i])
        {
                case 1: /* CHAR datatype: no change in length needed.
*/
                        break;
                case 2: /* NUMBER datatype: use sqlprc() to extract precision and scale.  */
                        sqlprc (&(select_dp->L[i]), &precision, &scale);
                        /* Allow for maximum size of NUMBER. */
                        if (precision == 0)  precision = 40;
                        /* Also allow for decimal point and possible sign.  */
                        select_dp->L[i] = precision + 2;
                        /* Allow for a negative scale.  */
                        if (scale <0 )
                                select_dp->L[i] += -scale;
                        break;
        }
        /* Allocate space for the select-list data values.  */
        select_dp->V[i] = (char *)malloc(select_dp->L[i]);
        /* Coerce all datatypes to character.  */
        select_dp->T[i] = 1;
        }
        printf("\n\n");

        /* Fetch each row selected  */
        EXEC SQL WHENEVER NOT FOUND GOTO end_query;
        j=MAX_SP-1;
        for (;;)
        {
                EXEC SQL FETCH C USING DESCRIPTOR select_dp;
                for (i=0; i<select_dp->F; i++)
                        sprintf(descriptor_value[i][j],"%-.*s",select_dp->L[i]+1,
                                select_dp->V[i]);
                j--;
                if (j<0) {
                        printf("\n Warning:Too many images meet the the search condition\n
                                The rest of images skipped.\n");
                        break;  }
        }

        end_query:
        printf("\nNumber of rows processed: %d\n",sqlca.sqlerrd[2]);
        num_of_Items=sqlca.sqlerrd[2];
        /* Free space used by descriptors.  */
        for (i=0; i<select_dp->F; i++)
                free (select_dp->V[i]);
```

```
        /* Free space used by the descriptors themselves.  */
        sqlclu (bind_dp);
        sqlclu (select_dp);

        EXEC SQL CLOSE C;
        EXEC SQL COMMIT WORK;
        puts(" HAVE A GOOD DAY!\n");

        /* disconnect from Oracle database engine */

        EXEC SQL COMMIT RELEASE;

        return;
        /* ORACLE error handler */
        sqlerror:
        printf("\n\n%.70s\n",sqlca.sqlerrm.sqlerrmc);
        EXEC SQL WHENEVER SQLERROR CONTINUE;
        EXEC SQL ROLLBACK WORK;
        EXEC SQL COMMIT RELEASE;

        }
```

9. misc.c

```
/* functions specific for user interface */

#include <ammon.h>
#include <retrieve.h>

/* go back to main (genus) list from species sublist */

extern void
main_list()
        {
        register int    i, j;
        char            tmp[10];

        i=display_from=num_control_point=0;
        j=seq[break_point_in_genus];
        Sublist[break_point_in_genus]=displayed[j]=0;
        j=1;
        for (i=break_point_in_genus+1; i<num_of_Items; i++)
                if (displayed[seq[i]]==MAIN_LIST)  {
```

```
                displayed[seq[i]]=0; /* make it ready for display */
                j++;                                    }
        else    {
                displayed[seq[i]]=1;
                Sublist[i]=0;
                }
    display_to=break_point_in_genus;
    num_remained=j;
    In_sublist=0;
    sprintf(tmp, "%d", num_remained);
    panel_set_value(wait_num,tmp);
    sprintf(tmp,"%d",num_of_Items);
    panel_set_value(show_num, tmp);
    next_page();

    }

/* jump to the specified specimen */

extern void
goto_specimen()
    {
    register int    i, j;

    display_from=0;
    j=image_index(control_point[0].x,control_point[0].y);
    num_control_point=displayed[j]=0;
    for (i=0; i<num_of_Items; i++)
        if (seq[i]==j)
            break; /* find it */
    display_to=i;
    num_remained += 1;
    next_page();

    }

/* get a species sublist out of the genus list by clicking on a specimen of the species */

extern void
find_species()
    {
    register int    i,i0,j, k;
    char            tmp[10];

/* locate the specimen */
```

```
        display_from=0;
        j=image_index(control_point[0].x,control_point[0].y);
        num_control_point=0;
        for (i=0; i<num_of_Items; i++)
                if (seq[i]==j) /* find it */
                        break;

        display_to=break_point_in_genus=i;
        for (i=0; i<num_in_page; i++)
                if (current_page[i] == j)
                        break; /* find its index on the current page */

        for (i0=i+1; i0<num_in_page; i0++)
                displayed[current_page[i0]]=MAIN_LIST;

        k=i0=0;
        while (descriptor_value[SYNSPECIES][rel[j]][i0] != '' &&
                descriptor_value[SYNSPECIES][rel[j]][i0])
                i0++;

        descriptor_value[SYNSPECIES][rel[j]][i0]=0;
        for (i=break_point_in_genus; i<num_of_Items; i++)
                if (!strncmp(descriptor_value[SYNSPECIES][rel[seq[i]]],
                                descriptor_value[SYNSPECIES][rel[j]],i0)) {

        /* find all specimens belonging to species descriptor_value[SYNSPECIES][rel[j]] */

                        displayed[seq[i]]=0;
                        Sublist[i]=1;
                        k++;            }
                else if (!displayed[seq[i]])    {
                        displayed[seq[i]]=MAIN_LIST; /* needed for going back to main list */
                        Sublist[i]=0;   }
        In_sublist=1; /* currect display status */
        num_remained=k;
        sprintf(tmp,"%d",k);
        panel_set_value(show_num,tmp);
        next_page();

        }

/* save area in the canvas defined by two corner points between[0] and between[1]
   as image file */

extern void
save_as()
```

```
{
Pixrect *image_out;
int     x0,y0;
char    *filename;
FILE    *fp;

filename = panel_get_value(file_name);
if (filename)   {
        if ((fp=fopen(filename, "w") == NULL);      {
                printf("Can't open file %s\n",filename);
                return;                             }
                x0=y0=0;
                if (between[1].x>between[0].x && between[1].y>between[0].y)     {
                        x0=between[0].x;
                        y0=between[0].y;
                        image_out=mem_create(between[1].x-x0,between[1].y-y0,8);
                        pw_read(image_out,0,0,between[1].x-x0,between[1].y-
                                y0,PIX_SRC,pw,x0,y0); }
                else    { /* save the whole image canvas */
                        image_out=mem_create(canvas_rect.width,
                                                canvas_rect.height,8);

pw_read(image_out,0,0,canvas_rect.width,canvas_rect.height,PIX_SRC,pw,0,0);  }

        pr_dump(image_out,fp,colormap, RT_STANDARD, 1);
        pr_close(image_out);
        fclose(fp);
                }
else
        printf("Please Choose a File Name!");

}
```

10. utility.h

```
/* this is a list of image file directories, data sources with a few plates are lumped into
directory 'others' */

static char     *imageDir[] = {"andrew", "crickmay", "dagis", "donovan", "erben",
                        "frebold","gabilly","geczy","guex","hall","hillebrandt",
                        "hirano", "howarth", "imlay", "jakob", "jaworski",
                        "jennifer", "jozsef", "kalacheva", "meister", "poulton",
                        "quinzio", "repin", "riccardi", "robert", "schlatter",
                        "schleg", "smith", "stankevich", "taylor", "tilmann",
                        "treatise", "westermann", "whiteaves", "white",
```

```
"wiedenmayer","zeiss","McLearn"};
```

11. utility.c

```
#include <ammon.h>
#include <utility.h>

/* specimen may have multiple images which need to be separated before displaying */

extern void
sort_image_list()
        {
        register int    i, k, connect, j=0;
        char            *buf, *buf1;
        char            bufy[30], tmp[30];

        /* image list starts with MAX_NUM-1 */

        for (i=MAX_SP-num_of_Items; i<MAX_SP; i++)  {
                buf1=(char *)strdup(descriptor_value[35][i]);

                if (j>MAX_IMAGE-5)
                        break;

                if (strchr(buf1,'|') != NULL)  { /* multiple images in buf1 separated by '|' */
                        buf=(char *)strtok(buf1,"|");

                /* save the first image into the sorted image list */

                        sprintf(image_name[j++],"%s",buf);
                        rel[j-1]=i;
                        sprintf(bufy,"%s",buf);

                /* find the base of the image file name */

                        if (strchr(bufy,'-') != NULL)  {

                        /* there is a '-' in the image name */

                                connect=1;
                                truncate(bufy,"-");     }

                        else    {

                        /* there is no '-' in the image name */
```

```
                connect=0;
                k=strlen(bufy)-1;
                while (isdigit(bufy[k]) || bufy[k]=='.')
                        k--;
                bufy[k+1]='\0'; }

/* repeat the above process */

        while ((buf=(char *)strtok(NULL,"|")) != NULL) {
                if (connect) {
                        if (isdigit(buf[0]) && strchr(buf,'-') != NULL) {
                                k=strlen(bufy)-1;
                                while (isdigit(bufy[k]))
                                        k--;
                                strncpy(tmp,bufy,k+1);
                                tmp[k+1]='\0';
                                sprintf(image_name[j++],"%s%s",tmp,buf);
                                rel[j-1]=i;
                                sprintf(bufy,"%s",image_name[j-1]);
                                truncate(bufy,"-");
                                                                }
                        else if (isdigit(buf[0])) {
                                sprintf(image_name[j++],"%s-%s",bufy,buf);
                                rel[j-1]=i;
                                                        }
                        else    {
                                k=strlen(bufy)-1;
                                while (isdigit(bufy[k]))
                                        k--;
                                if (bufy[k]=='p')
                                        k--;
                                strncpy(tmp,bufy,k+1);
                                tmp[k+1]='\0';
                                sprintf(image_name[j++],"%s%s",tmp,buf);
                                rel[j-1]=i;
                                if (strchr(buf,'-') != NULL)    {
                                        sprintf(bufy,"%s",image_name[j-1]);
                                        truncate(bufy,"-");     }
                                else {
                                        connect=0;
                                        bufy[++k]='f';
                                        bufy[++k]='\0';  }
                                }
                                } /* end of if (connect) */
                else    {
```

```
                        if (isdigit(buf[0]) && strchr(buf,'-') != NULL) {
                                sprintf(image_name[j++],"%s-%s",bufy,buf);
                                rel[j-1]=i;
                                connect=1;
                                sprintf(bufy,"%s",image_name[j-1]);
                                truncate(bufy,"-");
                        }
                        else if (isdigit(buf[0])) {
                                sprintf(image_name[j++],"%s%s",bufy,buf);
                                rel[j-1]=i;
                                                        }
                        else    {
                                k=strlen(bufy)-1;
                                if (bufy[k]=='p')
                                        k--;
                                strncpy(tmp,bufy,k+1);
                                tmp[k+1]='\0';
                                sprintf(image_name[j++],"%s%s",tmp,buf);
                                rel[j-1]=i;
                                if (strchr(buf,'-') != NULL)    {
                                        connect=1;
                                        sprintf(bufy,"%s",image_name[j-1]);
                                        truncate(bufy,"-");     }
                                else {
                                        if (bufy[k]=='p') {
                                                bufy[++k]='f';
                                                bufy[++k]='\0';}
                                        }
                                }
                        }

                k=0;
                while (image_name[j-1][k] != ' ')
                        ++k;
                image_name[j-1][k]='\0';
                                                }
        else    { /* single image in the name */
                k=0;
                while ((image_name[j][k]=buf1[k]) != ' ')
                        ++k;
                image_name[j][k]='\0';
                rel[j]=i;
                j++;
                }
} /* end of for loop */
```

```
        num_of_Items=j;

        /* prepare for show_descriptor */

        for (i=0; i<NUM_OF_DESCRIPTOR; i++)
                *descriptor_value[i][0]=0;

        /* none of the images have been displayed */

        for (i=0; i<num_of_Items; i++)
                displayed[i]=0;

        }

/* image name from Ammon is relative path, file dir need to be added as prefix */

extern void
build_full_name()
        {
        register int    i, num_of_dir;
        char            tmp[60];

        num_of_dir = sizeof imageDir / sizeof imageDir[0];

        for (i=0; i<num_of_Items; i++)          {
                tmp[0] = 0;
                for (j=0; j<num_of_dir; j++)
                        if (strstr(image_name[i], imageDir[j]))

        sprintf(tmp,"/home/images/%s/%s.ras",imageDir[j],image_name[i]);

                if (!tmp[0])
                        sprintf(tmp,"/home/images/others/%s.ras",image_name[i]);
                sprintf(image_name[i],"%s",tmp);
        }

/* adjust the image brightness by modifying colormap */

extern void
adjust_brightness()
        {
        register int    i, j, k;

        k=(int) panel_get_value(bright);
```

```
        for (i=1; i<252; i++)
                if ((j=cs+k+i)>0 && j<256) /* cs is the brightness of color entry 1 */
                        red[i]=green[i]=blue[i]=j;
                else if (j<0)
                        red[i]=green[i]=blue[i]=0;
                else if (j>255)
                        red[i]=green[i]=blue[i]=255;

        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
}

/* adjust the image brightness by modifying colormap */

extern void
adjust_contrast()
        {
        register int    i, j, k;

        k=(int) panel_get_value(contrast);

        for (i=1; i<252; i++)
                if ((j=cs+k*i/10)>0 && j<256) /* cs is the brightness of entry 1 */
                        red[i]=green[i]=blue[i]=j;
                else if (j<0)
                        red[i]=green[i]=blue[i]=0;
                else if (j>255)
                        red[i]=green[i]=blue[i]=255;

        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
}

/* produce negative of the image */

extern void
reverse()
        {
        register int    i;

        for (i=1; i<252; i++)
                red[i]=green[i]=blue[i]=255-red[i];

        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
```

```
        }

extern void
quit(item, event)
        {
        /* quit without user confirmation */

        if (text_win)
                textsw_reset(text_win,0,0);

        (void)window_set(command_frame, FRAME_NO_CONFIRM, TRUE, 0);
        (void)window_destroy(command_frame);

        (void)window_set(display_window, FRAME_NO_CONFIRM, TRUE, 0);
        (void)window_destroy(display_window);
        }

extern void
help(client, event)
        Notify_client   client;
        Event           *event;
        {

        alert_prompt(
                (Frame)client,
                NULL,
                ALERT_MESSAGE_STRINGS, "To be implemented", 0,
                ALERT_BUTTON_NO, "Cancel",
                ALERT_OPTIONAL, 1,
                ALERT_NO_BEEPING, 1,
                0);

        }

/* convert query statement into plain text */

extern query_to_plain(buf)
        char    *buf;
        {

        buf=substitute(buf,"="," = :");
        buf=substitute(buf,"like","like :");
        buf=substitute(buf,"> "," > :");
        buf=substitute(buf,"< "," < :");
        buf=substitute(buf,"","%");
```

```
        }

/* replace string pattern s2 with s1 in s */

static char *
substitute(s,s1,s2)
        char    *s, *s1, *s2;
        {
        register int      i, j;
        char              tmp[200], *buf, *bufy;

        buf=s;
        i=strlen(buf);
        j=strlen(s2);
        tmp[0]=0;
        while (bufy=(char *)strstr(buf,s2))     {
                strncat(tmp,buf,i-strlen(bufy));
                strcat(tmp,s1);
                buf=&bufy[j];
                i=strlen(buf);   }
        strcat(tmp,buf);
        s=strdup(tmp);

        return s;
        }


static char *
truncate(s1,s2)
        char    *s1, *s2;
{
        register int      i;

        i=0;
        while (s1[i++] != s2[0])
                ;
        s1[--i]='\0';

        return s1;
}

/* if lots of specimens meet the search condition, it will take a long time to complete
   the query, if you want to quit before it finishes simply press the STOP key */

extern void *
stop_op()
```

```
{
exit (0);
}
```

12. win.h

```
/* button locations */

#define COMMAND_START_ROW       12
#define COMMAND_INCREMENT       2

/* introductory text locations */

#define X_START             140
#define Y_START             90
#define Y_INCREMENT     25

static void     adjust_info_panel(Panel local_panel);
```

13. win.c

```
/* windows, buttons related functions */

#include <ammon.h>
#include <icon.h>
#include <win.h>

/* introductory text locations */

#define X_START 140
#define Y_START 90
#define Y_INCREMENT 25


extern void
create_command_button(frame_local)
        Frame   frame_local;
        {
        panel_create_item(frame_local,PANEL_BUTTON,
                PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW),
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_NOTIFY_PROC, query_from_text,
                PANEL_LABEL_IMAGE, panel_button_image
                (frame_local,"Combination",13,font),
```

```
            0);

        panel_create_item(frame_local,PANEL_MESSAGE,
            PANEL_ITEM_Y,
ATTR_ROW(COMMAND_START_ROW+COMMAND_INCREMENT),
            PANEL_LABEL_FONT, font,
            PANEL_LABEL_STRING,"Some Utility:",
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +2*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, next_page,
            PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Next Page",13,font),
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +3*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, previous_page,
            PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Previous Page",13,font),
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +4*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, goto_specimen,
            PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Goto Specimen",13,font),
            0);

        panel_create_item(frame_local,PANEL_TEXT,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +5*COMMAND_INCREMENT),
            PANEL_LABEL_STRING, "Brightness:",
            0);

        bright=panel_create_item(frame_local,PANEL_SLIDER,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +6*COMMAND_INCREMENT+1),
            PANEL_SHOW_VALUE, FALSE,
```

```
        PANEL_VALUE, 0,
        PANEL_MIN_VALUE, -50,
        PANEL_MAX_VALUE, 50,
        PANEL_SLIDER_WIDTH, 94,
        PANEL_NOTIFY_PROC, adjust_brightness,
        0);

panel_create_item(frame_local,PANEL_TEXT,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +7*COMMAND_INCREMENT+1),
        PANEL_LABEL_STRING, "Contrast:",
        0);

contrast=panel_create_item(frame_local,PANEL_SLIDER,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +8*COMMAND_INCREMENT),
        PANEL_SHOW_VALUE, FALSE,
        PANEL_VALUE, 10,
        PANEL_MIN_VALUE, 1,
        PANEL_MAX_VALUE, 20,
        PANEL_SLIDER_WIDTH, 94,
        PANEL_NOTIFY_PROC, adjust_contrast,
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +9*COMMAND_INCREMENT+1),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, main_list,
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Main List",13,font),
        0);

panel_create_item(frame_local, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                +10*COMMAND_INCREMENT+1),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, find_species,
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Speci Sublist",13,font),
        0);


panel_create_item(frame_local,PANEL_BUTTON,
```

```
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                    +11*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, reverse,
            PANEL_LABEL_IMAGE, panel_button_image
                                    (frame_local,"Reverse Image",13,font),
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                    +12*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, save_as,
            PANEL_LABEL_IMAGE, panel_button_image
                                    (frame_local,"Save Image As",13,font),
            0);

        file_name=panel_create_item(frame_local,PANEL_TEXT,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                    +13*COMMAND_INCREMENT),
            PANEL_LABEL_FONT, font,
            PANEL_LABEL_STRING, "File:",
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                    +14*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, help,
            PANEL_LABEL_IMAGE,panel_button_image(frame_local,"Help",4,font),
            0);

        button_Quit = panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW
                                    +14*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(12),
            PANEL_NOTIFY_PROC, quit,
            PANEL_LABEL_IMAGE,panel_button_image(frame_local,"Quit",4,font),
            0);

        }

extern void
init_info_panel(frame_local)
        Frame  frame_local;
        {
```

```
/* font for displaying title */

font=pf_open("/apps/sundesk/fonts/sunpaint/R.24");

/* create info window to display the introduction text */

info_panel=window_create(frame_local, PANEL,
        PANEL_LABEL_BOLD, FALSE,
        WIN_X, 0,
        WIN_Y, 0,
        WIN_WIDTH, screen_rect.width-command_rect.width,
        WIN_HEIGHT, info_rect.height,
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START-20,
        PANEL_ITEM_Y, Y_START-40,
        PANEL_LABEL_STRING, "WELCOME      TO    JURASSIC
                                AMMONITE    IMAGE    DATABASE",
        0);

/* font for displaying introductory text */

font=pf_open("/apps/sundesk/fonts/sunview/lucida-19.v");

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START+80,
        PANEL_ITEM_Y, Y_START,
        PANEL_LABEL_STRING, "You can retrieve Jurassic ammonites by:",
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START,
        PANEL_ITEM_Y, Y_START+Y_INCREMENT,
        PANEL_LABEL_STRING, "1. Taxonomy: Family, Genus, and Species.",
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START,
        PANEL_ITEM_Y, Y_START+2*Y_INCREMENT,
        PANEL_LABEL_STRING, "2. Stratigraphy: Stage and Standard Zone.",
```

```
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START,
            PANEL_ITEM_Y,Y_START+3*Y_INCREMENT,
            PANEL_LABEL_STRING, "3. Geography: Geographic Area and Country.",
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START,
            PANEL_ITEM_Y, Y_START+4*Y_INCREMENT,
            PANEL_LABEL_STRING, "4. Morphology: Volution, Expansion Rate,
                                                Whorl Shape, and Rib Form.",
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START,
            PANEL_ITEM_Y, Y_START+5*Y_INCREMENT,
            PANEL_LABEL_STRING, "5. Reference: in which the figure of
                                                the specimen appears.",
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START+40,
            PANEL_ITEM_Y, Y_START+6*Y_INCREMENT,
            PANEL_LABEL_STRING, "You can also retrieve by combining
                                                the above criteria. ",
            0);

        }

static  void
adjust_info_panel(local_panel)
        Panel local_panel
        {
        int             i;
        char            tmp[40], *buf;
        Panel_item      item;

        /* destroy all old panel items in the info panel */

        panel_each_item(local_panel, item)
```

```
        panel_destroy_item(item);
panel_end_each

info_rect.height = screen_rect.height-text_rect.height-canvas_rect.height;
info_rect.width = screen_rect.width-command_rect.width;
window_set(local_panel,
        WIN_HEIGHT, info_rect.height,
        WIN_WIDTH,  info_rect.width,
        0);

/* display the query statement of the menu selection */

if (num_of_Items)       {

/* translate from SQL query to plain text, eliminate ':' */

/* skip 'select * from  OPS$liang.ammon_image where' in query statement */

        buf=&query_statement[42];
        query_to_plain(buf);
        i=strlen(buf);
        buf[i-2]=0;
        tmp[0]=0;
        sprintf(tmp,"%s%s",buf,search_condition[0]);
                        }
else
        tmp[0]=0;

/* condition_text is the text field for query text display or input, you can
  type in a query and press the Combination button to search the database */

condition_text = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 180,
        PANEL_VALUE_DISPLAY_LENGTH, 200,
        PANEL_LABEL_X, ATTR_COL(0),
        PANEL_LABEL_Y, ATTR_ROW(0),
        PANEL_LABEL_STRING, "Search :",
        PANEL_VALUE, tmp,
        0);

/* display number of images retrieved */

if (num_of_Items)
        sprintf(tmp,"%4d",num_of_Items);

show_num = panel_create_item(local_panel, PANEL_TEXT,
```

```
            PANEL_VALUE_DISPLAY_LENGTH, 5,
            PANEL_LABEL_X, ATTR_COL(0),
            PANEL_LABEL_Y, ATTR_ROW(1),
            PANEL_LABEL_STRING, "Images Returned:",
            PANEL_VALUE, tmp,
            0);

        /* number of images wait to be displayed */

        wait_num =
            panel_create_item(local_panel, PANEL_TEXT,
            PANEL_VALUE_DISPLAY_LENGTH, 5,
            PANEL_LABEL_X, ATTR_COL(22),
            PANEL_LABEL_Y, ATTR_ROW(1),
            PANEL_LABEL_STRING, "Remained:",
            0);


        panel_paint(local_panel, PANEL_CLEAR);

}

extern void
adjust_display_window(local_panel)
        Panel   local_panel;
        {

        start_status=1;/* now the system has gone through at least one query */

        adjust_info_panel();

        /* adjust image_canvas size due to the creation of a new text window */

        window_set(image_canvas,
            WIN_X, 0,
            WIN_Y, screen_rect.height-canvas_rect.height,
            WIN_WIDTH, canvas_rect.width,
            WIN_HEIGHT, canvas_rect.height,
            0);

        text_win=window_create(display_window, TEXTSW,
            WIN_X, 0,
            WIN_Y, screen_rect.height-canvas_rect.height-text_rect.height,
            WIN_WIDTH, text_rect.width,
            WIN_HEIGHT, text_rect.height,
            0);
```

```
}

extern void canvas_handler(event)
        Event  *event;
        {
        register int      i, j;

        if (event_is_up(event))
                return;
        switch (event_action(event)) {
                case MS_LEFT: /* locate the image and display related info */
                i=image_index(event_x(event),event_y(event));
                if (i>=0)
                        show_descriptors(i);
                break;
        case MS_MIDDLE:
        /* mark the point with color 252 and determine the image scale */
                pw_put(pw,event_x(event),event_y(event),252);
                control_point[num_control_point].x=event_x(event);
                control_point[num_control_point].y=event_y(event);
                if (!num_control_point)         {
                        image_scale=atof(descriptor_value[SCALE][rel[image_index
                                                (event_x(event),event_y(event))]]);
                        if (!image_scale)
                                image_scale=1.0;        }
                num_control_point++;
        case MS_RIGHT: /* mark two points for ploting brightness profile */
                if (between[1].x || between[1].y)       {
                        i=between[1].x;
                        j=between[1].y;
                        between[1].x=event_x(event);
                        between[1].y=event_y(event);
                        between[0].x=i;
                        between[0].y=j;         }
                else if (between[0].x || between[0].y)  {
                        between[1].x=event_x(event);
                        between[1].y=event_y(event);    }
                else    {
                        between[0].x=event_x(event);
                        between[0].y=event_y(event);}
                break;
        default:
                break;
        }
}
```

```
extern void
clear_canvas()
        {
        (void)pw_writebackground(pw,0,0,(int)(LINT_CAST(window_get(image_canvas,
                                        CANVAS_WIDTH))),
        (int)(LINT_CAST(window_get(image_canvas, CANVAS_HEIGHT))), PIX_CLR);
        }
```

Appendix 3 Source code for program imageEdit

There are 5 files in this program: imagedit.h, main.c, utility.c, file.c, modify.c.

1. imagedit.h

```
/* header file for imageEdit
 *
 * Author: Bo Liang
 * Department of Geological Sciences
 * The University of British Columbia
 * Date: May 1991
 * Modifications: April 1994
 */

#include <stdio.h>
#include <math.h>
#include <suntool/tty.h>
#include <sys/types.h>
#include <sunwindow/attr.h>
#include <sunwindow/defaults.h>
#include <suntool/seln.h>
#include <suntool/expand_name.h>
#include <suntool/alert.h>
#include <suntool/frame.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/canvas.h>
#include <suntool/walkmenu.h>
#include <suntool/scrollbar.h>

/* maximum number of characters in the file path name */

#define MAX_PATH_LEN    1024
#define Pi              3.14159
#define MIN_WIDTH 700   /* minimum width of the window */
#define MAX_IMAGE       3000   /* maximum size of the image */

#define LOCATE    0     /* locate the area of interest */
#define CLEAN     1     /* clean the image */

Frame       base_frame;
Canvas      canvas;
Pixwin      *pw;

/* indicate the amount of change wanted */
```

```
Panel_item      rotate, bright, contrast;

/* clean or locate the image of interest */

Panel_item      status;

/* color used to erase noise or mark the border of the image */

Panel_item      border_color, eraser_color, eraser_width;

/* file name and directory */

Panel_item      dir_item, fname_item;

int             w, h, bpp;      /* image width, height and depth */

/* size of the base frame and canvas */

struct size {
        int width;
        int height;
}       frame_size = {700, 900};
struct size     canvas_size = {800, 1000};

unsigned char red [256], green [256], blue [256];
colormap_t      colormap = { RMT_EQUAL_RGB, 256, red, green, blue};

int             x[2], y[2]; /* corner coords */
int             edge[4][MAX_IMAGE]; /* buffer for old data of the four sides */
float           aver, var, fct[2];
float           *a; /* array to hold raw data of the image */

/* panel notify procs */

void    load_image();
void    save_image();
void    quit();
void    rotate_proc();
void    rightness_proc();
void    contrast_proc();
void    image_edit(Event*, short, short, short);
void    init_data();
void    reset_image();

/* canvas procs */
```

```
void    modify_canvas();
void    clear_canvas();
void    handle_event();
```

2. main.c

```
/* main entry for imageEdit */

#include "imagedit.h"

main(argc, argv)
        int argc;
        char **argv;
{
        int     i;
        void    init_panel(), init_canvas();

        /* create a base frame to hold the panel and image canvas. */

        base_frame =
                window_create((Window) 0, FRAME,
                        FRAME_LABEL, "INTERACTIVE IMAGE EDITOR",
                        WIN_WIDTH, frame_size.width,
                        WIN_HEIGHT, frame_size.height,
                        FRAME_ICON, icon_create(ICON_IMAGE, &canvas_demo_pr, 0),
                        FRAME_NO_CONFIRM, TRUE,
                        FRAME_ARGS, argc, argv,
                        0);

        /* initialize the info and command panel. */

        init_panel();

        init_canvas();

        window_main_loop(base_frame);

        exit(0);

        }


static void
init_panel()
{
```

```
Panel panel;
char current_dir[MAX_PATH_LEN];

panel = window_create(base_frame, PANEL, PANEL_LABEL_BOLD, TRUE, 0);

bright =
        panel_create_item(panel, PANEL_SLIDER,
                PANEL_LABEL_STRING, "Brightness Adjustment: ",
                PANEL_SLIDER_WIDTH, 100,
                PANEL_VALUE, 50,
                PANEL_MIN_VALUE, 0,
                PANEL_MAX_VALUE, 100,
                PANEL_NOTIFY_PROC, brightness_proc,
                0);

contrast =
        panel_create_item(panel, PANEL_SLIDER,
                PANEL_LABEL_STRING, "Contrast Adjustment:  ",
                PANEL_SLIDER_WIDTH, 100,
                PANEL_VALUE, 50,
                PANEL_MIN_VALUE, 0,
                PANEL_MAX_VALUE, 100,
                PANEL_NOTIFY_PROC, contrast_proc,
                0);

rotate =
        panel_create_item(panel, PANEL_SLIDER,
                PANEL_LABEL_STRING, "Rotate image:  ",
                PANEL_SLIDER_WIDTH, 100,
                PANEL_VALUE, 0,
                PANEL_MIN_VALUE, -90,
                PANEL_MAX_VALUE, 90,
                PANEL_NOTIFY_PROC, rotate_proc,
                0);

status =
        panel_create_item(panel, PANEL_CHOICE,
                PANEL_LABEL_X, ATTR_COL(0),
                PANEL_LABEL_Y, ATTR_ROW(3),
                PANEL_LABEL_STRING, "Edit Options:",
                PANEL_CHOICE_STRINGS, "Locate image", "Erase noise", 0,
                0);
border_color =
        panel_create_item(panel, PANEL_TEXT,
                PANEL_LABEL_X, ATTR_COL(0),
                PANEL_LABEL_Y, ATTR_ROW(4),
```

```
                PANEL_VALUE_DISPLAY_LENGTH, 3,
                PANEL_LABEL_STRING, "Border color:",
                0);

eraser_color =
        panel_create_item(panel, PANEL_TEXT,
                PANEL_LABEL_X, ATTR_COL(19),
                PANEL_LABEL_Y, ATTR_ROW(4),
                PANEL_VALUE_DISPLAY_LENGTH, 3,
                PANEL_LABEL_STRING, "Eraser color:",
                0);

eraser_width =
        panel_create_item(panel, PANEL_TEXT,
                PANEL_LABEL_X, ATTR_COL(36),
                PANEL_LABEL_Y, ATTR_ROW(4),
                PANEL_VALUE_DISPLAY_LENGTH, 3,
                PANEL_LABEL_STRING, "Eraser width:",
                0);

dir_item =
         panel_create_item(panel, PANEL_TEXT,
                        PANEL_LABEL_X, ATTR_COL(55),
                        PANEL_LABEL_Y, ATTR_ROW(0),
                        PANEL_VALUE_DISPLAY_LENGTH, 23,
                        PANEL_VALUE, getwd(current_dir),
                        PANEL_LABEL_STRING, "Dir: ",
                        0);

fname_item =
        panel_create_item(panel, PANEL_TEXT,
                        PANEL_LABEL_X, ATTR_COL(55),
                        PANEL_LABEL_Y, ATTR_ROW(1),
                        PANEL_VALUE_DISPLAY_LENGTH, 23,
                        PANEL_LABEL_STRING, "File:",
                        0);

(void) panel_create_item(panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(55),
        PANEL_LABEL_Y, ATTR_ROW(2),
        PANEL_LABEL_IMAGE, panel_button_image(panel, "Load", 0, 0),
        PANEL_NOTIFY_PROC, load_image,
        0);

(void) panel_create_item(panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(62),
```

```
            PANEL_LABEL_Y, ATTR_ROW(2),
            PANEL_LABEL_IMAGE, panel_button_image(panel, "Save", 0, 0),
            PANEL_NOTIFY_PROC, save_image,
            0);

        (void) panel_create_item(panel, PANEL_BUTTON,
            PANEL_LABEL_X, ATTR_COL(69),
            PANEL_LABEL_Y, ATTR_ROW(2),
            PANEL_LABEL_IMAGE, panel_button_image(panel, "Quit", 0, 0),
            PANEL_NOTIFY_PROC, quit,
            0);

        (void) panel_create_item(panel, PANEL_BUTTON,
            PANEL_LABEL_X, ATTR_COL(55),
            PANEL_LABEL_Y, ATTR_ROW(4),
            PANEL_LABEL_IMAGE, panel_button_image(panel, "Reset", 0, 0),
            PANEL_NOTIFY_PROC, reset_image,
            0);

        (void) panel_create_item(panel, PANEL_BUTTON,
            PANEL_LABEL_X, ATTR_COL(69),
            PANEL_LABEL_Y, ATTR_ROW(4),
            PANEL_LABEL_IMAGE, panel_button_image(panel, "Clear", 0, 0),
            PANEL_NOTIFY_PROC, clear_canvas,
            0);

        window_set(panel, PANEL_CARET_ITEM, fname_item, 0);

        (void) window_fit_height(panel);

        }

static void
init_canvas()
        {
        Scrollbar       vertical_sb, horizontal_sb;

        /* loaded image may be larger than the window size, need a scrollable canvas. */

        vertical_sb = scrollbar_create(SCROLL_LINE_HEIGHT, 5, (char *) 0);
        horizontal_sb = scrollbar_create(SCROLL_LINE_HEIGHT, 5, (char *) 0);

        canvas =
            window_create(base_frame, CANVAS,
                WIN_VERTICAL_SCROLLBAR, vertical_sb,
                WIN_HORIZONTAL_SCROLLBAR, horizontal_sb,
```

```
                CANVAS_RETAINED, FALSE,     /* necessary for the following
                                             * modification */
                CANVAS_AUTO_SHRINK, FALSE,
                WIN_EVENT_PROC, handle_event,
                CANVAS_FIXED_IMAGE, FALSE,
                CANVAS_WIDTH, canvas_size.width,
                CANVAS_HEIGHT, canvas_size.height,
                0);

        window_set(canvas, WIN_CONSUME_PICK_EVENT, LOC_TRAJECTORY, 0);

        /* declare the canvas has 256 colors */

        pw = (Pixwin *) window_get(canvas, WIN_PIXWIN);
        pw_setcmsname(pw, "rainbow");
        pw_putcolormap(pw, 0, 256, red, green, blue);

        /* make it a color, retained canvas. */

        window_set(canvas,
                CANVAS_RETAINED, TRUE,
                CANVAS_AUTO_CLEAR, TRUE,
                WIN_VERTICAL_SCROLLBAR, vertical_sb,
                WIN_HORIZONTAL_SCROLLBAR, horizontal_sb,
                0);
        }
```

3. utility.c

```
/* responses to mouse event */

#include "imagedit.h"

extern void
handle_event(canvas_local, event)
        Canvas canvas_local;
        Event          *event;
        {
        short          upper_corner = FALSE;
        short          lower_corner = FALSE;
        short          erase = FALSE;

        if (event_is_up(event))
                return;
        switch (event_action(event))  {
                case MS_LEFT:
```

```
                upper_corner = TRUE;
                break;
        case MS_MIDDLE:
                erase = TRUE;
                break;
        case MS_RIGHT:
                lower_corner = TRUE;
                break;
        case LOC_TRAJECTORY:
                if (window_get(canvas_local, WIN_EVENT_STATE,MS_MIDDLE))
                erase = TRUE;
                break;
        default:
                break;          }

    image_edit(event, lower_corner, upper_corner, erase);

    }


extern void
modify_canvas(width, height)
    int     width, height;
    {

    if (width<MIN_WIDTH)
        width=MIN_WIDTH;

    (void)window_set(canvas,
                CANVAS_WIDTH, width+20, /* 20 - margin */
                CANVAS_HEIGHT, height+20,
                0);

}

extern void
clear_canvas()
    {
    (void)pw_writebackground(pw,0,0,(int)(LINT_CAST(window_get(canvas,
CANVAS_WIDTH))),
    (int)(LINT_CAST(window_get(canvas, CANVAS_HEIGHT))), PIX_CLR);
    }

extern void
quit()
    {
```

```
        /* quit without user confirmation */

        (void)window_set(base_frame, FRAME_NO_CONFIRM, TRUE, 0);
        (void)window_destroy(base_frame);

        }
```

4. file.c

```
#include "imagedit.h"

/* load the specified image, and initialize parameters */

static void
load_image()
        {
        Pixrect *image_orig;
        FILE    *fp;
        char    *filename;

        if (!strlen(filename = panel_get_value(fname_item)))
                {
                printf("Please specify a file name  to load !");
                return;
                }

        clear_canvas();

        x[0]=x[1]=y[0]=y[1]=0; /* reset the corner coordinates */

        var=aver=0.0; /* initialization for the average and variance */

        if ((fp=fopen(filename,"r"))==NULL)
        {
                printf("Can't open this file !");
                return;
        }

        /* read image attribute */

        image_orig=pr_load(fp, colormap);
        w=image_orig->pr_size.x;
        h=image_orig->pr_size.y;
        bpp=image_orig->pr_depth;
```

```
        modify_canvas(w,h); /* adjust canvas size */

        panel_set_value(status,0); /* default status : locate area of interest */

        /* let color entry 255 be green, easily visible in a grey scale image */ */
        green[255]=255;
        red[255]=blue[255]=0;
        pw_setcmsname(pw, "rainbow");
        pw_putcolormap(pw, 0, 256, red, green, blue);

        /* copy the image to window pw */

        pw_rop(pw,10,10,w,h,PIX_SRC,image_orig,0,0);
        pr_close(image_orig);
        fclose(fp);
        }


        /* save the specified area of pw as an image file */

static void
save_image()
        {
        Pixrect *image_mody;
        int     i, j, k;
        char    *filename;
        FILE    *fp;

        filename = panel_get_value(fname_item); /* get the file name */

        if ((fp=fopen(filename, "w")) == NULL)
        {
                printf("Can't open this file !");
                return;
        }


        if (!x[1] && !y[1])     /* if no specified area of interest, save the whole pw */
        {
                x[0]=y[0]=0;
                x[1]=w;
                y[1]=h;
        }

        w=x[1]-x[0]-1;
        h=y[1]-y[0]-1;
```

```
        image_mody=mem_create(w,h,bpp);

        /* change the color map back to normal, entry 255 is white now */

        red[255]=green[255]=blue[255]=255;
        if (bpp>1)
        {
                /* x[0]+1,y[0]+1 help to skip the color border */

                pw_read(image_mody,0,0,w,h,PIX_SRC,pw,x[0]+1,y[0]+1);
                pr_dump(image_mody,fp,&colormap, RT_BYTE_ENCODED, 1);
        }
        else /* black and white image */
        {
                for (i=0; i<h; i++)
                        for (j=0; j<w; j++)     {
                                k = pw_get(pw,x[0]+1+j,y[0]+1+i);
                                if (k==255)
                                        pr_put(image_mody, j, i, 1);
                                                }
                pr_dump(image_mody,fp,NULL,RT_BYTE_ENCODED, 1);
        }
        pr_close(image_mody);
        clear_canvas();
        fclose(fp);

        }

static void
reset_image()
        {
        /* simply reload the image */

        load_image();
        }
```

5. modify.c

```
/* image edit related functions */

#include "imagedit.h"

/* rotate the image around the center by angle */

extern void
rotate_proc()
```

```
{
Pixrect *image_mody;
register int    i, j, k, w1, w0, h1, h0,
register int    xmin, xmax, ymin, ymax, xx[6], yy[6];
double          angle;

k = (int) panel_get_value(rotate);
if (k < 90) {
        angle = (double) -PI * k / 180; /* convert to radius */
        xx[0] = w / 2;
        yy[0] = h / 2;
        xmin = ymin = MAX_IMAGE;
        xmax = ymax = 0;

/* compute the coordinates of the four corners after rotation */

        xx[1] = -w / 2 * cos(angle) - h / 2 * sin(angle);
        yy[1] = w / 2 * sin(angle) - h / 2 * cos(angle);
        xx[2] = w / 2 * cos(angle) - h / 2 * sin(angle);
        yy[2] = -w / 2 * sin(angle) - h / 2 * cos(angle);
        xx[3] = -w / 2 * cos(angle) + h / 2 * sin(angle);
        yy[3] = w / 2 * sin(angle) + h / 2 * cos(angle);
        xx[4] = w / 2 * cos(angle) + h / 2 * sin(angle);
        yy[4] = -w / 2 * sin(angle) + h / 2 * cos(angle);

/* compute the coordinates of the four new corners */

        for (i = 1; i < 5; i++) {
                if (xx[i] < xmin)
                        xmin = xx[i];
                else if (xx[i] > xmax)
                        xmax = xx[i];
                if (yy[i] < ymin)
                        ymin = yy[i];
                else if (yy[i] > ymax)
                        ymax = yy[i];
        }
        w1 = xmax - xmin + 1;           /* width of the rotated image */
        h1 = ymax - ymin + 1; /* height of the rotated image */
        w0 = w1 / 2;
        h0 = h1 / 2;
        image_mody = mem_create(w1, h1, bpp);

        /* locate the coord. in the orig. image for each point in the new one */
        for (i = 0; i < w1; i++)
                for (j = 0; j < h1; j++) {
```

```
                        xx[5] = (i - w0) * cos(angle) - (j - h0) * sin(angle) + xx[0];
                        yy[5] = (i - w0) * sin(angle) + (j - h0) * cos(angle) + yy[0];
                        if (xx[5] >= 0 && xx[5] < w && yy[5] >= 0 && yy[5] < h)
                                k = pw_get(pw, xx[5], yy[5]);
                        else
                                k = 254;
                        pr_put(image_mody, i, j, k);
                }
        w = w1;
        h = h1;
    }
    else    { /* exactly 90 degree */
        i = h;
        h = w;
        w = i;
        image_mody = mem_create(w, h, bpp);
        for (i = 0; i < w; i++)
                for (j = 0; j < h; j++)
                        pr_put(image_mody, i, j, pw_get(pw, j, w - i));
        }

    modify_canvas(w, h);
    clear_canvas();
    pw_rop(pw, 0, 0, w, h, PIX_SRC, image_mody, 0, 0);
    pr_close(image_mody);

    }

extern void
brightness_proc()
    {
    register int    i, j, k;
    float           tmp;

    if (!aver)      /* the first time */
            init_data();
    k = (int) panel_get_value(bright);

    /*
     * increase the average brightness without change the contrast (variance)
     * it is much slower than simply change color map but offer some nice effect
     */

    fct[0] = k / 50.0; /* brightness scale factor */
    tmp = fct[0] * aver;
    tmp -= fct[1] * aver; /* fct[1] - contrast scale factor */
```

```
        for (i = 0; i < w; i++)
                for (j = 0; j < h; j++) {
                        k = fct[1] * a[i * h + j] + tmp;
                        if (k > 254)
                                k = 254; /* entry 255 is reserved */
                        else if (k < 1)
                                k = 1;
                        pw_put(pw, i, j, k);
                }

}

extern void
contrast_proc()
        {
        register int    i, j, k;
        float           tmp;

        if (!var) /* the first time */
                init_data();
        k = (int) panel_get_value(contrast);
        fct[1] = k / 50.0;
        tmp = (fct[0] - fct[1]) * aver;
        for (i = 0; i < w; i++)
                for (j = 0; j < h; j++) {
                        k = fct[1] * a[i * h + j] + tmp;
                        if (k > 254)
                                k = 254;
                        else if (k < 1)
                                k = 1;
                        pw_put(pw, i, j, k);
                }

        }

/* compute the average brightness and contrast (variance) for the image */

static void
init_data()
        {
        register int    i, j, k;
        float           tmp;

        if (a != NULL)
                free(a);
```

```
        /* allocate memory to hold the raw image data */

        a = (float *) malloc(w * h * sizeof(float));

        fct[0] = fct[1] = 1.0;
        for (i = 0; i < w; i++)
                for (j = 0; j < h; j++) {
                        a[i * h + j] = pw_get(pw, i, j);
                        aver += a[i * h + j];
                        var += sqr(a[i * h + j]);
                }
        tmp = aver;
        k = w * h;
        aver /= k;
        var = var + k * sqr(aver) - 2 * aver * tmp;
        var = sqrt(var / (k - 1));

        }

/* erase noise in the image or locate area of interest to save */

extern void
image_edit(event, lower_corner, upper_corner, erase)
        Event  *event;
        short  lower_corner, upper_corner, erase;
        {
        register int    i, color_index;
        long            op;

        brush = (struct pr_brush *) malloc(4);
        if (status == CLEAN) {
                brush->width = atoi(panel_get_value(eraser_width));
                if (brush->width < 0 || brush->width > w)
                        brush->width = w / 10; /* actually it is the height of the brush */
                brush_wid = w / 20; /* real width of the brush, fixed for each image */

                color_index = atoi(panel_get_value(eraser_color));
        }
        else
                color_index = atoi(panel_get_value(border_color));

        if (color_index < 0)
                color_index = 0;
        else if (color_index > 255)
                color_index = 255;
        if (bpp > 1)
```

```
        op = PIX_SRC | PIX_COLOR(color_index);
else
        op = PIX_SRC | PIX_COLOR(0); /* use bg color */

/* take proper action based on mouse event */

if (status == CLEAN && erase)
        (void) pw_line(pw, event_x(event) - brush_wid, event_y(event),
                event_x(event) + brush_wid, event_y(event), brush, NULL, op);
else if (status == LOCATE && upper_corner)      {
        if (!x[0] && !y[0]) { /* the first time */
                x[0] = event_x(event);
                y[0] = event_y(event);

        /* save original border data in edge */

                for (i = 0; i < h  - y[0]; i++)
                        edge[0][i] = pw_get(pw, x[0], y[0] + i);
                for (i = 0; i < w  - x[0]; i++)
                        edge[1][i] = pw_get(pw, x[0] + i, y[0]);

        /* draw the upper border using the color of color_index */

                (void) pw_vector(pw, x[0], y[0], x[0], h, op, 1);
                (void) pw_vector(pw, x[0], y[0], w, y[0], op, 1);

        } else {

        /* undo the old border before drawing a new one */

                for (i = 0; i < h - y[0]; i++)
                        pw_put(pw, x[0], y[0] + i, edge[0][i]);
                for (i = 0; i < w - x[0]; i++)
                        pw_put(pw, x[0] + i, y[0], edge[1][i]);

                x[0] = event_x(event);
                y[0] = event_y(event);
                for (i = 0; i < h - y[0]; i++)
                        edge[0][i] = pw_get(pw, x[0], y[0] + i);
                for (i = 0; i < w - x[0]; i++)
                        edge[1][i] = pw_get(pw, x[0] + i, y[0]);

        /* draw a new upper border using the color of color_index */

                (void) pw_vector(pw, x[0], y[0], x[0], h, op, 1);
                (void) pw_vector(pw, x[0], y[0], w, y[0], op, 1);
```

```
        }                                                    }
else if (status == LOCATE && lower_corner) {
        if (!x[1] && !y[1]) { /* the first time */
                x[1] = event_x(event);
                y[1] = event_y(event);

        /* save original border data in edge */

                for (i = 0; i < y[1]; i++)
                        edge[2][i] = pw_get(pw, x[1], i);
                for (i = 0; i < x[1]; i++)
                        edge[3][i] = pw_get(pw, i, y[1]);

        /* draw the lower border using the color of color_index */

                (void) pw_vector(pw, x[1], y[1], x[1], 0, op, 1);
                (void) pw_vector(pw, x[1], y[1], 0, y[1], op, 1);
        } else {
                for (i = 0; i < y[1]; i++)
                        pw_put(pw, x[1], i, edge[2][i]);
                for (i = 0; i < x[1]; i++)
                        pw_put(pw, i, y[1], edge[3][i]);
                x[1] = event_x(event);
                y[1] = event_y(event);
                for (i = 0; i < y[1]; i++)
                        edge[2][i] = pw_get(pw, x[1], i);
                for (i = 0; i < x[1]; i++)
                        edge[3][i] = pw_get(pw, i, y[1]);
                (void) pw_vector(pw, x[1], y[1], x[1], 0, op, 1);
                (void) pw_vector(pw, x[1], y[1], 0, y[1], op, 1);
                }
        }
}
```

Appendix 4 Source code for the interactive image measurement module

There are 15 files in this program: ammon.h, amenu.h, amenu.c, display.c, query.c, utility.h, utility.c, morph.h, morph.c, icon.h, measure.c, dbms.c, misc.c, win.h, win.c:

1. ammon.h
same as in Appendix 2.
2. amenu.h
same as in Appendix 2.
3. amenu.c
same as in Appendix 2.
4. display.c
same as in Appendix 2.
5. query.c
same as in Appendix 2.
6. utility.h
same as in Appendix 2.
7. utility.c
same as in Appendix 2.
8. morph.h

```
/* data , function, and structure declarations for image measurement module */

#define PIXEL_CM   39.0783 /* the number of screen pixels per centimeter */
#define Pi         3.141596

/* define the array index for descriptor_name */

#define DMAX          37
#define D             38
#define BISPACE       50
#define WWWH          46
#define WH            42
#define WHD           43
#define VOLUTION0     67
#define UD            39

/* the labels of icon items */

static char*   icon_list[]={"biconcave", "concave", "sinuous","falcate", "projected",
                  "straight","bundled", "falcoid", "zigzag", "intercalatory",
                  "bi", "tri", "multi", "poly","loop2", "loop3","loopn","angular",
                  "round", "strong","weak", "prorsiradiate", "gprorsiradiate",
                  "rectiradiate","grursiradiate", "rursiradiate", "uni", "bi",
                  "tri", "multi", "clavate", "bullate","high", "low", "shallow",
                  "steep","vertical", "undercut", "yes", "rounded","ellipsoid",
```

```
                    "wellipsoid", "oval","lanceolate", "ogival", "rectangular",
                    "wrectangular", "quadrate", "triangular","wtriangular",
                    "trapezoid", "coronate","plain", "carinate","sulcate",
                    "carinate-sulcate", "bicarinate","bicarinate-sulcate","yes"};

/* the conversion table of icon_list[] to descriptor_name[] */

int         icon_to_descriptor[]={59,59,59,59,59,59,59,59,59,59,61,61,61,61,61,61,
                                  61, 60,60,60,60, 58,58,58,58,58, 62,62,62,62,
                                  62,62,55,55, 56,56,56,56, 63, 54,54,54,54,54,54,
                                  54,54,54,54,54,54,54, 57,57,57,57,57,57,70};

/* the conversion table of text fields in the info panel to descriptor_name[] */

int         text_to_descriptor[]={43,68,42,64,49,50,51,65,52,66,67,47,53};

#define     NUM_OF_TEXT     sizeof text_to_descriptor / sizeof text_to_descriptor[0]
#define     NUM_OF_ICON     sizeof icon_list / sizeof icon_list[0];

/* text window size which is different from the one in retrieve.c, the new info panel
   size can be computed from command window size and text window size */

struct size     {
                int     width;
                int     height;
                }       text_rect = {263, 475};

struct points   {
        int     x;
        int     y;
        int     z;      /* different from definition for point */
        }       curve_points[10000];

int     num_point; /* number of points in the array curve_points *

Panel_item      inputIcon_panel, text_item[NUM_OF_TEXT], figure_num;
Panel_item      parameter_info; /* warning information */
Panel_item      num_of_whorls; /* the number of whorls separating P1 and P2 */

/* image measurement related functions */

extern float    distance();
extern void     bi_space();
extern void     d_max();
extern void     finess_ratio();
extern void     furc_pos();
```

```
extern void     max_wh();
extern void     rib_density();
extern void     rib_width();
extern float    ratio();
extern float    ratio_of_three();
extern void     simulate_coiling();
extern void     sr_hw();
extern void     t_hw();
extern void     unitub_pos();
static char     *density();
static void     plot(); /* no longer used */

/* update Ammon with the following functions */

static void     record_info_icon();
static void     record_info_text();
static void     build_modify_statement();
extern void     save();
extern void     modify_database();
extern void     reset_image();
extern void     reset_panel();
```

9. morph.c

```
/* main program of the interactive image measurement module */

#include <ammon.h>

main(argc, argv)
int argc;
char **argv;
{
        int             i, j;
        Scrollbar       vertical_sb;

        /* create a frame to host the left command window */

        command_frame =
                window_create(NULL, FRAME,
                        FRAME_LABEL, "Ammon Image Database - Measurement module",
                        FRAME_SHOW_LABEL, TRUE,
                        WIN_WIDTH, command_rect.width,
                        WIN_HEIGHT, command_rect.height,
                        WIN_X, 0,
                        WIN_Y, 0,
```

```
                FRAME_ICON, icon_create(ICON_IMAGE, &ammon_icon, 0),
                0);

    /* create the command window to host the menu and command buttons */

    command_win = window_create(command_frame, PANEL,
                PANEL_LABEL_BOLD, TRUE,
                PANEL_SHOW_MENU, TRUE,
                0);

    /* load font to label menu and command buttons */

    font=pf_open("/apps/sundesk/fonts/sunview/lucida-19.v");

    create_menu_button();

    create_command_button(command_win);

    init_menu();

    /* create a frame to host the  info and canvas window at the right side of the command
window */

    display_window =
        window_create(NULL, FRAME,
                FRAME_LABEL, " Welcome Window",
                FRAME_SHOW_LABEL, TRUE,
                WIN_X, command_rect.width,
                WIN_Y, 0,
                WIN_SHOW,TRUE,
                WIN_WIDTH, screen_rect.width-command_rect.width,
                WIN_HEIGHT, screen_rect.height,
                0);

    /* create the info panel giving an introduction to the system */

    init_info_panel(display_window);

    vertical_sb = scrollbar_create(SCROLL_LINE_HEIGHT, 5,
                                (char *)0); /* 5 is the width of the scrollbar */

    image_canvas=window_create(display_window, CANVAS,
        WIN_VERTICAL_SCROLLBAR, vertical_sb,
        CANVAS_RETAINED, FALSE,
        CANVAS_AUTO_SHRINK, FALSE,
        WIN_CONSUME_PICK_EVENT, MS_LEFT,
```

```
                WIN_EVENT_PROC, canvas_handler,
                WIN_X, 0,
                WIN_Y, info_rect.height+5,  /* canvas under info panel */
                WIN_WIDTH, screen_rect.width-command_rect.width,
                WIN_HEIGHT, screen_rect.height-info_rect.height-5,
                0);

        /* make the image_canvas a color canvas with 253 colors */

        pw=(Pixwin *) window_get(image_canvas, WIN_PIXWIN);

        pw_putcolormap(pw, 0, 253, red, green, blue); /* reserve 3 color entries for bg and fg
*/

        window_set(image_canvas,
                CANVAS_RETAINED, TRUE,
                0);

        /* load the interface image into the canvas */

        load_interface_image();

        /* allocate memory for retrieval results of database query */

        for (i=0; i<NUM_OF_DESCRIPTOR; i++)
                for (j=0; j<MAX_SP; j++)
                        descriptor_value[i][j] = (char *)malloc(len[i]+2);

        for (i=0; i<MAX_IMAGE; i++)
                image_name[i]=(char *)malloc(len[IMAGE]+2);

        signal(SIGURG, stop_op);

        window_main_loop(command_frame);

        exit(0);

        }

/* load the interface image into the image_canvas */

static void
load_interface_image()
        {
        Pixrect *image;
        FILE   *fp;
```

```
        register int      w, h;

        clear_canvas();

        /* don't move the interface image file to other location */

        if ((fp=fopen(interface_image,"r")) == NULL)        {
                printf("Can't open %s\n",interface_image);
                return;                                     }

        image=pr_load(fp, colormap);
        w=image->pr_size.x;
        h=image->pr_size.y;
        red[252]=255; /* color entry 252 gives pure red */
        green[252]=blue[252]=0;
        pw_setcmsname(pw, "image_cmp");
        pw_putcolormap(pw, 0, 253, red, green, blue);
        cs=red[1];      /* as an indicator of the brightness */

        /* display the image in the middle of the canvas */

        if (h<screen_rect.height-info_rect.height)
                pw_rop(pw,(screen_rect.width-command_rect.width-w)/2,
                        (screen_rect.height-info_rect.height-h)/2,w,h,PIX_SRC,image,0,0);
        else
                pw_rop(pw,(screen_rect.width-command_rect.width-w)/2,0,
                        w,h,PIX_SRC,image,0,0);
        pr_close(image);
        fclose(fp);
}
```

10. icon.h

```
static short rib1_image[] = {
#include "/home/liang/icon/rib-biconcave"
};
mpr_static(rib_biconcave, 64, 64, 1, rib1_image);

static short rib2_image[] = {
#include "/home/liang/icon/rib-concave"
};
mpr_static(rib_concave, 64, 64, 1, rib2_image);

static short rib3_image[] = {
#include "/home/liang/icon/rib-sinuous"
```

```
};
mpr_static(rib_sinuous, 64, 64, 1, rib3_image);

static short rib4_image[] = {
#include "/home/liang/icon/rib-falcate"
};
mpr_static(rib_falcate, 64, 64, 1, rib4_image);

static short rib8_image[] = {
#include "/home/liang/icon/rib-falcoid"
};
mpr_static(rib_falcoid, 64, 64, 1, rib8_image);

static short rib5_image[] = {
#include "/home/liang/icon/rib-projected"
};
mpr_static(rib_projected, 64, 64, 1, rib5_image);

static short rib6_image[] = {
#include "/home/liang/icon/rib-straight"
};
mpr_static(rib_straight, 64, 64, 1, rib6_image);

static short rib7_image[] = {
#include "/home/liang/icon/rib-bundled"
};
mpr_static(rib_bundled, 64, 64, 1, rib7_image);

static short rib9_image[] = {
#include "/home/liang/icon/rib-zigzag"
};
mpr_static(rib_zigzag, 64, 64, 1, rib9_image);

static short rib10_image[] = {
#include "/home/liang/icon/rib-intercalatory"
};
mpr_static(rib_intercalatory, 64, 64, 1, rib10_image);

static short furc1_image[] = {
#include "/home/liang/icon/furc-bi"
};
mpr_static(furc_bi, 64, 64, 1, furc1_image);

static short furc2_image[] = {
#include "/home/liang/icon/furc-tri"
};
```

```
mpr_static(furc_tri, 64, 64, 1, furc2_image);

static short furc3_image[] = {
#include "/home/liang/icon/furc-multi"
};
mpr_static(furc_multi, 64, 64, 1, furc3_image);

static short furc4_image[] = {
#include "/home/liang/icon/furc-poly"
};
mpr_static(furc_poly, 64, 64, 1, furc4_image);

static short furc5_image[] = {
#include "/home/liang/icon/furc-loop2"
};
mpr_static(furc_loop2, 64, 64, 1, furc5_image);

static short furc6_image[] = {
#include "/home/liang/icon/furc-loop3"
};
mpr_static(furc_loop3, 64, 64, 1, furc6_image);

static short furc7_image[] = {
#include "/home/liang/icon/furc-loopn"
};
mpr_static(furc_loopn, 64, 64, 1, furc7_image);

static short prof1_image[] = {
#include "/home/liang/icon/prof-angular"
};
mpr_static(prof_angular, 64, 64, 1, prof1_image);

static short prof2_image[] = {
#include "/home/liang/icon/prof-round"
};
mpr_static(prof_round, 64, 64, 1, prof2_image);

static short prof3_image[] = {
#include "/home/liang/icon/prof-strong"
};
mpr_static(prof_strong, 64, 64, 1, prof3_image);

static short prof4_image[] = {
#include "/home/liang/icon/prof-weak"
};
mpr_static(prof_weak, 64, 64, 1, prof4_image);
```

```
static short direction1_image[] = {
#include "/home/liang/icon/prorsiradiate"
};
mpr_static(prorsiradiate, 64, 64, 1, direction1_image);

static short direction2_image[] = {
#include "/home/liang/icon/gprorsiradiate"
};
mpr_static(gprorsiradiate, 64, 64, 1, direction2_image);

static short direction3_image[] = {
#include "/home/liang/icon/rectiradiate"
};
mpr_static(rectiradiate, 64, 64, 1, direction3_image);

static short direction4_image[] = {
#include "/home/liang/icon/grursiradiate"
};
mpr_static(grursiradiate, 64, 64, 1, direction4_image);

static short direction5_image[] = {
#include "/home/liang/icon/rursiradiate"
};
mpr_static(rursiradiate, 64, 64, 1, direction5_image);

static short tuberc1_image[] = {
#include "/home/liang/icon/tuberc-uni"
};
mpr_static(tuberc_uni, 64, 64, 1, tuberc1_image);


static short tuberc2_image[] = {
#include "/home/liang/icon/tuberc-bi"
};
mpr_static(tuberc_bi, 64, 64, 1, tuberc2_image);

static short tuberc3_image[] = {
#include "/home/liang/icon/tuberc-tri"
};
mpr_static(tuberc_tri, 64, 64, 1, tuberc3_image);

static short tuberc4_image[] = {
#include "/home/liang/icon/tuberc-multi"
};
mpr_static(tuberc_multi, 64, 64, 1, tuberc4_image);
```

```
static short tuberc5_image[] = {
#include "/home/liang/icon/tuberc-clavate"
};
mpr_static(tuberc_clavate, 64, 64, 1, tuberc5_image);

static short tuberc6_image[] = {
#include "/home/liang/icon/tuberc-bullate"
};
mpr_static(tuberc_bullate, 64, 64, 1, tuberc6_image);

static short umbilical1_image[] = {
#include "/home/liang/icon/shallow"
};
mpr_static(shallow, 64, 64, 1, umbilical1_image);

static short umbilical2_image[] = {
#include "/home/liang/icon/steep"
};
mpr_static(steep, 64, 64, 1, umbilical2_image);

static short umbilical3_image[] = {
#include "/home/liang/icon/vertical"
};
mpr_static(vertical, 64, 64, 1, umbilical3_image);

static short umbilical4_image[] = {
#include "/home/liang/icon/undercut"
};
mpr_static(undercut, 64, 64, 1, umbilical4_image);

static short umbilical5_image[] = {
#include "/home/liang/icon/uwallht-high"
};
mpr_static(uwallht_high, 64, 64, 1, umbilical5_image);

static short umbilical6_image[] = {
#include "/home/liang/icon/uwallht-low"
};
mpr_static(uwallht_low, 64, 64, 1, umbilical6_image);

static short whorl1_image[] = {
#include "/home/liang/icon/whorl-rounded"
};
mpr_static(whorl_rounded, 64, 64, 1, whorl1_image);
```

```
static short whorl2_image[] = {
#include "/home/liang/icon/whorl-ellipsoid"
};
mpr_static(whorl_ellipsoid, 64, 64, 1, whorl2_image);

static short whorl3_image[] = {
#include "/home/liang/icon/whorl-wellipsoid"
};
mpr_static(whorl_wellipsoid, 64, 64, 1, whorl3_image);

static short whorl4_image[] = {
#include "/home/liang/icon/whorl-oval"
};
mpr_static(whorl_oval, 64, 64, 1, whorl4_image);

static short whorl5_image[] = {
#include "/home/liang/icon/whorl-lanceolate"
};
mpr_static(whorl_lanceolate, 64, 64, 1, whorl5_image);

static short whorl6_image[] = {
#include "/home/liang/icon/whorl-ogival"
};
mpr_static(whorl_ogival, 64, 64, 1, whorl6_image);

static short whorl7_image[] = {
#include "/home/liang/icon/whorl-rectangular"
};
mpr_static(whorl_rectangular, 64, 64, 1, whorl7_image);

static short whorl8_image[] = {
#include "/home/liang/icon/whorl-wrectangular"
};
mpr_static(whorl_wrectangular, 64, 64, 1, whorl8_image);

static short whorl9_image[] = {
#include "/home/liang/icon/whorl-quadrate"
};
mpr_static(whorl_quadrate, 64, 64, 1, whorl9_image);

static short whorl10_image[] = {
#include "/home/liang/icon/whorl-triangular"
};
mpr_static(whorl_triangular, 64, 64, 1, whorl10_image);

static short whorl11_image[] = {
```

```
#include "/home/liang/icon/whorl-wtriangular"
};
mpr_static(whorl_wtriangular, 64, 64, 1, whorl11_image);

static short whorl12_image[] = {
#include "/home/liang/icon/whorl-trapezoid"
};
mpr_static(whorl_trapezoid, 64, 64, 1, whorl12_image);

static short whorl13_image[] = {
#include "/home/liang/icon/whorl-coronate"
};
mpr_static(whorl_coronate, 64, 64, 1, whorl13_image);


static short venter1_image[] = {
#include "/home/liang/icon/venter-plain"
};
mpr_static(venter_plain, 64, 64, 1, venter1_image);

static short venter2_image[] = {
#include "/home/liang/icon/venter-carinate"
};
mpr_static(venter_carinate, 64, 64, 1, venter2_image);

static short venter3_image[] = {
#include "/home/liang/icon/venter-sulcate"
};
mpr_static(venter_sulcate, 64, 64, 1, venter3_image);

static short venter4_image[] = {
#include "/home/liang/icon/carinate-sulcate"
};
mpr_static(carinate_sulcate, 64, 64, 1, venter4_image);

static short venter5_image[] = {
#include "/home/liang/icon/bicarinate"
};
mpr_static(bicarinate, 64, 64, 1, venter5_image);

static short venter6_image[] = {
#include "/home/liang/icon/tricarinate"
};
mpr_static(bicarinate_sulcate, 64, 64, 1, venter6_image);

static short ontogeny_image[] = {
```

```
#include "/home/liang/icon/ontogeny"
};
mpr_static(ontogeny, 64, 64, 1, ontogeny_image);

static short approx_image[] = {
#include "/home/liang/icon/approx"
};
mpr_static(approx, 64, 64, 1, approx_image);
```

11. measure.c

```
#include <math.h>
#include "ammon.h"
#include "morph.h"

/* the following functions are used to measure images */

/* compute BISPACE in Ammon */

extern void
bi_space()
        {
        char    tmp[10];
        float   tp;

        tp=ratio(); /* use the points in the buffer */
        if (tp) {
                sprintf(tmp,"%4.2f",tp);
                sprintf(descriptor_value[BISPACE][0], "%s", tmp);
                }
        }

/* measure the maximum diameter for the specimen */

extern void
d_max()
        {
        float   tp;

        tp = distance(control_point[num_control_point-1],control_point[num_control_point-
2]);
        sprintf(descriptor_value[DMAX][0],"%5.2f", tp/(image_scale*PIXEL_CM));
        }

/*  compute number of secondary ribs per half whorl */
```

```
extern void
sr_hw()
        {
        panel_set_value(text_item[5],density());
        }

/*  compute number of tubercles per half whorl */

extern void
t_hw()
        {
        panel_set_value(text_item[6],density());
        }

/*  compute number of primary ribs per half whorl */

extern void
rib_density()
        {
        panel_set_value(text_item[4],density());
        }

/* use the number of control points in the buffer and their angular distance to calculate the
density */

static char *
density()
        {
        register int    i;
        double          sigma1, sigma2, sigma;
        char            tmp[10];

        /* angles in terms of coiling axis */

        if (!control_point[0].x || !control_point[0].y) {
                panel_set_value(parameter_info, "Please mark coiling axis!");
                num_control_point=0;                            }
        else    {
                tmp[0]=0;
                sigma1=atan2((double)(control_point[1].y-control_point[0].y),
                                (double)(control_point[1].x-control_point[0].x));
                sigma2=atan2((double)(control_point[num_control_point-1].y-
                                control_point[0].y), (double)(control_point[num_control
                                _point-1].x-control_point[0].x));
                sigma=fabs(sigma2-sigma1);
                sigma=(sigma>Pi) ? 2*Pi-sigma:sigma;
```

```
            i=(num_control_point-2)*Pi/sigma;
            sprintf(tmp,"%d",i);
            num_control_point=1;/* reserve the coiling axis, reset other points */        }

        return tmp;

        }

/* measure the maximum whorl height */

extern void
max_wh()
        {
        char   tmp[10];
        double  tp, u, w;

        tp=distance(control_point[num_control_point-1],
                    control_point[num_control_point-2]);
        sprintf(tmp,"%4.2f",tp/(PIXEL_CM*image_scale));
        panel_set_value(text_item[3],tmp);

        /* extrapolate the max diameter of the specimen    */

        u=atof(panel_get_value(text_item[2])); /* read the volution from info panel */
        w=atof(panel_get_value(text_item[0])); /* read expansion rate from info panel */
        if (u && w)    {
                tp /= (1.0-u);
                tp *= (1.0+1.0/sqrt(w));
                tp /= (image_scale*PIXEL_CM);
                sprintf(descriptor_value[DMAX][0],"%5.2f",tp);
                        }
        }

/* click 4 points to mark the rib width and length, compute rib width / rib length */

extern void
rib_width()
        {
        char    tmp[10];
        float   tp;

        tp=ratio();
        if (tp) {
                sprintf(tmp,"%4.2f",tp);
                panel_set_value(text_item[7], tmp); /* show it in the relevant text field */
                }
```

```
        }

/* mark rib start, furcation and end points, compute the relative rib furcation position */

extern void
furc_pos()
        {
        char    tmp[10];
        float   tp;

        tp=ratio_of_three();
        if (tp) {
                sprintf(tmp,"%4.2f",tp);
                panel_set_value(text_item[9], tmp); /* show it in the relevant text field */
                }
        }

/* mark the umbilical seam, tubercle and venter, compute the relative uni-tuberculate position
*/

extern void
unitub_pos()
        {
        char    tmp[10];
        float   tp;

        tp=ratio_of_three();
        if (tp) {
                sprintf(tmp,"%4.2f",tp);
                panel_set_value(text_item[10], tmp);
                }
        }


extern void
finess_ratio()
        {
        char    tmp[10];
        double w_out, r[2], dist1, dist2, sigma;
        point   p;

        w_out=atof(panel_get_value(text_item[0]));

        /* compute the distance between C' and D' (see fig. 3-1) */
```

```
        dist1=distance(control_point[num_control_point-1], control_point[num_control_point-
2]);

        r[1] = dist1/(1+sqrt(1/w_out); /* radius with diameter dist1 */

        /* if p is the middle point between A' and B' */

        p.x = (control_point[num_control_point-3].x+
                control_point[num_control_point-4].x)/2;
        p.y = (control_point[num_control_point-3].y+
                control_point[num_control_point-4].y)/2;

        /* compute the distance between D' and p (see fig. 3-1) */

        dist2=distance(control_point[num_control_point-1], p);

        /* align OP3 with axis y */

        sigma=0.0;      /* starting point with radius r[1] */
        r[0]=r[1];
        y[0] = r[1] -dist2;       /* P3Y on page 26*/
        while (sigma > -Pi)     {
                sigma -= 1.0/r[0];
                r[0]=r[1]*pow(w_out, sigma/(2*Pi));
                y[1]=r[0]*cos(sigma);
                if (y[1]<y[0])  /* projection at point p */
                        break;
                                }
        dist2 = r[0]*(1+1/sqrt(w_out))          /* shell diameter at point p */

        dist1 = distance(control_point[num_control_point-3],
                        control_point[num_control_point-4]); /* whorl width at point p */

        /* compute fineness ratio */

        sprintf(tmp,"%4.2f",dist1/dist2);
        panel_set_value(text_item[11], tmp);

        /* compute whorl width / whorl height */

        u=atof(panel_get_value(text_item[2])); /* read the volution from info panel */
        dist2 = (1-u)*r[0];   /* whorl height     */
        sprintf(descriptor_value[WWWH][0],"%4.2f",dist1/dist2);

        num_control_point=1;
```

```
}

/* compute the ratio with 4 control points - P1P2/P3P4 */

extern float
ratio()
        {
        float   dist1, dist2;
        char    tmp[40];

        if (num_control_point<4)      {
                panel_set_value(parameter_info,"Too few control points!");
                num_control_point=1;
                return 0;                }
        else    {
                dist1=distance(control_point[num_control_point-3],
                        control_point[num_control_point-4]);
                dist2=distance(control_point[num_control_point-1],
                        control_point[num_control_point-2]);
                sprintf(tmp,"Ratio=%5.4f.",dist1/dist2);
                panel_set_value(parameter_info,tmp);
                num_control_point=1;

                return dist1/dist2;
                }
        }

/* compute the ratio with 3 control points P1P2/P1P3 */

extern float
ratio_of_three()
        {
        float   dist1, dist2;
        char    tmp[40];

        if (num_control_point<3)      {
                panel_set_value(parameter_info,"Too few control points!");
                num_control_point=1;
                return 0;                }
        else    {
                dist1= distance(control_point[num_control_point-2],
                                control_point[num_control_point-3]);
                dist2=distance(control_point[num_control_point-1],
                                control_point[num_control_point-3]);
                sprintf(tmp,"Ratio=%5.4f.",dist1/dist2);
                panel_set_value(parameter_info,tmp);
```

```
                num_control_point=1;
                return dist1/dist2;
                }
        }

/* compute the distance between p1 and p2 */

extern float
distance(p1, p2)
        point   p1, p2;
        {
        float   dist;
        char    tmp[40];

        dist=sqrt((double)sqr(p1.x-p2.x)+(double)sqr(p1.y-p2.y));
        sprintf(tmp,"Distance=%5.4f.",dist);
        panel_set_value(parameter_info,tmp);
        num_control_point=1; /* reserve coiling axis, reset other points */

        return dist;
        }

/* compute basic shell geometry */

extern void
simulate_coiling()
        {
        register int    i, num_whorls, x[3], y[3], clock_wise;
        double          r[5], sigma, sigma1, sigma2, tp, tp0;
        float           w_in, w_out, u;
        char            tmp[150];

        /* need 5 control points to simulate inner and outer coiling  curve */

        if (num_control_point<5)      {
                panel_set_value(parameter_info,"Not enough control points !");
                num_control_point=0;}
        else if (num_control_point>5) {
                panel_set_value(parameter_info,"Too many control points !");
                num_control_point=0;}
        else    {
                sprintf(tmp,"%s", panel_get_value(num_of_whorls));
                num_whorls=atoi(tmp); /* P1 and P2 are separated by num_whorls */
                tmp[0]=0;
                if (!num_whorls)        {
                        panel_set_value(parameter_info,"Please give number of whorls !");
```

```
goto end;     }

/* coiling axis */
x[0]=control_point[0].x;
y[0]=control_point[0].y;

/* calculate distance between each control point and coiling axis */

for (i=1; i<num_control_point; i++)
        r[i]=distance(control_point[i], control_point[0]);

/* compute whorl height  */

tp=(r[3]-r[2])/(PIXEL_CM*image_scale);
sprintf(descriptor_value[WH][0],"%5.2f",tp);

/* compute the angle between OP4 and OP3 */

sigma1=atan2((double)(control_point[3].y-y[0]),
                (double)(control_point[3].x-x[0]));
sigma2=atan2((double)(control_point[4].y-y[0]),
                (double)(control_point[4].x-x[0]));

/* determine coiling direction */

if (sigma2-sigma1>0 && sigma2-sigma1<Pi || sigma2-sigma1<-Pi)
        clock_wise=1;
else
        clock_wise=-1;
sigma=fabs(sigma2-sigma1);
if (sigma>Pi) /* angle between OP3 and OP4 must be less than Pi */
        sigma = 2*Pi-sigma;

/* compute expansion rate and volution */

u=r[2]/r[3]; /* U in Ammon */

 /* expansion rate from inner coiling curve */
w_in=pow(r[2]/r[1],1.0/(double)num_whorls);

w_out=pow(r[3]/r[4],2*Pi/sigma); /* expansion rate from outer coiling curve */

/* display expansion rate and volution in the relevant text field */

sprintf(tmp,"%4.2f",w_out);
panel_set_value(text_item[0],tmp);
```

```
                sprintf(tmp,"%4.2f",w_in);
                panel_set_value(text_item[1],tmp);
                sprintf(tmp,"%4.2f",u);
                panel_set_value(text_item[2],tmp);

                /* compute shell diameter   */

                tp0=r[3]*(1+1.0/sqrt(w_out))/(PIXEL_CM*image_scale);
                sprintf(descriptor_value[D][0],"%5.2f",tp0);

                /* compute whorl height/diameter */

                tp0=tp/tp0;
                sprintf(descriptor_value[WHD][0],"%5.2f",tp0);

                /* compute  quantitative volution */

                tp0=(r[3]/w_out-r[2])/(PIXEL_CM*image_scale);
                if (tp0<0.0)
                        tp=0.0;
                else
                        tp=tp0/tp;
                sprintf(descriptor_value[VOLUTION0][0],"%5.2f",tp);

                /* compute umbilical diameter  */

                tp0=r[2]*(1+1.0/sqrt(w_in))/(PIXEL_CM*image_scale);
                sprintf(descriptor_value[UD][0],"%5.2f",tp0);

                panel_set_value(num_of_whorls,"");

                }
        end:    ;
        }

/* superimpose the simulated coiling cure on top of the image */

static void
plot_coiling(x, clock_wise, w_in, w_out, r, sigma1)
        int     x[], clock_wise;
        float   w_in, w_out;
        double r[], sigma1;
        {
        register int    x[3], y[3];
        double          tp;
```

```
num_point=0; /* number of points on the coiling curve */
x[2]=y[2]=0;

/* plot the outer coiling curve */

sigma=1.0/r[3];        /* starting point */
while (sigma<2*Pi)    {
        tp=sigma1+clock_wise*sigma;
        r[0]=r[3]*pow(w_out,-sigma/(2*Pi));
                x[1]=x[0]+r[0]*cos(tp);
                y[1]=y[0]+r[0]*sin(tp);
                if (x[1] != x[2] || y[1] != y[2]) {
                        x[2]=curve_points[num_point].x=x[1];
                        y[2]=curve_points[num_point].y=y[1];

                /* curve_points can be used to restore the original image */

                        curve_points[num_point].z=pw_get(pw,x[1],y[1]);
                        pw_put(pw,x[1],y[1],252); /* color entry 252 - red */
                        num_point++;                    }
        sigma += 1.0/r[0];      /* angular increment step */   }

/* plot the inner coiling curve */

sigma1=atan2((double)(control_point[2].y-y[0]),(double)(control_point[2].x-x[0]));
x[2]=y[2]=0;
sigma=1.0/r[2];
r[0]=r[2];
r[1]=(r[1]<6) ? r[1]:6;
while (r[0]>r[1] && num_point<10000)      {
        tp=sigma1+clock_wise*sigma;
        r[0]=r[2]*pow(w_in,-sigma/(2*Pi));
        x[1]=x[0]+r[0]*cos(tp);
        y[1]=y[0]+r[0]*sin(tp);
        if (x[1] != x[2] || y[1] != y[2]) {
                x[2]=curve_points[num_point].x=x[1];
                y[2]=curve_points[num_point].y=y[1];
                curve_points[num_point].z=pw_get(pw,x[1],y[1]);
                pw_put(pw,x[1],y[1],252);
                num_point++;                    }
        sigma += 1.0/r[0];              }

}

/* plot brightness profile between two points */
```

```
static void
plot()
        {
        register int    i, k, dist, x, y;
        struct size     profile_rect;
        struct point    profile_start;

        if (between[1].x || between[1].y)       {
                dist= distance(between[0], between[1]);
                profile_rect.width = dist;
                profile_rect.height = 256;
                profile_start.x = screen_rect.width-command_rect.width-profile_rect.width;
                profile_start.y = screen_rect.height-info_rect.height-profile_rect.height;
                (void)pw_writebackground(pw, profile_start.x, profile_start.y,
                        profile_rect.width, profile_rect.height, PIX_CLR);

                for (i=0; i<profile_rect.width; i++)    {
                        x=between[0].x+(between[1].x-between[0].x)*i/dist;
                        y=between[0].y+(between[1].y-between[0].y)*i/dist;
                        k=pw_get(pw, x, y);
                        pw_put(pw, profile_start.x+i, screen_rect.height-
                                                info_rect.height-k, 251);       }

        }
        else if (between[0].x || between[0].y)
                panel_set(wait_num,
                        PANEL_VALUE, "Need one more point to plot!",
                        0);
        else
                panel_set(wait_num,
                        PANEL_VALUE, "Need two points to plot!",
                        0);
        }
```

12. dbms.pc

```
/* all database related operations are in this file which need to be compiled by Oracle
precompiler before proceeding with cc and combine with other source code */

#include <stdio.h>
#include <ammon.h>

EXEC SQL BEGIN DECLARE SECTION;
        extern char     *query_statement;
        extern char     *search_condition[12];
        char            oracleid = '/';
```

```
        VARCHAR modify_statement[500];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

/* Declare the bind and select descriptors. */
SQLDA               *bind_dp;
SQLDA               *select_dp;

extern SQLDA        *sqlald();

extern void     sqlprc();

extern void     sqlnul();
int             null_ok;


extern void
execute_query()
        {
        same as the one in Appendix 2
        }

extern void
modify_database()
        {
        register int     i, j, k;

        /* Automatic log on to ORACLE  */

        EXEC SQL CONNECT :oracleid;

        /* Set up ORACLE error handling.  */
        EXEC SQL WHENEVER SQLERROR GOTO sqlerror;


        bind_dp = sqlald (40, 20, 0);
        bind_dp->N = 40;


        /* Prepare the statement.  */

        EXEC SQL PREPARE S FROM :modify_statement;

        EXEC SQL DESCRIBE BIND VARIABLES FOR S INTO bind_dp;
```

```
        /* If F is negative, there were more bind variables than originally allocated by sqlald().
*/

        if (bind_dp->F <0)
        {
        printf("\nToo many bind variables for descriptor.");
        goto sqlerror;
        }

        /* Set the maximum number of array elements in the descriptor to the number found.
*/
        bind_dp->N = bind_dp->F;
        j=0;

        /* copy only morph paramters which have been changed */

        for (i=DMAX; i<NUM_OF_DESCRIPTOR; i++)
                if (*descriptor_value[i][0])    {
                        bind_dp->L[j] = strlen(descriptor_value[i][0]);
                        bind_dp->V[j] = malloc(bind_dp->L[j]+1);
                        strcpy (bind_dp->V[j], descriptor_value[i][0]);
                        bind_dp->I[j] = 0;
                        bind_dp->T[j++] = 1;
                                                }

        /* figure out the name of the selected image */

        k=rel[image_index(control_point[0].x,control_point[0].y)];
        i=0;
        while (image_name[k][i++] != ' ')       ;
        bind_dp->L[j] = i-1;    /* image name as condition. */
        bind_dp->V[j] = malloc(bind_dp->L[j]+1);
        strcpy (bind_dp->V[j], image_name[k]);
        bind_dp->I[j] = 0;
        bind_dp->T[j] = 1;

        EXEC SQL EXECUTE S USING DESCRIPTOR bind_dp;

        printf("\nNumber of rows processed: %d\n",sqlca.sqlerrd[2]);

        /* Free space used by descriptors.  */
        for (i=0; i<bind_dp->F; i++)
                free (bind_dp->V[i]);

        /* Free space used by the descriptors themselves.  */
```

```
sqlclu (bind_dp);

EXEC SQL COMMIT WORK;

/* disconnect from Oracle database engine */

EXEC SQL COMMIT RELEASE;

return;
/* ORACLE error handler */
sqlerror:
        printf("\n\n%.70s\n",sqlca.sqlerrm.sqlerrmc);
        EXEC SQL WHENEVER SQLERROR CONTINUE;
        EXEC SQL ROLLBACK WORK;
        EXEC SQL COMMIT RELEASE;

}
```

13. misc.c

```
/* miscellaneous functions used for updating database */

#include <ammon.h>
#include <morph.h>

/* save new data into the database */

extern void
        save()
        {

        /* find the morphological information from icon palette. */

        record_info_icon();

        /* find the morphological information from text items associated with button */

        record_info_text();

        /* build database update statement based on descriptors available. */

        build_modify_statement();
```

```
        modify_database();

        /* Reset */

        reset_panel();

        }

/* scan the icon panel, copy new data */

static void
record_info_icon()
        {
        register int i;

        for (i=0; i<NUM_OF_ICON; i++)
                if (panel_get(inputIcon_panel, PANEL_TOGGLE_VALUE, i))      {
                        if (! *descriptor_value[icon_to_descriptor[i]][0]) /* new data */
                                sprintf(descriptor_value[icon_to_descriptor[i]][0], icon_list[i]);
                        else    {
                                strcat(descriptor_value[icon_to_descriptor[i]][0],",");
                                strcat(descriptor_value[icon_to_descriptor[i]][0],icon_list[i]);
                                }                                       }

        }

/* scan the text fields in the info panel, copy data into an array */

static void
        record_info_text()
        {
        register int    i;
        char*           tmp;

        for (i=0; i<NUM_OF_TEXT; i++)
                if (*(tmp=panel_get_value(text_item[i]))) /* data available */
                        sprintf(descriptor_value[text_to_descriptor[i]][0], tmp);
        }

/* construct the update query statement for new data */

static void
build_modify_statement()
        {
        register int    i, j=0;
        char    tmp[3];
```

```
        sprintf(modify_statement.arr,"%s","update ammon_image set ");

        for (i=DMAX; i<NUM_OF_DESCRIPTOR; i++) /* DMAX - start of morph
parameters */
                if (*descriptor_value[i][0])    {
                        if (j)
                                strcat(modify_statement.arr,", ");
                        j++;
                        strcat(modify_statement.arr,descriptor_name[i]);
                        strcat(modify_statement.arr," =: v"); /* v - binding variable */
                        sprintf(tmp,"%d",i);
                        strcat(modify_statement.arr,tmp);
                                                }
        strcat(modify_statement.arr," where image = :v");
        modify_statement.len=strlen(modify_statement.arr);

        }

/* erase simulated coiling curve with pre-silmulate points */

extern void
reset_image()
        {
        register int    i;

        /* put the original data points back */

        for (i=0;i<num_point; i++)
                pw_put(pw,curve_points[i].x,curve_points[i].y,curve_points[i].z);
        num_point=num_control_point=0; /* reset counters */
        }

/* reset the icon panel and text fields in the info panel after saving the new data
  make them ready for new data input */

extern void
reset_panel()
        {
        register int    i, j;

        for (i=DMAX; i<NUM_OF_DESCRIPTOR; i++)
                *descriptor_value[i][0]=0;

        for (i=0; i<NUM_OF_TEXT; i++)
                if (*panel_get_value(text_item[i]))
```

```
                panel_set_value(text_item[i],"");

        for (i=0; i<NUM_OF_ICON; i++)
                if (panel_get(inputIcon_panel, PANEL_TOGGLE_VALUE, i))
                        panel_set(inputIcon_panel, PANEL_TOGGLE_VALUE, i, FALSE, 0);

        num_control_point=0;
        }

/* jump to display the specified image of figure_num */

extern void
goto_figure()
        {
        display_from=0;
        display_to=atoi(panel_get_value(show_num))-atoi(panel_get_value(figure_num));
        next_page();
        }
```

14. win.h

```
/* button locations */

#define COMMAND_START_ROW 12
#define COMMAND_INCREMENT 2

/* introductory text locations */

#define X_START         140
#define Y_START         70
#define Y_INCREMENT     25

/* info panel item locations */

static point pos[] = {{38, 1}, {55, 1}, {0, 2}, {11, 2}, {16, 2}, {26, 2}, {31, 2},
                {35, 2}, {40, 2}, {48, 2}, {54, 2}, {61, 2}, {66, 2}, {73, 2},
                {78, 2}, {84, 2}, {0, 3}, {12, 3}, {17, 3}, {27, 3}, {32, 3},
                {42, 3}, {47, 3}, {59, 3}, {64, 3}, {70, 3}, {75, 3}, {81, 3}};

static point label_pos[] = {{2, 100}, {2, 170}, {67, 240}, {461, 240}, {202, 304},
                                {10, 370}, {202, 432}};

static  void    adjust_info_panel(Panel local_panel);
static  void    create_icon_panel(Panel local_panel);
```

15. win.c

```
/* windows, buttons related functions */

#include "ammon.h"
#include "icon.h"
#include "win.h"

extern void
create_command_button(frame_local)
        Frame  frame_local;
        {
        panel_create_item(frame_local,PANEL_BUTTON,
                PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW),
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_NOTIFY_PROC, query_from_text,
                PANEL_LABEL_IMAGE,
                        panel_button_image(frame_local,"Combination",13,font),
                0);

        panel_create_item(frame_local,PANEL_MESSAGE,
                PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                COMMAND_INCREMENT),
                PANEL_LABEL_FONT, font,
                PANEL_LABEL_STRING,"Some Utility:",
                0);

        panel_create_item(frame_local,PANEL_BUTTON,
                PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                2*COMMAND_INCREMENT),
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_NOTIFY_PROC, next_page,
                PANEL_LABEL_IMAGE, panel_button_image
                                        (frame_local,"Next Page",13,font),
                0);

        panel_create_item(frame_local,PANEL_BUTTON,
                PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                        3*COMMAND_INCREMENT),
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_NOTIFY_PROC, previous_page,
                PANEL_LABEL_IMAGE, panel_button_image
                                        (frame_local,"Previous Page",13,font),
                0);

        panel_create_item(frame_local,PANEL_TEXT,
```

```
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                4*COMMAND_INCREMENT),
        PANEL_LABEL_STRING, "Brightness:",
        0);

bright=panel_create_item(frame_local,PANEL_SLIDER,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                4*COMMAND_INCREMENT+1),
        PANEL_SHOW_VALUE, FALSE,
        PANEL_VALUE, 0,
        PANEL_MIN_VALUE, -50,
        PANEL_MAX_VALUE, 50,
        PANEL_SLIDER_WIDTH, 94,
        PANEL_NOTIFY_PROC, adjust_brightness,
        0);

panel_create_item(frame_local,PANEL_TEXT,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                5*COMMAND_INCREMENT+1),
        PANEL_LABEL_STRING, "Contrast:",
        0);

contrast=panel_create_item(frame_local,PANEL_SLIDER,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                6*COMMAND_INCREMENT),
        PANEL_SHOW_VALUE, FALSE,
        PANEL_VALUE, 10,
        PANEL_MIN_VALUE, 1,
        PANEL_MAX_VALUE, 20,
        PANEL_SLIDER_WIDTH, 94,
        PANEL_NOTIFY_PROC, adjust_contrast,
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                6*COMMAND_INCREMENT+1),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, goto_figure,
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Goto Figure:",13,font),
        0);

figure_num = panel_create_item(frame_local, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                7*COMMAND_INCREMENT+1),
```

```
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_LABEL_STRING, "Number:",
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                8*COMMAND_INCREMENT),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, distance
        (control_point[num_control_point-1],control_point[num_control_point-2]),
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Distance",13,font),
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                9*COMMAND_INCREMENT),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, ratio,
        PANEL_LABEL_IMAGE, panel_button_image (frame_local,"Ratio",13,font),
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                10*COMMAND_INCREMENT),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, reset_panel,
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Reset Panel",13,font),
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                11*COMMAND_INCREMENT),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, reset_image,
        PANEL_LABEL_IMAGE, panel_button_image
                                (frame_local,"Reset image",13,font),
        0);

panel_create_item(frame_local,PANEL_BUTTON,
        PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                12*COMMAND_INCREMENT),
        PANEL_ITEM_X, ATTR_COL(0),
        PANEL_NOTIFY_PROC, reverse,
```

```
            PANEL_LABEL_IMAGE, panel_button_image
                                    (frame_local,"Reverse Image",13,font),
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                    13*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, save,
            PANEL_LABEL_IMAGE, panel_button_image
                                    (frame_local,"Save",13,font),
            0);

        panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                    14*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(0),
            PANEL_NOTIFY_PROC, help,
            PANEL_LABEL_IMAGE,panel_button_image
                                    (frame_local,"Help",4,font),
            0);

        button_Quit = panel_create_item(frame_local,PANEL_BUTTON,
            PANEL_ITEM_Y, ATTR_ROW(COMMAND_START_ROW+
                                    14*COMMAND_INCREMENT),
            PANEL_ITEM_X, ATTR_COL(12),
            PANEL_NOTIFY_PROC, quit,
            PANEL_LABEL_IMAGE,panel_button_image(frame_local,"Quit",4,font),
            0);

        }

extern void
init_info_panel(frame_local)
        Frame   frame_local;
        {

        Panel   info_panel;

        /* font for displaying title */

        font=pf_open("/apps/sundesk/fonts/sunpaint/R.24");

        /* create info window to display the introduction text */

        info_panel=window_create(frame_local, PANEL,
```

```
        PANEL_LABEL_BOLD, FALSE,
        WIN_X, 0,
        WIN_Y, 0,
        WIN_WIDTH, screen_rect.width-command_rect.width,
        WIN_HEIGHT, info_rect.height,
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START-30,
        PANEL_ITEM_Y, Y_START-30,
        PANEL_LABEL_STRING, "        Interactive    Ammonite    Image
                                        Measurement    Module",
        0);

/* font for displaying introductory text */

font=pf_open("/apps/sundesk/fonts/sunview/lucida-19.v");

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START+40,
        PANEL_ITEM_Y, Y_START,
        PANEL_LABEL_STRING, "First, retrieve the ammonites that you want
                                        to measure by:",
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START,
        PANEL_ITEM_Y, Y_START+Y_INCREMENT,
        PANEL_LABEL_STRING, "1. Taxonomy: Family, Genus, and Species.",
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START,
        PANEL_ITEM_Y, Y_START+2*Y_INCREMENT,
        PANEL_LABEL_STRING, "2. Stratigraphy: Stage and Standard Zone.",
        0);

panel_create_item(info_panel,PANEL_MESSAGE,
        PANEL_LABEL_FONT, font,
        PANEL_ITEM_X, X_START,
        PANEL_ITEM_Y,Y_START+3*Y_INCREMENT,
        PANEL_LABEL_STRING, "3. Geography: Geographic Area and Country.",
```

```
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START,
            PANEL_ITEM_Y, Y_START+4*Y_INCREMENT,
            PANEL_LABEL_STRING, "4. Morphology: Volution, Expansion Rate,
                                        Whorl Shape, and Rib Form.",
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START,
            PANEL_ITEM_Y, Y_START+5*Y_INCREMENT,
            PANEL_LABEL_STRING, "5. Reference: in which the figure of
                                        the specimen appears.",
            0);

        panel_create_item(info_panel,PANEL_MESSAGE,
            PANEL_LABEL_FONT, font,
            PANEL_ITEM_X, X_START+40,
            PANEL_ITEM_Y, Y_START+6*Y_INCREMENT,
            PANEL_LABEL_STRING, "Second, select the image and
                                        do the measurement. ",
            0);

        }

extern void
adjust_info_panel(local_panel)
        Panel local_panel
        {
        int             i;
        char            tmp[40], *buf;
        Panel_item      item;

        /* destroy all old panel items in the info panel */

        panel_each_item(local_panel, item)
            panel_destroy_item(item);
        panel_end_each

        info_rect.height = text_rect.height; /* increase height to accomondate icon panel */
        info_rect.width = screen_rect.width-command_rect.width-text_rect.width;
        window_set(local_panel,
            WIN_HEIGHT, info_rect.height,
```

```
        WIN_WIDTH,  info_rect.width,
        0);

/* display the query statement of the menu selection */

if (num_of_Items)      {

/* translate from SQL query to plain text, eliminate ':' */

/* skip 'select * from  OPS$liang.ammon_image where' in query statement */

        buf=&query_statement[42];
        query_to_plain(buf);
        i=strlen(buf);
        buf[i-2]=0;
        tmp[0]=0;
        sprintf(tmp,"%s%s",buf,search_condition[0]);
                        }
else
        tmp[0]=0;

/* condition_text is the text field for query text display or input, you can
   type in a query and press the Combination button to search the database */

condition_text = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 180,
        PANEL_VALUE_DISPLAY_LENGTH, 200,
        PANEL_LABEL_X, ATTR_COL(0),
        PANEL_LABEL_Y, ATTR_ROW(0),
        PANEL_LABEL_STRING, "Search :",
        PANEL_VALUE, tmp,
        0);

/* display number of images retrieved */

if (num_of_Items)
        sprintf(tmp,"%4d",num_of_Items);

show_num = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 5,
        PANEL_LABEL_X, ATTR_COL(0),
        PANEL_LABEL_Y, ATTR_ROW(1),
        PANEL_LABEL_STRING, "Images Returned:",
        PANEL_VALUE, tmp,
        0);
```

```
/* number of images wait to be displayed */

wait_num =
        panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 5,
        PANEL_LABEL_X, ATTR_COL(22),
        PANEL_LABEL_Y, ATTR_ROW(1),
        PANEL_LABEL_STRING, "Remained:",
        0);

/* when simulate the inner coiling curve, the user need to specify the number of
   whorls which seperate the two control points (P1 & P2) */

num_of_whorls =
        panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_LABEL_X, ATTR_COL(pos[0].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[0].y),
        PANEL_LABEL_STRING, "Whorls:",
        0);

/* text field for displaying some parameter info */

parameter_info =
        panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 25,
        PANEL_LABEL_X, ATTR_COL(pos[1].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[1].y),
        0);

/* create buttons and text fields for image measurement and result display */

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[2].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[2].y),
        PANEL_LABEL_IMAGE, panel_button_image
                                (local_panel,"W (out) =",0, NULL),
        PANEL_NOTIFY_PROC, simulate_coiling,
        0);

text_item[0] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[3].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[3].y),
        0);
```

```
panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[4].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[4].y),
        PANEL_LABEL_IMAGE, panel_button_image
                                (local_panel,"W (in) =",0, NULL),
        PANEL_NOTIFY_PROC, simulate_coiling,
        0);

text_item[1]=panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[5].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[5].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[6].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[6].y),
        PANEL_LABEL_IMAGE, panel_button_image(local_panel,"U=",0, NULL),
        PANEL_NOTIFY_PROC, simulate_coiling,
        0);

text_item[2] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[7].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[7].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[8].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[8].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"MaxWH=",0, NULL),
        PANEL_NOTIFY_PROC, max_wh,
        0);

text_item[3] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 5,
        PANEL_LABEL_X, ATTR_COL(pos[9].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[9].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[10].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[10].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"PRHW=",0, NULL),
```

```
        PANEL_NOTIFY_PROC, rib_density,
        0);

text_item[4] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[11].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[11].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[12].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[12].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"SRHW=",0, NULL),
        PANEL_NOTIFY_PROC, sr_hw,
        0);

text_item[5] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[13].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[13].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[14].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[14].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"THW=",0, NULL),
        PANEL_NOTIFY_PROC, t_hw,
        0);

text_item[6] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[15].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[15].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[16].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[16].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"Rib Width=",0, NULL),
        PANEL_NOTIFY_PROC, rib_width,
        0);

text_item[7] = panel_create_item(local_panel, PANEL_TEXT,
```

```
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[17].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[17].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[18].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[18].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"WW/WH",0, NULL),
        PANEL_NOTIFY_PROC, bi_space,
        0);

text_item[8] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[19].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[19].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[20].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[20].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"Furcpos=",0, NULL),
        PANEL_NOTIFY_PROC, furc_pos,
        0);

text_item[9] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[21].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[21].y),
        0);

panel_create_item(local_panel, PANEL_BUTTON,
        PANEL_LABEL_X, ATTR_COL(pos[22].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[22].y),
        PANEL_LABEL_IMAGE, panel_button_image
        (local_panel,"Unitubpos=",0, NULL),
        PANEL_NOTIFY_PROC, unitub_pos,
        0);

text_item[10] = panel_create_item(local_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 4,
        PANEL_LABEL_X, ATTR_COL(pos[23].x),
        PANEL_LABEL_Y, ATTR_ROW(pos[23].y),
        0);
```

```
        panel_create_item(local_panel, PANEL_BUTTON,
                PANEL_LABEL_X, ATTR_COL(pos[24].x),
                PANEL_LABEL_Y, ATTR_ROW(pos[24].y),
                PANEL_LABEL_IMAGE, panel_button_image
                (local_panel,"WWD=",0, NULL),
                PANEL_NOTIFY_PROC, finess_ratio,
                0);

        text_item[11] = panel_create_item(local_panel, PANEL_TEXT,
                PANEL_VALUE_DISPLAY_LENGTH, 4,
                PANEL_LABEL_X, ATTR_COL(pos[25].x),
                PANEL_LABEL_Y, ATTR_ROW(pos[25].y),
                0);

        panel_create_item(local_panel, PANEL_BUTTON,
                PANEL_LABEL_X, ATTR_COL(pos[26].x),
                PANEL_LABEL_Y, ATTR_ROW(pos[26].y),
                PANEL_LABEL_IMAGE, panel_button_image
                (local_panel,"CHW=",0, NULL),
                PANEL_NOTIFY_PROC, c_hw,
                0);

        text_item[12] = panel_create_item(local_panel, PANEL_TEXT,
                PANEL_VALUE_DISPLAY_LENGTH, 4,
                PANEL_LABEL_X, ATTR_COL(pos[27].x),
                PANEL_LABEL_Y, ATTR_ROW(pos[27].y),
                0);

        create_icon_panel();

        panel_paint(local_panel, PANEL_CLEAR);

}

static void
create_icon_panel(local_panel)
        Panel local_panel
        {

        /* create an icon panel to facilitate the input of morph data */

        inputIcon_panel =
                panel_create_item(local_panel, PANEL_TOGGLE,
                PANEL_CHOICE_IMAGES, &rib_biconcave, &rib_concave, &rib_sinuous,
                        &rib_falcate, &rib_projected, &rib_straight, &rib_bundled,
```

```
                &rib_falcoid, &rib_zigzag, &rib_intercalatory, &furc_bi,
                &furc_tri, &furc_multi, &furc_poly,&furc_loop2, &furc_loop3,
                &furc_loopn,&prof_angular, &prof_round, &prof_strong,
                &prof_weak,&prorsiradiate, &gprorsiradiate, &rectiradiate,
                &grursiradiate, &rursiradiate, &tuberc_uni, &tuberc_bi,
                &tuberc_tri, &tuberc_multi, &tuberc_clavate, &tuberc_bullate,
                &uwallht_high, &uwallht_low, &shallow, &steep, &vertical,
                &undercut, &ontogeny, &whorl_rounded, &whorl_ellipsoid,
                &whorl_wellipsoid, &whorl_oval, &whorl_lanceolate,
                &whorl_ogival, &whorl_rectangular, &whorl_wrectangular,
                &whorl_quadrate, &whorl_triangular, &whorl_wtriangular,
                &whorl_trapezoid, &whorl_coronate, &venter_plain,
                &venter_carinate, &venter_sulcate, &carinate_sulcate,
                &bicarinate, &bicarinate_sulcate, &approx, 0,
        PANEL_CHOICE_XS, 76,140,204,268,332,396,460,524,588,652,76,140,204,
                268,332,396,460,524,588,652,1,140,204,268,332,396,524,588,652,1,
                65, 129, 268,332,396,460,524,588,652,76,140,204,268,332,396,460,
                524, 588,652,1, 65,129,268,332,396,460,524,588,652,0,
        PANEL_CHOICE_YS, 88, 88, 88, 88, 88, 88, 88, 88, 88, 88, 152,152,152,
                152,152,152,152,152,152,152,216,216, 216, 216, 216, 216,216,
                216, 216, 280, 280, 280, 280, 280, 280,280, 280, 280,280, 344,
                344, 344, 344, 344, 344, 344, 344, 344, 344, 408, 408, 408,
                408, 408, 408, 408, 408, 408, 408, 408, 0,
        0);

/* label the icon panel */

panel_create_item(local_panel, PANEL_MESSAGE,
        PANEL_LABEL_X, label_pos[0].x,
        PANEL_LABEL_Y, label_pos[0].y,
        PANEL_LABEL_STRING, "RIB FORM",
        0);

panel_create_item(local_panel, PANEL_MESSAGE,
        PANEL_LABEL_X, label_pos[1].x,
        PANEL_LABEL_Y, label_pos[1].y,
        PANEL_LABEL_STRING, "RIB FURC",
        0);

panel_create_item(local_panel, PANEL_MESSAGE,
        PANEL_LABEL_X, label_pos[2].x,
        PANEL_LABEL_Y, label_pos[2].y,
        PANEL_LABEL_STRING, "RIB TREND",
        0);

panel_create_item(local_panel, PANEL_MESSAGE,
```

```
                PANEL_LABEL_X, label_pos[3].x,
                PANEL_LABEL_Y, label_pos[3].y,
                PANEL_LABEL_STRING, "TUBERCLE",
                0);

        panel_create_item(local_panel, PANEL_MESSAGE,
                PANEL_LABEL_X, label_pos[4].x,
                PANEL_LABEL_Y, label_pos[4].y,
                PANEL_LABEL_STRING, "UWALL",
                0);

        panel_create_item(local_panel, PANEL_MESSAGE,
                PANEL_LABEL_X, label_pos[5].x,
                PANEL_LABEL_Y, label_pos[5].y,
                PANEL_LABEL_STRING, "WHORL",
                0);

        panel_create_item(local_panel, PANEL_MESSAGE,
                PANEL_LABEL_X, label_pos[6].x,
                PANEL_LABEL_Y, label_pos[6].y,
                PANEL_LABEL_STRING, "VENTER",
                0);

}

extern void
adjust_display_window(local_panel)
        Panel   local_panel;
        {

        start_status=1;/* now the system has gone through at least one query */

        adjust_info_panel();

        /* adjust image_canvas size due to the creation of a new text window */

        window_set(image_canvas,
                WIN_WIDTH, screen_rect.width-command_rect.width-8,
                WIN_HEIGHT, screen_rect.height-info_rect.height,
                WIN_Y, info_rect.height,
                0);

        text_win=window_create(display_window, TEXTSW,
                WIN_X, screen_rect.width-text_rect.width,
                WIN_Y, 0,
                WIN_WIDTH, text_rect.width,
```

```
                WIN_HEIGHT, text_rect.height,
                0);

        panel_set(local_panel, PANEL_CARET_ITEM, num_of_whorls, 0);
}

extern void canvas_handler(event)
        Event   *event;
        {
        register int      i, j;

        if (event_is_up(event))
                return;
        switch (event_action(event)) {
                case MS_LEFT: /* locate the image and display related info */
                i=image_index(event_x(event),event_y(event));
                if (i>=0)
                        show_descriptors(i);
                break;
        case MS_MIDDLE:

        /* mark the point with color 252 and determine the image scale */

                pw_put(pw,event_x(event),event_y(event),252);
                control_point[num_control_point].x=event_x(event);
                control_point[num_control_point].y=event_y(event);
                if (!num_control_point)         {
                        image_scale=atof(descriptor_value[SCALE][rel[image_index
                                                (event_x(event), event_y(event))]]);
                if (!image_scale)
                        image_scale=1.0;        }
                num_control_point++;
        case MS_RIGHT: /* mark two points for ploting brightness profile */
                if (between[1].x || between[1].y)       {
                        i=between[1].x;
                        j=between[1].y;
                        between[1].x=event_x(event);
                        between[1].y=event_y(event);
                        between[0].x=i;
                        between[0].y=j;         }
                else if (between[0].x || between[0].y)  {
                        between[1].x=event_x(event);
                        between[1].y=event_y(event);    }
                else    {
                        between[0].x=event_x(event);
                        between[0].y=event_y(event);}
```

```
                break;
        default:
                break;
        }
}

extern void
clear_canvas()
        {
        (void)pw_writebackground(pw,0,0,(int)(LINT_CAST(window_get(image_canvas,
CANVAS_WIDTH))),
        (int)(LINT_CAST(window_get(image_canvas, CANVAS_HEIGHT))), PIX_CLR);
        }
```

Appendix 5 Source code for automated image measurement module

There are 9 files in this program: auto.h, main.c, canny.c, thin.c, trace.h, trace.c, utility.c, smooth_form.c, ribbed_form.c.

1. auto.h

```
/*
 * Header file for automated image measurement module.
 *
 * Author: Bo Liang
 * Department of Geological Sciences
 * The University of British Columbia
 * Date: May 1991
 * Modifications: April 1994
 */

#include <stdio.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>

#define MAXNUM   6000   /* maxium number of segments in the image */
#define MAXWDTH  2048
#define MINLEN   10     /* minimum number of points in segment */

#define Pi       3.1416
#define RIBANG   Pi/12  /* radial */
#define COILANG  2*Pi/5 /* spiral */

/* label for line segments */

#define RIB      1
#define COILING  2
#define UNKNOWN  3

#define N   0
#define NE  1
#define E   2
#define SE  3
#define S   4
#define SW  5
#define W   6
#define NW  7
#define O   8

#define MAX(x, y)   ( x > y ) ? x : y
#define MIN(x, y)   (x < y) ? x: y
```

```
#define SQR(i)          i*i
#define D(x0,y0,x1,y1) sqrt(SQR(x1-x0)+SQR(y1-y0))
#define CLOSE(x0,y0,x1,y1) (abs(x0-x1)<2 && abs(y0-y1)<2)

/* angle conversion macro */

#define POSITIVE(alpha)          (alpha < 0) ? 2*Pi+alpha:alpha
#define LESSPI(alpha)            (alpha > Pi) ? 2*Pi-alpha:alpha
#define TWOPI(alpha)             (alpha > 2*Pi) ? alpha-2*Pi:alpha
#define HALFPI(alpha)            (alpha > Pi/2) ? Pi-alpha:alpha
#define LARGER(alpha, belta)     (alpha > belta) ? alpha-belta : 2*Pi+alpha-belta
#define TRUEDIF(alpha, belta)    (belta>0) ? alpha-belta : alpha+belta
#define ANGLE(x0, y0, x1, y1)    (x0 == x1) ? Pi/2 : atan((y1-y0)/(x1-x0))


int             imageSize;
int             num_of_seg;  /* number of line segments in the image */
int             *seq;
short int       neighbor[2][MAXNUM];
float           yo, xo;          /* coordinates of the coiling axis */
float           threshold;
float           *theta;
float           ox[MAXNUM], oy[MAXNUM], r[MAXNUM];
float           rep[MAXNUM], label[MAXNUM];

int             *num_of_point;       /* number of points in the line segment */

struct points {
        short int        x;
        short int        y;
        short int        z;
}       *segment[MAXNUM]
        struct point     *newseg[MAXNUM], *buf;

        Pixrect          *new_image;
```

2. main.c

```
#include <stdio.h>
#include "auto.h"

main(argc, argv)
        int      argc;
        char     *argv[];
{
        FILE             *fpi, *fpo;
```

```
char            *image_in, *image_out;
Pixrect         *orig_image, *screen[2];
colormap_t      *colormap;
register         argn;
int             nrows, ncols,;


if (argn = getarg("i", argc, argv))
        image_in = argv[++argn];
if (argn = getarg("o", argc, argv))
        image_out = argv[++argn];

/* load original ammon image */

fpi = fopen(image_in, "r");
orig_image = pr_load(fpi, colormap);
if (orig_image == NULL) {
        fprintf(stderr, "Can't open %s as image file\n", image_in);
        exit(1);
}
ncols = orig_image->pr_size.x;
nrows = orig_image->pr_size.y;
imageSize = MAX(nrows, ncols);

if ((fpo = fopen(image_out, "w")) == NULL);
fprintf(stderr, "Can't open %s as output file\n", image_out);
exit(1);

/* create a memory pixrect to host the processed image of orig_image */

new_image = mem_create(ncols, nrows, 1);
if (new_image == NULL) {
        fprintf(stderr, "2: can't open new_image as image file\n");
        exit(1);
}
/* prepare screen for display */

screen[0] = pr_open("/dev/fb");
screen[1] = pr_open("/dev/fb");


/* remove some of the background noise */

remove_noise(orig_image);

/* use Canny operator to detect edges */
```

```
edge_detector(orig_image, new_image);

/* skeletonize edges potentially wider than one pixel */

edge_thinning(new_image);

/* group edge points into line segment */

trace_segment(ncols, nrows);

/* after all the changes on orig_image, restore it to original state */

pr_close(orig_image);
orig_image = pr_load(fpi, colormap);

/* connect short line segments based on distance and colinearity */

connect();

/* order segments according to number of points in segment */

order_segment();


/*
 * remove redundant segments based on complete overlap along x & y as
 * well as closeness of end-points
 */

remove_redundant_segs();

/*
 * use the underlying pattern of ribs and coiling curves to locate
 * coiling axis
 */

locate_coiling_axis(ncols, nrows);

/*
 * assign segments to one of the following classes : Rib, coiling
 * curve and Unknown
 */

if (label_seg() == RIB)          /* ribbed form */
        process_ribbed_form();
```

```
        else                    /* smooth form */
                process_smooth form();

        exit(0);

}
```

3. canny.c

provided by the Department of Computer Science, UBC.

4. thin.c

provided by the Department of Computer Science, UBC.

5. trace.h

```
static int      xx, yy;

struct point    {
        int     x;
        int     y;
};

/* orientation data array in terms of {delta x, delta y} for trace_from */

struct point    main_O[]={{0,1},{1,0},{1,1},{1,-1}};
struct point    main_SE[]={{1,1},{1,0},{1,1}};
struct point    main_S[]={{1,0},{1,-1},{1,1}};
struct point    main_SW[]={{1,-1},{1,0},{0,-1}};
struct point    main_W[]={{0,-1},{1,-1},{-1,-1}};
struct point    main_NW[]={{-1,-1},{0,-1},{-1,0}};
struct point    main_N[]={{-1,0},{-1,1},{-1,-1}};
struct point    main_NE[]={{-1,1},{0,1},{-1,0}};
struct point    main_E[]={{0,1},{1,1},{-1,1}};

/*  orientation data array for further */

struct point    further_SE[]={{2,2},{1,2},{2,1},{0,2},{2,0},{3,3},{2,3},{3,2},{1,3},{3,1},
                                {0,3},{3,0}};
struct point    further_S[]={{2,0},{2,1},{2,-1},{2,2},{2,-2},{1,2},{1,-2},{3,0},{3,1},
                                {3,-1},{3,2},{3,-2}};
struct point    further_SW[]={{2,-2},{2,-1},{1,-2},{2,0},{0,-2},{3,-3},{3,-2},{2,-3},{3,-1},
                                {1,-3},{3,0},{0,-3}};
struct point    further_N[]={{-2,0},{-2,1},{-2,-1},{-2,2},{-2,-2},{-1,2},{-1,-2},{0,2},{0,-2},
                                {-3,0},{-3,1},{-3,-1}};
```

```
struct point    further_NE[]={{-2,2},{-1,2},{-2,1},{0,2},{-2,0},{-3,3},{-3,2},{-2,3},{-3,1},
                                {-1,3},{0,3},{-3,0}};
struct point    further_NW[]={{-2,-2},{-2,-1},{-1,-2},{-2,0},{0,-2},{-3,-3},{-2,-3},{-3,-2},
                                {-1,-3},{-3,-1},{0,-3},{-3,0}};
struct point    further_E[]={{0,2},{1,2},{-1,2},{2,2},{-2,2},{2,1},{2,0},{0,3},{1,3},{-1,3},
                                {2,3},{-2,3}};
struct point    further_W[]={{0,-2},{1,-2},{-1,-2},{2,-2},{-2,-2},{2,-1},{-2,-1},{2,0},{0,-3},
                                {-1,-3},{1,-3},{2,-3}};

static void     trace_from(int*, int*, int);
static void     further(int*, int*, int);
static int      any_direction(int*, int*);
static void     update_seg(int*, int*, int, int, int);
static void     round_up(int*, int*);
static int      convert(double);
```

6. trace.c

```
/*
 * This Program is used to trace line segments from output image of Canny
 * Operator.
 */

#include "auto.h"
#include "trace.h"

void
trace_segment(h, w)
        int             h, w;
{
        int             i, j, k, orientation;


        segment[0] = (struct point *) malloc(imageSize * sizeof(points));
        buf = segment[0];
        for (i = 1; i < h; i++)
                for (j = 1; j < w; j++) {
                        if ((k = pr_get(new_image, i, j))) {
                                segment[0]->x = i;
                                segment[0]->y = j;
                                segment[0]->z = k;
                                ++segment[0];
                                pr_put(new_image, i, j, 0);
                                orientation = O;
                                num_of_point[num_of_seg] = 0;
                                trace_from(&i, &j, orientation);
```

```
                }
        }
}

/* start tracing from point (i, j) along orientation */

static void
trace_from(i, j, orientation)
        int             *i, *j, orientation;
{
        register int    k, m, n;

        xx = yy = 0.0;

        switch (orientation) {
        case O:
                n = sizeof main_O / sizeof main_O[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + main_O.x, *j + main_O.y)) > 0) {
                                update_seg(i, j, k, main_O.x, main_O.y);
                                further(i, j, orientation);
                                round_up(i, j);
                                break;
                        }
                break;
        case SE:
                n = sizeof main_SE / sizeof main_SE[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + main_SE.x, *j + main_SE.y)) > 0) {
                                update_seg(i, j, k, main_SE.x, main_SE.y);
                                break;
                        }
                break;
        case S:
                n = sizeof main_S / sizeof main_S[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + main_S.x, *j + main_S.y)) > 0) {
                                update_seg(i, j, k, main_S.x, main_S.y);
                                break;
                        }
                break;
        case SW:
                n = sizeof main_SW / sizeof main_SW[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + main_SW.x, *j + main_SW.y)) > 0) {
                                update_seg(i, j, k, main_SW.x, main_SW.y);
```

```
                        break;
                }
        break;
    case W:
        n = sizeof main_W / sizeof main_W[0];
        for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + main_W.x, *j + main_W.y)) > 0) {
                        update_seg(i, j, k, main_W.x, main_W.y);
                        break;
                }
        break;
    case NW:
        n = sizeof main_NW / sizeof main_NW[0];
        for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + main_NW.x, *j + main_NW.y)) > 0) {
                        update_seg(i, j, k, main_NW.x, main_NW.y);
                        break;
                }
        break;
    case N:
        n = sizeof main_N / sizeof main_N[0];
        for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + main_N.x, *j + main_N.y)) > 0) {
                        update_seg(i, j, k, main_N.x, main_N.y);
                        break;
                }
        break;
    case NE:
        n = sizeof main_NE / sizeof main_NE[0];
        for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + main_NE.x, *j + main_NE.y)) > 0) {
                        update_seg(i, j, k, main_NE.x, main_NE.y);
                        break;
                }
        break;
    case E:
        n = sizeof main_E / sizeof main_E[0];
        for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + main_E.x, *j + main_E.y)) > 0) {
                        update_seg(i, j, k, main_E.x, main_E.y);
                        break;
                }
    }                   /* for switch */
}                       /* for the function */
```

```
/* find point 2-5 pixels away along the prefered direction */

static void
further(i, j, orientation)
        int             *i, *j, orientation;
{
        register int    k, m, n;

        /* fine tuned for each direction, don't try to generalize it */

        switch (orientation) {
        case SE:
                n = sizeof further_SE / sizeof further_SE[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + further_SE[m].x,
                                                *j + further_SE[m].y)) > 0) {
                                update_seg(i, j, k, further_SE[m].x, further_SE[m].y);
                                break;
                        }
                break;
        case S:
                n = sizeof further_S / sizeof further_S[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + further_S[m].x, *j + further_S[m].y)) > 0)
{
                                update_seg(i, j, k, further_S[m].x, further_S[m].y);
                                break;
                        }
                break;
        case SW:
                n = sizeof further_SW / sizeof further_SW[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + further_SW[m].x,
                                                *j + further_SW[m].y)) > 0) {
                                update_seg(i, j, k, further_SW[m].x, further_SW[m].y);
                                break;
                        }
                break;
        case N:
                n = sizeof further_N / sizeof further_N[0];
                for (m = 0; m < n; m++)
                        if ((k = pr_get(new_image, *i + further_N[m].x, *j + further_N[m].y)) >
0) {
                                update_seg(i, j, k, further_N[m].x, further_N[m].y);
                                break;
                        }
```

```
            break;
        case NE:
            n = sizeof further_NE / sizeof further_NE[0];
            for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + further_NE[m].x,
                                        *j + further_NE[m].y)) > 0) {
                    update_seg(i, j, k, further_NE[m].x, further_NE[m].y);
                    break;
                }
            break;
        case NW:
            n = sizeof further_NW / sizeof further_NW[0];
            for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + further_NW[m].x,
                                        *j + further_NW[m].y)) > 0) {
                    update_seg(i, j, k, further_NW[m].x, further_NW[m].y);
                    break;
                }
            break;
        case E:
            n = sizeof further_E / sizeof further_E[0];
            for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + further_E[m].x, *j + further_E[m].y)) > 0)
{
                    update_seg(i, j, k, further_E[m].x, further_E[m].y);
                    break;
                }
            break;
        case W:
            n = sizeof further_W / sizeof further_W[0];
            for (m = 0; m < n; m++)
                if ((k = pr_get(new_image, *i + further_W[m].x,
                                        *j + further_W[m].y)) > 0) {
                    update_seg(i, j, k, further_W[m].x, further_W[m].y);
                    break;
                }
        }                   /* for the switch   */

}                           /* for the function */


/*
 * trace along all directions if the number of points in the current segment
 * is small
 */
```

```
int
any_direction(i, j)
        int     *i, *j;
{
        int     k, ii, jj;

        for (ii = -1; ii <= 1; ii++)
                for (jj = -1; jj <= 1; jj++)
                        if ((k = pr_get(new_image, *i + ii, *j + jj))) {
                                update_seg(i, j, k, ii, jj);
                                return 1;
                        }       /* succeed in finding a bridge */
        return 0;
}


/* search along direction (ii, jj) */

static void
update_seg(i, j, k, ii, jj)
        int     *i, *j, k, ii, jj
        {
                double          sigma;

                *i = *i + ii;
                *j = *j + jj;
                segment[0]->x = *i;
                segment[0]->y = *j;
                segment[0]->z = k;
                ++segment[0];
                pr_put(new_image, *i, *j, 0);
                xx += ii;
                yy += jj;
                sigma = atan2(yy, xx);
                orientation = convert(sigma);
                ++num_of_point[num_of_seg];
                linetrace(i, j, orientation);

                return;
        }

/* explore all directions and store selected path in a new segment */

        static void
        round_up(i, j)
```

```
        int     *i, *j;
{
        register int    k;

        k = 1;
        while (k && num_of_point[num_of_seg] < 5)       /* less than 5 pixels
                                                        * away */
                k = any_direction(i, j);
        segment[num_of_seg] = (struct point *) malloc(num_of_point[num_of_seg]
                                                        * sizeof(points));
        for (k = 0; k <= num_of_point[num_of_seg]; k++) {
                *segment[num_of_seg] = *(segment[0] - num_of_point[num_of_seg] - 1 + k);
                ++segment[num_of_seg];
        }
        theta[num_of_seg] = atan(yy / xx);
        num_of_seg++;
        segment[0] = buf;

}


/* convert angle to orientation */

int
convert(sigma)
        double          sigma;
{
        register int    orientation;

        switch (nint((4 * sigma) / Pi)) {
        case 0:
                orientation = S;
                break;
        case 1:
                orientation = SE;
                break;
        case 2:
                orientation = E;
                break;
        case 3:
                orientation = NE;
                break;
        case 4:
        case -4:
                orientation = N;
                break;
```

```
        case -1:
                orientation = SW;
                break;
        case -2:
                orientation = W;
                break;
        case -3:
                orientation = NW;
                break;
        }

        return orientation;
```

7. utility.c

```
/*
 * remove background noise based on the average and standard deviation of the
 * backgroud brightness intensity which are estimated from a three-pixel wide
 * margin of the image
 */

extern void
remove_noise(orig_image)
        Pixrect         *orig_image;
{
        register int    i, j, num;
        double          sum, sum_sqr, mean, std;

        sum = sum_sqr = 0.0;

        for (i = 0; i < ncols; i++) {
                for (j = 0; j < 3; j++) {
                        sum += pr_get(orig_image, i, j);
                        sum_sqr += sqr(pr_get(orig_image, i, j));
                        num++;
                }
                for (j = nrows - 3; j < nrows; j++) {
                        sum += pr_get(orig_image, i, j);
                        sum_sqr += sqr(pr_get(orig_image, i, j));
                        num++;
                }
        }
        for (j = 0; j < nrows; j++) {
                for (i = 0; i < 3; i++) {
                        sum += pr_get(orig_image, i, j);
```

```
                sum_sqr += sqr(pr_get(orig_image, i, j));
                num++;
            }
            for (i = ncols - 3; i < ncols; i++) {
                sum += pr_get(orig_image, i, j);
                sum_sqr += sqr(pr_get(orig_image, i, j));
                num++;
            }
        }

        mean = sum / num;
        std = sum_sqr - num * SQR(mean);
        std = sqrt(std / (num - 1.0));

        /* if mean<128 (black background) and  z < mean+std then z =0 */

        if (mean < 128) {
            for (i = 0; i < ncols; i++)
                for (j = 0; j < nrows; j++)
                    if (pr_get(orig_image, i, j) < mean + std)
                        pr_set(orig_image, i, j, 0);
        } else {
            for (i = 0; i < ncols; i++)
                for (j = 0; j < nrows; j++)
                    if (pr_get(orig_image, i, j) > mean - std)
                        pr_set(orig_image, i, j, 0);
        }
}


/* compute portion of overlap between segment z0-z1 and z2-z3 */

float
overlap(z0, z1, z2, z3)
        float   z0, z1, z2, z3;
{
        int     i, j;
        float   ovlp, lap[2];

        if (z0 > z1) {
            lap[0] = z1;
            z1 = z0;
            z0 = lap[0];
        }
        if (z2 > z3) {
            lap[0] = z3;
```

```
                z3 = z2;
                z2 = lap[0];
        }
        if (z3 > z1)
                lap[0] = z1 - z2;       /* the length of the overlap */
        else
                lap[0] = z3 - z0;

        lap[1] = (z1 - z0 < z3 - z2) ? z1 - z0 : z3 - z2;       /* the length of the
                                                                 * shortest segment */

        if (tmp < 0)
                ovlp = 0.0;
        else if (!lap[1])
                ovlp = 1.0;
        else
                ovlp = lap[0] / lap[1];

        return ovlp;
}

/*
 * remove redundant segments based on complete overlap along x & y as well as
 * closeness of end-points
 */

static void
remove_redundant_segs()
{
        register int    i, j, k, j1, j2;
        float           x[5], y[5];

        for (i = 2; i < num_of_seg; i++) {
                x[1] = (newseg[seq[i]] - 1)->x;
                y[1] = (newseg[seq[i]] - 1)->y;
                x[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->x;
                y[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->y;
                for (j = 1; j < i; j++) {
                        x[3] = (newseg[seq[j]] - 1)->x;
                        y[3] = (newseg[seq[j]] - 1)->y;
                        x[4] = (newseg[seq[j]] - num_of_point[seq[j]] - 1)->x;
                        y[4] = (newseg[seq[j]] - num_of_point[seq[j]] - 1)->y;
                        if (overlap(x[1], x[2], x[3], x[4]) > 0.99 && overlap(y[1], y[2], y[3], y[4])
>                               0.99) {
                                j1 = j2 = 0;
                                for (i1 = 0; i1 <= num_of_point[seq[j]]; i1++) {
```

```
                                x[0] = (newseg[seq[j]] - i1 - 1)->x;
                                y[0] = (newseg[seq[j]] - i1 - 1)->y;
                                if (D(x[1], y[1], x[0], y[0]) < 3.0)
                                        j1 = 1;
                                if (D(x[2], y[2], x[0], y[0]) < 3.0)
                                        j2 = 1;
                                if (j1 && j2)
                                        break;
                        }
                        if (j1 && j2) {
                                num_of_point[seq[i]] = 0;
                                break;
                        }
                }
        }               /* end of j loop */
}                       /* end of i loop  */
        return;
}


/* connect short segments based on distance and colinearity */

void
connect()
{
        register int    i, j, k, x[5], y[5];
        double          tmp, sigma;

        /* allocate space for new segment array */

        newseg[0] = (struct point *) malloc(imageSize * sizeof(points));
        buf = newseg[0];        /* record where it starts */

        k = 0;
        for (i = 1; i < num_of_seg; i++)
                if (num_of_point[i]) {
                        k++;
                        x[0] = (segment[i] - 1)->x;
                        y[0] = (segment[i] - 1)->y;
                        x[1] = (segment[i] - num_of_point[i] - 1)->x;
                        y[1] = (segment[i] - num_of_point[i] - 1)->y;
                        for (j = 0; j <= num_of_point[i]; j++) {
                                *newseg[0] = *(segment[i] - num_of_point[i] - 1 + j);
                                ++newseg[0];
                        }
```

```
for (j = i + 1; j < num_of_seg; j++)
        if (num_of_point[j]) {
                tmp = fabs(theta[i] - theta[j]);
                sigma = HALFPI(tmp);
                x[2] = (segment[j] - 1)->x;
                y[2] = (segment[j] - 1)->y;
                x[3] = (segment[j] - num_of_point[j] - 1)->x;
                y[3] = (segment[j] - num_of_point[j] - 1)->y;
                if (fabs(theta[i]) < Pi / 4.0)
                        tmp = overlap(x[0], x[1], x[2], x[3]);
                else
                        tmp = overlap(y[0], y[1], y[2], y[3]);

                /*
                 * if few overlap and segment i,j are
                 * colinear, or the two segments are
                 * very short, connect the two
                 */

                if (num_of_point[j] > 2 && num_of_point[i] > 2 &&
                    sigma < Pi / 9.0 && tmp < 0.1 || num_of_point[i] < 3 ||
                    num_of_point[j] < 3) {

                        /*
                         * check the four end points
                         * of the two segments,
                         * determine how to connect
                         * them
                         */

                        if (CLOSE(x[0], y[0], x[2], y[2])) {
                                join_segment(j, 1);  /* 1 - start from front */
                        } else if (CLOSE(x[0], y[0], x[3], y[3])) {
                                join_segment(j, -1); /* 1 - start from back */
                                x[0] = x[2];
                                y[0] = y[2];
                        } else if (CLOSE(x[1], y[1], x[2], y[2])) {
                                reverse_segment(i);
                                join_segment(j, 1);
                                x[1] = x[0];
                                y[1] = y[0];
                                x[0] = x[3];
                                y[0] = y[3];
                        } else if (CLOSE(x[1], y[1], x[3], y[3])) {
                                reverse_segment(i);
                                join_segment(j, -1);
```

```
                                        x[1] = x[0];
                                        y[1] = y[0];
                                        x[0] = x[2];
                                        y[0] = y[2];
                                }
                                theta[i] = ANGLE(x[0], y[0], x[1], y[1]);
                                j = i;
                        }
                }
                newseg[k] = (struct point *) malloc(num_of_point[i] * sizeof(points));
                for (j = 0; j <= num_of_point[i]; j++) {
                        *newseg[k] = *(newseg[0] - num_of_point[i] - 1 + j);
                        ++newseg[k];
                }
                newseg[0] = buf;        /* reset the memory location
                                         * for newseg[0] */
                num_of_point[k] = num_of_point[i];
                theta[k] = theta[i];
        }
    num_of_seg = k + 1;

}

/* join segment j to newseg[0] from direction k */

static void
join_segment(j, k)
        int     j, k;
{
        register int    i;
        if (k > 0)
                for (i = 0; i <= num_of_point[j]; i++) {
                        *newseg[0] = *(segment[j] - i - 1);
                        ++newseg[0];
                }

        else
                for (i = num_of_point[j]; i >= 0; i--) {
                        *newseg[0] = *(segment[j] - i - 1);
                        ++newseg[0];
                }
        num_of_point[i] += num_of_point[j] + 1;
        num_of_point[j] = 0;
}

/* reverse the order of points in segment */
```

```
static void
reverse_segment(i)
        int             i;
{
        register int    j;

        for (j = 0; j < (num_of_point[i] + 1) / 2; j++) {
                *(newseg[0] - j) = *(newseg[0] - num_of_point[i] - 1 + j);
                *(newseg[0] - num_of_point[i] - 1 + j) = *(newseg[0] - j - 1);
        }
        for (j = 0; j < (num_of_point[i] + 1) / 2; j++)
                *(newseg[0] - (num_of_point[i] + 1) / 2 + j) =
                        *(newseg[0] - (num_of_point[i] + 1) / 2 + 1 + j);

}

/*
 * order segments based on number of points in the segment, eliminate the one
 * with less than MINLEN points and save order information in seq[]
 */

static void
order_segment()
{
        register int    i, j, k;

        for (i = 1; i < num_of_seg; i++)
                seq[i] = i;
        for (i = 1; i < num_of_seg; i++)
                for (j = i + 1; j < num_of_seg; j++)
                        if (num_of_point[seq[i]] < num_of_point[seq[j]]) {
                                k = seq[i];
                                seq[i] = seq[j];
                                seq[j] = k;
                        }
        for (i = 1; i < num_of_seg; i++)
                if (num_of_point[seq[i]] < MINLEN)
                        num_of_seg--;

}

/*
 * locate coiling axis according to support from rib and coiling curve
 * candidates
 */
```

```
static void
locate_coiling_axis(ncols, nrows)
        int             ncols, nrows;
{
        register int    i, j, k, x[5], y[5];
        double          sigma[2], contrib[2];

        contrib[1] = 0;
        for (i = x[0] - ncols / 4; i < x[0] + ncols / 4; i += 5)   /* 5 - scanning
                                                                    * resolution */
                for (j = y[0] - nrows / 4; j < y[0] + nrows / 4; j += 5)
                        if (i > 0 && j > 0) {
                                contrib[0] = 0;
                                for (k = 1; k < num_of_seg; k++) {
                                        x[1] = (newseg[seq[k]] - num_of_point[seq[k]] / 2)->x - i;
                                        y[1] = (newseg[seq[k]] - num_of_point[seq[k]] / 2)->y - j;
                                        if (x[1]) {
                                                sigma[0] = atan((double) y[1] / x[1]);
                                                sigma[1] = fabs(theta[seq[k]] - sigma[0]);
                                                sigma[0] = HALFPI(sigma[1]);
                                        } else {        /* middle point vector
                                                         * is in vertical
                                                         * position */
                                                sigma[0] = POSITIVE(theta[seq[k]]);
                                        }
                                        if (sigma[0] < Pi / 12.0)       /* a rib candidate */
                                                contrib[0] += num_of_point[seq[k]];
                                        else if (num_of_point[seq[k]] < imageSize / 6) {
                                                if (sigma[0] > 0.4 * Pi)        /* short coiling curve
                                                                                 * segment */
                                                        contrib[0] += num_of_point[seq[k]];
                                        } else {        /* get a short segment
                                                         * of the long coiling
                                                         * curve candidate */
                                                x[1] = (newseg[seq[k]] - num_of_point[seq[k]] / 2
                                                  + imageSize / 20)->x - (newseg[seq[k]] -
                                                  num_of_point[seq[k]] / 2 - imageSize / 20)->x;
                                                y[1] = (newseg[seq[k]] - num_of_point[seq[k]] / 2
                                                  + imageSize / 20)->y - (newseg[seq[k]] -
                                                  num_of_point[seq[k]] / 2 - imageSize / 20)->y;
                                                if (x[1]) {
                                                  sigma[1] = fabs(Pi / 2 - atan((double) y[1] / x[1]));
                                                  sigma[0] = HALFPI(sigma[1]);
                                                } else
                                                        sigma[0] = 0;
```

```
                                        if (sigma[0] > 0.4 * Pi)
                                                contrib[0] += num_of_point[seq[k]];
                                }
                        }
                        if (contrib[0] > contrib[1]) {  /* the location geting
                                                         * the largest support
                                                         * is coiling axis */
                                contrib[1] = contrib[0];
                                xo = i;
                                yo = j;
                        }
                }
}

int
label_seg()
{
        register int    i, k, x[5], y[5];
        double          sigma;

        for (i = 1; i < num_of_seg; i++)
                label[i] = 0;
        for (i = 1; i < num_of_seg; i++) {
                x[1] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->x - xo;
                y[1] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->y - yo;
                if (x[1]) {
                        sigma = fabs(theta[seq[i]] - atan(y[1] / x[1]));
                        sigma = HALFPI(sigma);
                } else
                        sigma = TRUEDIF(Pi / 2, theta[seq[i]]);
                theta[seq[i]] = POSITIVE(atan2(y[1], x[1]));
                if (sigma < RIBANG)
                        label[i] = RIB;
                else if (sigma > COILANG)
                        label[i] = COILING;
                else
                        label[i] = UNKNOWN;
        }
        k = 0;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] == RIB)
                        k++;

        if (k > 0.3 * num_of_seg)
                return RIB;     /* ribbed form */
        else
```

```
                return COILING;      /* smooth shell */

}

bright_between(i0, j0, x0, y0, r0, r1)
        int     i0, j0, x0, y0;
        float   r0, r1;
{
        register int    i, j, i1 = 0, j1, x[5], y[5], f[400][100], count[400];
        float           s[5], n[5];

        for (i = 0; i < 5; i++)
                n[i] = 0.0;
        for (i = 0; i <= num_of_point[seq[i0]]; i++) {
                x[1] = (newseg[seq[i0]] - i - 1)->x;
                y[1] = (newseg[seq[i0]] - i - 1)->y;
                s[1] = D(x0, y0, x[1], y[1]);
                if (s[1] >= r0 && s[1] <= r1) {
                        for (j = 0; j <= num_of_point[seq[j0]]; j++) {
                                x[2] = (newseg[seq[j0]] - j - 1)->x;
                                y[2] = (newseg[seq[j0]] - j - 1)->y;
                                s[2] = D(x0, y0, x[2], y[2]);
                                if (fabs(s[2] - s[1]) < 1.0)
                                        break;
                        }
                        s[3] = atan2((double) y[1] - y0, (double) x[1] - x0);
                        s[4] = atan2((double) y[2] - y0, (double) x[2] - x0);
                        if (fabs(s[3] - s[4]) < Pi / 4.0) {
                                if (s[3] > s[4]) {
                                        s[0] = s[3];
                                        s[3] = s[4];
                                        s[4] = s[0];
                                }
                        } else {
                                if (s[3] < 0.0) {
                                        s[0] = s[4];
                                        s[4] = 2 * Pi + s[3];
                                        s[3] = s[4];
                                } else if (s[4] < 0.0)
                                        s[4] += 2 * Pi;
                        }
                        j1 = 1;
                        i1++;
                        for (s[0] = s[3]; s[0] <= s[4]; s[0] += 0.5 / s[1]) {
                                x[3] = x0 + s[1] * cos(s[0]);
                                y[3] = y0 + s[1] * sin(s[0]);
```

```
                        if (!(x[3] == x[4] && y[3] == y[4])) {
                                x[4] = x[3];
                                y[4] = y[3];
                                f[i1][j1++] = pr_get(x[3], y[3]);
                        }
                }
                count[i1] = j1;
        }
}
for (i = 1; i <= i1; i++) {
        n[0] = count[i] / 2;
        n[1] += f[i][(int) n[0]];
        n[2] += SQR(f[i][(int) n[0]]);
}
if (i1) {
        n[3] = n[1] / i1;
        s[0] = n[2] + i1 * SQR(n[3]) - 2 * n[3] * n[1];
        n[4] = 2 * sqrt(s[0] / i1);
        n[1] = n[2] = n[0] = 0.0;
        for (i = 1; i <= i1; i++) {
                j1 = 0;
                for (j = 1; j <= count[i] / 2; j++)
                        if (j1 || f[i][j] < (int) n[3] + n[4] && f[i][j] > (int) n[3] - n[4]) {
                                j1 = 1;
                                n[1] += f[i][j];
                                n[2] += SQR(f[i][j]);
                                n[0] += 1.0;
                        }
        }
        for (i = 1; i <= i1; i++) {
                j1 = 0;
                for (j = count[i] - 1; j > count[i] / 2; j--)
                        if (j1 || f[i][j] < (int) n[3] + n[4] && f[i][j] > (int) n[3] - n[4]) {
                                j1 = 1;
                                n[1] += f[i][j];
                                n[2] += SQR(f[i][j]);
                                n[0] += 1.0;
                        }
        }
        if (n[0] > 4.0)
                n[3] = n[1] / n[0];
        else
                return -2;
        s[0] = n[2] + n[0] * SQR(n[3]) - 2 * n[3] * n[1];
        s[0] = sqrt(s[0] / n[0]);
        if (s[0] < threshold)
```

```
                return (int) n[3];
        else
                return -1;
    } else
        return -2;
}

bright_nearby(i0, j0, x0, y0, r0, i2)
    int     i0, j0, i2, x0, y0;
    float   r0;
{
    register int   i, j, i1 = 0, j1, x[5], y[5], f[400][100], count[400];
    float          s[5], n[5];
    for (i = 0; i < 5; i++)
        n[i] = 0.0;
    s[2] = 0.0;
    for (i = 0; i <= num_of_point[seq[i0]]; i++) {
        x[1] = (newseg[seq[i0]] - i - 1)->x;
        y[1] = (newseg[seq[i0]] - i - 1)->y;
        s[1] = D(x0, y0, x[1], y[1]);
        if (fabs(s[1] - s[2]) > 0.5) {
            s[2] = s[1];
            s[3] = atan2((double) y[1] - y0, (double) x[1] - x0);
            j1 = 1;
            i1++;
            if (j0 < 0)
                for (s[0] = s[3]; s[0] <= s[3] + r0; s[0] += 0.5 / s[1]) {
                    x[3] = x0 + s[1] * cos(s[0]);
                    y[3] = y0 + s[1] * sin(s[0]);
                    if (!(x[3] == x[4] && y[3] == y[4])) {
                        x[4] = x[3];
                        y[4] = y[3];
                        f[i1][j1++] = pr_get(orig_image, x[3], y[3]);
                    }
                }
            else
                for (s[0] = s[3]; s[0] >= s[3] - r0; s[0] -= 0.5 / s[1]) {
                    x[3] = x0 + s[1] * cos(s[0]);
                    y[3] = y0 + s[1] * sin(s[0]);
                    if (!(x[3] == x[4] && y[3] == y[4])) {
                        x[4] = x[3];
                        y[4] = y[3];
                        f[i1][j1++] = pr_get(orig_image, x[3], y[3]);
                    }
                }
            count[i1] = j1;
```

```
            }
        }
        for (i = 1; i <= i1; i++) {
            n[0] = count[i] / 2;
            n[1] += f[i][(int) n[0]];
            n[2] += SQR(f[i][(int) n[0]]);
        }
        if (i1) {
            n[3] = n[1] / i1;
            s[0] = n[2] + i1 * SQR(n[3]) - 2 * n[3] * n[1];
            n[4] = 2 * sqrt(s[0] / i1);
            n[1] = n[2] = n[0] = 0;
            for (i = 1; i <= i1; i++) {
                j1 = 0;
                for (j = 1; j <= count[i] / 2; j++)
                    if (j1 || f[i][j] < (int) n[3] + n[4] && f[i][j] > (int) n[3] - n[4]) {
                        j1 = 1;
                        n[1] += f[i][j];
                        n[2] += SQR(f[i][j]);
                        n[0] += 1.0;
                    }
            }
            for (i = 1; i <= i1; i++) {
                j1 = 0;
                for (j = count[i] - 1; j >.count[i] / 2; j--)
                    if (j1 || f[i][j] < (int) n[3] + n[4] && f[i][j] > (int) n[3] - n[4]) {
                        j1 = 1;
                        n[1] += f[i][j];
                        n[2] += SQR(f[i][j]);
                        n[0] += 1.0;
                    }
            }
            if (n[0] > 4.0)
                n[3] = n[1] / n[0];
            else
                return -2;
            s[0] = n[2] + n[0] * SQR(n[3]) - 2 * n[3] * n[1];
            s[0] = sqrt(s[0] / n[0]);
            if (i2 || s[0] < threshold)
                return (int) n[3];
            else
                return -1;
        } else
            return -2;
}
```

```
countpoints_between(i0, j0)
        int     i0, j0;
{
        register int    i, j, j1, n = 0, x[5], y[5];

        if (theta[seq[i0]] > 0.25 * Pi && theta[seq[i0]] < 0.75 * Pi || theta[seq[i0]] > 1.25 * Pi
                && theta[seq[i0]] < 1.75 * Pi)
                for (i = 0; i <= num_of_point[seq[i0]]; i++) {
                        x[1] = (newseg[seq[i0]] - i - 1)->x;
                        y[1] = (newseg[seq[i0]] - i - 1)->y;
                        x[2] = 0;
                        for (j = 0; j <= num_of_point[seq[j0]]; j++)
                                if (y[1] == (newseg[seq[j0]] - j - 1)->y) {
                                        x[2] = (newseg[seq[j0]] - j - 1)->x;
                                        break;
                                }
                        if (x[2]) {
                                if (x[1] > x[2]) {
                                        x[3] = x[2];
                                        x[2] = x[1];
                                        x[1] = x[3];
                                }
                                for (x[3] = x[1] + 3; x[3] < x[2] - 2; x[3]++)
                                        if (pr_get(x[3], y[1])) {
                                                j1 = 1;
                                                for (j = 0; j <= num_of_point[seq[i0]]; j++)
                                                        if (x[3] == (newseg[seq[i0]] - j - 1)->x &&
                                                                y[1] == (newseg[seq[i0]] - j - 1)->y) {
                                                                j1 = 0;
                                                                break;
                                                        }
                                                if (j1)
                                                        for (j = 0; j <= num_of_point[seq[j0]]; j++)
                                                                if (x[3] == (newseg[seq[j0]] - j - 1)->x
                                                                  && y[1] == (newseg[seq[j0]] - j -
                                                                  1)->y) {
                                                                        j1 = 0;
                                                                        break;
                                                                }
                                                if (j1)
                                                        n++;
                                        }
                        }
                }
        else
                for (i = 0; i <= num_of_point[seq[i0]]; i++) {
```

```
                        x[1] = (newseg[seq[i0]] - i - 1)->x;
                        y[1] = (newseg[seq[i0]] - i - 1)->y;
                        y[2] = 0;
                        for (j = 0; j <= num_of_point[seq[j0]]; j++)
                                if (x[1] == (newseg[seq[j0]] - j - 1)->x) {
                                        y[2] = (newseg[seq[j0]] - j - 1)->y;
                                        break;
                                }
                        if (y[2]) {
                                if (y[1] > y[2]) {
                                        y[3] = y[2];
                                        y[2] = y[1];
                                        y[1] = y[3];
                                }
                                for (y[3] = y[1] + 3; y[3] < y[2] - 2; y[3]++)
                                        if (pr_get(x[1], y[3])) {
                                                j1 = 1;
                                                for (j = 0; j <= num_of_point[seq[i0]]; j++)
                                                        if (x[1] == (newseg[seq[i0]] - j - 1)->x &&
                                                                y[3] == (newseg[seq[i0]] - j - 1)->y) {
                                                                j1 = 0;
                                                                break;
                                                        }
                                                if (j1)
                                                        for (j = 0; j <= num_of_point[seq[j0]]; j++)
                                                                if (x[1] == (newseg[seq[j0]] - j - 1)->x
                                                                && y[3] == (newseg[seq[j0]] - j -
                                                                1)->y) {
                                                                        j1 = 0;
                                                                        break;
                                                                }
                                                if (j1)
                                                        n++;
                                        }
                        }
                }
        return n;
}

getarg(desig, argc, argv)
        int     argc;
        char    *desig, *argv[];
{
        int     argn;
        char    minus[2], lookfor[12];
```

```
        strcpy(minus, "-");
        strcpy(lookfor, minus);
        strcat(lookfor, desig);

        for (argn = 1; argn < argc; argn++)
                if (!strcmp(argv[argn], lookfor))
                        return (argn);

        return (0);
}
```

8. smooth_form.c

```
/* This program processes ammonites with few ribs. It was written several years ago and was in
poor         shape. It is a typical example of bad coding practice, difficult to read and maintain */

#include "auto.h"

process_smooth_form()
{
        register int    i, j, k, x[110], y[110];
        float           tmp[6], x[1] = xx[1][pos[i2]];
        int             ORIENTATION;

        y[1] = yy[1][pos[i2]];
        x[2] = xx[1][pos[i2] + n[i2] - 1];
        y[2] = yy[1][pos[i2] + n[i2] - 1];
        r[0] = D(xo, yo, x[1], y[1]);
        r[100] = D(xo, yo, x[2], y[2]);
        if (r[100] > r[0])
                ORIENTATION = 0;
        else
                ORIENTATION = 1;
        n[201] = n[100] / 20;
        if (n[i2] > n[201]) {
                x[2] = xx[1][pos[i2] + n[201]];
                y[2] = yy[1][pos[i2] + n[201]];
                r[100] = D(xo, yo, x[2], y[2]);
        }
        if (x[1] - xo)
                tmp[1] = atan2((float) (y[1] - yo), (float) (x[1] - xo));
        else
                tmp[1] = (y[1] > yo) ? Pi / 2.0 : -Pi / 2.0;
        if (x[2] - xo)
                tmp[2] = atan2((float) (y[2] - yo), (float) (x[2] - xo));
```

```
else
        tmp[2] = (y[2] > yo) ? Pi / 2.0 : -Pi / 2.0;
if (tmp[1] * tmp[2] > 0.0)
        tmp[0] = fabs(tmp[2] - tmp[1]);
else
        tmp[0] = LESSPI(fabs(tmp[2] - tmp[1]));
tmp[3] = fabs(log(r[100] / r[0]) / tmp[0]);
j = 1;
for (i = 1; i <= num_of_point[0]; i++)
        if (j && len[i] > pos[i2] - 2) {
                n[101] = i;
                j = 0;
        } else if (!j && len[i] > pos[i2] + n[i2] - 3) {
                n[102] = i;
                break;
        }
for (i = 0; i <= 72; i++)
        ox[i] = oy[i] = -1.0;
n[201] = num_of_point[0] / 20;
ox[80] = 10.0;
for (j = 0; j < n[201] / 3; j++) {
        x[1] = (segment[0] - n[101] - 1 - j)->x;
        y[1] = (segment[0] - n[101] - 1 - j)->y;
        r[0] = D(xo, yo, x[1], y[1]);
        if (n[i2] > n[201])
                n[202] = n[101] + n[201] - j;
        else
                n[202] = n[102] - j;
        x[2] = (segment[0] - n[202])->x;
        y[2] = (segment[0] - n[202])->y;
        r[1] = D(xo, yo, x[2], y[2]);
        if (x[1] - xo)
                tmp[1] = atan2((float) (y[1] - yo), (float) (x[1] - xo));
        else
                tmp[1] = (y[1] > yo) ? Pi / 2.0 : -Pi / 2.0;
        if (x[2] - xo)
                tmp[2] = atan2((float) (y[2] - yo), (float) (x[2] - xo));
        else
                tmp[2] = (y[2] > yo) ? Pi / 2.0 : -Pi / 2.0;
        if (tmp[1] * tmp[2] > 0.0)
                tmp[0] = fabs(tmp[2] - tmp[1]);
        else
                tmp[0] = LESSPI(fabs(tmp[2] - tmp[1]));
        if (ORIENTATION && r[1] < r[0]) {
                ox[0] = log(r[0] / r[1]) / tmp[0];
                if (fabs(ox[0] - tmp[3]) < ox[80]) {
```

```
                        ox[36] = ox[0];
                        oy[36] = -tmp[0];
                        ox[80] = fabs(ox[0] - tmp[3]);
                        j1 = j;
                }
        } else if (!ORIENTATION && r[1] > r[0]) {
                ox[0] = log(r[1] / r[0]) / tmp[0];
                if (fabs(ox[0] - tmp[3]) < ox[80]) {
                        ox[36] = ox[0];
                        oy[36] = tmp[0];
                        ox[80] = fabs(ox[0] - tmp[3]);
                        j1 = j;
                }
        }
}
a[0] = r[0];
n[202] += n[201];
i2 = 37;
x[3] = x[2];
y[3] = y[2];
r[3] = r[1];
tmp[3] = tmp[2];
j0 = 0;
for (i = n[202]; i < num_of_point[0]; i += n[201]) {
        ox[80] = ox[i2 - 1];
        for (j = 0; j < n[201] / 2; j++) {
                x[4] = (segment[0] - i + j)->x;
                y[4] = (segment[0] - i + j)->y;
                r[4] = D(xo, yo, x[4], y[4]);
                if (x[4] - xo)
                        tmp[4] = atan2((float) (y[4] - yo), (float) (x[4] - xo));
                else
                        tmp[4] = (y[4] > yo) ? Pi / 2.0 : -Pi / 2.0;
                if (tmp[3] * tmp[4] > 0.0)
                        tmp[0] = fabs(tmp[4] - tmp[3]);
                else
                        tmp[0] = LESSPI(fabs(tmp[4] - tmp[3]));
                if (ORIENTATION && r[4] < r[3]) {
                        ox[0] = log(r[3] / r[4]) / tmp[0];
                        if (fabs(ox[0] - ox[i2 - 1]) < ox[80]) {
                                ox[80] = fabs(ox[0] - ox[i2 - 1]);
                                ox[i2] = ox[0];
                                oy[i2] = -tmp[0];
                                x[6] = x[4];
                                y[6] = y[4];
                                r[6] = r[4];
```

```
                        tmp[5] = tmp[4];
                    }
                } else if (!ORIENTATION && r[4] > r[3]) {
                    ox[0] = log(r[4] / r[3]) / tmp[0];
                    if (fabs(ox[0] - ox[i2 - 1]) < ox[80]) {
                        ox[80] = fabs(ox[0] - ox[i2 - 1]);
                        ox[i2] = ox[0];
                        oy[i2] = tmp[0];
                        x[6] = x[4];
                        y[6] = y[4];
                        r[6] = r[4];
                        tmp[5] = tmp[4];
                    }
                }
            }
            if (ox[80] < ox[i2 - 1]) {
                x[3] = x[6];
                y[3] = y[6];
                r[3] = r[6];
                tmp[3] = tmp[5];
                i2++;
            }
        }
        if (i2 == 37) {
            if (oy[36] > 0.0) {
                x[99] = x[2];
                y[99] = y[2];
                c[0] = tmp[2];
            } else {
                x[98] = x[2];
                y[98] = y[2];
            }
        } else {
            if (oy[36] > 0.0) {
                x[99] = x[3];
                y[99] = y[3];
                c[0] = tmp[3];
            } else {
                x[98] = x[3];
                y[98] = y[3];
            }
        }
        n[202] = n[101] + j1 - n[201];
        i1 = 35;
        x[3] = x[1];
        y[3] = y[1];
```

```
tmp[3] = tmp[1];
r[3] = r[0];
for (i = n[202]; i > 0; i -= n[201]) {
        ox[80] = ox[i1 + 1];
        for (j = 0; j < n[201] / 2; j++) {
                x[4] = (segment[0] - i - j)->x;
                y[4] = (segment[0] - i - j)->y;
                r[4] = D(xo, yo, x[4], y[4]);
                c1 = 1;
                if (x[4] - xo)
                        tmp[4] = atan2((float) (y[4] - yo), (float) (x[4] - xo));
                else
                        tmp[4] = (y[4] > yo) ? Pi / 2.0 : -Pi / 2.0;
                if (tmp[3] * tmp[4] > 0.0)
                        tmp[0] = fabs(tmp[4] - tmp[3]);
                else
                        tmp[0] = LESSPI(fabs(tmp[4] - tmp[3]));
                if (ORIENTATION && r[4] > r[3]) {
                        ox[0] = log(r[4] / r[3]) / tmp[0];
                        if (fabs(ox[0] - ox[i1 + 1]) < ox[80]) {
                                ox[80] = fabs(ox[0] - ox[i1 + 1]);
                                ox[i1] = ox[0];
                                oy[i1] = tmp[0];
                                x[6] = x[4];
                                y[6] = y[4];
                                r[6] = r[4];
                                tmp[5] = tmp[4];
                        }
                } else if (!ORIENTATION && r[4] < r[3]) {
                        ox[0] = log(r[3] / r[4]) / tmp[0];
                        if (fabs(ox[0] - ox[i1 + 1]) < ox[80]) {
                                ox[80] = fabs(ox[0] - ox[i1 + 1]);
                                ox[i1] = ox[0];
                                oy[i1] = -tmp[0];
                                x[6] = x[4];
                                y[6] = y[4];
                                r[6] = r[4];
                                tmp[5] = tmp[4];
                        }
                }
        }
        if (ox[80] < ox[i1 + 1]) {
                x[3] = x[6];
                y[3] = y[6];
                r[3] = r[6];
                tmp[3] = tmp[5];
```

```
                i1--;
        }
}
if (i1 == 35) {
        if (oy[36] < 0.0) {
                x[99] = x[1];
                y[99] = y[1];
                c[0] = tmp[1];
        } else {
                x[98] = x[1];
                y[98] = y[1];
        }
} else {
        if (oy[36] < 0.0) {
                x[99] = x[3];
                y[99] = y[3];
                c[0] = tmp[3];
        } else {
                x[98] = x[3];
                y[98] = y[3];
        }
}

if (!ORIENTATION)
        ORIENTATION = -1;
tmp[0] = 0.0;
for (i = i1 + 1; i < i2; i++)
        tmp[0] += fabs(oy[i]);
r[0] = D(xo, yo, x[98], y[98]);
r[100] = D(xo, yo, x[99], y[99]);
tmp[1] = x[99] - xo;
tmp[2] = y[99] - yo;
tmp[3] = atan2(tmp[2], tmp[1]);
angle = POSITIVE(tmp[3]);
a[0] = fabs(log(r[100] / r[0]) / tmp[0]);
b[0] = log(r[100]) / a[0];
r[0] = r[100] / exp(a[0] * 2 * Pi);
c1 = 0;
i2 = 1;
for (i = 1; i < i1; i++) {
        j1 = 0;
        x[21] = (newseg[seq[i]] - 1)->x;
        y[21] = (newseg[seq[i]] - 1)->y;
        x[22] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->x;
        y[22] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->y;
        r[1] = D(xo, yo, x[21], y[21]);
```

```
        if (r[1] < r[0]) {
                r[200] = D(xo, yo, x[22], y[22]);
                tmp[1] = atan2((double) y[21] - yo, (double) x[21] - xo);
                tmp[1] = POSITIVE(tmp[1]);
                tmp[2] = atan2((double) y[22] - yo, (double) x[22] - xo);
                tmp[2] = POSITIVE(tmp[2]);
                tmp[3] = fabs(tmp[2] - tmp[1]);
                if (tmp[3] > Pi) {
                        tmp[3] = 2 * Pi - tmp[3];
                        if (ORIENTATION > 0 && (tmp[1] > tmp[2] && r[200] > r[1] ||
                                tmp[1] < tmp[2] && r[200] < r[1]))
                                j1 = 1;
                        else if (ORIENTATION < 0 && (tmp[1] > tmp[2] && r[200] <
                                r[1] || tmp[1] < tmp[2] && r[200] > r[1]))
                                j1 = 1;
                } else {
                        if (ORIENTATION > 0 && (tmp[1] > tmp[2] && r[200] < r[1] ||
                                tmp[1] < tmp[2] && r[200] > r[1]))
                                j1 = 1;
                        else if (ORIENTATION < 0 && (tmp[1] > tmp[2] && r[200] >
                                r[1] || tmp[1] < tmp[2] && r[200] < r[1]))
                                j1 = 1;
                }
                tmp[4] = fabs(log(r[200] / r[1])) / tmp[3];
                if (j1 && tmp[4] < 3 * a[0]) {
                        label[i] = -2;
                        x[i2] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->x;
                        y[i2] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->y;
                        tmp[1] = x[i2] - xo;
                        tmp[2] = y[i2] - yo;
                        tmp[2] = atan2(tmp[2], tmp[1]);
                        theta[seq[i]] = POSITIVE(tmp[2]);
                        i2++;
                        if (i2 > 10)
                                break;
                }
        }
}
for (i = 1; i < i1; i++)
        if (label[i] > 0)
                label[i] = 0;
tmp[0] = 0.0;
for (i = 1; i < i2 - 1; i++)
        for (j = i + 1; j < i2; j++) {
                r[200] = D(xo, yo, x[j], y[j]);
                r[201] = D(xo, yo, x[i], y[i]);
```

```
tmp[1] = atan2((double) y[j] - yo, (double) x[j] - xo);
tmp[1] = POSITIVE(tmp[1]);
tmp[2] = atan2((double) y[i] - yo, (double) x[i] - xo);
tmp[2] = POSITIVE(tmp[2]);
tmp[4] = 0.0;
if (r[201] > r[200]) {
        for (j0 = 0; j0 < 5; j0++) {
                if (ORIENTATION > 0 && (tmp[0] = 2 * j0 * Pi + tmp[2]
                        tmp[1]) > Pi / 3)
                        tmp[4] = log(r[201] / r[200]) / tmp[0];
                else if (ORIENTATION < 0 && (tmp[0] = 2 * j0 * Pi +
                        tmp[1] - tmp[2]) > Pi / 3)
                        tmp[4] = log(r[201] / r[200]) / tmp;
                if (tmp[4] > a[0] / 3 && tmp[4] < 1.5 * a[0]) {
                        c[1] = tmp[4];
                        d[1] = log(r[201]) / c[1];
                        tmp[4] = log(r[100]) / c[1] + Pi / 2.0;
                        tmp[3] = tmp[2];
                        j1 = 0;
                        for (i0 = (tmp[4] - d[1]) / (2 * Pi) - 2; i0 <= (tmp[4]
                                - d[1]) / (2 * Pi) + 2; i0++) {
                                r[4] = exp(c[1] * (d[1] + 2 * Pi * i0 +
                                        ORIENTATION * (angle - tmp[3])));
                                if (r[4] < r[100] + (r[100] - r[0]) / 2 &&
                                        r[4] > (r[100] + r[0]) / 2) {
                                        j1 = 1;
                                        break;
                                }
                        }
                        if (j1) {
                                tmp[1] = tmp[2] = 0.0;
                                for (j1 = 1; j1 < i1; j1++)
                                        if (label[j1] == -1) {
                                          x[190] = (newseg[seq[j1]] - 1)->x;
                                          y[190] = (newseg[seq[j1]] - 1)->y;
                                          x[191] = (newseg[seq[j1]] -
                                              num_of_point[seq[j1]] - 1)->x;
                                          y[191] = (newseg[seq[j1]] -
                                              num_of_point[seq[j1]] - 1)->y;
                                         r[190] = D(xo, yo, x[190], y[190]);
                                         r[191] = D(xo, yo, x[191], y[191]);
                                         if (r[190] > r[191]) {
                                                r[192] = r[191];
                                                r[191] = r[190];
                                                r[190] = r[192];
                                                }
```

```
                                for (i0 = -d[1] / (2 * Pi) - 1.0; i0 <=
                                        (tmp[4] - d[1]) / (2 * Pi) + 1; i0++) {
                                        r[2] = exp(c[1] * (d[1] + 2 * Pi * i0 +
                                                ORIENTATION *(theta[seq[j1]] - tmp[3])));
                                        if (r[2] < r[0] && r[2] > r[190] && r[2] < r[191]) {
                                                tmp[2] += num_of_point[seq[j1]];
                                                break;
                                                }
                                        }
                                } else if (label[j1] == -2) {
                                                x[190] = (newseg[seq[j1]] -
                                                        num_of_point[seq[j1]] / 2)->x;
                                                y[190] = (newseg[seq[j1]] -
                                                num_of_point[seq[j1]] / 2)->y;
                                                r[190] = D(xo, yo, x[190], y[190]);
                                                for (i0 = -d[1] / (2 * Pi) - 1; i0 <= (tmp[4] -
                                                        d[1]) / (2 * Pi) + 1.0; i0++) {
                                                        r[2] = exp(c[1] * (d[1] + 2 * Pi * i0
                                                                + ORIENTATION *
                                                                (theta[seq[j1]] - tmp[3])));
                                                if (r[2] < r[0] && fabs(r[2] - r[190]) < 5.0)
{
                                                        tmp[1] += num_of_point[seq[j1]];
                                                        break;
                                                        }
                                                }
                                        }
                                if (tmp[1] / tmp[2] > tmp[0]) {
                                        tmp[0] = tmp[1] / tmp[2];
                                        c[0] = c[1];
                                        d[0] = d[1];
                                        sigma = tmp[3];
                                }
                        }
                }
        }
} else {
        for (j0 = 0; j0 < 5; j0++) {
            if (ORIENTATION > 0 && (tmp[0] = 2 * j0 * Pi + tmp[1] - tmp[2]) > Pi / 3)
                        tmp[4] = log(r[200] / r[201]) / tmp[0];
                else if (ORIENTATION < 0 && (tmp[0] = 2 * j0 * Pi + tmp[2] - tmp[1]) > Pi
                        / 3)
                        tmp[4] = log(r[200] / r[201]) / tmp[0];
                    if (tmp[4] > a[0] / 3 && tmp[4] < 1.5 * a[0]) {
                        c[1] = tmp[4];
                        d[1] = log(r[200]) / c[1];
```

```
            tmp[4] = log(r[100]) / c[1] + Pi / 2.0;
            tmp[3] = tmp[1];
            for (i0 = (tmp[4] - d[1]) / (2 * Pi) - 2; i0 <= (tmp[4] - d[1]) / (2 * Pi) + 2; i0++) {
                r[4] = exp(c[1] * (d[1] + 2 * Pi * i0 +
                ORIENTATION * (angle - tmp[3])));
                if (r[4] < r[100] + (r[100] - r[0]) / 2 && r[4] > (r[100] + r[0]) / 2) {
                j1 = 1;
                break;
            }
        }
            if (j1) {
                tmp[1] = tmp[2] = 0.0;
                for (j1 = 1; j1 < i1; j1++)
                    if (label[j1] == -1) {
                        x[190] = (newseg[seq[j1]] - 1)->x;
                        y[190] = (newseg[seq[j1]] - 1)->y;
                        x[191] = (newseg[seq[j1]] - num_of_point[seq[j1]] - 1)->x;
                        y[191] = (newseg[seq[j1]] - num_of_point[seq[j1]] - 1)->y;
                        r[190] = D(xo, yo, x[190], y[190]);
                        r[191] = D(xo, yo, x[191], y[191]);
                        if (r[190] > r[191]) {
                            r[192] = r[191];
                            r[191] = r[190];
                            r[190] = r[192];
                        }
                        for (i0 = -d[1] / (2 * Pi) - 1.0; i0 <= (tmp[4] - d[1]) / (2 *
                            Pi) + 1.0; i0++) {
                            r[2] = exp(c[1] * (d[1] + 2 * Pi * i0 +
                            ORIENTATION * (theta[seq[j1]] - tmp[3])));
                            if (r[2] < r[0] && r[2] > r[190] && r[2] < r[191]) {
                                tmp[2] += num_of_point[seq[j1]];
                                break;
                            }
                        }
                    } else if (label[j1] == -2) {
                        x[190] = (newseg[seq[j1]] - num_of_point[seq[j1]] / 2)->x;
                        y[190] = (newseg[seq[j1]] - num_of_point[seq[j1]] / 2)->y;
                        r[190] = D(xo, yo, x[190], y[190]);
                        for (i0 = -d[1] / (2 * Pi) - 1.0; i0 <= (tmp[4] - d[1]) / (2 *
                            Pi) + 1.0; i0++) {
                            r[2] = exp(c[1] * (d[1] + 2 * Pi * i0 +
                            ORIENTATION * (theta[seq[j1]] - tmp[3])));
                            if (r[2] < r[0] && fabs(r[2] - r[190]) < 5.0) {
                                tmp[1] += num_of_point[seq[j1]];
                                break;
                            }
```

```
                                        }
                                }
                        if (tmp[1] / tmp[2] > tmp[0]) {
                                tmp[0] = tmp[1] / tmp[2];
                                c[0] = c[1];
                                d[0] = d[1];
                                sigma = tmp[3];
                                        }
                                }
                                }
                        }
                }
        tmp[4] = log(r[100]) / c[0] + Pi;
        tmp[0] = r[100];
        for (tmp[0] = 0.0; tmp[0] < tmp[4]; tmp[0] += 0.5 / r[1]) {
                r[1] = exp(c[0] * tmp[0]);
                i1 = xo + r[1] * cos(sigma + ORIENTATION * (tmp[0] - d[0]));
                j1 = yo + r[1] * sin(sigma + ORIENTATION * (tmp[0] - d[0]));
                pr_put(new_image, i1, j1, 1);
        }
        pr_dump(new_image, fpo, NULL, RT_STANDARD, 1);
        pr_close(orig_image);
        pr_close(new_image);
        pr_close(screen[0]);
        pr_close(screen[1]);
        system("stime");
}
```

9. ribbed_form.c

```
#include "auto.h"

void
process_ribbed_form()
{
        register int      i, j, k;

        /* coiling curves are nearly normal to radius vectors */

        confirm_coiling_curve();

        /* adjacent ribs should have large overlap along x or y direction */

        confirm_rib_by_overlap();
```

```
        /* divide into 12 sectors around coiling axis */

        analysis_by_sector();

        /* remove small ribs located within Pi/10 of big ribs */
        remove_small_rib();

        confine_coiling_curve();

        simulate_morph();

}

/* coiling curves are nearly normal to radius vectors (<75) */

static void
confirm_coiling_curve();
{
        register int    i, j, x[5], y[5];
        double          r[5], sigma[2], w;

        for (i = 1; i < num_of_seg; i++)
                if (label[i] != RIB) {
                        j = 0;
                        x[1] = (newseg[seq[i]] - 1)->x;
                        y[1] = (newseg[seq[i]] - 1)->y;
                        x[2] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->x;
                        y[2] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->y;
                        r[1] = D(xo, yo, x[1], y[1]);
                        r[2] = D(xo, yo, x[2], y[2]);
                        sigma[0] = atan2((double) y[1] - yo, (double) x[1] - xo);
                        sigma[0] = POSITIVE(sigma[0]);
                        sigma[1] = atan2((double) y[2] - yo, (double) x[2] - xo);
                        sigma[1] = POSITIVE(sigma[1]);
                        sigma[0] = fabs(sigma[1] - sigma[0]);
                        sigma[0] = LESSPI(sigma[0]);
                        w = fabs(log(r[2] / r[1])) / sigma[0];  /* expansion rate */
                        if (w < 1 / tan(75 * Pi / 180)) {
                                label[i] = COILING;
                                x[2] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->x;
                                y[2] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->y;
                                sigma[0] = atan2((double) y[2] - yo, (double) x[2] - xo);
                                theta[seq[i]] = POSITIVE(sigma[0]);
                        }
                }
}
```

```
/* adjacent ribs should have large overlap along x or y direction */

static void
confirm_rib_by_overlap();
{
        register int    i, j, x[5], y[5];
        register int    num_lap;/* number of ribs that overlap with the
                                 * current rib */
        double          delta;
        double          d[100];/* distance between ribs */

        for (i = 1; i < num_of_seg; i++)
                if (label[i] == RIB) {
                        x[1] = (newseg[seq[i]] - 1)->x;
                        y[1] = (newseg[seq[i]] - 1)->y;
                        x[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->x;
                        y[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->y;
                        num_lap = 1;

                        /*
                         * the length of each rib should be consistent with
                         * its neighboring ribs
                         */

                        for (j = i + 1; j < num_of_seg; j++)
                                if (num_of_point[seq[j]] > 0.4 * num_of_point[seq[i]] && label[j]
                                        == RIB) {
                                        delta = fabs(theta[seq[i]] - theta[seq[j]]);
                                        delta = LESSPI(delta);
                                        if (delta < Pi / 10) {
                                            x[3] = (newseg[seq[j]] - 1)->x;
                                            y[3] = (newseg[seq[j]] - 1)->y;
                                            x[4] = (newseg[seq[j]] - num_of_point[seq[j]] - 1)->x;
                                            y[4] = (newseg[seq[j]] - num_of_point[seq[j]] - 1)->y;
                                            if (abs(y[4] - y[3]) <= abs(x[4] - x[3]) / 10)
                                                d[num_lap] = y[1] - y[3];       /* distance between
                                                                                 *the two ribs */
                                            else
                                                d[num_lap] = (x[1] - x[3] - (x[4] - x[3]) * (y[1] -
                                                        y[3]) / (y[4] - y[3])) * (y[2] - y[1]) / D(x[1],
                                                        y[1], x[2], y[2]);
                                            if (abs(d[num_lap]) < 2 * num_of_point[seq[j]]) {
                                                if (abs(y[2] - y[1]) > abs(x[2] - x[1]) &&
                                                        overlap(y[1], y[2], y[3], y[4]) > 0.4)
                                                        num_lap++;
```

```
                                        else if (abs(y[2] - y[1]) <= abs(x[2] - x[1]) &&
                                        overlap(x[1], x[2], x[3], x[4]) > 0.4)
                                                num_lap++;
                                }
                        }
                }
        if (num_lap < 4)        /* doesn't get enough support */
                label[i] = UNKNOWN;
                }
}

/* find the farthest and the second farthest ribs in each of the 12 sectors */

static void
analysis_by_sector()
{
        register int    i, j, x[5], y[5];
        double          tmp[2];

        for (i = 1; i < num_of_seg; i++)
                if (label[i] == RIB || label[i] == UNKNOWN) {
                        x[0] = (newseg[seq[i]] - 1)->x;
                        y[0] = (newseg[seq[i]] - 1)->y;
                        x[1] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->x;
                        y[1] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->y;
                        tmp[0] = D(xo, yo, x[0], y[0]);
                        tmp[1] = D(xo, yo, x[1], y[1]);
                        if (tmp[0] > tmp[1]) {
                                r[2 * i] = tmp[0];
                                r[2 * i - 1] = tmp[1];
                        } else {
                                r[2 * i] = tmp[0];
                                r[2 * i - 1] = tmp[1];
                        }
                }
        for (i = 0; i < 100; i++) {
                rep[i] = 0;
                ox[i] = 0.0;
        }
        j = 0;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] == -1 || label[i] > 0) {
                        tmp[0] += num_of_point[seq[i]];
                        tmp[1] += r[2 * i - 1];
                        j++;
                        if (j > 6)
```

```
                        break;
                }
        tmp[0] /= j;            /* tha average length of the six longest rib
                                 * segments */
        ribToaxis /= j;         /* tha average distance of the six longest
                                 * rib segments from coiling axis */
        j = tmp[0] / 4;
        tmp[0] = Pi / 6.0;
        for (i = 1; i < num_of_seg; i++)
                if ((label[i] == RIB || label[i] == UNKNOWN) && num_of_point[seq[i]] > j) {
                        k = theta[seq[i]] / tmp[0] + 1;
                        if (r[2 * i] > ox[k])
                                ox[k] = r[2 * i];
                }
        for (i = 1; i < num_of_seg; i++)
                if ((label[i] == -1 || label[i] > 0) && num_of_point[seq[i]] > j) {
                        k = theta[seq[i]] / tmp[0];
                        if (r[2 * i] > ox[20 + k] && r[2 * i] < ox[k]) {
                                ox[20 + k] = r[2 * i];
                                rep[k] = i;
                        }
                }
}

static void
remove_small_rib()
{
        register int    i, j, k, big, rt, lt;
        double          tmp[2];

        for (i = 0; i < 12; i++)
                if (ox[20 + i] > 0.0) {
                        k = (i + 1 < 12) ? i + 1 : i + 1 - 12;
                        if (ox[20 + big] > 0.0) {
                                tmp[1] = fabs(theta[seq[rep[i]]] - theta[seq[rep[big]]]);
                                if (tmp[1] < Pi / 10.0 || tmp[1] > 2 * Pi - Pi / 12.0) {
                                        if (ox[20 + i] < ox[20 + big])
                                                ox[20 + i] = 0.0;
                                        else
                                                ox[20 + big] = 0.0;
                                }
                        }
                }
        tmp[1] = ox[20 + 0];
        big = 0;
        for (i = 1; i < 12; i++)
```

```
            if (ox[20 + i] > tmp[1]) {
                    tmp[1] = ox[20 + i];
                    big = i;
            }               /* big is Big rib */
    for (i = 1; i < 12; i++) {
            rt = (big + i < 12) ? big + i : big + i - 12;
            if (rep[rt])
                    break;
    }                       /* Big rib's right neighbour */
    for (i = 1; i < 12; i++) {
            lt = (big - i < 0) ? 12 + big - i : big - i;
            if (rep[lt])
                    break;
    }                       /* Big rib's left neighbour */

    /*
     * if a rib candidate is closer to coiling axis then both its left
     * and right ribs, it should be removed
     */

    for (i = 0; i < 12; i++)
            if (rep[i] && i != rt && i != lt) {
                    for (j = 1; j < 12; j++) {
                            k = (i + j < 12) ? i + j : i + j - 12;
                            if (rep[k])
                                    break;
                    }
                    for (j = 1; j < 12; j++) {
                            k = (i - j < 0) ? 12 + i - j : i - j;
                            if (rep[k])
                                    break;
                    }
                    if (ox[20 + i] < ox[20 + k] && ox[20 + i] < ox[20 + k])
                            ox[20 + i] = 0.0;
            }
    tmp[0] = tmp[1] = 0;
    for (i = 0; i < 12; i++)
            if (ox[20 + i] > 0.0 && i != big)
                    for (j = 1; j < 12; j++) {
                            rt = (i + j < 12) ? i + j : i + j - 12;
                            if (ox[20 + rt] > 0.0) {
                                    delta = theta[seq[rep[rt]]] - theta[seq[rep[i]]];
                                    delta = POSITIVE(delta);
                                    if (ox[20 + rt] < ox[20 + i] && ox[20 + rt] > ox[20 + i] /
exp(0.4                                             * delta))
                                            tmp[0] += ox[20 + i];
```

```
                    else if (ox[20 + rt] > ox[20 + i] && ox[20 + rt] < ox[20 + i] *
                            exp(0.4 * tmp[1]))
                            tmp[1] += ox[20 + i];
                    break;
                }
        }

if (tmp[0] > tmp[1]) {
        ORIENTATION = 1;/* right-lateral */
        for (j = 1; j < 11; j++) {
                rt = (big - j < 0) ? 12 + big - j : big - j;
                if (ox[20 + rt] > 0.0) {
                        delta = theta[seq[rep[big]]] - theta[seq[rep[rt]]];
                        delta = POSITIVE(delta);
                        if (ox[20 + rt] > ribToaxis * exp(0.4 * delta))
                                ox[20 + rt] = 0.0;
                        else
                                break;
                }
        }
        for (i = 0; i < 12; i++)
                if (ox[20 + i] > 0.0 && i != rt) {
                        for (j = 1; j < 12; j++) {
                                lt = (i + j < 12) ? i + j : i + j - 12;
                                if (ox[20 + lt] > 0.0)
                                        break;
                        }
                        delta = theta[seq[rep[lt]]] - theta[seq[rep[i]]];
                        delta = POSITIVE(delta);
                        if (ox[20 + i] < ox[20 + lt])
                                ox[20 + i] = 0.0;
                        else if (ox[20 + lt] < ox[20 + i] / exp(0.4 * delta))
                                ox[20 + lt] = 0.0;
                }
} else {
        ORIENTATION = -1;/* left-lateral */
        for (j = 1; j < 11; j++) {
                rt = (big + j < 12) ? big + j : big + j - 12;
                if (ox[20 + rt] > 0.0) {
                        delta = theta[seq[rep[rt]]] - theta[seq[rep[big]]];
                        delta = POSITIVE(delta);
                        if (r[2 * rep[rt]] > ribToaxis * exp(0.4 * delta))
                                ox[20 + rt] = 0.0;
                        else
                                break;
                }
```

```
                }
                for (i = 0; i < 12; i++)
                        if (ox[20 + i] > 0.0 && i != rt) {
                                for (j = 1; j < 12; j++) {
                                        lt = (i - j < 0) ? 12 + i - j : i - j;
                                        if (ox[20 + lt] > 0.0)
                                                break;
                                }
                                delta = theta[seq[rep[i]]] - theta[seq[rep[lt]]];
                                delta = POSITIVE(delta);
                                if (ox[20 + i] < ox[20 + lt])
                                        ox[20 + i] = 0.0;
                                else if (ox[20 + lt] < ox[20 + i] / exp(0.4 * delta))
                                        ox[20 + lt] = 0.0;
                        }
        }
}

static void
confine_coiling_curve()
{
        register int    i, j, k, i0, x[110], y[110];
        double          tmp[5];

        tmp[1] = Pi * imageSize / 100.0;
        for (i0 = 0; i0 < 12; i0++)
                if (ox[20 + i0] > 0.0) {
                        tmp[3] = tmp[4] = imageSize / 15;
                        k = 0;
                        x[19] = (newseg[seq[rep[i0]]] - 1)->x;
                        y[19] = (newseg[seq[rep[i0]]] - 1)->y;
                        x[20 + i0] = (newseg[seq[rep[i0]]] - num_of_point[seq[rep[i0]]] - 1)->x;
                        y[20 + i0] = (newseg[seq[rep[i0]]] - num_of_point[seq[rep[i0]]] - 1)->y;
                        if (D(xo, yo, x[19], y[19]) > D(xo, yo, x[20 + i0], y[20 + i0])) {
                                x[20 + i0] = x[19];
                                y[20 + i0] = y[19];
                        }
                        for (i = 1; i < num_of_seg; i++)
                                if (label[i] == -2 && num_of_point[seq[i]] > tmp[1]) {
                                        x[100] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->x;
                                        y[100] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->y;
                                        ox[100] = D(xo, yo, x[100], y[100]);
                                        tmp[2] = fabs(theta[seq[rep[i0]]] - theta[seq[i]]);
                                        tmp[2] = LESSPI(tmp[2]);
                                        ox[101] = imageSize / 15;
                                        if (ox[100] > ox[20 + i0] / 2.0 && tmp[2] < Pi / 4.0)
```

```
                                        for (j = 0; j <= num_of_point[seq[i]]; j++) {
                                                x[100] = (newseg[seq[i]] - j - 1)->x;
                                                y[100] = (newseg[seq[i]] - j - 1)->y;
                                                ox[99] = D(xo, yo, x[100], y[100]);
                                                if (ox[99] > ox[20 + i0]) {
                                                        ox[100] = D(x[20 + i0], y[20 + i0],
                                                                x[100], y[100]);
                                                        if (ox[100] < ox[101]) {
                                                                ox[101] = ox[100];
                                                                x[101] = x[100];
                                                                y[101] = y[100];
                                                        }
                                                }
                                        }
                                if (ox[101] < tmp[3]) {
                                        tmp[3] = ox[101];
                                        x[102] = x[101];
                                        y[102] = y[101];
                                        rep[20 + i0] = i;
                                } else if (ox[101] < tmp[4]) {
                                        tmp[4] = ox[101];
                                        x[103] = x[101];
                                        y[103] = y[101];
                                        k = i;
                                }
                        }
                if (k > 0) {
                        if (tmp[4] * num_of_point[seq[rep[20 + i0]]] < tmp[3] *
                                        num_of_point[seq[k]]) {
                                rep[20 + i0] = k;
                                x[20 + i0] = x[103];
                                y[20 + i0] = y[103];
                        } else if (rep[20 + i0] > 0) {
                                x[20 + i0] = x[102];
                                y[20 + i0] = y[102];
                        }
                } else if (rep[20 + i0] > 0) {
                        x[20 + i0] = x[102];
                        y[20 + i0] = y[102];
                }
        }
}

/*
 * This program was written several years ago, typical spaghetti code, very
 * difficult to read. It may need a few weeks to clean it up
```

```
*/

static void
simulate_morph()
{
        int     x[200], y[200], pos[1000], n[200];
        int     var1, var2;
        float   sigma, angle, sclefact, xx[2][MAXNUM], yy[2][MAXNUM];
        float   a[200], b[200], c[200], d[200];         /* morph parameters */
        float   r[MAXNUM], ox[MAXNUM], oy[MAXNUM];
        float   tmp[6];
        register        i, j, i1, j1, i0, j0, i2;

        for (i = 0; i < 12; i++)
                if (ox[20 + i] > 0.0) {
                        ox[20 + i] = D(xo, yo, x[20 + i], y[20 + i]);
                        oy[20 + i] = atan2((double) y[20 + i] - yo, (double) x[20 + i] - xo);
                        oy[20 + i] = POSITIVE(oy[20 + i]);
                }
        ox[1] = tmp[2] = ox[20 + n[3]];
        oy[1] = tmp[1] = oy[20 + n[3]];
        x[1] = x[20 + n[3]];
        y[1] = y[20 + n[3]];
        i0 = 1;
        for (i = 1; i < 12; i++) {
                if (ORIENTATION > 0)
                        n[4] = (n[3] + i < 12) ? n[3] + i : n[3] + i - 12;
                else
                        n[4] = (n[3] - i < 0) ? 12 + n[3] - i : n[3] - i;
                if (ox[20 + n[4]] > 0.0) {
                        if (ORIENTATION > 0)
                                tmp[3] = oy[20 + n[4]] - tmp[1];
                        else
                                tmp[3] = tmp[1] - oy[20 + n[4]];
                        tmp[3] = POSITIVE(tmp[3]);
                        tmp[4] = log(tmp[2] / ox[20 + n[4]]);
                        if (tmp[4] < 0.05 * tmp[3] || tmp[4] > 0.4 * tmp[3])
                                ox[20 + n[4]] = 0.0;
                        else if (i0 > 2 && (tmp[4] > 3.0 * tmp[3] * a[i0 - 1] || tmp[4] < tmp[3] *
                                a[i0 - 1] / 3.0))
                                ox[20 + n[4]] = 0.0;
                        else {
                                a[i0] = tmp[4] / tmp[3];
                                i0++;
                                x[i0] = x[20 + n[4]];
                                y[i0] = y[20 + n[4]];
```

```
                        tmp[2] = ox[i0] = ox[20 + n[4]];
                        tmp[1] = oy[i0] = oy[20 + n[4]];
                }
        }
}
a[0] = tmp[4] = 0.0;
b[100] = ox[1];
i0++;
for (i = 2; i < i0; i++) {
        tmp[1] = log(ox[i]) / a[i - 1];
        tmp[2] = log(ox[i - 1]) / a[i - 1];
        tmp[3] = oy[i - 1];
        if (i == 2)
                c[100] = oy[0] = tmp[3];
        a[99 + i] = a[i - 1];
        b[99 + i] = ox[i];
        c[99 + i] = oy[i];
        tmp[4] += tmp[2] - tmp[1];
        a[0] += a[i - 1] * (tmp[2] - tmp[1]);
        for (tmp[0] = tmp[1]; tmp[0] <= tmp[2]; tmp[0] += 0.5 / ox[i - 1]) {
                r[0] = exp(a[i - 1] * tmp[0]);
                i2 = xo + r[0] * cos(tmp[3] - ORIENTATION * (tmp[0] - tmp[2]));
                j0 = yo + r[0] * sin(tmp[3] - ORIENTATION * (tmp[0] - tmp[2]));
                pr_put(new_image, i2, j0, 1);
        }
        printf("i=%d k=%f \n", i - 1, a[i - 1]);
}
a[0] /= tmp[4];
var2 = i0 - 1;
for (i = 1; i < num_of_seg - 1; i++)
        if (label[i] > 0 && label[i] != 2) {
                rep[100] = i;
                j1 = 101;
                for (j = i + 1; j < num_of_seg; j++)
                        if (label[j] > 0) {
                                tmp[3] = (r[2 * i] - r[2 * i - 1] > 120.0) ? (r[2 * i] - r[2 * i -
                                        1]) / 20.0 : 6.0;
                                tmp[4] = fabs(r[2 * j] - r[2 * i]);
                                if (tmp[4] < tmp[3]) {
                                        tmp[3] = fabs(theta[seq[i]] - theta[seq[j]]);
                                        tmp[3] = LESSPI(tmp[3]);
                                        if (tmp[3] < Pi / 10.0)
                                                rep[j1++] = j;
                                }
                        }
                if (j1 > 102)
```

```
                    for (j = 100; j < j1; j++)
                        label[rep[j]] = 2;
        }
for (i = 1; i < num_of_seg - 1; i++)
        if (label[i] > 0 && label[i] != 3) {
                rep[100] = i;
                j1 = 101;
                for (j = i + 1; j < num_of_seg; j++)
                        if (label[j] > 0) {
                                tmp[3] = (r[2 * i] - r[2 * i - 1] > 120.0) ? (r[2 * i] - r[2 * i -
                                        1]) / 20.0 : 6.0;
                                tmp[4] = fabs(r[2 * j - 1] - r[2 * i - 1]);
                                if (tmp[4] < tmp[3]) {
                                        tmp[3] = fabs(theta[seq[i]] - theta[seq[j]]);
                                        tmp[3] = LESSPI(tmp[3]);
                                        if (tmp[3] < Pi / 10.0)
                                                rep[j1++] = j;
                                }
                        }
                if (j1 > 102) {
                        for (j = 100; j < j1; j++)
                                if (label[rep[j]] == 2)
                                        label[rep[j]] = 4;
                                else
                                        label[rep[j]] = 3;
                }
        }
j = 0;
for (i = 1; i < num_of_seg; i++)
        if (label[i] > 2) {
                ox[100 + j] = r[2 * i - 1];
                j++;
                if (j > 6)
                        break;
        }
for (i = 100; i < 106; i++)
        for (j = i + 1; j < 107; j++)
                if (ox[i] < ox[j]) {
                        ox[107] = ox[i];
                        ox[i] = ox[j];
                        ox[j] = ox[107];
                }
ox[107] = 0.0;
for (i = 101; i < 106; i++)
        ox[107] += ox[i];
ox[107] /= 5.0;
```

```
tmp[0] = ox[107] + (ox[1] - ox[107]) / 15.0;
n[4] = n[5] = 0;
for (i = 1; i < num_of_seg; i++)
        if ((label[i] == 3 || label[i] == 4) && r[2 * i - 1] < ox[107]) {
                n[4]++;
                for (j = 1; j < num_of_seg; j++)
                        if (label[j] > 0 && r[2 * i - 1] > r[2 * j] && r[2 * i - 1] - r[2 * j] <
                                r[2 * i] - r[2 * i - 1]) {
                                tmp[0] = fabs(theta[seq[i]] - theta[seq[j]]);
                                tmp[0] = (tmp[0] > 1.8 * Pi) ? 2 * Pi - tmp[0] : tmp[0];
                                if (tmp[0] < Pi / 18.0) {
                                        n[5]++;
                                        break;
                                }
                        }
        }
if (n[5] > 0.2 * n[4]) {
        tmp[0] = 0.0;
        tmp[1] = 2 * Pi * imageSize / 100.0;
        for (i = 1; i < 500; i++) {
                rep[i] = 0;
                ox[i] = 1000.0;
        }
        i0 = 1;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] == 2 || label[i] == 4) {
                        tmp[2] = Pi / 18;
                        for (j = 1; j < num_of_seg; j++)
                                if (label[j] > 0 && r[2 * j - 1] > r[2 * i] && r[2 * j - 1] -
                                        r[2 * i] < r[2 * i] - r[2 * i - 1]) {
                                        tmp[0] = fabs(theta[seq[i]] - theta[seq[j]]);
                                        tmp[0] = (tmp[0] > 1.8 * Pi) ? 2 * Pi - tmp[0] :
                                                tmp[0];
                                        if (tmp[0] < tmp[2]) {
                                                tmp[2] = tmp[0];
                                                rep[2 * i0] = j;
                                                ox[i0] = r[2 * j - 1] - r[2 * i];
                                        }
                                }
                        if (ox[i0] < r[2 * i] - r[2 * i - 1]) {
                                rep[2 * i0 - 1] = i;
                                i0++;
                        }
                }
        for (i = 1; i < num_of_seg; i++)
                if (label[i] == 3 || label[i] == 4) {
```

```
                tmp[2] = Pi / 18;
                for (j = 1; j < num_of_seg; j++)
                        if (label[j] > 0 && r[2 * i - 1] > r[2 * j] && r[2 * i - 1] -
                                r[2 * j] < r[2 * i] - r[2 * i - 1]) {
                                tmp[0] = fabs(theta[seq[i]] - theta[seq[j]]);
                                tmp[0] = (tmp[0] > 1.8 * Pi) ? 2 * Pi - tmp[0] :
                                        tmp[0];
                                if (tmp[0] < tmp[2]) {
                                        tmp[2] = tmp[0];
                                        rep[2 * i0] = j;
                                        ox[i0] = r[2 * i - 1] - r[2 * j];
                                }
                        }
                if (ox[i0] < r[2 * i] - r[2 * i - 1]) {
                        rep[2 * i0 - 1] = i;
                        i0++;
                }
        }
        for (i = 1; i < 2 * i0 - 1; i++)
                pos[i] = i;
        for (i = 1; i < i0 - 1; i++)
                for (j = i + 1; j < i0; j++)
                        if (ox[i] > ox[j]) {
                                ox[0] = ox[i];
                                ox[i] = ox[j];
                                ox[j] = ox[0];
                                pos[0] = pos[i];
                                pos[i] = pos[j];
                                pos[j] = pos[0];
                        }

        /* remove false rib-between points */

for (i = 1; i < i0; i++) {
        tmp[3] = theta[seq[rep[2 * pos[i]]]];
        if (r[2 * rep[2 * pos[i]]] > r[2 * rep[2 * pos[i] - 1]])
                tmp[4] = (r[2 * rep[2 * pos[i] - 1]] + r[2 * rep[2 * pos[i]] - 1]) / 2.0;
        else
                tmp[4] = (r[2 * rep[2 * pos[i]]] + r[2 * rep[2 * pos[i] - 1] - 1]) / 2.0;
        tmp[1] = tmp[2] = 0.0;
        for (j = 1; j < num_of_seg; j++)
                if (label[j] > 1) {
                        tmp[0] = fabs(theta[seq[j]] - tmp[3]);
                        tmp[0] = LESSPI(tmp[0]);
                                if (tmp[0] < Pi / 4.0) {
```

```
                        if (r[2 * j] > tmp[4] && r[2 * j - 1] < tmp[4] && tmp[4] -
                                r[2 * j - 1] > 0.10 * (r[2 * j] - r[2 * j - 1]) &&
                                tmp[4] - r[2 * j - 1] < 0.90 * (r[2 * j] - r[2 * j - 1]))
                                tmp[2] += r[2 * j] - r[2 * j - 1];
                        else if (fabs(r[2 * j] - tmp[4]) < 0.1 * (r[2 * j] - r[2 * j - 1])
                                || fabs(r[2 * j - 1] - tmp[4]) < 0.1 * (r[2 * j] - r[2 * j
                                - 1]))
                                tmp[1] += r[2 * j] - r[2 * j - 1];
                }
        }
if (tmp[2] >= tmp[1] / 2.0)
        ox[i] = 1000.0;
}
i2 = 1;
n[3] = 1;       /* number of control points-xx-yy */
for (i = 1; i < i0; i++)
        if (ox[i] < 1000.0) {
                j0 = 0;
                tmp[3] = yy[0][n[3]] = (theta[seq[rep[2 * pos[i]]]] + theta[seq[rep[2 *
                        pos[i] - 1]]]) / 2.0;
                if (r[2 * rep[2 * pos[i]]] > r[2 * rep[2 * pos[i] - 1]]) {
                        xx[0][n[3]] = r[2 * rep[2 * pos[i] - 1]];
                        xx[1][n[3]] = r[2 * rep[2 * pos[i]] - 1];
                } else {
                        xx[0][n[3]] = r[2 * rep[2 * pos[i]]];
                        xx[1][n[3]] = r[2 * rep[2 * pos[i] - 1] - 1];
                }
                tmp[4] = (xx[0][n[3]] + xx[1][n[3]]) / 2.0;
                n[3]++;
                tmp[0] = fabs(tmp[3] - oy[0]);
                tmp[0] = LESSPI(tmp[0]);
                if (tmp[0] < Pi / 4.0)
                        tmp[1] = tmp[0];
                else {
                        if (ORIENTATION > 0) {
                                if (tmp[3] > oy[0])
                                        tmp[1] = tmp[3] - oy[0];
                                else
                                        tmp[1] = 2 * Pi - oy[0] + tmp[3];
                        } else if (ORIENTATION < 0) {
                                if (tmp[3] < oy[0])
                                        tmp[1] = oy[0] - tmp[3];
                                else
                                        tmp[1] = 2 * Pi - tmp[3] + oy[0];
                        }
                        tmp[0] = tmp[0] / exp(a[0] * tmp[1] / 2.0);
```

```
                }
                if (tmp[4] < tmp[0] && i2 == 1)
                        j0 = 1;
                else if (tmp[4] < tmp[0]) {
                        j0 = 1;
                        for (j = 1; j < i2; j++) {
                                tmp[0] = fabs(tmp[3] - oy[j]);
                                tmp[0] = LESSPI(tmp[0]);
                                if (tmp[0] < Pi / 6.0 && fabs(log(oy[100 + j] / tmp[4])) <
                                        exp(0.3)) {
                                        j0 = 0;
                                        break;
                                }
                        }
                }
                if (j0) {
                        n[100 + 2 * i2] = rep[2 * pos[i]];
                        n[100 + 2 * i2 - 1] = rep[2 * pos[i] - 1];
                        x[i2] = xo + tmp[4] * cos(tmp[3]);
                        y[i2] = yo + tmp[4] * sin(tmp[3]);
                        oy[i2] = tmp[3];
                        ox[i2] = ox[i];
                        oy[i2 + 100] = tmp[4];
                        i2++;
                }
                if (i2 > 10)
                        break;
                else if (i2 > 5 && ox[i] > imageSize / 20.0)
                        break;
}
tmp[0] = 1000.0;
for (i = 1; i < num_of_seg; i++)
        if (label[i] > 2 && r[2 * i - 1] < tmp[0]) {
                tmp[0] = r[2 * i - 1];
                n[100 + 2 * i2 - 1] = i;
        }
x[100] = (newseg[seq[n[100 + 2 * i2 - 1]]] - 1)->x;
y[100] = (newseg[seq[n[100 + 2 * i2 - 1]]] - 1)->y;
x[101] = (newseg[seq[n[100 + 2 * i2 - 1]]] - num_of_point[seq[n[100 + 2 * i2 - 1]]] -
        1)->x;
y[101] = (newseg[seq[n[100 + 2 * i2 - 1]]] - num_of_point[seq[n[100 + 2 * i2 - 1]]] -
        1)->y;
oy[i2] = theta[seq[n[100 + 2 * i2 - 1]]];
ox[i2] = imageSize / 60;
tmp[1] = D(xo, yo, x[100], y[100]);
tmp[2] = D(xo, yo, x[101], y[101]);
```

```
                if (tmp[1] < tmp[2]) {
                        x[i2] = x[100];
                        y[i2] = y[100];
                        oy[i2 + 100] = tmp[1] - 3.0;
                } else {
                        x[i2] = x[101];
                        y[i2] = y[101];
                        oy[i2 + 100] = tmp[2] - 3.0;
                }
                for (i = 1; i <= i2; i++)
                        pos[i] = i;
                for (i = 1; i < i2; i++)
                        for (j = i + 1; j <= i2; j++)
                                if (ox[i] > ox[j]) {
                                        ox[0] = ox[i];
                                        ox[i] = ox[j];
                                        ox[j] = ox[0];
                                        pos[0] = pos[i];
                                        pos[i] = pos[j];
                                        pos[j] = pos[0];
                }
d[62] = ox[i2 - 1];
for (i = 1; i < i2 - 1; i++)
        if (ox[i] < 100.0)
                for (j = i + 1; j < i2; j++)
                        if (ox[j] < 100.0) {
                                tmp[1] = oy[100 + pos[i]];
                                tmp[2] = oy[100 + pos[j]];
                                if (tmp[1] > tmp[2]) {
                                        tmp[0] = tmp[1];
                                        tmp[1] = tmp[2];
                                        tmp[2] = tmp;
                                        tmp[0] = oy[pos[i]];
                                        tmp[5] = oy[pos[j]];
                                } else {
                                        tmp[0] = oy[pos[j]];
                                        tmp[5] = oy[pos[i]];
                                        d[60] = 1.0;
                                        d[61] = 0.0;
                                        tmp[0] = 0.0;
                                        for (i0 = -4; i0 < 5; i0++) {
                                                d[50 + i0] = log(tmp[2] / tmp[1]) / (2 * Pi * i0 -
                                                        ORIENTATION * (tmp[0] - tmp[5]));
                                                if (d[50 + i0] < 0.3 && d[50 + i0] > 0.05) {
                                                        n[4] = n[5] = n[6] = 1;
                                                        for (i1 = 1; i1 < n[3]; i1++)
```

```
                                        if (!(xx[0][i1] > tmp[2] || xx[1][i1] <
                                        tmp[1])) {
                                                for (j1 = 0; j1 < 5; j1++) {
                                                        tmp[3] = 2 * Pi * j1 -
                                ORIENTATION * (tmp[0] - yy[0][i1]);
                                tmp[4] = tmp[2] / exp(d[50 + i0] * tmp[3]);
                                if (tmp[4] < tmp[2] && tmp[4] > tmp[1] &&
                                tmp[4] > xx[0][i1] && tmp[4] < xx[1][i1]) {
                                                        n[4]++;
                                                        break;
                                                }
                                                if (tmp[4] < tmp[1])
                                                        break;
                                        }
                                }
                        for (i1 = 1; i1 < num_of_seg; i1++)
                                if (label[i1] > 0 && !(r[2 * i1 - 1] > tmp[2] && r[2
                                        * i1] < tmp[1])) {
                                        r[0] = (r[2 * i1] - r[2 * i1 - 1]) / 8.0;
                                        for (j1 = 0; j1 < 5; j1++) {
                        tmp[3] = 2 * Pi * j1 - ORIENTATION * (tmp[0] - theta[seq[i1]]);
                        tmp[4] = tmp[2] / exp(d[50 + i0] * tmp[3]);
                        if (tmp[4] > tmp[1] && tmp[4] < tmp[2]) {
                                tmp[3] = MIN(fabs(tmp[4] - r[2 * i1]), fabs(tmp[4] - r[2 *
                                        i1 - 1]));
                                if (tmp[4] > r[2 * i1 - 1] + r[0] && tmp[4] < r[2 * i1] -
                                        r[0]) {
                                        n[5] += num_of_point[seq[i1]];
                                        break;
                                } else if (tmp[3] < d[62])
                                        n[6] += num_of_point[seq[i1]];
                        }
                        if (tmp[4] < tmp[1])
                                break;
                }
                }
                tmp[3] = (float) n[6] / n[5];
                if (tmp[3] > d[61] || tmp[3] == d[61] && fabs(d[50 + i0] - a[0]) < d[60])
                        {
                        d[60] = fabs(d[50 + i0] - a[0]);
                        tmp[0] = d[50 + i0];
                        d[61] = tmp[3];
                }
        }
}
if (d[60] > 0.4 || tmp[0] > 0.3)
```

```
                ox[j] = 100.0;
        else {
                for (i0 = j + 1; i0 <= i2; i0++)
                        if (oy[100 + pos[i0]] > tmp[1] && oy[100 + pos[i0]] < tmp[2]) {
                                if (r[2 * n[100 + 2 * pos[i0]]] > r[2 * n[100 + 2 * pos[i0] - 1]]) {
                                        tmp[3] = r[2 * n[100 + 2 * pos[i0] - 1]];
                                        tmp[5] = r[2 * n[100 + 2 * pos[i0]] - 1];
                                } else {
                                        tmp[3] = r[2 * n[100 + 2 * pos[i0]]];
                                        tmp[5] = r[2 * n[100 + 2 * pos[i0] - 1] - 1];
                                }
                                for (j0 = -4; j0 < 5; j0++) {
                                        tmp[0] = 2 * Pi * j0 - ORIENTATION * (oy[pos[i]] -
                                                oy[pos[i0]]);
                                        tmp[0] = oy[100 + pos[i]] * exp(tmp[0] * tmp);
                                        if (tmp[0] > tmp[3] && tmp[0] < tmp[5])
                                                ox[i0] = 100.0;
                                }
                        }
                }
                                        }
for (i = 101; i <= 100 + i2; i++)
        pos[i] = i - 100;
        for (i = 1; i <= i2; i++)
                if (ox[i] == 100.0)
                        pos[100 + pos[i]] = 0;
        for (i = 1; i < i2; i++)
                if (pos[100 + i] > 0)
                        for (j = i + 1; j <= i2; j++)
                                if (pos[100 + j] > 0)
                                        if (oy[pos[100 + i] + 100] < oy[pos[100 + j] + 100]) {
                                                pos[100] = pos[100 + i];
                                                pos[100 + i] = pos[100 + j];
                                                pos[100 + j] = pos[100];
                                        }
        j0 = 1;
        for (i = 101; i < 100 + i2; i++)
                if (pos[i] > 0) {
                        for (j = i + 1; j <= 100 + i2; j++)
                                if (pos[j] > 0) {
                                        i0 = j;
                                        break;
                                }
                        tmp[3] = oy[100 + pos[i]];
                        tmp[4] = oy[100 + pos[i0]];
                        a[j0] = 0.0;
```

```
                d[61] = 0.0;
                d[60] = 1.0;
                for (j = 0; j < 5; j++) {
                        d[50 + j] = log(tmp[3] / tmp[4]) / (2 * Pi * j - ORIENTATION *
                                (oy[pos[i]] - oy[pos[i0]]));
                        if (d[50 + j] < 0.3 && d[50 + j] > 0.05) {
                                n[4] = n[5] = n[6] = 1;
                                for (i1 = 1; i1 < n[3]; i1++)
                                        if (!(xx[0][i1] > tmp[3] || xx[1][i1] < tmp[4])) {
                                                for (j1 = 0; j1 < 5; j1++) {
                                                        tmp[1] = 2 * Pi * j1 -
                                                        ORIENTATION * (oy[pos[i]] -
                                                        yy[0][i1]);
                                                        tmp[2] = tmp[3] / exp(d[50 + j] *
                                                                tmp[1]);
                                                        if (tmp[2] > tmp[4] && tmp[2] <
                                                        tmp[3] && tmp[2] > xx[0][i1] &&
                                                        tmp[2] < xx[1][i1]) {
                                                                n[4]++;
                                                                break;
                                                        }
                                                        if (tmp[2] < tmp[4])
                                                                break;
                }
        }
for (i1 = 1; i1 < num_of_seg; i1++)
        if (label[i1] > 0 && !(r[2 * i1 - 1] > tmp[3] && r[2 * i1] < tmp[4])) {
                r[0] = (r[2 * i1] - r[2 * i1 - 1]) / 8.0;
                for (j1 = 0; j1 < 5; j1++) {
                        tmp[1] = 2 * Pi * j1 - ORIENTATION * (oy[pos[i]] -
                theta[seq[i1]]);
                tmp[2] = tmp[3] / exp(d[50 + j] * tmp[1]);
                if (tmp[2] < tmp[3] && tmp[2] > tmp[4]) {
                      tmp[1] = MIN(fabs(tmp[2] - r[2 * i1]), fabs(tmp[2] - r[2 * i1 - 1]));
                        n[6] += num_of_point[seq[i1]];
                        if (tmp[2] > r[2 * i1 - 1] + r[0] && tmp[2] < r[2 * i1] - r[0]) {
                                n[5] += num_of_point[seq[i1]];
                                break;
                                } else if (tmp[1] < d[62])
                                        n[6] += num_of_point[seq[i1]];
                                } else if (tmp[2] < tmp[4])
                                        break;
                                                }
                                        }
        tmp[2] = (float) n[6] / n[5];
        if (tmp[2] > d[61] || tmp[2] == d[61] && fabs(d[50 + i0] - a[0]) < d[60]) {
```

```
                        d[61] = tmp[2];
                        d[60] = fabs(d[50 + j] - a[0]);
                        a[j0] = d[50 + j];
                }
        }
}
if (a[j0] > 0.05) {
        tmp[1] = log(tmp[4]) / a[j0];
        tmp[2] = log(tmp[3]) / a[j0];
        if (j0 == 1) {
                b[0] = tmp[3];
                c[0] = oy[pos[i]];
        }
        b[j0] = tmp[4];
        c[j0] = oy[pos[i0]];
        tmp[3] = oy[pos[i]];
        for (tmp[0] = tmp[2]; tmp[0] >= tmp[1]; tmp[0] -= 0.5 / r[0]) {
                r[0] = exp(a[j0] * tmp[0]);
                i1 = xo + r[0] * cos(tmp[3] - ORIENTATION * (tmp[0] - tmp[2]));
                j1 = yo + r[0] * sin(tmp[3] - ORIENTATION * (tmp[0] - tmp[2]));
                        pr_put(new_image, i1, j1, 1);
        }
        j0++;
}
                }
        var1 = j0;
        b[var1 - 1] /= exp(a[var1 - 1] * 2 * Pi);
} else {
        for (i = 1; i < 100; i++)
                rep[i] = 0;
        i0 = 1;
        c1 = 1;
        ox[0] = tmp[0];
        tmp[0] = 0.0;
        while (c1) {
                ox[i0] = ox[0];
                for (i = 1; i < num_of_seg; i++)
                        if ((label[i] == 3 || label[i] == 4) && r[2 * i - 1] < ox[i0]) {
                                if (i0 == 1)
                                        j0 = 1;
                                else if (i0 == 2) {
                                        tmp[0] = fabs(theta[seq[i]] - theta[seq[rep[2 * (i0 - 1) -
                                                        1]]]);
                                        tmp[0] = LESSPI(tmp[0]);
                                        tmp[4] = fabs(log(r[2 * i - 1] / ox[i0 - 1]));
                                        if (tmp[0] > Pi / 6.0 && tmp[4] > 0.05 * tmp[0])
```

```
                    j0 = 1;
                else
                    j0 = 0;
            } else if (i0 > 2) {
                tmp[4] = fabs(log(r[2 * i - 1] / ox[i0 - 1]));
                if (ORIENTATION < 0) {
                    tmp[0] = LARGER(theta[seq[i]], theta[seq[rep[2 *
                    (i0 - 1) - 1]]]);
                    if (tmp[4] < 0.5 * tmp[0] && tmp[4] > 0.05 * tmp[0] &&
                        theta[seq[i]] > theta[seq[rep[2 * (i0 - 1) - 1]]] &&
                        tmp[0] > Pi / 6.0 && tmp[0] + tmp < 2 * Pi)
                        j0 = 1;
                    else if (tmp[4] < 0.5 * tmp[0] && tmp[4] > 0.05 * tmp[0]
                        && theta[seq[i]] < theta[seq[rep[2 * (i0 - 1) - 1]]]
                        && theta[seq[rep[2 * (i0 - 1) - 1]]] - theta[seq[i]]
                        < 2 * Pi - Pi / 6.0 && theta[seq[rep[1]]] -
                        theta[seq[i]] > Pi / 6.0 && tmp[0] + tmp < 2 * Pi)
                            j0 = 1;
                        else
                            j0 = 0;
                    } else {
                        tmp[0] = LARGER(theta[seq[rep[2 * (i0 - 1) - 1]]],
                        theta[seq[i]]);
                if (tmp[4] < 0.5 * tmp[0] && tmp[4] > 0.05 * tmp[0] && theta[seq[i]] <
                    theta[seq[rep[2 * (i0 - 1) - 1]]] && tmp[0] > Pi / 6.0 && tmp[0] +
                    tmp < 2 * Pi)
                    j0 = 1;
                else if (tmp[4] < 0.5 * tmp[0] && tmp[4] > 0.05 * tmp[0] &&
                    theta[seq[i]] > theta[seq[rep[2 * (i0 - 1) - 1]]] && theta[seq[i]] -
                    theta[seq[rep[2 * (i0 - 1) - 1]]] < 2 * Pi - Pi / 6.0 && theta[seq[i]]
                    - theta[seq[rep[1]]] > Pi / 6.0 && tmp[0] + tmp < 2 * Pi)
                    j0 = 1;
                else
                    j0 = 0;
                }
            }
            if (j0) {
                ox[i0] = r[2 * i - 1];
                x[101] = (newseg[seq[i]] - 1)->x;
                x[102] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->x;
                y[102] = (newseg[seq[i]] - 1 - num_of_point[seq[i]])->y;
                tmp[3] = D(xo, yo, x[101], y[101]);
                tmp[4] = D(xo, yo, x[102], y[102]);
                if (tmp[3] > tmp[4]) {
                    x[i0] = x[102];
                    y[i0] = y[102];
```

```
                    } else {
                        x[i0] = x[101];
                        y[i0] = y[101];
                    }
                    rep[2 * i0 - 1] = i;
                }
            }
if (i0 > 1 && rep[2 * i0 - 1] > 0) {
        tmp = theta[seq[rep[2 * i0 - 1]]] - theta[seq[rep[2 * (i0 - 1) - 1]]];
        if (ORIENTATION < 0)
            tmp[0] += POSITIVE(tmp);
        else
            tmp[0] += POSITIVE(-tmp);
}
if (i0 > 2 && rep[2 * i0 - 1] > 0) {
        tmp = fabs(theta[seq[rep[2 * i0 - 1]]] - theta[seq[rep[1]]]);
        tmp = LESSPI(tmp);
}
if (i0 > 2 && (tmp < Pi / 4 || rep[2 * i0 - 1] == 0))
        c1 = 0;
else {
        tmp[3] = tmp[4] = imageSize / 15;
        j1 = 0;
        for (i = 1; i < num_of_seg; i++)
            if (label[i] == -2) {
                x[100] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->x;
                y[100] = (newseg[seq[i]] - num_of_point[seq[i]] / 2)->y;
                ox[100] = D(xo, yo, x[100], y[100]);
                ox[101] = imageSize / 15;
                if (ox[100] < 2 * ox[i0])
                    for (j = 0; j <= num_of_point[seq[i]]; j++) {
                        x[100] = (newseg[seq[i]] - j - 1)->x;
                        y[100] = (newseg[seq[i]] - j - 1)->y;
                        ox[99] = D(xo, yo, x[100], y[100]);
                        tmp[0] = atan2((double) y[100] - yo, (double) x[100] - xo);
                        tmp[0] = POSITIVE(tmp[0]);
                        tmp[0] = fabs(theta[seq[rep[2 * (i0 - 1) - 1]]] - tmp[0]);
                        tmp[0] = LESSPI(tmp[0]);
                        if (ox[99] < ox[i0] && (log(ox[99] / ox[i0 - 1]) > 0.05 *
                        tmp[0] || i0 == 1)) {
                            ox[100] = D(x[i0], y[i0], x[100], y[100]);
                            if (ox[100] < ox[101]) {
                                ox[101] = ox[100];
                                x[101] = x[100];
                                y[101] = y[100];
                            }
```

```
                        }
                    }
                if (ox[101] < tmp[3]) {
                        tmp[3] = ox[101];
                        x[102] = x[101];
                        y[102] = y[101];
                        rep[2 * i0] = i;
                } else if (ox[101] < tmp[4]) {
                        tmp[4] = ox[101];
                        x[103] = x[101];
                        y[103] = y[101];
                        j1 = i;
                }
        }
if (j1 > 0) {
        if (tmp[4] * num_of_point[seq[rep[2 * i0]]] < tmp[3] * num_of_point[seq[j1]]) {
                rep[2 * i0] = j1;
                x[i0] = x[103];
                y[i0] = y[103];
        } else if (rep[2 * i0] > 0) {
                x[i0] = x[102];
                y[i0] = y[102];
        }
} else if (rep[2 * i0] > 0) {
        x[i0] = x[102];
        y[i0] = y[102];
                }
                if (i0 > 1) {
                        tmp[3] = atan2((double) y[i0 - 1] - yo, (double) x[i0 - 1] - xo);
                        tmp[3] = POSITIVE(tmp[3]);
                        tmp[4] = atan2((double) y[i0] - yo, (double) x[i0] - xo);
                        tmp[4] = POSITIVE(tmp[4]);
                        if (ORIENTATION < 0)
                                tmp[0] = LARGER(tmp[4], tmp[3]);
                        else
                                tmp[0] = LARGER(-tmp[4], -tmp[3]);
                        tmp[3] = D(xo, yo, x[i0 - 1], y[i0 - 1]);
                        tmp[4] = D(xo, yo, x[i0], y[i0]);
                        d[i0 - 1] = fabs(log(tmp[3] / tmp[4])) / tmp[0];
                }
                ox[i0++] = D(xo, yo, x[i0], y[i0]);
        }
}
for (i = 1; i < i0; i++)
        ox[i] = D(xo, yo, x[i], y[i]);
        /*
```

```
 * for (i=1; i<2*i0-1; i++) if (rep[i]) for (j=0;
 * j<=num_of_point[seq[rep[i]]]; j++) {
 * i2=(newseg[seq[rep[i]]]-j-1)->x;
 * j0=(newseg[seq[rep[i]]]-j-1)->y;
 * pr_put(new_image,i2,j0,1); }
 */
a[0] = tmp[4] = 0.0;
tmp[1] = log(ox[1]) / d[1];
tmp[3] = atan2((double) y[1] - yo, (double) x[1] - xo);
tmp[3] = POSITIVE(tmp[3]);
if (ORIENTATION < 0) {
        if (tmp[3] > oy[0])
                tmp[2] = tmp[1] - (tmp[3] - oy[0]);
        else
                tmp[2] = tmp[1] - (tmp[3] + 2 * Pi - oy[0]);
} else {
        if (tmp[3] < oy[0])
                tmp[2] = tmp[1] - (oy[0] - tmp[3]);
        else
                tmp[2] = tmp[1] - (oy[0] + 2 * Pi - tmp[3]);
}
b[i0 - 2] = exp(d[1] * tmp[2]);
c[i0 - 2] = tmp[3] + ORIENTATION * (tmp[2] - tmp[1]);
if (c[i0 - 2] < 0.0)
        c[i0 - 2] += 2 * Pi;
if (c[i0 - 2] > 2 * Pi)
        c[i0 - 2] -= 2 * Pi;
for (tmp[0] = tmp[2]; tmp[0] < tmp[1]; tmp[0] += 0.5 / ox[1]) {
        r[0] = exp(d[1] * tmp[0]);
        i2 = xo + r[0] * cos(tmp[3] - ORIENTATION * (tmp[0] - tmp[1]));
        j0 = yo + r[0] * sin(tmp[3] - ORIENTATION * (tmp[0] - tmp[1]));
        pr_put(new_image, i2, j0, 1);
}
for (i = 2; i < i0; i++) {
        tmp[2] = log(ox[i]) / d[i - 1];
        tmp[1] = log(ox[i - 1]) / d[i - 1];
        a[i0 - i] = d[i - 1];
        if (i > 2)
                b[i0 - i] = ox[i - 1];
        if (i == i0 - 1) {
                b[0] = ox[i];
                c[0] = atan2((double) y[i] - yo, (double) x[i] - xo);
                c[0] = POSITIVE(c[0]);
        }
        tmp[3] = atan2((double) y[i - 1] - yo, (double) x[i - 1] - xo);
        tmp[3] = POSITIVE(tmp[3]);
```

```
                if (i > 2)
                        c[i0 - i] = tmp[3];
                tmp[4] += tmp[2] - tmp[1];
                a[0] += d[i - 1] * (tmp[2] - tmp[1]);
                for (tmp[0] = tmp[1]; tmp[0] <= tmp[2]; tmp[0] += 0.5 / ox[i - 1]) {
                        r[0] = exp(d[i - 1] * tmp[0]);
                        i2 = xo + r[0] * cos(tmp[3] - ORIENTATION * (tmp[0] - tmp[1]));
                        j0 = yo + r[0] * sin(tmp[3] - ORIENTATION * (tmp[0] - tmp[1]));
                        pr_put(new_image, i2, j0, 1);
                }
                printf("i=%d k=%f\n", i - 1, d[i - 1]);
        }
        var1 = i0 - 1;
        a[0] /= tmp[4];
        }
tmp[1] = 0.0;
tmp[2] = c[0];
for (i = 1; i < num_of_seg; i++)
        if ((label[i] == -1 || label[i] > 0) && r[2 * i - 1] > b[0]) {
                if (ORIENTATION > 0)
                        tmp[0] = c[0] - theta[seq[i]];
                else
                        tmp[0] = theta[seq[i]] - c[0];
                tmp[0] = POSITIVE(tmp[0]);
                tmp[3] = b[0] * exp(a[1] * tmp[0]);
                if (tmp[0] < Pi / 2.0 && tmp[0] > tmp[1] && tmp[3] < (r[2 * i] + r[2 * i -
                1]) / 2.0 && r[2 * i - 1] - tmp[3] < r[2 * i] - r[2 * i - 1]) {
                        tmp[0] = 2 * (r[2 * i] - r[2 * i - 1]) / (r[2 * i] + r[2 * i - 1]);
                        i1 = j1 = 0;
                        for (j = 1; j < num_of_seg; j++)
                        if (j != i && (label[j] == -1 || label[j] > 0) && r[2 * j - 1] > b[0] &&
                                num_of_point[seq[j]] < 3 * num_of_point[seq[i]] &&
                                num_of_point[seq[j]] > num_of_point[seq[i]] / 3 && overlap(r[2 *
                                i - 1], r[2 * i], r[2 * j - 1], r[2 * j]) > 0.7) {
                        if (ORIENTATION > 0)
                                tmp[3] = theta[seq[j]] - theta[seq[i]];
                        else
                                tmp[3] = theta[seq[i]] - theta[seq[j]];
                        tmp[3] = POSITIVE(tmp[3]);
                        if (tmp[3] < tmp[0])
                                i1 = 1;
                        else {
                                if (ORIENTATION > 0)
                                        tmp[3] = theta[seq[i]] - theta[seq[j]];
                                else
                                        tmp[3] = theta[seq[j]] - theta[seq[i]];
```

```
                        tmp[3] = POSITIVE(tmp[3]);
                        if (tmp[3] < tmp[0])
                                j1 = 1;
                }
                if (i1 && j1) {
                        tmp[1] = tmp[0];
                        tmp[2] = theta[seq[i]];
                        break;
                }
        }
}
        }
    tmp[3] = fabs(tmp[2] - c[0]);
    tmp[3] = LESSPI(tmp[3]);
    tmp[1] = log(b[0]) / a[1];
    tmp[4] = b[0];
    for (tmp[0] = tmp[1] + tmp[3]; tmp[0] > tmp[1]; tmp[0] -= 0.5 / tmp[4]) {
        tmp[4] = exp(a[1] * tmp[0]);
        i1 = xo + tmp[4] * cos(tmp[2] - ORIENTATION * (tmp[0] - tmp[1] - tmp[3]));
        j1 = yo + tmp[4] * sin(tmp[2] - ORIENTATION * (tmp[0] - tmp[1] - tmp[3]));
        pr_put(new_image, i1, j1, 1);
    }
    b[0] *= exp(a[1] * tmp[3]);
    if (ORIENTATION > 0) {
        tmp[3] = tmp[2] - c[var2 + 99];
        tmp[3] = POSITIVE(tmp[3]);
    } else {
        tmp[3] = c[var2 + 99] - tmp[2];
        tmp[3] = POSITIVE(tmp[3]);
    }
    tmp[4] = b[var2 + 99];
    tmp[1] = log(b[var2 + 99]) / a[var2 + 99];
    for (tmp[0] = tmp[1] - tmp[3]; tmp[0] < tmp[1]; tmp[0] += 0.5 / tmp[4]) {
        tmp[4] = exp(a[var2 + 99] * tmp[0]);
        i1 = xo + tmp[4] * cos(c[var2 + 99] - ORIENTATION * (tmp[0] - tmp[1]));
        j1 = yo + tmp[4] * sin(c[var2 + 99] - ORIENTATION * (tmp[0] - tmp[1]));
        pr_put(new_image, i1, j1, 1);
    }
    b[var2 + 99] /= exp(a[var2 + 99] * tmp[3]);
    if (ORIENTATION > 0) {
        tmp[3] = c[100] - tmp[2];
        tmp[3] = POSITIVE(tmp[3]);
    } else {
        tmp[3] = tmp[2] - c[100];
        tmp[3] = POSITIVE(tmp[3]);
    }
```

```
tmp[4] = b[100];
tmp[1] = log(b[100]) / a[101];
for (tmp[0] = tmp[1]; tmp[0] < tmp[1] + tmp[3]; tmp[0] += 0.5 / tmp[4]) {
        tmp[4] = exp(a[101] * tmp[0]);
        i1 = xo + tmp[4] * cos(c[100] - ORIENTATION * (tmp[0] - tmp[1]));
        j1 = yo + tmp[4] * sin(c[100] - ORIENTATION * (tmp[0] - tmp[1]));
        pr_put(new_image, i1, j1, 1);
}
b[100] *= exp(a[101] * tmp[3]);
c[0] = c[100] = c[var2 + 99] = tmp[2];
for (i = 1; i < 2 * num_of_seg; i++)
        ox[i] = 0.0;
for (i = 1; i < num_of_seg; i++)
        if (label[i] == -1 || label[i] > 0) {
                if (ORIENTATION > 0)
                        tmp[0] = theta[seq[i]] - c[0];
                else
                        tmp[0] = c[0] - theta[seq[i]];
                tmp[0] = POSITIVE(tmp[0]);
                tmp[1] = tmp[3] = 0.0;
                for (j = 1; j < var1; j++)
                        if (!ox[2 * i - 1]) {
                                tmp[2] = log(b[j - 1] / b[j]) / a[j];
                                if (tmp[1] + tmp[2] > tmp[0]) {
                                        tmp[0] -= tmp[1];
                                        tmp[4] = b[j - 1] * exp(-a[j] * tmp[0]);
                                        if (tmp[4] < (r[2 * i] + r[2 * i - 1]) / 2) {
                                                ox[2 * i - 1] = tmp[4];
                                                if (tmp[3])
                        ox[2 * i] = tmp[3];
                break;
        } else {
                tmp[3] = tmp[4];
                tmp[1] = tmp[2] - tmp[0];
                tmp[0] = 2 * Pi;
                j0 = tmp[1] / (2 * Pi);
                if (j0)
                        for (j1 = 1; j1 <= j0; j1++) {
                                tmp[4] = tmp[3] * exp(-a[j] * 2 * Pi);
                                if (tmp[4] < (r[2 * i] + r[2 * i - 1]) / 2) {
                                        ox[2 * i - 1] = tmp[4];
                                        ox[2 * i] = tmp[3];
                                        break;
                                } else
                                        tmp[3] = tmp[4];
                        }
```

```
                    tmp[1] -= j0 * 2 * Pi;
                }
        } else
                tmp[1] += tmp[2];
}
if (ox[2 * i - 1] && !ox[2 * i]) {
        if (ORIENTATION > 0)
                tmp[0] = theta[seq[i]] - c[100];
        else
                tmp[0] = c[100] - theta[seq[i]];
        tmp[0] = POSITIVE(tmp[0]);
        tmp[1] = 0.0;
        for (j = 101; j < 100 + var2; j++) {
                tmp[2] = log(b[j - 1] / b[j]) / a[j];
                if (tmp[1] + tmp[2] > tmp[0]) {
                        tmp[0] -= tmp[1];
                        ox[2 * i] = b[j - 1] * exp(-a[j] * tmp[0]);
                        break;
                } else
                        tmp[1] += tmp[2];
        }
        if (ox[2 * i] < 0.9 * r[2 * i] && r[2 * i] - r[2 * i - 1] < (ox[2 * i] - ox[2 * i - 1]) / 2)
                ox[2 * i] = ox[2 * i - 1] = 0.0;
                        }
        if (overlap(r[2 * i - 1], r[2 * i], ox[2 * i - 1], ox[2 * i]) < 0.5)
                ox[2 * i] = ox[2 * i - 1] = 0.0;
        }
        for (i = 1; i < num_of_seg; i++)
                if (ox[2 * i])
                        for (j = i + 1; j < num_of_seg; j++)
        if (ox[2 * j]) {
                if (ox[2 * rep[i] - 1] < ox[2 * rep[j] - 1]) {
                        rep[0] = rep[i];
                        rep[i] = rep[j];
                        rep[j] = rep[0];
                }
        }
for (i = 1; i < num_of_seg; i++)
        if (ox[2 * rep[i]]) {
                for (j = 1; j < num_of_seg; j++)
                    if (ox[2 * rep[j]] >= b[100] && ox[2 * rep[j] - 1] < 0.9 * ox[2 * rep[i] - 1]) {
                                label[rep[j]] = 0;
                                ox[2 * rep[j]] = ox[2 * rep[j] - 1] = 0.0;
                        }
                break;
        }
```

```
for (i = 100; i < 200; i++)
        n[i] = 0;
for (i = 1; i < 200; i++)
        rep[i] = 0;
for (i = 1; i < num_of_seg; i++)
        if (ox[2 * i]) {
                i1 = rep[0] = i;
                j1 = 1;
                c1 = 1;
                i0 = num_of_point[seq[i]] / 2;
                tmp[0] = r[2 * i - 1] * (exp(a[1] * Pi / 4.0) - 1.0);
                while (c1) {
                        d[j1] = tmp[0];
                        for (j = 1; j < num_of_seg; j++)
                                if (ox[2 * j] && num_of_point[seq[j]] > i0 && ox[2 * i1 - 1] >
                                        ox[2 * j - 1] && ox[2 * i1 - 1] - ox[2 * j - 1] < d[j1]) {
                                        d[j1] = ox[2 * i1 - 1] - ox[2 * j - 1];
                                        rep[j1] = j;
                                }
                        if (d[j1] < tmp[0] && (j1 < 2 || d[j1] > d[j1 - 1] / 3 && d[j1] < 3 * d[j1 - 1])) {
                                i1 = rep[j1];
                                j1++;
                        } else if (j1 > 5) {
                                tmp[1] = imageSize;
                                tmp[2] = 0.0;
                                for (j = 0; j < j1; j++) {
                                        if (ox[2 * rep[j] - 1] < tmp[1]) {
                                                rep[900] = rep[j];
                                                tmp[1] = ox[2 * rep[j] - 1];
                                        }
                                        if (ox[2 * rep[j] - 1] > tmp[2]) {
                                                        rep[901] = rep[j];
                                                        tmp[2] = ox[2 * rep[j] - 1];
                                                }
                                                label[rep[j]] = 5;
                                        }
                                        tmp[3] = fabs(theta[seq[rep[901]]] - theta[seq[rep[900]]]);
                                        tmp[3] = LESSPI(tmp[3]);
                                        sigma = tmp[3] / (j1 - 1);
                                        c1 = 0;
                                } else
                                        break;
                        }
                        if (!c1)
                                break;
                }
```

```
for (i = 1; i < num_of_seg; i++)
        if (label[i] == -1 || label[i] > 0) {
                xo = (r[2 * i] - r[2 * i - 1]) / 20.0;
                break;
        }
n[1] = 0;
for (i = 1; i < num_of_seg; i++)
        if (ox[2 * i]) {
                j0 = 0;
                tmp[0] = 6 * sigma;
                if (r[2 * i] - r[2 * i - 1] > xo)
                        i2 = 1;
                else
                        i2 = 0;
                for (j = 1; j < num_of_seg; j++)
                    if (j != i && ox[2 * j] && r[2 * j] - r[2 * j - 1] > (r[2 * i] - r[2 * i -
                     1]) / 6.0 && overlap(r[2 * i - 1], r[2 * i], r[2 * j - 1], r[2 * j]) > 0.5) {
                                if (r[2 * i] - r[2 * i - 1] > r[2 * j] - r[2 * j - 1]) {
                                        tmp[2] = (r[2 * j] + r[2 * j - 1]) / 2.0;
                                        for (j1 = 0; j1 <= num_of_point[seq[i]]; j1++) {
                                                x[1] = (newseg[seq[i]] - j1 - 1)->x;
                                                y[1] = (newseg[seq[i]] - j1 - 1)->y;
                                                tmp[3] = D(xo, yo, x[1], y[1]);
                                                if (fabs(tmp[3] - tmp[2]) < 2.0) {
                                                        tmp[4] = atan2((double) y[1] - yo,
                                                                (double) x[1] - xo);
                                                        tmp[4] = POSITIVE(tmp[4]);
                                                        tmp[4] = theta[seq[j]] - tmp[4];
                                                        tmp[4] = POSITIVE(tmp[4]);
                                                        break;
                                                }
                                        }
        } else {
                tmp[2] = (r[2 * i] + r[2 * i - 1]) / 2.0;
                for (j1 = 0; j1 <= num_of_point[seq[j]]; j1++) {
                        x[1] = (newseg[seq[j]] - j1 - 1)->x;
                        y[1] = (newseg[seq[j]] - j1 - 1)->y;
                        tmp[3] = D(xo, yo, x[1], y[1]);
                        if (fabs(tmp[3] - tmp[2]) < 2.0) {
                                tmp[4] = atan2((double) y[1] - yo, (double) x[1] - xo);
                                tmp[4] = POSITIVE(tmp[4]);
                                tmp[4] = tmp[4] - theta[seq[i]];
                                tmp[4] = POSITIVE(tmp[4]);
                                break;
                        }
                }
```

```
                }
                if (tmp[4] < tmp[0]) {
                        xx[0][i] = tmp[0] = tmp[4];
                        j0 = j;
                }
        }
if (j0) {
        n[1] = bright_between(j0, i, xo, yo, r[2 * i - 1], r[2 * i]);
        if (n[1] > 0)
                neighbor[0][i] = j0;
        else
                n[1] = bright_nearby(i, -1, xo, yo, 0.8 * sigma, i2);
} else
        n[1] = bright_nearby(i, -1, xo, yo, 0.8 * sigma, i2);
j0 = 0;
tmp[0] = 6 * sigma;
for (j = 1; j < num_of_seg; j++)
        if (j != i && ox[2 * j] && r[2 * j] - r[2 * j - 1] > (r[2 * i] - r[2 * i - 1]) / 6.0 &&
                overlap(r[2 * i - 1], r[2 * i], r[2 * j - 1], r[2 * j]) > 0.5) {
                if (r[2 * i] - r[2 * i - 1] > r[2 * j] - r[2 * j - 1]) {
                        tmp[2] = (r[2 * j] + r[2 * j - 1]) / 2.0;
                        for (j1 = 0; j1 <= num_of_point[seq[i]]; j1++) {
                                x[1] = (newseg[seq[i]] - j1 - 1)->x;
                                y[1] = (newseg[seq[i]] - j1 - 1)->y;
                                tmp[3] = D(xo, yo, x[1], y[1]);
                                if (fabs(tmp[3] - tmp[2]) < 2.0) {
                                        tmp[4] = atan2((double) y[1] - yo, (double) x[1] - xo);
                                        tmp[4] = POSITIVE(tmp[4]);
                                        tmp[4] = tmp[4] - theta[seq[j]];
                                        tmp[4] = POSITIVE(tmp[4]);
                                        break;
                                }
                        }
                } else {
                        tmp[2] = (r[2 * i] + r[2 * i - 1]) / 2.0;
                        for (j1 = 0; j1 <= num_of_point[seq[j]]; j1++) {
                                x[1] = (newseg[seq[j]] - j1 - 1)->x;
                                y[1] = (newseg[seq[j]] - j1 - 1)->y;
                                tmp[3] = D(xo, yo, x[1], y[1]);
                                if (fabs(tmp[3] - tmp[2]) < 2.0) {
                                        tmp[4] = atan2((double) y[1] - yo, (double) x[1] - xo);
                                        tmp[4] = POSITIVE(tmp[4]);
                                        tmp[4] = theta[seq[i]] - tmp[4];
                                        tmp[4] = POSITIVE(tmp[4]);
                                        break;
                                }
```

```
                    }
                }
                if (tmp[4] < tmp[0]) {
                    xx[1][i] = tmp[0] = tmp[4];
                    j0 = j;
                }
            }
if (j0) {
        n[2] = bright_between(j0, i, xo, yo, r[2 * i - 1], r[2 * i]);
        if (n[2] > 0)
            neighbor[1][i] = j0;
        else
            n[2] = bright_nearby(i, 1, xo, yo, 0.8 * sigma, i2);
} else
        n[2] = bright_nearby(i, 1, xo, yo, 0.8 * sigma, i2);
if (n[1] >= 0 && n[2] >= 0 && (n[2] - n[1] > 30 || i2 && n[2] > n[1]))
        label[i] = 6;
else if (n[1] >= 0 && n[2] >= 0 && (n[1] - n[2] > 30 || i2 && n[1] > n[2]))
        label[i] = 7;
            }
        j1 = 1;
        for (i = 1; i < num_of_seg; i++)
            if (label[i] > 5 && r[2 * i] - r[2 * i - 1] > 0.6 * (ox[2 * i] - ox[2 * i - 1]) && r[2 *
                i] - r[2 * i - 1] > 38.0) {
                n[j1++] = i;
                if (j1 > 150)
                    break;
            }
        for (i = 0; i < 20; i++)
            d[i] = 0.0;
        for (i = 20; i < 40; i++)
            d[i] = -1.0;
        for (i = 280; i < 300; i++)
            n[i] = 0;
        for (i = 1; i < j1; i++) {
            tmp[2] = (ox[2 * n[i]] - ox[2 * n[i] - 1]) / 19.0;
            for (j = 0; j <= num_of_point[seq[n[i]]]; j++) {
                x[1] = (newseg[seq[n[i]]] - j - 1)->x;
                y[1] = (newseg[seq[n[i]]] - j - 1)->y;
                tmp[1] = D(xo, yo, x[1], y[1]);
                for (j0 = 0; j0 < 20; j0++)
                    if (fabs(tmp[1] - ox[2 * n[i] - 1] - j0 * tmp[2]) < 3.0) {
                        d[20 + j0] = atan2((double) y[1] - yo, (double) x[1] - xo);
                        d[20 + j0] = POSITIVE(d[20 + j0]);
                        j += tmp[2] / 2.0;
                        break;
```

```
                        }
                }
                tmp[0] = d[30];
                for (j = 20; j < 40; j++)
                        if (d[j] >= 0.0) {
                                d[j] -= tmp[0];
                                if (d[j] > Pi)
                                        d[j] -= 2 * Pi;
                                else if (d[j] < -Pi)
                                        d[j] += 2 * Pi;
                                d[j - 20] += d[j];
                                n[260 + j]++;
                        }
                for (j = 20; j < 40; j++) {
                        printf("j=%d d=%f ", j - 20, d[j]);
                        d[j] = -1.0;
                }
                printf(" \n");
        }
        for (i = 0; i < 20; i++)
                if (n[280 + i] > 2)
                        d[i] /= n[280 + i];
                else
                        d[i] = 0.0;
        i1 = -1;
        for (i = 0; i < 20; i++)
                if (d[i] != 0.0) {
                        i1 = i;
                        break;
                }
        for (i = 0; i < i1; i++)
                d[i] = d[i1];
        i1 = 20;
        for (i = 19; i > 9; i--)
                if (d[i] != 0.0) {
                        i1 = i;
                        break;
                }
        for (i = i1 + 1; i < 20; i++)
                d[i] = d[i1];
        for (i = 20; i < 200; i++)
                d[i] = -1.0;
        for (i = 1; i < num_of_seg; i++)
                oy[i] = -1.0;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] > 5) {
```

```
            tmp[1] = (ox[2 * i] + ox[2 * i - 1]) / 2.0;
            if (tmp[1] >= r[2 * i - 1] && tmp[1] <= r[2 * i]) {
                    for (j = 0; j <= num_of_point[seq[i]]; j++) {
                            x[1] = (newseg[seq[i]] - j - 1)->x;
                            y[1] = (newseg[seq[i]] - j - 1)->y;
                            tmp[2] = D(xo, yo, x[1], y[1]);
                            if (fabs(tmp[2] - tmp[1]) < 3.0)
                                    break;
                    }
                    oy[i] = atan2((double) y[1] - yo, (double) x[1] - xo);
                    oy[i] = POSITIVE(oy[i]);
            } else {
                    tmp[2] = (ox[2 * i] - ox[2 * i - 1]) / 20.0;
                    for (j1 = 0; j1 <= num_of_point[seq[i]]; j1++) {
                            x[1] = (newseg[seq[i]] - j1 - 1)->x;
                            y[1] = (newseg[seq[i]] - j1 - 1)->y;
                            tmp[1] = D(xo, yo, x[1], y[1]);
                            for (i1 = 0; i1 < 20; i1++)
                                if (fabs(tmp[1] - ox[2 * i - 1] - i1 * tmp[2]) < 3.0) {
                                    d[20 + i1] = atan2((double) y[1] - yo, (double) x[1]
                                                    - xo);
                                    d[20 + i1] = POSITIVE(d[20 + i1]);
                                    j1 += tmp[2] / sqrt(2.0);
                                    break;
                                    }
                    }
                    i1 = 0;
                    tmp[3] = 2 * Pi;
                    tmp[4] = d[40] = 0.0;
                    for (j1 = 20; j1 < 40; j1++)
                            if (d[j1] >= 0.0) {
                                    tmp[0] = d[j1] - d[j1 - 20];
                                    if (tmp[0] < 0.0)
                                            tmp[0] += 2 * Pi;
                                    else if (tmp[0] > 2 * Pi)
                                            tmp[0] -= 2 * Pi;
                                    if (tmp[0] < tmp[3])
                            tmp[3] = tmp[0];
                    if (tmp[0] > tmp[4])
                            tmp[4] = tmp[0];
                    d[40] += tmp[0];
                    i1++;
            }
    if (i1) {
            if (tmp[4] - tmp[3] < Pi)
                    oy[i] = d[40] / i1;
```

```
                else {
                        d[40] = 0.0;
                        for (j1 = 20; j1 < 40; j1++)
                                if (d[j1] >= 0.0) {
                                        tmp[0] = d[j1] - d[j1 - 20];
                                        if (tmp[0] < 0.0 || tmp[0] < Pi)
                                                tmp[0] += 2 * Pi;
                                        d[40] += tmp[0];
                                }
                        d[40] /= i1;
                        d[40] = TWOPI(d[40]);
                        oy[i] = d[40];
                }
        } else {
                j1 = 20 * ((r[2 * i] + r[2 * i - 1]) / 2.0 - ox[2 * i - 1]) / (ox[2 * i] - ox[2 * i - 1]);
                d[40] = theta[seq[i]] - d[j1];
                if (d[40] > 2 * Pi)
                        d[40] -= 2 * Pi;
                else if (d[40] < 0.0)
                        d[40] += 2 * Pi;
                oy[i] = d[40];
                }
        for (j1 = 20; j1 < 40; j1++)
                d[j1] = -1.0;
        }
}
        for (i = 1; i < 2 * num_of_seg; i++)
                ox[i] = 0.0;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] > 5) {
                        if (ORIENTATION > 0)
                                tmp[0] = oy[i] - c[0];
                        else
                                tmp[0] = c[0] - oy[i];
                        tmp[0] = POSITIVE(tmp[0]);
                        tmp[1] = tmp[3] = 0.0;
                        for (j = 1; j < var1; j++)
                                if (!ox[2 * i - 1]) {
                tmp[2] = log(b[j - 1] / b[j]) / a[j];
                if (tmp[1] + tmp[2] > tmp[0]) {
                        tmp[0] -= tmp[1];
                        tmp[4] = b[j - 1] * exp(-a[j] * tmp[0]);
                        if (tmp[4] < (r[2 * i] + r[2 * i - 1]) / 2) {
                                ox[2 * i - 1] = tmp[4];
                                if (tmp[3])
                                        ox[2 * i] = tmp[3];
```

```
                        break;
                    } else {
                        tmp[3] = tmp[4];
                        tmp[1] = tmp[2] - tmp[0];
                        tmp[0] = 2 * Pi;
                        j0 = tmp[1] / (2 * Pi);
                        if (j0)
                            for (j1 = 1; j1 <= j0; j1++) {
                                tmp[4] = tmp[3] * exp(-a[j] * 2 * Pi);
                                if (tmp[4] < (r[2 * i] + r[2 * i - 1]) / 2) {
                                    ox[2 * i - 1] = tmp[4];
                                    ox[2 * i] = tmp[3];
                                    break;
                                } else
                                    tmp[3] = tmp[4];
                            }
                        tmp[1] -= j0 * 2 * Pi;
                    }
                } else
                    tmp[1] += tmp[2];
        }
if (ox[2 * i - 1] && !ox[2 * i]) {
        if (ORIENTATION > 0)
            tmp[0] = oy[i] - c[100];
        else
            tmp[0] = c[100] - oy[i];
        tmp[0] = POSITIVE(tmp[0]);
        tmp[1] = 0.0;
        for (j = 101; j < 100 + var2; j++) {
            tmp[2] = log(b[j - 1] / b[j]) / a[j];
            if (tmp[1] + tmp[2] > tmp[0]) {
                tmp[0] -= tmp[1];
                ox[2 * i] = b[j - 1] * exp(-a[j] * tmp[0]);
                break;
            } else
                tmp[1] += tmp[2];
        }
        if (ox[2 * i] < 0.9 * r[2 * i])
            ox[2 * i] = b[100];
                }
                if (overlap(r[2 * i - 1], r[2 * i], ox[2 * i - 1], ox[2 * i]) < 0.5)
                    ox[2 * i] = ox[2 * i - 1] = 0.0;
            }
        for (i = 1; i < num_of_seg; i++)
            rep[i] = i;
        for (i = 1; i < num_of_seg; i++)
```

```
        if (ox[2 * i])
                for (j = i + 1; j < num_of_seg; j++)
                        if (ox[2 * j]) {
                                if (ox[2 * rep[i] - 1] < ox[2 * rep[j] - 1]) {
                                        rep[0] = rep[i];
                                        rep[i] = rep[j];
                                        rep[j] = rep[0];
                                }
                        }
for (i = 1; i < num_of_seg; i++)
        if (rep[i] != i)
                for (j = 1; j < num_of_seg; j++)
                        if (rep[j] == i)
                                neighbor[0][i] = j;
for (i = 1; i < num_of_seg; i++)
        if (label[i] == 6 && r[2 * i] - r[2 * i - 1] > (ox[2 * i] - ox[2 * i - 1]) / 2.0)
                label[i] = 9;
for (i = 1; i < num_of_seg; i++)
        if (label[i] == 6) {
                i2 = 0;
                tmp[1] = (r[2 * i] + r[2 * i - 1]) / 2;
                tmp[2] = r[2 * i] - r[2 * i - 1];
                tmp[0] = 2 * tmp[2] / tmp[1];
                tmp[0] = MIN(tmp[0], 5 * sigma);
                for (j = i + 1; j < num_of_seg; j++)
                        if (label[j] > 5 && r[2 * j] - r[2 * j - 1] > 0.25 * tmp[2] &&
                                overlap(r[2 * i - 1], r[2 * i], r[2 * j - 1], r[2 * j]) > 0.5) {
                                tmp[3] = fabs(oy[i] - oy[j]);
                                tmp[3] = LESSPI(tmp[3]);
                                if (tmp[3] < tmp[0])
                                        i2++;
                                if (i2 > 1)
                                        break;
                        }
                if (i2 < 2)
                        for (j = i - 1; j > 0; j--)
                                if (label[j] > 5 && r[2 * j] - r[2 * j - 1] < 4 * tmp[2] &&
                                   overlap(r[2 * i - 1], r[2 * i], r[2 * j - 1], r[2 * j]) > 0.5) {
                                        tmp[3] = fabs(oy[i] - oy[j]);
                                        tmp[3] = LESSPI(tmp[3]);
                                if (tmp[3] < tmp[0])
                                        i2++;
                                if (i2 > 1)
                                        break;
                        }
        if (i2 > 1)
```

```
                label[i] = 8;
        }
c1 = 1;
while (c1) {
        c1 = 0;
        for (i = 1; i < num_of_seg; i++)
                if (label[i] > 5 && label[i] != 7) {
                        tmp[1] = MAX(xo, (ox[2 * i] - ox[2 * i - 1]) / 5.0);
                        x[1] = (newseg[seq[i]] - 1)->x;
                        y[1] = (newseg[seq[i]] - 1)->y;
                        x[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->x;
                        y[2] = (newseg[seq[i]] - num_of_point[seq[i]] - 1)->y;
                        if (r[2 * i] > D(xo, yo, x[2], y[2])) {
                                x[3] = x[2];
                                y[3] = y[2];
                                x[2] = x[1];
                                y[2] = y[1];
                                x[1] = x[3];
                                y[1] = y[3];
                        }
                        tmp[0] = atan2((double) y[2] - y[1], (double) x[2] - x[1]);
                        tmp[0] = POSITIVE(tmp[0]);
                        j1 = 100;
                        tmp[3] = r[2 * i - 1];
                        tmp[4] = r[2 * i];
                        for (j = 1; j < neighbor[0][i]; j++)
                        if (label[rep[neighbor[0][i] - j]] > 5 && label[rep[neighbor[0][i] - j]] != 7) {
                                if (num_of_point[seq[rep[neighbor[0][i] - j]]] < num_of_point[seq[i]] &&
                                        overlap(r[2 * i - 1], r[2 * i], r[2 * rep[neighbor[0][i] - j] - 1], r[2 *
                                        rep[neighbor[0][i] - j]]) < 0.01) {
                                        x[3] = (newseg[seq[rep[neighbor[0][i] - j]]] - 1)->x;
                                        y[3] = (newseg[seq[rep[neighbor[0][i] - j]]] - 1)->y;
                                        x[4] = (newseg[seq[rep[neighbor[0][i] - j]]] -
                                                num_of_point[seq[rep[neighbor[0][i] - j]]] - 1)->x;
                                        y[4] = (newseg[seq[rep[neighbor[0][i] - j]]] -
                                                num_of_point[seq[rep[neighbor[0][i] - j]]] - 1)->y;
                                        if (r[2 * i] > r[2 * rep[neighbor[0][i] - j]]) {
                                            if (D(x[1], y[1], x[3], y[3]) < D(x[1], y[1], x[4], y[4]))
                                                tmp[5] = atan2((double) y[1] - y[3], (double) x[1] - x[3]);
                                            else
                                                tmp[5] = atan2((double) y[1] - y[4], (double) x[1] - x[4]);
                                        } else {
                                            if (D(x[2], y[2], x[3], y[3]) < D(x[2], y[2], x[4], y[4]))
                                                tmp[5] = atan2((double) y[2] - y[3], (double) x[2] - x[3]);
                                                else
                                        tmp[5] = atan2((double) y[2] - y[4], (double) x[2] - x[4]);
```

```
        }
        tmp[5] = POSITIVE(tmp[5]);
        tmp[2] = fabs(tmp[5] - tmp[0]);
        tmp[2] = LESSPI(tmp[2]);
        if (tmp[2] < 8 * sigma) {
                r[2 * i] += (r[2 * rep[neighbor[0][i] - j]] - r[2 * rep[neighbor[0][i]
                                - j] - 1]) / 2;
                        r[2 * i - 1] -= (r[2 * rep[neighbor[0][i] - j]] - r[2 *
                                                rep[neighbor[0][i] - j] - 1]) / 2;
                        if (label[i] < 8 && r[2 * i] - r[2 * i - 1] > tmp[1])
                                label[i] = 8;
                        if (label[i] > 7) {
                                c1 = 1;
                                label[rep[neighbor[0][i] - j]] = 1;
                        }
                }
        } else
                break;
        }
for (j = 1; j < num_of_seg - neighbor[0][i]; j++)
        if (label[rep[neighbor[0][i] + j]] > 5 && label[rep[neighbor[0][i] + j]] != 7) {
                if (num_of_point[seq[rep[neighbor[0][i] + j]]] < num_of_point[seq[i]] &&
                        overlap(r[2 * i - 1], r[2 * i], r[2 * rep[neighbor[0][i] + j] - 1], r[2 *
                        rep[neighbor[0][i] + j]]) < 0.01) {
                        x[3] = (newseg[seq[rep[neighbor[0][i] + j]]] - 1)->x;
                        y[3] = (newseg[seq[rep[neighbor[0][i] + j]]] - 1)->y;
                        x[4] = (newseg[seq[rep[neighbor[0][i] + j]]] -
                                num_of_point[seq[rep[neighbor[0][i] + j]]] - 1)->x;
                        y[4] = (newseg[seq[rep[neighbor[0][i] + j]]] -
                                num_of_point[seq[rep[neighbor[0][i] + j]]] - 1)->y;
                        if (r[2 * i] > r[2 * rep[neighbor[0][i] + j]]) {
                            if (D(x[1], y[1], x[3], y[3]) < D(x[1], y[1], x[4], y[4]))
                                tmp[5] = atan2((double) y[1] - y[3], (double) x[1] - x[3]);
                            else
                                tmp[5] = atan2((double) y[1] - y[4], (double) x[1] - x[4]);
                } else {
                        if (D(x[2], y[2], x[3], y[3]) < D(x[2], y[2], x[4], y[4]))
                                tmp[5] = atan2((double) y[2] - y[3], (double) x[2] - x[3]);
                        else
                                tmp[5] = atan2((double) y[2] - y[4], (double) x[2] - x[4]);
}
                        tmp[5] = POSITIVE(tmp[5]);
                        tmp[2] = fabs(tmp[5] - tmp[0]);
                        tmp[2] = LESSPI(tmp[2]);
                        if (tmp[2] < 8 * sigma) {
```

```
                        r[2 * i] += (r[2 * rep[neighbor[0][i] + j]] - r[2 * rep[neighbor[0][i]
                                + j] - 1]) / 2;
                        r[2 * i - 1] -= (r[2 * rep[neighbor[0][i] + j]] - r[2 *
                                rep[neighbor[0][i] + j] - 1]) / 2;
                        if (label[i] < 8 && r[2 * i] - r[2 * i - 1] > tmp[1])
                                label[i] = 8;
                        if (label[i] > 7) {
                                c1 = 1;
                                label[rep[neighbor[0][i] + j]] = 1;
                        }
                }
        } else
                break;
    }
}
    }
    for (i = 1; i < num_of_seg; i++)
        if (label[i] == 6) {
            for (j = i + 1; j < num_of_seg; j++)
                if (label[j] == 6 && ox[2 * j - 1] < 1.1 * ox[2 * i - 1] && ox[2 * j - 1] >
                        0.9 * ox[2 * i - 1] && overlap(tmp[3], tmp[4], r[2 * j - 1], r[2 *
                        j]) < 0.1) {
                        tmp[2] = fabs(oy[i] - oy[j]);
                        tmp[2] = LESSPI(tmp[2]);
                        if (tmp[2] < sigma / 4) {
                                n[j1++] = j;
                                tmp[4] += num_of_point[seq[j]] / 2 + 1;
                                tmp[3] -= num_of_point[seq[j]] / 2;
                        }
                }
            if (tmp[4] - tmp[3] > tmp[1]) {
                label[i] = 8;
                r[2 * i] = tmp[4];
                r[2 * i - 1] = tmp[3];
                for (i1 = 100; i1 < j1; i1++) {
                        /*
                         * num_of_point[seq[i]] +=
                         * num_of_point[seq[n[i1]]]+1;
                         */
                        label[n[i1]] = 1;
                }
            }
    }
for (i = 1; i < num_of_seg; i++)
    if (label[i] == 8) {
            for (j = 1; j < neighbor[0][i]; j++)
```

```
                if (label[rep[neighbor[0][i] - j]] == 9) {
                        tmp[1] = fabs(oy[i] - oy[rep[neighbor[0][i] - j]]);
                        tmp[1] = LESSPI(tmp[1]);
                        tmp[2] = 1.0 - (r[2 * i] - r[2 * i - 1]) / (ox[2 * i] - ox[2 * i - 1]);
                        if (tmp[1] < 2 * sigma * SQR(tmp[2]) || tmp[1] < 2 * sigma *
                                tmp[2] && overlap(r[2 * i - 1], r[2 * i], r[2 *rep[neighbor[0][i] -
                                j] - 1], r[2 * rep[neighbor[0][i] - j]]) < 0.5)
                                label[i] = 1;
                        break;
                }
                for (j = 1; j < num_of_seg - neighbor[0][i]; j++)
                        if (label[rep[neighbor[0][i] + j]] == 9) {
                                tmp[1] = fabs(oy[i] - oy[rep[neighbor[0][i] + j]]);
                                tmp[1] = LESSPI(tmp[1]);
                                tmp[2] = 0.5 - (r[2 * i] - r[2 * i - 1]) / (ox[2 * i] - ox[2 * i - 1]);
                                if (tmp[1] < 8 * sigma * SQR(tmp[2]) || tmp[1] < 4 * sigma *
                                  tmp[2] && overlap(r[2 * i - 1], r[2 * i], r[2 *rep[neighbor[0][i]
-                                 j] - 1], r[2 * rep[neighbor[0][i] - j]]) < 0.5)
                                      label[i] = 1;
                                break;
                        }
                if (label[i] == 8)
                        label[i] = 9;
        }
j0 = 0;
tmp[1] = tmp[2] = 0.0;
for (i = 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9) {
                j0++;
                if (j0 == 1)
                        tmp[1] = tmp[2] = oy[rep[i]];
                else {
                        if (ORIENTATION > 0)
                                tmp[0] = LARGER(oy[rep[i]], tmp[2]);
                        else
                                tmp[0] = LARGER(tmp[2], oy[rep[i]]);
                        if (tmp[0] < 3 * sigma)
                                tmp[2] = oy[rep[i]];
                        else if (j0 > 7) {
                                if (ORIENTATION > 0)
                                        tmp[0] = LARGER(tmp[2], tmp[1]);
                                else
                                        tmp[0] = LARGER(tmp[1], tmp[2]);
                                printf("sigma1=%f j0=%d\n", tmp[0] / (j0 - 1), j0 - 1);
                                tmp[1] = tmp[2] = oy[rep[i]];
                                j0 = 1;
```

```
                    } else {
                        tmp[1] = tmp[2] = oy[rep[i]];
                        j0 = 1;
                    }
                }
        }
tmp[0] = 0.3 * sigma;
for (i = 0; i < 15; i++)
        n[i] = 0;
for (i = 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9)
                for (j = i + 1; j < num_of_seg; j++)
                    if (label[rep[j]] == 9) {
                        tmp[1] = fabs(oy[rep[i]] - oy[rep[j]]);
                        tmp[1] = LESSPI(tmp[1]);
                        i1 = tmp[1] / tmp[0];
                        if (i1 > 14)
                            i1 = 14;
                        n[i1]++;
                        break;
                    }
printf("\n");
n[15] = 0;
for (i = 0; i < 14; i++) {
        if (n[i] > n[15]) {
                n[15] = n[i];
                j = i;
        }
}
for (i = 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9)
                for (j = i + 1; j < num_of_seg; j++)
                  if (label[rep[j]] == 9) {
                        tmp[1] = fabs(oy[rep[i]] - oy[rep[j]]);
                        tmp[1] = LESSPI(tmp[1]);
                        if (tmp[1] > 3.6 * sigma && tmp[1] < 5 * sigma) {
                            label[rep[(i + j) / 2]] = 9;
                            oy[rep[(i + j) / 2]] = (oy[rep[i]] + oy[rep[j]]) / 2.0;
                            ox[2 * rep[(i + j) / 2]] = sqrt(ox[2 * rep[i]] * ox[2 * rep[j]]);
                            ox[2 * rep[(i + j) / 2] - 1] = sqrt(ox[2 * rep[i] - 1] * ox[2 * rep[j] -1]);
                        }
                        break;
                }
        for (i = 1; i < num_of_seg; i++)
                if (label[rep[i]] == 9) {
                        i0 = i;
```

```
                break;
        }
i1 = i0;
j1 = 0;
for (i = i0 + 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9) {
                tmp[1] = fabs(oy[rep[i]] - oy[rep[i1]]);
                tmp[1] = LESSPI(tmp[1]);
                for (j = i + 1; j < num_of_seg; j++)
                        if (label[rep[j]] == 9) {
                                j1 = j;
                                break;
                        }
                tmp[2] = fabs(oy[rep[i]] - oy[rep[j1]]);
                tmp[2] = LESSPI(tmp[2]);
                if (tmp[1] < 3.6 * sigma && tmp[2] < 3.6 * sigma && (tmp[1] > 1.4 *
                        tmp[2] || tmp[1] < 0.7 * tmp[2])) {
                        if (fabs(oy[rep[i1]] - oy[rep[j1]]) < Pi)
                                oy[rep[i]] = (oy[rep[i1]] + oy[rep[j1]]) / 2.0;
                        else {
                                oy[rep[i]] = (2 * Pi + oy[rep[i1]] + oy[rep[j1]]) / 2.0;
                                oy[rep[i]] = TWOPI(oy[rep[i]], 2 * Pi);
                        }
                        ox[2 * rep[i]] = sqrt(ox[2 * rep[i1]] * ox[2 * rep[j1]]);
                        ox[2 * rep[i] - 1] = sqrt(ox[2 * rep[i1] - 1] * ox[2 * rep[j1] - 1]);
                        i1 = i = j1;
                } else
                        i1 = i;
        }
i2 = j0 = 0;
for (i = 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9) {
                tmp[1] = tmp[2] = oy[rep[i]];
                break;
        }
for (i = 1; i < num_of_seg; i++)
        if (label[rep[i]] == 9) {
                n[++j0] = rep[i];
                if (ORIENTATION > 0) {
                        tmp[0] = LARGER(oy[rep[i]], tmp[1]);
                        tmp = LARGER(oy[rep[i]], tmp[2]);
                } else {
                        tmp[0] = LARGER(tmp[1], oy[rep[i]]);
                        tmp = LARGER(tmp[2], oy[rep[i]]);
}
if (tmp < 4 * sigma && tmp[0] < Pi / 2)
```

```
        tmp[2] = oy[rep[i]];
    else if (j0 > 6) {
        if (ORIENTATION > 0)
            tmp = LARGER(tmp[2], tmp[1]);
        else
            tmp = LARGER(tmp[1], tmp[2]);
        tmp[3] = tmp / (j0 - 2);
        tmp[2] = 0.0;
        i1 = 0;
        for (j = 2; j < j0; j++) {
            if (ORIENTATION > 0)
                tmp = LARGER(oy[n[j]], oy[n[j - 1]]);
            else
                    tmp = LARGER(oy[n[j - 1]], oy[n[j]]);
            if (tmp > 1.2 * tmp[3]) {
                tmp[2] += tmp;
                i1++;
                if (tmp[2] > (i1 + 0.9) * tmp[3] && i1 < 5) {
                        i0 = i1 + 1;
                        tmp[4] = log(ox[2 * n[j - i1] - 1] / ox[2 * n[j] - 1]) / tmp[2];
                        for (j1 = 1; j1 < i1; j1++) {
                                if (ORIENTATION > 0) {
                                        oy[n[j - j1]] = oy[n[j]] - j1 * tmp[2] / i0;
                                        oy[n[j - j1]] = POSITIVE(oy[n[j - j1]]);
                                } else {
                                        oy[n[j - j1]] = oy[n[j]] + j1 * tmp[2] / i0;
                                        oy[n[j - j1]] = TWOPI(oy[n[j - j1]]);
                                }
                                ox[2 * n[j - j1]] = ox[2 * n[j]] * exp(tmp[4] * j1 * tmp[2] /
                                                i0);
                                ox[2 * n[j - j1] - 1] = ox[2 * n[j] - 1] * exp(tmp[4] * j1 *
                                                tmp[2] / i0);
                        }
                for (j1 = i1; j1 < i0; j1++) {
                                i2++;
                                label[num_of_seg + i2] = 9;
                                if (ORIENTATION > 0) {
                                        oy[num_of_seg + i2] = oy[n[j]] - j1 * tmp[2] / i0;
                                        oy[num_of_seg + i2] = POSITIVE(oy[num_of_seg
                                                        + i2]);
                                } else {
                                        oy[num_of_seg + i2] = oy[n[j]] + j1 * tmp[2] / i0;
                                        oy[num_of_seg + i2] = TWOPI(oy[num_of_seg +
                                                                i2]);
                                }
```

```
                        ox[2 * (num_of_seg + i2)] = ox[2 * n[j]] * exp(tmp[4] * j1 *
                        tmp[2] / i0);
                        ox[2 * (num_of_seg + i2) - 1] = ox[2 * n[j] - 1] * exp(tmp[4] * j1
                        * tmp[2] / i0);
                }
                i1 = 0;
                tmp[2] = 0.0;
        }
    } else {
        i1 = 0;
        tmp[2] = 0.0;
    }
    }
    tmp[1] = tmp[2] = oy[rep[i]];
    n[1] = rep[i];
    j0 = 1;
} else {
    tmp[1] = tmp[2] = oy[rep[i]];
    n[1] = rep[i];
    j0 = 1;
}
        }
    for (i = 1; i < num_of_seg + i2 + 1; i++)
        rep[i] = i;
    for (i = 1; i < num_of_seg + i2 + 1; i++)
        if (label[i] == 9)
            for (j = i + 1; j < num_of_seg + i2 + 1; j++)
                if (label[j] == 9) {
                    if (ox[2 * rep[i] - 1] < ox[2 * rep[j] - 1]) {
                        rep[0] = rep[i];
                        rep[i] = rep[j];
                        rep[j] = rep[0];
                    }
                }
    j0 = 0;
    for (i = 1; i < num_of_seg + i2 + 1; i++)
        if (label[rep[i]] == 9) {
            if (!j0)
                tmp = 2 * sigma;
            else if (ORIENTATION > 0)
                tmp = LARGER(oy[rep[i]], tmp[3]);
            else
                tmp = LARGER(tmp[3], oy[rep[i]]);
            j0++;
            tmp[3] = oy[rep[i]];
            tmp[1] = (ox[2 * rep[i]] - ox[2 * rep[i] - 1]) / 19.0;
```

```
                for (j = 0; j < 19; j++) {
            x[1] = xo + (ox[2 * rep[i] - 1] + j * tmp[1]) * cos(oy[rep[i]] + d[j]);
            y[1] = yo + (ox[2 * rep[i] - 1] + j * tmp[1]) * sin(oy[rep[i]] + d[j]);
            x[2] = xo + (ox[2 * rep[i] - 1] + (j + 1) * tmp[1]) * cos(oy[rep[i]] + d[j + 1]);
            y[2] = yo + (ox[2 * rep[i] - 1] + (j + 1) * tmp[1]) * sin(oy[rep[i]] + d[j + 1]);
            if (abs(y[2] - y[1]) > abs(x[2] - x[1])) {
                tmp[2] = (float) (x[2] - x[1]) / (y[2] - y[1]);
                if (y[2] > y[1]) {
                    y[3] = y[1];
                    y[4] = y[2];
                } else {
                    y[3] = y[2];
                    y[4] = y[1];
                }
                for (j1 = y[3]; j1 <= y[4]; j1++) {
                    i1 = x[2] + tmp[2] * (j1 - y[2]);
                    pr_put(new_image, i1, j1, 1);
                }
            } else {
                tmp[2] = (float) (y[2] - y[1]) / (x[2] - x[1]);
                if (x[2] > x[1]) {
                    x[3] = x[1];
                    x[4] = x[2];
                } else {
                    x[3] = x[2];
                    x[4] = x[1];
                }
                for (i1 = x[3]; i1 <= x[4]; i1++) {
                    j1 = y[2] + tmp[2] * (i1 - x[2]);
                    pr_put(new_image, i1, j1, 1);
                }
            }
        }
}
}

/* fitting rib form */

static void
regress(ox, oy)
        double          ox[], oy[];
{
        register int    i, j, j1;
        double          a[6][100], r[6][6], sum[6], mean[6], s[6], partial[5][5],
        double          t[100];
```

```
        for (i = 0; i < 6; i++)
                sum[i] = s[i] = coe[n[6]][i] = 0.0;
        for (i = 1; i <= n[5]; i++) {
                a[0][i] = oy[i];
                sum[0] += a[0][i];
                for (j = 1; j < 6; j++) {
                        a[j][i] = pow(ox[i], (double) j);
                        sum[j] += a[j][i];
                }
        }
        for (i = 0; i < 6; i++)
                mean[i] = sum[i] / n[5];
        for (i = 0; i < 6; i++) {
                for (j = 1; j <= n[5]; j++)
                        s[i] += sqr(a[i][j] - mean[i]);
                s[i] = sqrt(s[i]);
        }
        for (i = 0; i < 6; i++)
                for (j = i; j < 6; j++) {
                        r[i][j] = 0.0;
                        for (j1 = 1; j1 <= n[5]; j1++)
                                r[i][j] += (a[i][j1] - mean[i]) * (a[j][j1] - mean[j]);
                        r[i][j] /= (s[i] * s[j]);
                        r[j][i] = r[i][j];
                }
        for (i = 1; i < 6; i++)
                for (j = 1; j < 6; j++)
                        partial[i - 1][j - 1] = r[i][j];
        if (!invert(partial, 5))
                fprintf(stderr, "Singular partial correlation matrix");
        for (i = 1; i < 6; i++) {
                for (j = 1; j < 6; j++)
                        coe[n[6]][i] += partial[i - 1][j - 1] * r[0][j];
                coe[n[6]][i] = coe[n[6]][i] * s[0] / s[i];
        }
        coe[n[6]][0] = mean[0];
        for (i = 1; i < 6; i++)
                coe[n[6]][0] -= coe[n[6]][i] * mean[i];
        return;
}

float
invert(matrix, size)
        int             size;
        double          matrix[5][5];
{
```

```
        int             j, k, l;
        double          pivot, temp, det = 1.0;

        for (j = 0; j < size; j++) {
                pivot = matrix[j][j];
                det *= pivot;
                matrix[j][j] = 1.0;
                for (k = 0; k < size; k++)
                        if (pivot)
                                matrix[j][k] /= pivot;
                        else
                                return (0.0);
                for (k = 0; k < size; k++)
                        if (k != j) {
                                temp = matrix[k][j];
                                matrix[k][j] = 0.0;
                                for (l = 0; l < size; l++)
                                        matrix[k][l] -= matrix[j][l] * temp;
                        }
        }
        return ((float) det);
}
```

Appendix 6 Source code for Fourier analysis

There are 6 files in this program: fourier.h, fourier.c, canny.c, trace.c, kuhl.c, foote.c.

1. fourier.h

```
/*
 * Header file for Fourier analysis
 *
 * Author: Bo Liang
 * Department of Geological Sciences
 * The University of British Columbia
 * Date: March 1991
 * Modifications: April 1994
 *
 */
#include <stdio.h>
#include <ctype.h>
#include <pixrect/pixrect_hs.h>
#include <math.h>

#define MAX         100    /* maximum number of line segments allowed */
#define KUHL        0
#define FOOTE       1
#define CLOSE       8      /* tha maximum gap in a closed contour is 8 pixels */

#define Pi      3.14159
#define sqr(i) ((double)(i)*(double)(i))


enum boolean {
        NO, YES
};

struct point {
        short int       x;
        short int       y;
        short int       z;
}       *segment[MAX];      /* hold points in line segments */

short   seg_points[MAX];    /* number of points in each segment */

short   num_of_seg;         /* the number of line segments in the image */

int     outer;              /* index for the outer contour */
```

```
double *a, *b, *c, *d;          /* fourier coefficients. */


Pixrect         *outline_image;
FILE            *fo, *coef_data;;

/* two different Fourier techniques */

/* elliptical fourier analysis, see Kuhl 1982 for detail */

void            kuhl(int num_of_harmonic, int num_of_point);

/* perimeter-based fourier analysis, see Foote 1989 for detail  */

void            foote(int num_of_harmonic, int num_of_point)
```

2. fourier.c

```
#include "fourier.h"

main(argc, argv)
        int     argc;
        char    *argv[];
{
        register int    i, j, nrows, ncols;
        boolean         option;
        int             num_of_harmonic;
        FILE            *fp;
        Pixrect         *whorl_image;


        if (argc < 4) {
                fprintf(stderr, "Please specify a cross section file, a data file to host
                                fourier coefficients and the output image file name");
                exit(1);
        }
        fp = fopen(argv[1], "r");
        if ((whorl_image = pr_load(fp, colormap)) == NULL) {
                fprintf(stderr, "Can't open %s as image file", image_file);
                exit(2);
        }
        ncols = whorl_image->pr_size.x;
        nrows = whorl_image->pr_size.y;

        coef_data = fopen(argv[2], "w");
```

```
outline_image = mem_create(ncols, nrows, bpp);
fo = fopen(image_out, "w");
if ((fo = fopen(argv[3], "w")) == NULL) {
        fprintf(stderr, "Can't open %s for output", argv[3]);
        exit(3);
}
printf("Please give the following parameters:\n");
printf(" 1. Number of harmonic: \n");
scanf("%d", &num_of_harmonic);
printf(" 2. Elliptic (0) / Perimeter-based (1) \n");
scanf("%d", &option);

/* use Canny edge detector to mark the outline of the whorl section */

edge_detect(whorl_image, out_image, ncols, nrows);

/* group edge points into line contour */

trace_segment(ncols, nrows);

/*
 * there are two contours due to the fact that whorl thickness > 1
 * pixel, find the outer contour
 */

j = seg_points[0];
outer = 0;
for (i = 0; i < num_of_seg; i++) {
        if (seg_points[i] > j) {
                j = seg_points[i];
                outer = i;
        }
}

/* check the two end points of the contour, see whether it is closed */

j = sqrt(sqr((segment[outer] - 1)->y - (segment[outer] - seg_points[outer] - 1)->y) +
        sqr((segment[outer] - 1)->x - (segment[outer] - seg_points[outer] - 1)->x));
if (j > CLOSE) {
        fprintf(stderr, "the segment selected is not a closed contour\n");
        exit(4);
}
if (!option) {
        fprintf(coef_data, "Elliptical Fourier coefficients for %s\n", argv[1]);
        kuhl(num_of_harmonic, seg_points[outer]);
```

```
        } else {
                fprintf(coef_data, "Perimeter-based Fourier coefficients for %s\n", argv[1]);
                foote(num_of_harmonic, seg_points[outer]);
        }

        fclose(coef_data);
        pr_dump(outline_image, fo, NULL, RT_STANDARD, 1);
        pr_close(whorl_image);
        pr_close(outline_image);

        exit(0);

}
```

3. canny.c

provided by Department of Computer Science, UBC.

4. trace.c

same as in Appendix 5.

5. kuhl.c

```
/* perform elliptical fourier analysis for images of whorl cross sections
 * the whorl image must be a single closed contour to proceed; based on a
 * paper by Kuhl & Giardina 1982
 */
#include "fourier.h"

extern void
kuhl(num_of_harmonic, num_of_point)
        int     num_of_harmonic, num_of_point;
{

        register int    i, j, n, ymax, ymin;
        int             p[3], xo, yo, xx, yy;
        boolean         cont;
        float           sigfact; /* significance factor for whorl flank */
        double          *x[2], *y[2], *len, x_sum[2], y_sum[2];
        double          *aa, *bb, *cc, *dd;     /* fourier coef. with
                                                 * normalization */
        double          θ, φ, sigma[2], tmp[2];

        /* allocate memory for x, y and len */

        x[0] = (double) malloc((num_of_point + 1) * sizeof(double));
        y[0] = (double) malloc((num_of_point + 1) * sizeof(double));
```

```
x[1] = (double) malloc((num_of_point + 1) * sizeof(double));
y[1] = (double) malloc((num_of_point + 1) * sizeof(double));
len = (double) malloc((num_of_point + 1) * sizeof(double));

/*
 * read coords of points in whorl outline segment into x[0], y[0];
 * len[i] is the curve length at point i;
 */

len[0] = 0.0;
x[0][1] = (segment[outer] - 1)->x;
y[0][1] = ymin = ymax = (segment[outer] - 1)->y;
x_sum[0] = x[0][1];
y_sum[0] = y[0][1];
for (i = 2; i <= num_of_point + 1; i++) {
        x[0][i] = (segment[outer] - i)->x;
        y[0][i] = (segment[outer] - i)->y;
        if (y[0][i] < ymin)
                ymin = y[0][i];
        if (y[0][i] > ymax)
                ymax = y[0][i];
        len[i - 1] = len[i - 2] + sqrt(sqr(y[0][i] - y[0][i - 1]) + sqr(x[0][i] - x[0][i - 1]));
        x_sum[0] += x[0][i];
        y_sum[0] += y[0][i];
}

/* approximate venter shoulder with two points p[1], p[2] */

cont = YES;
for (i = 1; i <= num_of_point + 1; i++) {
        if (cont && y[0][i] < ymin + (ymax - ymin) / 10.0) {
                p[1] = i;
                cont = NO;
        }
        if (cont == NO && y[0][i] > ymin + (ymax - ymin) / 10.0) {
                p[2] = i;
                break;
        }
}

/* compute the raw fourier coefficients */

for (i = 1; i < num_of_harmonic; i++) {
        a[i] = b[i] = c[i] = d[i] = 0.0;
        for (j = 1; j <= num_of_point; j++) {
                sigma[0] = 2 * Pi * i * len[j] / len[num_of_point];
```

```
                sigma[1] = 2 * Pi * i * len[j - 1] / len[num_of_point];
                a[i] += (x[0][j + 1] - x[0][j]) * (cos(sigma[0]) - cos(sigma[1])) /
                        (len[j] - len[j - 1]);
                b[i] += (x[0][j + 1] - x[0][j]) * (sin(sigma[0]) - sin(sigma[1])) /
                        (len[j] - len[j - 1]);
                c[i] += (y[0][j + 1] - y[0][j]) * (cos(sigma[0]) - cos(sigma[1])) /
                        (len[j] - len[j - 1]);
                d[i] += (y[0][j + 1] - y[0][j]) * (sin(sigma[0]) - sin(sigma[1])) /
                        (len[j] - len[j - 1]);
        }
        a[i] *= len[num_of_point] / (2 * i * i * Pi * Pi);
        b[i] *= len[num_of_point] / (2 * i * i * Pi * Pi);
        c[i] *= len[num_of_point] / (2 * i * i * Pi * Pi);
        d[i] *= len[num_of_point] / (2 * i * i * Pi * Pi);
}

/* normalize fourier coefficients */

θ = atan2(2 * (a[1] * b[1] + c[1] * d[1]), (a[1] * a[1] + c[1] * c[1] -
                b[1] * b[1] - d[1] * d[1]));
if (θ < 0.0)
        θ += 2 * Pi;
θ = θ / 2.0;
tmp[0] = a[1] * cos(θ) + b[1] * sin(θ);
tmp[1] = c[1] * cos(θ) + d[1] * sin(θ);
φ = atan2(tmp[0], tmp[1]);
if (φ < 0.0)
        φ += 2 * Pi;
magnitude = sqrt(sqr(tmp[0]) + sqr(tmp[1]));
for (i = 1; i < num_of_harmonic; i++) {
        aa[i] = a[i] * cos(φ) * cos(i * θ) + c[i] * sin(φ) * cos(i * θ) +
                b[i] * cos(φ) * sin(i * θ) + d[i] * sin(φ) * sin(i * θ);
        bb[i] = -a[i] * cos(φ) * sin(i * θ) - c[i] * sin(φ) * sin(i * θ) +
                b[i] * cos(φ) * cos(i * θ) + d[i] * sin(φ) * cos(i * θ);
        cc[i] = -a[i] * sin(φ) * cos(i * θ) + c[i] * cos(φ) * cos(i * θ) -
                b[i] * sin(φ) * sin(i * θ) + d[i] * cos(φ) * sin(i * θ);
        dd[i] = a[i] * sin(φ) * sin(i * θ) - c[i] * cos(φ) * sin(i * θ) -
                b[i] * sin(φ) * cos(i * θ) + d[i] * cos(φ) * cos(i * θ);
}

/* compute a[0] and c[0] */

for (i = 1; i < num_of_harmonic; i++) {
        a[i] = aa[i];
        b[i] = bb[i];
```

```
        c[i] = cc[i];
        d[i] = dd[i];
}
x[0][1] = y[0][1] = 0;
tmp[0] = θ * len[num_of_point] / (2 * Pi);
for (i = 1; i < num_of_harmonic; i++) {
        sigma[0] = 2 * Pi * (len[1] - tmp[0]) * i / len[num_of_point];
        x[1][1] += a[i] * cos(sigma[0]) + b[i] * sin(sigma[0]);
        y[1][1] += c[i] * cos(sigma[0]) + d[i] * sin(sigma[0]);
}
xo = (segment[outer] - 1)->x;
yo = (segment[outer] - 1)->y;
a[0] = xo * cos(φ) - yo * sin(φ) - x[1][1];
c[0] = xo * sin(φ) + yo * cos(φ) - y[1][1];

/* give different significance for venter and flank */

x_sum[1] = 0;
y_sum[1] = 0;
for (i = 1; i <= num_of_point; i++) {
        x[1][i] = a[0];
        y[1][i] = c[0];
        for (j = 1; j < num_of_harmonic; j++) {
                sigma[0] = 2 * Pi * (len[i] - tmp[0]) * j / len[num_of_point];
                x[1][i] += a[j] * cos(sigma[0]) + b[j] * sin(sigma[0]);
                y[1][i] += c[j] * cos(sigma[0]) + d[j] * sin(sigma[0]);
        }
        x_sum[1] += x[1][i] * cos(φ) - y[1][i] * sin(φ);
        y_sum[1] += x[1][i] * sin(φ) + y[1][i] * cos(φ);
}
xo = -x_sum[1] / num_of_point + x_sum[0] / (num_of_point + 1);
yo = -y_sum[1] / num_of_point + y_sum[0] / (num_of_point + 1);
x_sum[0] = y_sum[0] = 0;
sigfact = num_of_point / (4 * (p[2] - p[1]));
for (i = 1; i <= num_of_point; i++) {
        tmp[1] = (x[0][i] - xo) * cos(φ) + (y[0][i] - yo) * sin(φ);
        y[0][i] = -(x[0][i] - xo) * sin(φ) + (y[0][i] - yo) * cos(φ);
        x[0][i] = tmp[1];
        if (i > p[1] && i < p[2]) {
                x_sum[0] += sigfact * fabs(x[1][i] - x[0][i]);
                y_sum[0] += sigfact * fabs(y[1][i] - y[0][i]);
        } else {
                x_sum[0] += fabs(x[1][i] - x[0][i]);
                y_sum[0] += fabs(y[1][i] - y[0][i]);
        }
}
```

```
/* eliminate coef. that don't have significant contribution */

cont = YES;
while (cont) {
        tmp[1] = 100 * x_sum[0];
        for (j = num_of_harmonic - 1; j > 1; j--) {
                if (fabs(a[j]) > 0.0) {
                        x_sum[1] = 0.0;
                        for (i = 1; i <= num_of_point; i++) {
                                sigma[0] = 2 * Pi * (len[i] - tmp[1]) * j /
                                                len[num_of_point];
                                if (i > p[1] && i < p[2])
                                        x_sum[1] += sigfact * fabs(x[1][i] - a[j] *
                                                cos(sigma[0]) - x[0][i]);
                                else
                                        x_sum[1] += fabs(x[1][i] - a[j] *
                                                cos(sigma[0]) - x[0][i]);
                        }
                        if (fabs(tmp[1]) > x_sum[1]) {
                                tmp[1] = x_sum[1];    /* contribution by a[n] */
                                n = j;
                        }
                }
                if (fabs(b[j]) > 0.0) {
                        x_sum[1] = 0.0;
                        for (i = 1; i <= num_of_point; i++) {
                                sigma[0] = 2 * Pi * (len[i] - tmp[0]) * j /
                                                len[num_of_point];
                                if (i > p[1] && i < p[2])
                                        x_sum[1] += sigfact * fabs(x[1][i] - b[j] *
                                                sin(sigma[0]) - x[0][i]);
                                else
                                        x_sum[1] += fabs(x[1][i] - b[j] * sin(sigma[0]) -
                                                x[0][i]);
                        }
                        if (fabs(tmp[1]) > x_sum[1]) {
                                tmp[1] = -x_sum[1];   /* contribution by b[n] */
                                n = j;
                        }
                }
        }
        if (fabs(tmp[1]) < 1.5 * x_sum[0]) {  /* not siginificant */
                if (tmp[1] > 0) {
                        for (i = 1; i <= num_of_point; i++)
```

```
                        x[1][i] -= a[n] * cos(2 * Pi * (len[i] - tmp[0]) * n /
                                len[num_of_point]);   /* re-compute x after

                 * eliminating a[n] */
                        a[n] = 0;
                } else {
                        for (i = 1; i <= num_of_point; i++)
                                x[1][i] -= b[n] * sin(2 * Pi * (len[i] - tmp[0]) * n /
                                len[num_of_point]);   /* re-compute x after

                 * eliminating b[n] */
                        b[n] = 0;
                }
        } else
                cont = NO;
}

cont = YES;
while (cont) {
        tmp[1] = 100 * y_sum[0];
        for (j = num_of_harmonic - 1; j > 1; j--) {
                if (fabs(c[j]) > 0.0) {
                        y_sum[1] = 0.0;
                        for (i = 1; i <= num_of_point; i++) {
                                sigma[0] = 2 * Pi * (len[i] - tmp[0]) * j /
                                                len[num_of_point];
                                if (i > p[1] && i < p[2])
                                        y_sum[1] += sigfact * fabs(y[1][i] - c[j] *
                                                cos(sigma[0]) - y[0][i]);
                                else
                                        y_sum[1] += fabs(y[1][i] - c[j] * cos(sigma[0]) -
                                                y[0][i]);
                        }
                        if (fabs(tmp[1]) > y_sum[1]) {
                                tmp[1] = y_sum[1];    /* contribution by c[n] */
                                n = j;
                        }
                }
                if (fabs(d[j]) > 0.0) {
                        y_sum[1] = 0.0;
                        for (i = 1; i <= num_of_point; i++) {
                                sigma[0] = 2 * Pi * (len[i] - tmp[0]) * j /
                                                len[num_of_point];
                                if (i > p[1] && i < p[2])
                                        y_sum[1] += sigfact * fabs(y[1][i] - d[j] *
                                                sin(sigma[0]) - y[0][i]);
```

```
                                else
                                        y_sum[1] += fabs(y[1][i] - d[j] * sin(sigma[0]) -
                                                y[0][i]);
                        }
                        if (fabs(tmp[1]) > y_sum[1]) {
                                tmp[1] = -y_sum[1];   /* contribution by d[n] */
                                n = j;
                        }
                }
        }
        if (fabs(tmp[1]) < 1.5 * y_sum[0]) {  /* not siginificant */
                if (tmp[1] > 0) {
                        for (i = 1; i <= num_of_point; i++)
                                y[1][i] -= c[n] * cos(2 * Pi * (len[i] - tmp[0]) * n /
                                        len[num_of_point]);   /* re-compute y after
                                                               * eliminating c[n] */
                        c[n] = 0;
                } else {
                        for (i = 1; i <= num_of_point; i++)
                                y[1][i] -= d[n] * sin(2 * Pi * (len[i] - tmp[0]) * n /
                                        len[num_of_point]);   /* re-compute y after
                                                               * eliminating d[n] */
                        d[n] = 0;
                }
        } else
                cont = NO;
    }

}

/* save normalized Fourier coefficients */

b[1] = c[1] = 0;                /* TRUE if Fourier coefficients has been
                                 * normalized */

fprintf(coef_data, "A0=%f C0=%f\n", xo, yo);
for (i = 1; i < num_of_harmonic; i++) {
    if (fabs(a[i]) > 0.0 || fabs(b[i]) > 0.0 || fabs(c[i]) > 0.0 || fabs(d[i]) > 0.0) {
        fprintf(coef_data, "harmonic%d ", i);
        if (fabs(a[i]) > 0.0)
                fprintf(coef_data, "  a=%f ", a[i] / magnitude);
        if (fabs(b[i]) > 0.0)
                fprintf(coef_data, "  b=%f ", b[i] / magnitude);
        if (fabs(c[i]) > 0.0)
                fprintf(coef_data, "  c=%f ", c[i] / magnitude);
        if (fabs(d[i]) > 0.0)
```

```
                fprintf(coef_data, "  d=%f ", d[i] / magnitude);
            fprintf(coef_data, "  \n");
        }
}

/* plot Fourier Rep of whorl in outline_image */

for (i = 1; i <= num_of_point; i++) {
        x[1][i] = a[0];
        y[1][i] = c[0];
        for (j = 1; j < num_of_harmonic; j++) {
            sigma[0] = 2 * Pi * (len[i] - tmp[0]) * j / len[num_of_point];
            x[1][i] += a[j] * cos(sigma[0]) + b[j] * sin(sigma[0]);
            y[1][i] += c[j] * cos(sigma[0]) + d[j] * sin(sigma[0]);
        }
        xx = x[1][i] * cos(φ) - y[1][i] * sin(φ) + xo;
        yy = x[1][i] * sin(φ) + y[1][i] * cos(φ) + yo;
        pr_put(outline_image, xx, yy, 1);

        free(x[0]);
        free(x[1]);
        free(y[0]);
        free(y[1]);
        free(len);


}
```

6. foote.c

```
/* perform perimeter-based fourier analysis for images of whorl cross sections
 * the whorl image must be a single closed contour to proceed; based on a
 * paper by Foote 1989
 */
#include "fourier.h"

foote(num_of_harmonic, num_of_point)
        int             num_of_harmonic, num_of_point;
{
        register int    i, j, ymin, ymax, int x, y, xo, yo, p[3];
        double          *r, *θ, *len;
        double          sigma[2], φ, rr,;

        /* allocate memory for r, s, and len */

        r = (double) malloc((num_of_point + 1) * sizeof(double));
```

```
θ = (double) malloc((num_of_point + 1) * sizeof(double));
len = (double) malloc((num_of_point + 1) * sizeof(double));

/* locate the geometric center of the whorl section */

ymin = (segment[outer] - 1)->y;
ymax = (segment[outer] - 1)->y;
for (i = 2; i < num_of_point / 2; i++) {
        j = (segment[outer] - i)->y;
        if (j > ymax) {
                p[0] = i;       /* lowest point at umbilical shoulder */
                ymax = j;
        } else if (j < ymin) {
                p[1] = i;       /* highest point at venter */
                ymin = j;
        }
}
ymax = (segment[outer] - num_of_point)->y;
for (i = num_of_point - 1; i > num_of_point / 2; i--) {
        j = (segment[outer] - i)->y;
        if (j > ymax) {
                p[2] = i;       /* lowest point at umbilical shoulder */
                ymax = j;
        } else if (y < ymin) {
                p[1] = i;
                ymin = j;
        }
}
xo = ((segment[outer] - p[1])->x + ((segment[outer] - p[0])->x +
        (segment[outer] - p[2])->x) / 2) / 2;
yo = ((segment[outer] - p[1])->y + ((segment[outer] - p[2])->y +
        (segment[outer] - p[1])->y) / 2) / 2;

/* convert from x-y coord to r-s system, start from p[0] */

len[0] = 0.0;
j = 1;
for (i = p[0]; i >= 1; i--) {
        x = (segment[outer] - i)->x;
        y = (segment[outer] - i)->y;
        r[j] = sqrt(sqr(y - yo) + sqr(x - xo));
        len[j] = len[j - 1] + sqrt(sqr(y - (segment[outer] - i - 1)->y) + sqr(x -
                (segment[outer] - i - 1)->x));
        if (y == yo) {
                if (x > xo)
                        θ[j] = 0.0;
```

```
                else
                        θ[j] = -Pi;
        } else
                θ[j] = atan2((double) y - yo, (double) x - xo);
        j++;
}
for (i = num_of_point - 1; i >= p[2]; i--) {
        x = (segment[outer] - i)->x;
        y = (segment[outer] - i)->y;
        r[j] = sqrt(sqr(y - yo) + sqr(x - xo));
        len[j] = len[j - 1] + sqrt(sqr(y - (segment[outer] - i - 1)->y) +
                sqr(x - (segment[outer] - i - 1)->x));
        if (y == yo) {
                if (x > xo)
                        θ[j] = 0.0;
                else
                        θ[j] = -Pi;
        } else
                θ[j] = atan2((double) y - yo, (double) x - xo);
        j++;
}


/* normalize r and θ */

num_of_point = j - 1;
sigma[1] = 2 * Pi - fabs(θ[1] - θ[num_of_point]);
magnitude = len[num_of_point] / sigma[1];
sigma[0] = θ[1];
if (sigma[0] - Pi / 2) {
        for (i = 1; i <= num_of_point; i++) {
                r[i] = r[i] / magnitude;
                if (θ[i] - sigma[0] >= 0.0)
                        θ[i] = θ[i] - sigma[0] - len[i] * sigma[1] / len[num_of_point];
                else
                        θ[i] = 2 * Pi + θ[i] - sigma[0] - len[i] * sigma[1] /
                                len[num_of_point];
        }
} else {
        for (i = 1; i <= num_of_point; i++) {
                r[i] = r[i] / magnitude;
                if (-θ[i] + sigma[0] >= 0.0)
                        θ[i] = -θ[i] + sigma[0] - len[i] * sigma[1] / len[num_of_point];
                else
```

```
                    θ[i] = 2 * Pi - θ[i] + sigma[0] - len[i] * sigma[1] /
                            len[num_of_point];
        }
}

/* compute Fourier coefficients */

for (i = 0; i < num_of_harmonic; i++) {
        a[i] = b[i] = c[i] = d[i] = 0.0;
        for (j = 1; j <= num_of_point; j++) {
                a[i] += r[j] * cos(2 * Pi * i * len[j] / (double) len[num_of_point]);
                b[i] += r[j] * sin(2 * Pi * i * len[j] / (double) len[num_of_point]);
                c[i] += θ[j] * cos(2 * Pi * i * len[j] / (double) len[num_of_point]);
                d[i] += θ[j] * sin(2 * Pi * i * len[j] / (double) len[num_of_point]);
        }
        a[i] = 2 * a[i] / (double) len[num_of_point];
        b[i] = 2 * b[i] / (double) len[num_of_point];
        c[i] = 2 * c[i] / (double) len[num_of_point];
        d[i] = 2 * d[i] / (double) len[num_of_point];
}

/* plot simulated whorl using Fourier Rep. */

for (i = 1; i <= num_of_point; i++) {
        rr = a[0] / 2.0;
        φ = c[0] / 2.0;
        for (j = 1; j < num_of_harmonic; j++) {
                rr += a[j] * cos(2 * Pi * len[i] * j / (double) len[num_of_point]) +
                        b[j] * sin(2 * Pi * len[i] * j / (double) len[num_of_point]);
                φ += c[j] * cos(2 * Pi * len[i] * j / (double) len[num_of_point]) +
                        d[j] * sin(2 * Pi * len[i] * j / (double) len[num_of_point]);
        }
        if (sigma[1] - Pi / 2) {
                x = magnitude * rr * cos((double) φ + len[i] * sigma[1] /
                        len[num_of_point] + sigma[0]) + xo;
                y = magnitude * rr * sin((double) φ + len[i] * sigma[1] /
                        len[num_of_point] + sigma[0]) + yo;
        } else {
                x = magnitude * rr * cos((double) sigma[0] - φ - len[i] * sigma[1] /
                len[num_of_point]) + xo;
                y = magnitude * rr * sin((double) sigma[0] - φ - len[i] * sigma[1] /
                len[num_of_point]) + yo;
        }
        pr_put(image_out, x + 20, y + 20, 1);
}
```

```
        free(r);
        free(s);
        free(len);

}
```

Appendix 7 Source code for the animation of whorl shape changes

This is a simple whorl shape change simulation program.

```
/* This program is used to interpolate or extrapolate sequence of whorl shape */

#include <stdio.h>
#include <math.h>
#include <pixrect/pixrect_hs.h>

#define MAXCOEF   100
#define NUMPOINT  2000
#define PI        3.14159
#define INTP      4          /* number of steps during interpolation */
#define EXTP      8          /* number of steps during exterpolation */

enum boolean {
        NO, YES
};

struct rect {
        int     origin_x;
        int     origin_y;
        int     width;
        int     height;
}       disp_rect = {50, 50, 600, 600 };

double          *a[MAXCOEF], *b[MAXCOEF], *c[MAXCOEF], *d[MAXCOEF];
double          x[NUMPOINT], y[NUMPOINT];

main(argc, argv)
        int     argc;
        char    *argv[];
{
        register int    i, j;
        int             num_of_section, num_of_harmonic;
        boolean         option, cont;
        FILE            *coef_data;
        Pixrect         *whorl_section, *scrn;

        if (argc < 2) {
                fprintf(stderr, "Please specify the Fourier coefficient data file.");
                exit(1);
        }
        if ((coef_data = fopen(argv[1], "r")) == NULL) {
                printf(stderr, "Can't open %s", argv[1]);
```

```
            exit(1);
        }
        fscanf(coef_data, "%d", &num_of_section);
        fscanf(coef_data, "%d", &i);
        num_of_harmonic = i / 4;/* each harmonic has a, b, c, d components */

        /* allocate memory to host the fourier coefficients */

        for (i = 0; i < num_of_section; i++) {
            a[i] = (double *) malloc(4 * (num_of_harmonic + 1) * sizeof(double));
            b[i] = (double *) malloc(4 * (num_of_harmonic + 1) * sizeof(double));
            c[i] = (double *) malloc(4 * (num_of_harmonic + 1) * sizeof(double));
            d[i] = (double *) malloc(4 * (num_of_harmonic + 1) * sizeof(double));
        }

        /* prepare screen for display */

        whorl_section = mem_create(disp_rect.width, disp_rect.height, 1);
        scrn = pr_open("/dev/fb");

        /* load Fourier coefficients */

        for (i = 1; i <= num_of_section; i++)  /* reserve i = 0 as buffer */
            for (j = 0; j < num_of_harmonic; j++)
                fscanf(coef_data, "%f %f %f %f", &a[i][j], &b[i][j], &c[i][j], &d[i][j]);

        cont = YES;
        while (cont) {

            printf("visualize the whole sequence(0)/use two sections for extrapolation (1)");
            scanf("%d", &option);

            if (option)
                extrapolate(num_of_harmonic, num_of_section);
            else
                interpolate(num_of_harmonic, num_of_section);
        }

        pr_close(scrn);
}

static void
extrapolate(num_of_harmonic, num_of_section)
        int     num_of_harmonic, num_of_section
        {
        register int     i, j;
```

```
        int             section[2];

        printf("Which part of the sequence you want to see? i.e. 4,5. 1-%d\n",
                num_of_section);
        scanf("%d", &section[0]);
        scanf("%d", &section[1]);

        /*
         * section[1] > section[0], extrapolate forward; section[1] <section[0],
         * extrapolate backward;
         */

        /* when i = INTP, whorl change from section[0] to section [1],
         * i > INTP for extrapolation */

        for (i = 0; i <= EXTP; i++)
{               for (j = 0; j < num_of_harmonic; j++) {
                        a[0][j] = a[section[0]][j] + i * (a[section[1]][j] -
                                a[section[0]][j]) / INTP;
                        b[0][j] = b[section[0]][j] + i * (b[section[1]][j] -
                                b[section[0]][j]) / INTP;
                        c[0][j] = c[section[0]][j] + i * (c[section[1]][j] -
                                c[section[0]][j]) / INTP;
                        d[0][j] = d[section[0]][j] + i * (d[section[1]][j] -
                                d[section[0]][j]) / INTP;
}

/* compute x, y of the Fourier Rep of the whorl section */

                compute_fourier(num_of_harmonic);

                display(x, y, i);
}

        }

static void
interpolate(num_of_harmonic, num_of_section)
        int     num_of_harmonic, num_of_section
        {
        register int    i, j, k;


        for (i = 1; i < num_of_section; i++)
                for (k = 0; k < INTP; k++)    {
                        for (j = 0; j < num_of_harmonic; j++) {
```

```
                        a[0][j] = a[i][j] + k * (a[i + 1][j] - a[i][j]) / INTP;
                        b[0][j] = b[i][j] + k * (b[i + 1][j] - b[i][j]) / INTP;
                        c[0][j] = c[i][j] + k * (c[i + 1][j] - c[i][j]) / INTP;
                        d[0][j] = d[i][j] + k * (d[i + 1][j] - d[i][j]) / INTP;
}

        /* compute x, y of the Fourier Rep of the whorl section */

        compute_fourier(num_of_harmonic);
        display(x, y, k);
}
    }

static void
compute_fourier(num_of_harmonic)
    int     num_of_harmonic
    {
    register int    i, j;
    double          tp;

    for (i = 1; i < NUMPOINT; i++)
    {
        x[i] = 0.0;
        y[i] = 0.0;
        tp = tp;
        for (j = 1; j < num_of_harmonic + 1; j++) {
            x[i] += a[0][j - 1] * cos(tp) + b[0][j - 1] * sin(tp);
            y[i] += c[0][j - 1] * cos(tp) + d[0][j - 1] * sin(tp);
                                                    }
    }
    }

static void
display(x, y, order_in_sequence)
int     x[], y[], order_in_sequence;
{
    register int    i, j;
    register int    xmin, xmax, ymin, ymax,     /* determine the dimention of the
                                                 * whorl section */
    xmin = xmax = x[1];
    ymin = ymax = y[1];
    for (j = 2; j < NUMPOINT; j++) {
        if (xmin > x[j])
            xmin = x[j];
        if (xmax < x[j])
```

```
                xmax = x[j];
        if (ymin > y[j])
                ymin = y[j];
        if (ymax < y[j])
                ymax = y[j];
}

if (order_in_sequence == 0) {
        scale = (xmax - xmin) > (ymax - ymin) ? (xmax - xmin) : (ymax - ymin);
                        scale /= displ_rect.width;
                }
for (j = 1; j < NUMPOINT; j++) {
        x[j] = (x[j] - xmin) / scale;
        y[j] = (y[j] - ymin) / scale;
        pr_put(whorl_section, (int) x[j], (int) y[j], 1);
                }
pr_rop(scrn, disp_rect.orig_x, disp_rect.orig_y, displ_rect.width, displ_rect.height,
        PIX_SRC, whorl_section, 0, 0);
        }
}
```