An Executive System for Modeling with Rational B-Splines

by

Glenn Roy Hottel, SR
B.S. and M.S., Nuclear Engineering, Purdue University (1978)

Submitted to the Department of

OCEAN ENGINEERING

In partial fulfillment of the
requirements for the Degree of

NAVAL ENGINEER

and

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE of TECHNOLOGY

MAY , 1989

Certified By :

Professor N. M. Patrikalakis
Department of Ocean Engineering
Thesis Supervisor

Professor P. E. Sullivan
Department of Ocean Engineering
Thesis Supervisor

Professor D. C. Gossard
Mechanical Engineering Department
Thesis Reader

Department of Ocean Engineering
Author, May, 1989

Accepted By :

A. Douglas Carmichael, Chairman
Department Graduate Committee
Department of Ocean Engineering

Ain A. Sonin, Chairman
Committee on Graduate Studies,
Mechanical Engineering Department

An Executive System for Modeling With Rational B-Splines

by

Glenn Roy Hottel, SR

Submitted to the Department of Ocean Engineering and Department of Mechanical Engineering on May 6, 1989, in partial fulfillment of the requirements for the Degree of Naval Engineer and Degree of Master of Science in Mechanical Engineering

ABSTRACT

An editor for use in the modeling of surfaces and curves with non-uniform rational B-splines (NURBS) was developed. A comprehensive menu structure has been generated and a method for interfacing future modules into this structure was developed and discussed with examples.

Surface modules interfaced include: Gaussian, mean, normal and principal curvature presentations; shading with light source and color editing; presentation editing with full positioning and rotation capabilities; and, isophote line calculation.

Curve modules interfaced include entering and editing points in the parametric space of a B-spline surface; generation of a B-spline curve interpolating these points while staying on the surface; and, fairing of the curve to get a smoother shape for a curve on a surface.

Implementation of the editor uses a DEC VAX Station II with the NAG Mark 11 library. The display graphics are performed on a Silicon Graphics IRIS 3030 Workstation networked with the VAX station.

Thesis Supervisor : N. M. Patrikalakis, Ph.D.
Title : Assistant Professor of Ocean Engineering

Thesis Supervisor : P. E. Sullivan
Title : Associate Professor of Ocean Engineering

A-1

ACKNOWLEDGMENTS

The United States Navy provided tuition and salary during my stay at M.I.T and for this I am truly appreciative. Without this assistance I would not have been able to have this valuable educational experience.

The MIT Ocean Engineering Design Laboratory research in the area of this thesis is supported by the Naval Sea Systems Command of the U.S. Navy and the MIT Sea Grant College Program under contract number NA86AA-D-SG089.

Special thanks go to Mr. Mike Drooker who gave me hours of assistance in many different areas, all of which were crucial to my work. The majority of the routines interfaced into the editor are the work of Mr. Panagiotis Alourdas. His assistance in helping me understand their inner workings is greatly appreciated.

A number of the members of the Design Laboratory unselfishly answered questions and gave assistance to me on the many occasions when I became lost with the various operations of the laboratory. To Mr. George Kriezis, Mr. Bradley Moran and Mr. Seamus Tuohy, again, thank you for your help.

DEDICATION

To my wife Janet and four children, Kelly, Chris, Emily and Glenn Roy, JR, whose love and support made this possible and without whom it would all be meaningless.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION AND OUTLINE 21

CHAPTER 2 - DATA AND FILE STRUCTURES 23
2.1 General 23
2.2 B-Spline Curves 24
2.3 B-Spline Surfaces 28
2.4 Existing Structures 30
2.4.1 egeom 30
2.4.2 fgeom 31
2.5 Developed Structures 32
2.5.1 Mouse_Words 33
2.5.2 Menu_Entry 35
2.5.3 Stats 37
2.5.4 Message 38
2.5.5 Choice_List 39
2.5.6 FulCurv 39
2.5.7 FulSurf 42
2.6 File Naming Conventions 46

CHAPTER 3 - HELP FILE 49
3.1 Input Help File 49
3.2 Help File Processing Program 51
3.3 Generated Files 53
3.3.1 Message Files 53
3.3.2 Pointer Files 53
3.3.3 Include File 54
3.3.3.1 Define Lines 54
3.3.3.2 Extern Lines 54
3.3.3.3 Array Set Ups 55

CHAPTER 4 - MENU FILES 57
4.1 Menu Data Structure 57
4.2 Menu Tree Structure 58
4.3 Menu Interaction 62

CHAPTER 5 - EDITOR PROGRAMS AND LAYOUT 65
5.1 CHANGING A VIEW POINT 67
5.2 MAIN MENU 68
5.2.1 INPUT ROUTINES 68
5.2.1.1 CURVE (3-D) 68
5.2.1.1.1 ENTER FORM KEYBOARD 68
5.2.1.1.2 RECALL FRO LOCAL FILE 68
5.2.1.1.3 RECALL IGES FILE 68
5.2.1.1.4 INTERACTIVE INPUT 68
5.2.1.2 SURFACE 68
5.2.1.2.1 ENTER FROM KEYBOARD 68
5.2.1.2.2 RECALL FROM LOCAL FILE 68
5.2.1.2.3 RECALL IGES FILE 69
5.2.1.2.4 INTERACTIVE INPUT 69

5.2.1.3 CURVE ON SURFACE 70
5.2.1.3.1 ENTER FROM KEYBOARD 70
5.2.1.3.2 RECALL FROM LOCAL FILE 70
5.2.1.3.3 RECALL IGES FILE 70
5.2.1.4 ALGEBRAIC SURFACE 70
5.2.1.4.1 ENTER FROM KEYBOARD 70
5.2.1.4.2 RECALL FROM LOCAL FILE 70
5.2.1.5 GRID OF POINTS 70
5.2.1.5.1 ENTER FROM KEYBOARD 70
5.2.1.5.2 RECALL FROM LOCAL FILE 70
5.2.1.6 FUNCTION ON CURVE 70
5.2.1.6.1 ENTER FROM KEYBOARD 70
5.2.1.6.2 RECALL FROM LOCAL FILE 70
5.2.1.7 LIST OF POINTS 70
5.2.1.7.1 ENTER FROM KEYBOARD 70
5.2.1.7.2 RECALL FROM LOCAL FILE 70
5.2.1.7.3 INTERACTIVE INPUT 70
5.2.1.8 LIST OF LISTS 70
5.2.1.8.1 RECALL FROM LOCAL FILE 70
5.2.1.8.2 INTERACTIVE INPUT 70
5.2.1.9 LIST OF POINTS (3-D) 70
5.2.1.9.1 ENTER FROM KEYBOARD 70
5.2.1.9.2 RECALL FROM LOCAL FILE 70
5.2.1.9.3 INTERACTIVE INPUT 71
5.2.2 GEOMETRY GENERATION 71
5.2.2.1 CURVES 71
5.2.2.1.1 FIT POINTS IN 3-D 71
5.2.2.1.2 APPROXIMATE WITH NURBS 71
5.2.2.1.3 OFFSET OF A PLANAR CURVE 71
5.2.2.1.4 OFFSET NORMAL TO PATCH 71
5.2.2.2 SURFACES 71
5.2.2.2.1 OFFSET OF ANOTHER SURFACE 71
5.2.2.2.2 RULED SURFACE 71
5.2.2.2.3 FIT/APPROXIMATE n ISOPARAMETER LINES 71
5.2.2.2.4 FIT/APPROXIMATE GRID OF POINTS ... 71
5.2.2.2.5 CONVERT ALGEBRAIC TO NURBS 71
5.2.2.3 CURVES ON SURFACE 71
5.2.2.3.1 FIT/APPROXIMATE LIST OF POINTS --> calls submenu (see Chapter 5.3) 71
5.2.2.3.2 FIT/APPROXIMATE LIST OF LISTS 71
5.2.2.3.3 VARIABLE OFFSET OF ANOTHER CURVE ON SURFACE 71
5.2.2.4 BLEND 71
5.2.2.4.1 BOUNDARY CONDITIONS 71
5.2.2.4.1.1 POSITION 71
5.2.2.4.1.2 NORMAL 71
5.2.2.4.1.3 CURVATURE 72
5.2.2.4.2 DEFINE SURFACE 72
5.2.2.4.3 DEFINE CURVES 72
5.2.2.4.4 EXECUTE BLEND 72

5.2.3 GEOMETRY INTERROGATION 72
5.2.3.1 CURVES 72
5.2.3.1.1 VISUALIZATION 72
5.2.3.1.1.1 RESOLUTION 72
5.2.3.1.1.2 COLOR 72
5.2.3.1.1.3 VIEWPOINT 72
5.2.3.1.2 CURVATURE VALUES 72
5.2.3.1.2.1 RESOLUTION 72
5.2.3.1.2.2 SHOW CURVATURE MAP 72
5.2.3.1.3 STATUS 72
5.2.3.1.3.1 ON 72
5.2.3.1.3.2 OF 72
5.2.3.2 CURVES ON SURFACE 72
5.2.3.2.1 VISUALIZATION 72
5.2.3.2.1.1 RESOLUTION 72
5.2.3.2.1.2 LINETYPE 72
5.2.3.2.1.3 VIEWPOINT 72
5.2.3.2.2 CURVATURE MAP 72
5.2.3.2.2.1 RESOLUTION 72
5.2.3.2.2.2 SHOW 72
5.2.3.2.3 STATUS 73
5.2.3.2.3.1 ON 73
5.2.3.2.3.2 OFF 73
5.2.3.3 SURFACE 73
5.2.3.3.1 VISUALIZATION 73
5.2.3.3.1.1 RESOLUTION 73
5.2.3.3.1.2 COLOR 73
5.2.3.3.1.3 VIEWPOINT 73
5.2.3.3.2 PLANE CONTOURS 74
5.2.3.3.2.1 SET # PLANES 74
5.2.3.3.2.2 SET START PLANE 74
5.2.3.3.2.3 SET PLANE DISTANCE 74
5.2.3.3.2.4 INTERSECTION ACCURACY 74
5.2.3.3.2.4.1 2-D 74
5.2.3.3.2.4.2 3-D 74
5.2.3.3.3 CYLINDER CONTOURS 74
5.2.3.3.3.1 SET # CYLINDERS 74
5.2.3.3.3.2 SET START CYLINDER 74
5.2.3.3.3.3 CYLINDER DISTANCE 74
5.2.3.3.3.4 INTERSECTION ACCURACY 74
5.2.3.3.3.4.1 2-D 74
5.2.3.3.3.4.2 3-D 74
5.2.3.3.4 SHADED IMAGE 74
5.2.3.3.4.1 READ IMAGE 74
5.2.3.3.4.2 CALCULATE IMAGE 74
5.2.3.3.4.3 COLOR 75
5.2.3.3.4.4 SET LIGHT SOURCE 77
5.2.3.3.4.5 SET VIEWPOINT 77
5.2.3.3.5 RAY TRACE 78
5.2.3.3.5.1 READ TRACE 78
5.2.3.3.5.2 CALCULATE TRACE 78
5.2.3.3.5.3 SET COLOR 78

5.2.3.3.6 CURVATURE 78
5.2.3.3.6.1 READ CURVATURE 78
5.2.3.3.6.2 CHANGE VIEW 78
5.2.3.3.6.3 ALL CURVATURES 78
5.2.3.3.6.4 GAUSSIAN 79
5.2.3.3.6.5 MEAN 80
5.2.3.3.6.6 ABSOLUTE 80
5.2.3.3.6.7 MAXIMUM PRINCIPAL 80
5.2.3.3.6.8 MINIMUM PRINCIPAL 80
5.2.3.3.6.9 NORMAL U 80
5.2.3.3.6.10 NORMAL V 81
5.2.3.3.7 ISOPHOTES 81
5.2.3.3.7.1 SET NUMBER 81
5.2.3.3.7.2 READ ISOPHOTE 81
5.2.3.3.7.3 CALCULATE ISOPHOTES 81
5.2.3.3.7.4 SHOW ISOPHOTES 81
5.2.3.3.8 REFLECTION LINES 82
5.2.3.3.8.1 SET NUMBER 82
5.2.3.3.8.2 READ IN LINES 82
5.2.3.3.8.3 CALCULATE LINES 82
5.2.3.3.8.4 SHOW LINES 82
5.2.3.3.9 GEODESICS 82
5.2.3.3.9.1 READ IN 82
5.2.3.3.9.2 CALCULATE 82
5.2.3.3.9.3 SHOW 82
5.2.3.3.10 SURFACE ON/OFF 82
5.2.4 GEOMETRY PROCESSING 82
5.2.4.1 CURVES 82
5.2.4.1.1 APPROXIMATE NURBS 82
5.2.4.1.1.1 SET ORDER 82
5.2.4.1.1.2 SET ACCURACIES 82
5.2.4.1.1.3 RUN 82
5.2.4.1.2 FAIRING 82
5.2.4.1.2.1 KNOT 82
5.2.4.1.2.2 AUTOMATED 82
5.2.4.1.2.3 RUN 82
5.2.4.1.3 CONTROL POINT EDIT 82
5.2.4.1.4 CHOOSE EXACT DEGREE 82
5.2.4.1.5 SUBDIVIDE 82
5.2.4.1.6 SPLIT CURVE 82
5.2.4.2 CURVE ON SURFACE 83
5.2.4.2.1 CONVERT COS TO NURBS 83
5.2.4.2.1.1 SET ACCURACIES 83
5.2.4.2.1.1.1 POSITION 83
5.2.4.2.1.1.2 CURVATURE 83
5.2.4.2.1.1.3 SLOPE 83
5.2.4.2.1.2 RUN CONVERT 83
5.2.4.2.2 FAIRING --> calls submenu (see Chapter 5.4) 83
5.2.4.2.3 EDITING 83
5.2.4.2.4 SUBDIVIDE IN UV 83
5.2.4.2.5 SPLIT IN UV 83

5.2.4.3 SURFACE 83
5.2.4.3.1 APPROXIMATE NURBS 83
5.2.4.3.1.1 SET ORDER 83
5.2.4.3.1.2 SET ACCURACIES 83
5.2.4.3.1.2.1 POSITION 83
5.2.4.3.1.2.2 CURVATURE 83
5.2.4.3.1.2.3 SLOPE 83
5.2.4.3.1.3 RUN 83
5.2.4.3.2 FAIRING 83
5.2.4.3.2.1 KNOT 83
5.2.4.3.2.2 AUTOMATED 83
5.2.4.3.2.3 RUN FAIRING 83
5.2.4.3.3 EDITING 84
5.2.4.3.4 DEGREE ELEVATION 84
5.2.4.3.5 SUBDIVIDE 84
5.2.4.3.6 SPLIT 84
5.2.4.4 INTERSECTIONS 84
5.2.4.4.1 LISTS 2-D 84
5.2.4.4.2 LISTS 3-D 84
5.2.5 QUIT 84
5.3 UV_MENU 84
5.3.1 INPUT U-V POINTS 84
5.3.2 OUTPUT U-V POINTS 85
5.3.3 SHOW U-V POINTS 85
5.3.4 ADD U-V POINTS 86
5.3.5 INSERT U-V POINTS 87
5.3.6 DELETE U-V POINTS 88
5.3.7 MOVE U-V POINTS 89
5.3.8 SELECT WINDOW 90
5.3.9 FIT POINTS 91
5.3.10 MAKE SYSTEM CURVE 92
5.3.11 SET STEPS 93
5.3.12 START AGAIN 94
5.3.13 QUIT 94
5.4 COSFAIR_MENU 94
5.4.1 FAIR CHILD - SINGLE 94
5.4.2 FAIR CHILD - AUTO 95
5.4.3 SET SCALE 96
5.4.4 SET STEPS 96
5.4.5 REDRAW CURVES 96
5.4.6 CHANGE VIEW, USE PARENT 96
5.4.7 CHANGE VIEW, USE CHILD 96
5.4.8 KEEP CHILD 97
5.4.9 SHOW WIRE FRAME/CURVE 97
5.4.10 SHOW SURFACE/CURVE 97
5.4.11 SET VIEW OF COS 98
5.4.12 QUIT FAIRING 98
5.5 WORLD_MENU 98
5.5.1 SET BACKGROUND COLOR 98
5.5.2 SELECT CURRENT SURFACE 99
5.5.3 SELECT CURRENT CURVE 100

5.5.4 STATUS ROUTINES 100
5.5.4.1 LIST JOBS 100
5.5.4.2 SUSPEND JOBS 100
5.5.4.3 ACTIVATE JOBS 100
5.5.4.4 KILL JOBS 100
5.5.5 SET PERSPECTIVE 100
5.5.6 TOGGLE BELL 101

CHAPTER 6 - NEW MODULES - EDITOR EXPANSION 103
6.1 Help File Additions 103
6.2 Menu File Additions 105
6.2.1 Menu Entry Formats 106
6.2.2 Menu Addition Examples 107
6.2.3 Menu Replacement Examples 107
6.3 Interface Examples 109
6.3.1 Add Call To Existing Routine 109
6.3.2 Add New Simple Entry 109
6.3.3 Add New Compound Entry 111
6.3.4 Consolidation Of Routines 114
6.3.5 Multiple Related Routines Requiring Identical Setup 115
6.3.6 Add Routine To Run External Job 120

CHAPTER 7 - DEMONSTRATION OF EDITOR 127
7.1 Screen Layout 127
7.2 Starting A Design - Input Data 130
7.3 Surface Operations 130
7.3.1 View Changing 132
7.3.2 All Curvatures 134
7.3.3 Segmentation Selection 134
7.3.4 Shaded Image 137
7.3.5 System Level Routines 140
7.3.6 External Jobs 144
7.3.7 Further Development In Surface Area 144
7.4 Curve Operations - Entering and Editing 146
7.4.1 Input Of Data Points From File 146
7.4.2 Windowing 148
7.4.3 Adding New Points 148
7.4.4 Help Screens 151
7.4.5 Insert New Points 151
7.4.6 Data Point Corrections 154
7.4.6.1 Move A Point 154
7.4.6.2 Delete A Point 154
7.4.7 Fit Of Points and Step Size 157
7.4.8 Making a System Curve and Quitting 157
7.5 Curve Operations - Fairing 159
7.5.1 Actual Fairing 159
7.5.2 Setting the Scale 159
7.5.3 Porcupine View 161
7.5.4 Curve on Wire Frame 161
7.6 Open Parametric Curve 164

CHAPTER 8 - SUMMARY **167**

CHAPTER 9 - REFERENCES **169**

CHAPTER 10 - APPENDICES **171**
10.1 MAIN MENU DATA FILE 171
10.2 UV_MENU.DAT DATA FILE 187
10.3 PROGRAM MAKEFILE 189

TABLE OF FIGURES

Sample Menu Structure 37
Help File Entries 49
Menu Data Structure Interaction 59
Menu Tree Skeleton 60
Complete Editor Menu Tree 66

New Menu Entry Formats 106
Editor Screen Presentations 129
Initial Gaussian Curvature Map 131
View Selection Display : Before Change 131
View Selection Display : After Change 133

Gaussian Curvature Map : New View 133
Display of All Curvature Maps 135
Surface Rendering Segmentation Selection 135
Wire Frame Segmentation Selection 136
All Curvature Maps : Increased Segmentation 136

Gaussian Curvature Map : Specific Area 138
Shaded Image : Default Color 138
Light Source Positioning Presentation 139
Shaded Image : Modified Light Source 139
Shaded Image : Color Changed (CYAN) 141

Shaded Image : Color Changed (YELLOW) 141
Shaded Image : WOOD Coloring (CYAN) 142
Shaded Image : WOOD Coloring (MULTI) 142
Menu Showing SYSTEM SELECTION Header 143
Shaded Image Set Up for Screen Dump 143

External Job : Status Box 145
External Job : Complete Job 145
Curve on Surface Input Prompt 147
Curve on Surface : Check for Valid Input 147
Initial Windowing Presentation 149

Windowing Intermediate Presentation 149
Point Adding : Initial Presentation 150
Point Adding : After Addition 150
Example Help Screen 152
Insert Points : Initial Presentation 152

Insert Points : After Insertions 153
Move : After Point Selection, Before Move 153
Move : After Move 155
Delete : After Selection, Before Deletion 155
Delete : After Deletion 156

Curve Fit Drawn With 25 Steps 156
Curve Fit Drawn With 100 Steps 158
Data Saving Prompt 158
Initial Fairing Screen Presentation 160
Fairing Screen With Scale of 0.1 160

Parent and Child Curves : After Fairing 162
View Change of Porcupine Curves 162
Faired Curve on Surface Wire Frame 163
Curve and Wire Frame : Transformed View 163
Open Curve in Parametric Space 165

Open Curve : Before Fairing 165
Open Curve : After Fairing 166
Open Curve on Surface Wire Frame 166

TABLE OF TABLES

Structure : egeom 31
Structure : fgeom 32
Structure : Mouse_Words 34
Structure : Menu_Entry 36
Structure : Stats 38

Structure : Message 38
Structure : Choice_List 39
Structure : FulCurv 40
Structure : FulSurf 43
Example Menu Data File 61

All Curvatures Screen Arrangement 79
Menu Replacement Examples 108

CHAPTER 1
INTRODUCTION AND OUTLINE

Non-uniform rational B-spline (NURBS) curves and surfaces are used extensively in modern computer aided design and research continues in this area for advanced applications. Many routines have been developed in the Ocean Engineering Design Laboratory at M.I.T. as practical demonstrations of many of the theories about the use of B-splines. However, since this development has been done by many different people to support individual research projects, there has been very little integration of the different programs into a system that allows the different programs to work together.

The object of this thesis is to :

1. Develop the data and menu structures for a general editor designed to use B-spline curves and surfaces.

2. Develop a method by which future additions to the editor can be done easily, helping to ensure the editor is kept up to date and useful.

3. Interface many of the basic visualization routines that will be needed for all future modules.

4. Develop a method for running external jobs separately from the editor.

The philosophy behind all of the various areas of development was to make all of the modules easy to use by even a novice designer. The basic structure of the thesis is as follows:

Chapter 2 will first discuss the basic information necessary to understand B-spline curves and surfaces. Then, all of the data structures used in the editor are listed with explanations of their data fields and use.

Chapters 3 and 4 explain the make up and use of the help file and menu files respectively. The contents of these files, as well as how they are generated, is discussed.

Chapter 5 explains each of the interfaced modules. The chapter also includes a title entry for those modules not yet interfaced. This is done to give the reader a sense of the full menu layout.

Chapter 6 gives detailed direction and examples of how to expand the editor. All possible types of expansion are discussed.

Chapter 7 shows an example of how the editor can be used during a design. Included are photographs of the computer screen as the example develops.

Chapters 8 and 9 are a summary of the work completed in the thesis and the list of references used during the thesis respectively.

Chapter 10 is the appendices which include a copy of the main menu data file, **main_menu.dat**, as Appendix 10.1, a copy of a submenu data file, **uv_menu.dat**, as Appendix 10.2 and a copy of the editor make file, **Makefile**, as Appendix 10.3. Appendix 10.1 and 10.3 are annotated as discussed in various places of Chapter 6.

CHAPTER 2
DATA AND FILE STRUCTURES

2.1 General

For ease and clarity in programming, large blocks of related data should be grouped into a single data structure. This makes it much easier to manipulate the related data in all aspects of the program. A considerable effort must be expended at the beginning of any programming project to ensure that the data structures developed are both adequate for the task at hand and have sufficient growth capability to allow small adjustments for unplanned changes.

The system that is used for development of a computer program will usually have a major influence upon the layout of the program and the data structures used in the program and this was the case with this editor. The computer set up for this development was a Silicon Graphics IRIS 3030 Workstation used for the display of the graphical output and a DEC VAX Station II for computations. In addition to the remote IRIS Graphics Library installed on the DEC VAX station, many of the routines developed use functions of the NAG Mark 11 Fortran library. The editor primarily employs the C programming language with occasional calls to the ULTRIX operating system.

As important as data structures are the file structures used for data storage and the system of naming these storage files. A suggested system for file naming developed for this editor is discussed at the end of this section.

To support this editor it was necessary to develop many new structures. Previously developed structures were utilized whenever possible to aid in compatibility across application boundaries. Because they are essentially a subset of the new, larger structures, the preexisting structures used in the editor are briefly discussed first.

The central idea of this editor is the use of non-uniform rational B-spline curves and surfaces for modeling of free-form (sculptured) shapes. The most basic data structures that hold the information needed to define these curves and surfaces are **egeom** and **fgeom** respectively. These data structures are listed in detail after a brief review of the terms needed to understand the definition of B-splines. The discussion here comes mainly from **[1]**.

2.2 B-Spline Curves

If $T=(t_0,t_1,\ldots,t_k)$ is a set of real numbers such that $t_i \le t_{i+1}$, then a real valued function $f(t)$ defined in the domain $[t_0,t_k]$ is called a **spline** of order M, or degree $M-1$, if $f(t)$ is a polynomial of degree $M-1$ on each sub interval $[t_i,t_{i+1}]$ and its first $M-2$ derivatives are continuous in the entire interval $[t_0,t_k]$. More

important, the higher derivatives of a spline function are continuous everywhere except at $t_i, 0 \le i \le k$. The values $t_0, t_1, \ldots, t_k$ are the knots of the B-spline function and T is the knot vector.

A basis for the vector space spanned by spline functions was found by Curry and Schoenberg **[2]** and an expression of this basis best suited for numerical evaluation has been provided by De Boor **[3]** and Cox **[4]**. The definition of this basis function is recursive and is listed below:

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$N_{i,M}(t) = \frac{t - t_i}{t_{i+M-1} - t_i} N_{i,M-1}(t) + \frac{t_{i+M} - t}{t_{i+M} - t_{i+1}} N_{i+1,M-1}(t) \qquad \text{if } M > 1 \tag{2}$$

The functions defined through (1) and (2) are called B-spline basis functions over T. In evaluating (2) the convention 0/0 = 0 is used whenever such a ratio appears. $N_{i,M}(t)$ is a weighted average of the B-spline functions associated with knots i and $i+1$. Each weight is the ratio of the distance between the parameter and the corresponding end of the support, t_i or t_{i+M}, to the length of $M-1$ spans starting from t_i, t_{i+1}, respectively. Thus, $N_{i,M}(t)$ extends over M spans of the knot vector covered by the interval $[t_i, t_{M+i}]$. $N_{i,M}(t)$ is zero in the remainder of $[t_0, t_k]$.

Non-uniform knot vectors offer certain advantages. Local manipulation of the curve shape is better achieved by a finer knot mesh in areas where accurate shape control is desirable. In applications where interpolation of unevenly spaced points

is required using the B-spline basis, a better parameterization results with a non-uniform knot vector. Curve and surface subdivision (refinement) for enhanced control also necessitates use of a non-uniform knot vector.

These concepts can be extended to vector-valued functions, i.e. spline functions $\mathbf{f}(t) : [t_0, t_k] \rightarrow \mathbf{R}^3$. In this case, $\mathbf{f}(t)$ is a vector, $\mathbf{f}(t)$ = (x(t) y(t) z(t)), where x(t), y(t) and z(t) are scalar spline functions over the same knot vector $T = (t_0, t_1, \ldots, t_k)$. The basis functions are the same as those given by equations (1) and (2). The parametric representation of curves with vector-valued spline functions offers certain advantages with respect to explicit methods such as independence of coordinate systems, easy mathematical formulation of multiple-valued shapes and representation of derivative singularity within the same formulation.

A class of spline functions with the so called open knot vector

$$T = [\tau_0, \tau_0, \ldots, \tau_0, \tau_1, \tau_2, \ldots, \tau_{n-M}, \tau_{n-M+1}, \tau_{n-M+1}, \ldots, \tau_{n-M+1}] = (t_0, t_1, \ldots, t_{n+M-1}) \qquad (3)$$

where τ_0 and τ_{n-M+1} each repeat M times

is of particular interest for design applications. This curve representation can be expressed as

$$\mathbf{R}_M(t) = \sum_{i=0}^{n-1} \mathbf{P}_i N_{i,M}(t) \qquad (4)$$

where $\mathbf{P}_i$ are the n vertices of the associated control polygon (P) described in terms of their (x,y,z) coordinates in a Cartesian system.

One point on a B-spline curve for the parameter value t can be computed recursively through the Cox-De Boor's algorithm. Let $T = (t_0, t_1, \ldots, t_{n+M-1})$ be the knot vector and i an index such that $t_i \le t < t_{i+1}$. Then

$$\mathbf{R}_M(t) = \mathbf{P}_i^{[M-1]}(t) \tag{5}$$

$$\text{where} \quad \mathbf{P}_i^{[k]}(t) = \begin{cases} \mathbf{P}_i & \text{if} \quad k = 0 \\ \lambda\, \mathbf{P}_i^{[k-1]}(t) + (1-\lambda)\, \mathbf{P}_{i-1}^{[k-1]}(t) & \text{if} \quad k > 0 \end{cases} \tag{6}$$

$$\text{and} \quad \lambda = \frac{t - t_i}{t_{i-k+M} - t_i} \tag{7}$$

One vertex of the control polygon of a general B-spline curve according to (4) affects M consecutive intervals $[t_i, t_{i+1}], \ldots, [t_{i+M-1}, t_{i+M}]$ and one interval is affected by M consecutive vertices. Hence, local control of the B-spline curve is possible by shifting only a limited number of vertices.

Rational B-spline curves **[5]** provide a generalization of integral or polynomial B-spline curves. They permit the representation of a wider class of free-form curves as well as classical algebraic curves such as conics, in particular circular arc segments as a special case. A rational B-spline curve of order M over the control polygon P with n vertices and knot vector T is defined as

$$\mathbf{R}_M(t) = \frac{\sum_{i=0}^{n-1} h_i \mathbf{P}_i N_{i,M}(t)}{\sum_{i=0}^{n-1} h_i N_{i,M}(t)} \tag{8}$$

where the h_i are positive real numbers (weights). The integral B-spline curve is a special case of the rational. It is obtained by setting h_i equal to 1 and observing

$$\sum_{i=0}^{n-1} N_{i,M}(t) = 1 \quad , \quad t_0 \le t \le t_{n+M-1}$$

Rational B-splines have all the properties of integral B-splines. In addition, they are closed under bilinear transformations, i.e. transformations of the form $t = (at'+b)/(ct'+d)$ with a,b,c,d all real.

2.3 B-Spline Surfaces

The B-spline patch is the surface analogue of the B-spline curve and is a tensor product surface defined by a topologically rectangular set of control points $\mathbf{P}_{i,j}$, $0 \le i \le m-1$, $0 \le j \le n-1$, which are the vertices of the control polyhedron, P, and two knot vectors, T and S, associated with each parameter, u, v.

Let

$$T = (t_0, t_1, \ldots, t_{m+M-1}) \tag{9}$$

$$S = (s_0, s_1, \ldots, s_{n+N-1}) \tag{10}$$

be the knot vectors, where M and N are the orders in u and v respectively. The corresponding integral B-spline patch is given by

$$\mathbf{S}_{M,N}(u,v)=\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\mathbf{P}_{i,j}N_{i,M}(u)N_{j,N}(v) \tag{11}$$

where $N_{i,M}(u)$ and $N_{j,N}(v)$ are obtained from (1) and (2) by replacing the parameter t with u and v respectively.

Isoparametric or, simply, parametric lines on a B-spline patch are obtained by letting u = constant or v = constant. A parametric line with $u = u_0$ is a B-spline curve in v with S as its knot vector and vertices $\mathbf{Q}_j$, $0 \le j \le n-1$ given by

$$\mathbf{Q}_j=\sum_{i=0}^{m-1}\mathbf{P}_{i,j}N_{i,M}(u_0) \tag{12}$$

Some of the properties of B-spline curves can be easily extended to patches. The support of the basis functions extends over a rectangular area of $M \times N$ adjacent intervals of the parametric space. Surface discontinuities can also be represented by using multiple internal knots in either knot vector.

Rational B-spline suraces are generalizations of integral B-spline patches. Given a control polyhedron P with m, n vertices in each parametric direction and the knot vectors T and S, the corresponding rational B-spline patch of orders M, N in u, v is

$$\mathbf{R}_{M,N}(u,v)=\frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}h_{i,j}\mathbf{P}_{i,j}N_{i,M}(u)N_{j,N}(v)}{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}h_{i,j}N_{i,M}(u)N_{j,N}(v)} \tag{13}$$

where $h_{i,j}$ are positive real numbers. By taking $h_{i,j}=1$ in (13) the rational B-spline patch is reduced to an integral tensor product patch. The properties of integral B-spline patches are easily extended to rational patches **[5]**. Rational B-spline patches can be employed to represent a wider class of free-form surfaces in comparison to integral B-spline patches. In addition, they allow representation of classical algebraic surfaces such as quadrics, torii and surfaces of revolution with planar rational B-spline profiles.

2.4 Existing Structures

The existing structures used in this editor are the core of the interaction between all of the Design Laboratory programs developed for B-spline curves and surfaces. The names of the structures are **egeom** and **fgeom** for edge (curve) geometry and face (surface) geometry respectively. These structures are given in Chapters 2.4.1 and 2.4.2 and are defined in the include file **gen.h**.

2.4.1 egeom

This structure is used to hold the information needed for three-dimensional non-uniform rational B-spline curves. There are many variables in the editor with the type definition ParCurv. This type definition is a short hand notation for the egeom structure.

Table 2.1 - Structure : egeom	
int type	Currently used to code the type of egeom the data has been generated for. For example, 261326 would be a periodic parametric curve.
short order	The order of the B-spline curve.
short ncontpts	The number of control points for the curve.
short kmem	The assigned size of the knots array ≥ (order + ncontpts)
short pmem	The assigned size of the control points array ≥ ncontpts
double2 *knots	Pointer to array of knots ordered from the smallest parametric value to the largest.
vector **contpts	Pointer to array of control points for the curve.

2.4.2 fgeom

This structure is used to hold the basic information needed for a three-dimensional non-uniform rational B-spline surface. The variables in the editor with the type definition ParSurf are this type of structure.

Table 2.2 - Structure : fgeom	
int type	Currently used to code the specific kind of fgeom the data has been generated for. For example, the code for a parametric surface, periodic in u, is 262931.
short uorder	The order of the B-spline surface in the u direction.
short vorder	Same as uorder but for the v direction.
short ucontpts	The number of control points in the u direction.
short vcontpts	Same as ucontpts but for the v direction.
short ukmem	The assigned size of uknots array ≥ (uorder + ucontpts) .
short vkmem	Same as ukmem but for the v direction.
short upmem	The assigned size of contpts array ≥ ucontpts.
short vpmem	Same as upmem but for the v direction.
double2 *uknots	Pointer to the knot vector in the u direction ordered from the smallest parametric value to the largest parametric value.
double2 *vknots	Same as uknots but for the v direction.
vector ***contpts	Pointer to the vector array holding the values for the control points for the surface.

2.5 Developed Structures

The structures developed specifically for the editor include:

1. Mouse_Words - Used for setting the mouse icon wording.

2. Menu_Entry - Used to build the linked menu structure.

3. Stats - Keeps track of external jobs.

4. Message - Used to send messages between external processes and the main program of the editor.

5. Choice_List - Used to implement a popup menu structure.

6. FulCurv - Incorporates the egeom structure discussed in Chapter 2.4.1 into a full system curve.

7. FulSurf - Incorporates the fgeom structure discussed in Chapter 2.4.2 into a full system surface.

Each new structure is discussed separately below.

2.5.1 Mouse_Words

This structure is used anytime the routine for labeling the mouse icon is called. The following items are included in the structure:

Table 2.3 - Structure : Mouse_Words	
int press	1/0 : there is/is not a valid press operation for some mouse button
int release	1/0 : there is/is not a valid release operation for some mouse button
char *text[8]	The left and right mouse buttons have four lines of text associated with their operation, two each for press and release. Each line can be up to eleven characters long. Since the different operations are color coded, it is important to use the correct lines.
char *middle[2]	Serves the same purpose for the middle mouse button as 'text' serves for the other two mouse buttons. Only one entry (vice two) is needed for the middle mouse button operations because these lines can be much longer, 24 characters.

A typical use of this variable is shown in the following code (assumes mousew has been declared of type Mouse_Words). This would tell the user that some sort of action will be taken as a result of the press or subsequent release of the left mouse button.

```
/*  press operation  */
mousew.text[0] = strdup("SELECT");
mousew.text[1] = strdup(" POINT");
press = 1;

/*  release operation  */
mousew.text[2] = strdup("ACCEPT");
mousew.text[3] = strdup(" POINT");
release = 1;

mousewords(&mousew,1);
```

The routine **strdup()** allocates memory for a copy of the argument string. The routine **mousewords()** actually labels the mouse icon and if called with a **1** releases the memory allocated by the **strdup** call. Calling with a **0** will stop the freeing of this memory.

2.5.2 Menu_Entry

This structure is used to build the menu tree structure used throughout the program. Any number of these variables can be made and interfaced in adding future modules. Currently four such menus are used. This structure contains pointers to occurrences of itself so it is essentially a recursive definition. To facilitate this, the original definition is in terms of **menu_entry** (no caps) and the definition used throughout the program is **Menu_Entry**.

Table 2.4 - Structure : Menu_Entry	
char entry_name[25]	The text printed when this item is part of a menu.
char menu_title[25]	The text printed when this item is the head of a menu.
int num_subs	The number of menu selections directly available in the menu that this entry is the head of. Can range from 1, if this entry has no submenu items, to an upper limit of 14, based upon screen layout limitations.
char help[5]	The help file index abbreviation for this selection.
struct menu_entry *from	Points to the menu entry that directly called this entry.
struct menu_entry *first_sub	Points to the first member of any subentries.
struct menu_entry *next	If this is a subentry, points to the next entry if it exists.
struct menu_entry *head	Points to the calling menu item of the submenu that this item is a part of.

Shown next is a sample menu structure. For this structure the various values of the Menu_Entry CIRCLE are listed.

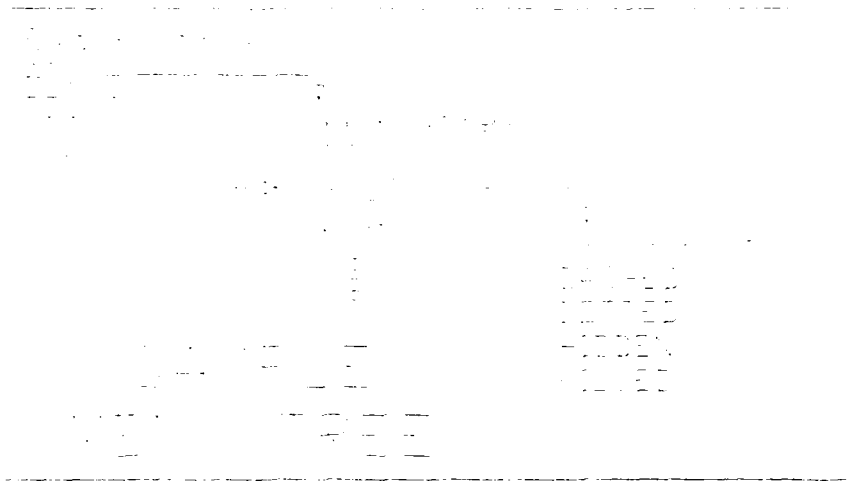


Figure 2.1 - Sample Menu Structure

entry_name : CIRCLE

menu_title : Circle Routines

num_subs : 4

from : points to LINE entry

next : points to SQUARE entry

head : points to Edit Routines entry

The storage requirements of a menu are dynamically allocated when the program is called. This allows the menu arrangement to be changed external to the program. For a more in depth discussion of the menu structure used in the program see Chapter 4.

2.5.3 Stats

This structure is used to keep track of the status of external jobs started by this program. It consists of the following elements:

Table 2.5 - Structure : Stats	
int colors	The color of a specific entry, where colors are used to code progress of jobs.
char *file_name	Pointer to the name of the externally running program.
char *path_name	Pointer to a string containing the full path of the externally running program.

This data structure will need to be expanded as the routines that check and manage external jobs are completed and added to the editor.

2.5.4 Message

This structure is used to read and write system messages. This is the method used for communication between processes in the editor. With these messages it is possible for the main calling program to keep track of the progress of the externally running programs.

Table 2.6 - Structure : Message	
long type	The numerical type identifier of the message.
char text[1024]	The actual message text.

As with the Stats data structure, as the message handling characteristics of the editor are extended this structure will be expanded.

2.5.5 Choice_List

This structure is used to generate popup menus that have the same appearance as the menus generated in the menu tree. They are used in a slightly different way though.

Table 2.7 - Structure : Choice_List	
short type	The number that the entry will be in the menu.
char *text	Pointer to text string holding the entry name.

This structure has only limited use in the editor. See Chapter 4.3 for a further discussion.

2.5.6 FulCurv

This is one of the two major structures of the editor. It contains a pointer to the curve data structure discussed previously (egeom) and various other data fields to make the curve into a full, system curve. Since the structure has pointers to structures of the same type as itself, it is necessary to use an initial name for the structure of **curves**. The name used throughout the program to reference this type of data structure however is **FulCurv**. The data items are listed below.

Table 2.8 - Structure : FulCurv	
ParCurv *egeom	Pointer to curve geometry data structure
double2 **pts	Pointer to two dimensional, dynamically allocated array of u and v points from which the curve was initially generated (if that is the method used for curve generation).
int parent_number	The number in the system curve array where the parent to this curve is stored if there is a parent.
int n	The number of u,v points in the pts array.
double2 umin	Minimum u parameter value of the space over which this curve is defined.
double2 umax	Maximum u parameter value of the space over which this curve is defined.
double2 vmin	Minimum v parameter value of the space over which this curve is defined.
double2 vmax	Maximum v parameter value of the space over which this curve is defined.
Object obj[9]	An array of objects saved during editor operation. However, no objects are currently saved or stored. This is generally due to the fact that objects needed for curve presentation can be more quickly generated than those needed for surface presentation. The array has been included for future expansion.
char *parent_name	Pointer to the name of this curve's parent (if the parent exists).

struc curves *child_of	Pointer to curve structure this is a child of.
struct curves *copy_of	Pointer to curve structure this is a copy of. There is no distinction made at this time between a "child_of" and a "copy_of". Because of this, only the child_of pointer is currently used.
struct curves *offset_of	Pointer to curve that this is an offset of.
char *surface_name	Pointer to the name of the file where the surface that this curve belongs to is stored or if the surface has not been stored, a description of the surface.
char *show_it	Points to a string of 1's and 0's that signify which parts of the structure are to be shown in the various sections of the editor.
char *has_parts	Points to a string of 1's and 0's that signify which parts of the structure have already been generated.
char *parts_saved	Points to a string of 1's and 0's that signify which parts of the structure have been saved to a file.
char *copy_name	Pointer to the file name where the curve this is a copy of is stored.
char *offset_name	Pointer to the file name where the curve this is an offset of is stored.
char *scurve_map	Pointer to the file name where the curvature map data has been stored.
char *sgrowth_1	Pointers for future growth.
char *sgrowth_2	" " " "
char *sgrowth_3	" " " "

char *sgrowth_4	" " " "
char *sgrowth_5	" " " "
char *saved_as	Pointer to the file name where this curve has been saved.
char *desc	Pointer to a description of this curve.
int sys_number	Location in system curve array.
int color	Color to be used with the curve when it is drawn.
char *have_obj	String of 1's and 0's that signify which of the objects in the obj[] array have been generated.
int steps	The number of steps to use when rendering the curve.
int is_open	Signifies the type of curve, 1 is open (non-periodic) and 0 is closed (periodic). This same information in a different format is available in the **egeom** structure.

2.5.7 FulSurf

This is the second major structure of the editor. It contains a pointer to the surface data structure discussed previously (fgeom) and various other data fields to make the surface into a full, system surface. Since the structure has pointers to structures of the same type as itself, it is necessary to use an initial name for the structure of **surfaces**. The name used throughout the program to reference this type of data structure however is **FulSurf**. The data items are listed below.

Table 2.9 - Structure : FulSurf	
ParSurf *fgeom	Pointer to surface data structure.
struct surfaces *copy_of	Pointer to surface this is a copy of.
struct surfaces *offset_of	Pointer to surface this is an offset of.
char *show_it	Points to a string of 1's and 0's that signify which parts of the structure are to be shown in the various sections of the editor.
char *has_parts	String of 1's and 0's that signify which parts above have been generated.
char *parts_saved	String of 1's and 0's that signify which parts have been saved. The following are the positions in the has_parts and parts_saved strings: 0 : surface (fgeom) 1 : surface (auxiliary data) 2 : curvature values 3 : shaded image 4 : ray traced image 5 : wire frame 6 : open 7 : open 8 : open
char *copy_name	Pointer to the file name where the surface this is a copy of is stored.
char *offset_name	Pointer to the file name where the surface this is an offset of is stored.
char *scurvature	These are all pointers to file names where the given data has been stored.
char *sshaded;	same as above
char *sray_trace;	same as above

char *sgrowth_1;	same as above but for future growth
char *sgrowth_2;	for future growth
char *sgrowth_3;	for future growth
char *sgrowth_4;	for future growth
char *sgrowth_5;	for future growth
char *saved_as;	Pointer to file name where this surface has been saved
char *desc	Pointer to description of this surface.
double2 box[6]	Array of minimum and maximum x, y and z values of the surface control points.
double2 light[3]	Array of light vectors used for making shaded image.
float seelight[3]	Source intensity, azimuthal angle and incident angle of light source. Used in setting light source position routine. After the routine, the values are transformed into light[] values.
int nsegu	The number of segments to be used in the u direction when producing shaded and curvature surfaces.
int nsegv	Same as nsegv but for the v direction.
int npointsu	The number of segments to be used in the u direction when making the wire frame model.
int npointsv	Same as npointsu but for the v direction.
int sys_number	The location of this surface in the system surface array.
float ssee[4]	Perspective values to be used when viewing shaded image alone.

float csee[4]	Perspective values to be used when viewing curvature images.
float ctrans[4]	Translation values to be used when viewing curvature images.
float strans[4]	Translation values to be used when viewing shaded image alone.
int port[4]	Viewport values used to view images.
int color	Color to be used when producing shaded image and wire frame.
Object obj[9]	Array for objects generated during the various portions of the editor.
Object key[9]	Color keys for the objects generated.
char *have_obj	String of 1's and 0's that signifies which objects above have been generated. The following positions are used in the obj and key arrays and in the have_obj string: 0 : wire frame 1 : shaded image 2 : Gaussian curvature 3 : mean curvature 4 : absolute curvature 5 : maximum principal curvature 6 : minimum principal curvature 7 : normal U curvature 8 : normal V curvature
double2 ***curv	Curvature values along the u and v directions. Curvatures K1, K2, normal U and normal V are stored for later use.
double2 ***pt	Three dimensional array of calculated points lying on the actual surface.

double2 ***norms	Normals at the points on the surface stored in pt array.
int **intensity	Intensity values at points on the surface. Each intensity value translates into a specific color for the shaded image.

2.6 File Naming Conventions

Because many external programs will be used with the editor and the primary method of passing information between these programs will be through data files, a file naming convention must be adopted so that the history of a file can be somewhat extracted from the name. This will help users go from one session of the editor to the next.

If a curve or surface is started from scratch, the user will be prompted for the file name without any extension, for example **surface_1**. The program will append the extension 'base' to the name entered, making the complete file name **surface_1.base**. This basic name will be used throughout all operations on this surface or curve.

As operations are performed on this curve or surface, the program will append additional extensions onto the file name as needed. For instance, if the surface is shaded, the shaded surface file will be named **surface_1.shade** and if the surface has the curvature mapping done, the name of the map will be **surface_1.map**.

If a completely new surface is generated using an existing surface (for example an offsetting process) the new surface will retain some of the previous surface's name. For example, a surface named **surface_1.base** would generate the name **surface_1_offset1.base** if the offset routine were performed upon it. The **base** extension is used since this new surface can have all of the routines (curvature, shading, etc.) performed upon it that could have been performed upon the original surface. The numerical identifier is used since a surface can have many offsets run for it and they must be kept separate. While it is possible to generate a surface that would have a name such as **surface_1_offset1_offset1.base** (or longer), it is recommended that this not be done. Besides being confusing, the directory for the files would become unusable. If more than one level of offsetting is desired, it is recommended that the first offset file be copied to another surface name before the offsetting is done.

CHAPTER 3
HELP FILE

The actual help files used by the editor are built by running the program **makemsg** on the input help file. Once these files have been built, it is not necessary to run the **makemsg** program again unless the input help file is modified. The structure of the input help file and a discussion of the making and use of the generated files is given below.

3.1 Input Help File

The input help file (for the current editor this file is called **helpfile.m**) contains an entry for every menu item in all of the menus of the editor, including header items. Each entry is of one of the following two forms:

Executable Routine Format	Menu Header Format
Vxxx program_name line count number - n line 1 line 2 line 3 . . . line n !	Vxxx line count number - n line 1 line 2 line 3 . . . line n !

Figure 3.1 - Help File Entries

The form on the right is used for header items. These items do not have program names associated with them. They are only used to give help information to the user about the items in the menu.

The form on the left is used for all help file items that actually call a routine. The function of each part of these forms is as follows:

1. The Vxxx is called the help file identifier. The make up of the identifiers is as follows:

   a. The identifier letter signifies the type of the routine called. **V** is for void, **F** for float, **D** for double and **I** for integer. Currently only the void type is used in the editor. The **N** type (for NONE) is only used for menu headers.

   b. The xxx is the identifier number. These numbers must appear sequentially within a specific group, without missing numbers. This means the sequence (V0,V1,D0,D1,V2,F0,V3) would be correct since the three groups represented are all in order with respect to their own group (V0,V1,V2,V3;D0,D1;F0). However, the sequence (V0,V1,D0,V3,D2,V2) is wrong for two reasons: the V sequence is out of order (V0,V1,V3,V2) and the D sequence skips a number (D0,____ ,D2).

2. The number called line count is the number of help lines in the item. It is a total of all lines between the number line and the exclamation line. Any blank lines included are counted.

3. The help lines are the specific text lines that will be given to the user when this item is referenced for help. There is no current provision made for multiple pages of help. It is suggested that only 35 lines of text be given here since that is the maximum number that will show on the help screen.

4. The exclamation point is used to signify the end of a specific entry. This must appear in the first column of the first line after the last help line in an item.

3.2 Help File Processing Program

The program **makemsg** is used to process the input help file into the files needed for system operation. The call of this program is as follows:

makemsg helpfile.m GOOD hottel

where

helpfile.m is the input help file generated for the editor

GOOD is a four letter code used to name the output files from this program

hottel is an include file generated by the program (it will have the file extension ".h" added to the entered name)

The names used can all be changed as long as the following changes are also made:

1. The name of the input file must match the name listed in the program call exactly, including file extension.

2. The four letter abbreviation **GOOD** is in the include file **defines.h** as a definition for the variable MAIN_HELP_FILE. If the abbreviation is to be changed in the program call, the entry in **defines.h** must also be changed.

3. The include file **hottel.h** is part of the program **mainmenu.c** and is part of the dependencies in the system make file **Makefile**. If the name is changed in the **makemsg** program call, the **mainmenu.c** program and the **Makefile** program must also be changed.

The following error checking is performed on the input help file by the **makemsg** program:

1. The entered line count for all entries is checked against the actual line count. If there is a problem, the program stops and prints the message

Error : # of lines incorrect at Vxxx

where the xxx is the number where the problem was detected. Two things can cause this error. Since the width of the help screen is limited, the length of any given input line must be limited. This limit is 78 characters. This includes any leading and trailing blanks. If a longer input line than this is given, the program will automatically break the too long line into two lines. This will usually cause

the line count to be too high and an error will result. The other way for this error to occur is simply to miscount the input lines.

2. If one of the sequences is improperly ordered, the following message will be printed where the error is detected:

Error : V routines out of order at Vxxx

where again the xxx is the number where the problem was detected.

3.3 Generated Files

When the program **makemsg** is run, various files will be generated. All of these files are discussed in the following section.

3.3.1 Message Files

The message text entered in **helpfile.m** is split into five different files. Each file has a name of the form **GOODmsgs.V** where only the file extension changes for the different types of messages. Only the help text is kept in these files.

3.3.2 Pointer Files

In order to recover the text stored in the message text files, a file of pointer offsets is generated for each of the message text files with a name of the form **GOODptrs.V**. There are two pointers for every menu item - the pointer

offset to the start of the applicable message text and the pointer offset to just past the applicable message text. This method of retrieval is very quick regardless of the size of either the pointer or message files.

3.3.3 Include File

The include file **hottel.h** is also generated by the **makemsg** program. This file has three types of lines : define lines, extern lines and array set ups.

3.3.3.1 Define Lines

All help file identifiers in **helpfile.m**, except type N, cause a line of the following form to be generated in **hottel.h** :

```
#define V21 list_of_lists_interactive
```

The identifier given in **helpfile.m** is paired with its routine to generate this entry. This allows the use of the identifier in all other areas of the file which shortens the file considerably.

3.3.3.2 Extern Lines

There is also one extern line generated for each of the define lines above and these extern lines have the following format :

```
extern void V21();
```

where the void will be the correct type corresponding to the V portion of the identifier. This ensures that the pointer to each of the routines is available to this file.

3.3.3.3 Array Set Ups

Each routine in the editor has a specific pointer associated with its location in memory. By setting these pointers into arrays, it is possible to call the routines by only making reference to one array location.

The type definitions for the arrays are in the **struct.h** file and are listed below:

```
typedef int (*INT_FUNCTION_PTR)();
typedef void (*VOID_FUNCTION_PTR)();
typedef float (*FLOAT_FUNCTION_PTR)();
typedef double (*DOUBLE_FUNCTION_PTR)();
```

The meaning of these definitions is that INT_FUNCTION_PTR types a variable as an array of pointers to functions that return integer values. The other definitions have similar meaning. These type definitions are then used in setting up the pointer arrays in the following manner (example shown is for void type) :

```
VOID_FUNCTION_PTR    v_routine_ptr[] =
  {
    V0 ,
    V1 ,
    V2 ,
    .
    .
    .
    Vxxx,
  };
```

Each of the array entries corresponds to one of the routine entries referenced in **helpfile.m**.

The use of these arrays is as follows. When either a help request or a program selection is made, the identifier letter is used to determine which array of function pointers to use (V for v_routine_ptr[], etc.) or which message and pointer files to use (GOODmsgs.V and GOODptrs.V, etc.) and the identifier number is used to get the correct function pointer or message pointers from the array or files.

CHAPTER 4
MENU FILES

There is one main menu file used with the editor, **main_menu.dat**, and three other smaller menu files, **uv_menu.dat**, **world_menu.dat** and **cosfair_menu.dat**. While these files all have the same general format, each is tailored for a specific use. The general layout and use of all of these menu structures are discussed below. Chapter 4.1 lists the menu data structure used for making menu trees and Chapter 4.2 discusses how the menu data files should be written to make these trees. Chapter 4.3 discusses the interaction between the various types of menus available in the editor.

4.1 Menu Data Structure

The menu files are designed to be read into the following data structure:

```
structure menu_entry
    {
        char entry_name[25];
        char menu_title[25];
        int num_subs;
        char help[5];

        struct menu_entry *from;
        struct menu_entry *first_sub;
        struct menu_entry *next;
        struct menu_entry *head;
    } Menu_Entry;
```

For a description of each of the fields of the structure, refer to Chapter 2.5.2.

4.2 Menu Tree Structure

The menu data structure is used by the program **menu_allocate** to generate the menu tree structure used in the editor. Shown in Figure 4.1 is the interaction of the fields of the data structure. Shown is only a very small portion of a generated tree structure. For an explanation of the data fields, see Chapter 2.5.2.

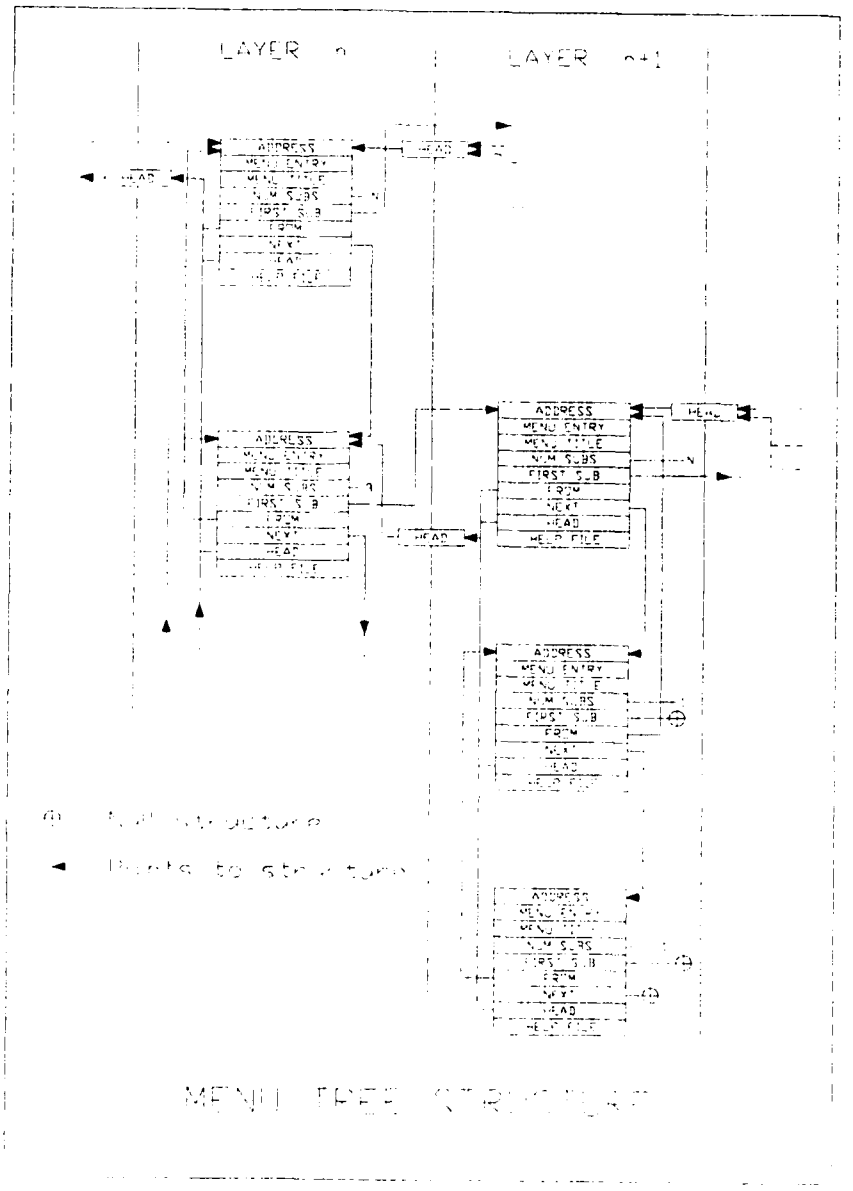


Figure 4.1 - Menu Data Structure Interaction

An example menu tree skeleton is shown below in Figure 4.2. The numbers inside the boxes are the help identifiers. The larger boxes drawn around the small rectangles signify the boundaries of a given menu presentation. The menu items are inside the larger box and will be listed from left to right in the box for a top to bottom menu presentation. The top item of the menu will be the rectangle attached to all members of the menu. The data file to get this tree is shown in Table 4.1.

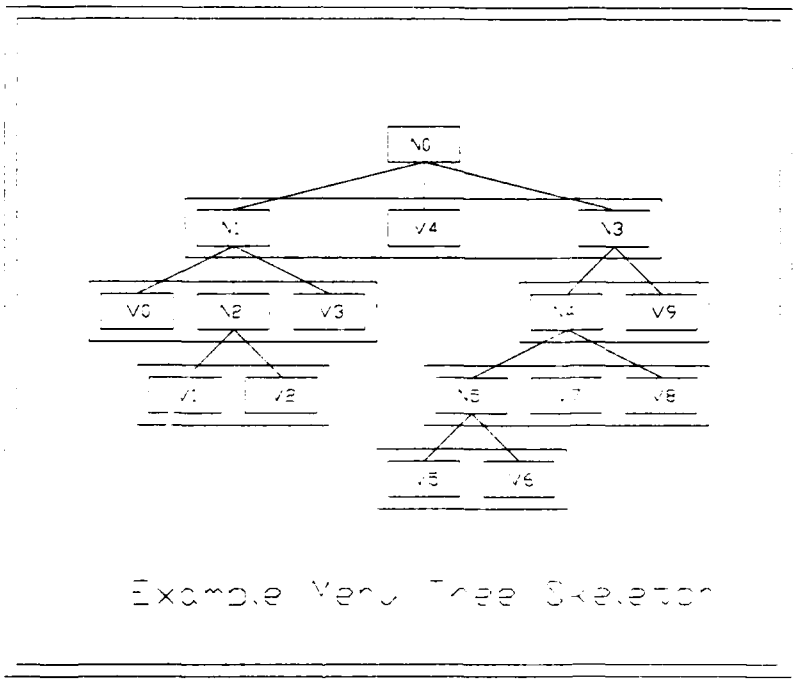


Figure 4.2 - Menu Tree Skeleton

The abbreviations used are MI for menu item and MT for menu title. The {} brackets would be replaced with actual titles and names.

Table 4.1 - Example Menu Data File

```
3
{ MI - N0 }
{ MT - N0 }
N0
    3
    { MI - N1 }
    { MT - N1 }
    N1
        1
        { MI - V0 }
        V0
        2
        { MI - N2 }
        { MT - N2 }
        N2
            1
            { MI - V1 }
            V1
            1
            { MI - V2 }
            V2
        1
        { MI - V3 }
        V3
    1
    { MI - V4 }
    V4
    2
    { MI - N3 }
    { MT - N3 }
    N3
        3
        { MI - N4 }
        { MT - N4 }
        N4
            2
            { MI - N5 }
            { MT - N5 }
            N5
                1
                { MI - V5 }
                V5
                1
                { MI - V6 }
                V6
            1
            { MI - V7 }
            V7
            1
            { MI - V8 }
            V8
        1
        { MI - V9 }
        V9
0
END
```

The indentation scheme used in the data file is for clarity only in developing the file. All leading blanks and tab characters are ignored when the file is read in.

Entries with no actual subroutines have a '1' listed for the number of subroutines. This signifies to the input routine that the entry has no subroutines and therefore will not have a menu title line. As discussed previously, the help identifier is actually the entry location in the routine pointer array.

This method of a linked-tree menu generation was chosen to allow the development of a program structure that has the following characteristics:

1. The position in the menu structure can always be reconstructed by back tracking through the tree. This aids the development of position sensitive help files.
2. The menu structure is more compact and entries are more easily changed as an application is updated.

4.3 Menu Interaction

There are three basic types of menus used in the editor. The main menu is loaded when the program is first started. The majority of the routines are called from this menu. All routines called directly from the main menu must use global variables for information passing because no provision is made for variable set up once the menu looping program **linked_menu** is entered. Because there are situations where a certain set of programs need the same type of set up and the routines needed would work

quicker and more straight forward by passing parameters, a method for separate menu structure generation is included with the editor.

The separate menus are of two types. The first to be discussed is special in that it is available from all other menus directly. This menu has been interfaced through the menu header line of all menus. The menu header line normally displays the name of the menu. However, if the cursor is taken to this entry, the name will change to SYSTEM SELECTIONS (see Figure 7.18) and these system selections can be chosen from all menus. This was necessary because there are some functions in the editor that must be available at all times - for example, choosing the system background color, choosing a specific curve or surface and checking the status of external jobs. If the system menu is selected, the user is kept from recursively calling it again from inside itself.

One of the unique things about this system menu selection is that if the cursor is placed at the header line and the help function is chosen, the help given is for the header entry and NOT THE SYSTEM ROUTINE.

The third type of menu available is part of a normally called routine. The called routine includes the code to generate another menu structure in addition to the main menu. This new menu is transient in that it is released when the routine completes. This type of menu is normally used when the routines

that are to be used in a submenu require some sort of set up that is not appropriate for the main menu program to perform. A discussion of how to set this structure up is given in Chapter 6.3.5.

There is one other menu form available in the editor. The **popup** menu routine included as part of the IRIS documentation **[6]** has been extensively modified to be used as another type of menu. It is currently only used in the color choosing portion of the editor. Its main usefulness is to allow the user to choose a specific option from a list which can be presented in the same way as a normal menu. This menu form should not be used with options that are not entirely straight forward since no help is available. Most appropriate are single item choices, for example choose the first item to get a one, choose the second for a two and so on, where the number gotten is used for a simple and straight forward option selection, such as color selection.

CHAPTER 5
EDITOR PROGRAMS AND LAYOUT

The editor menu structure is built as shown in Figure 5.1 The dashed lines joining some of the menu items signify menus that are generated from auxiliary menu data files. The smaller menu placed alone is the SYSTEM SELECTION menu that is available from all other menus.

This section will discuss all of the routines that have been interfaced into the editor. If some of the routines in a sub section have been interfaced and some have not, those that have not been interfaced are specifically listed as "Not interfaced at this time." If no routine in a sub section has been interfaced, all entries are left blank.

Because it is called from many different places in the editor, the view changing routine is discussed first in Chapter 5.1. All references to this routine direct the user back to this section.

The remaining layout of this section follows the layout of Figure 5.1 with the main menu starting with Chapter 5.2 and the **uv_menu**, **cosfair_menu** and **world_menu** submenus discussed in Chapters 5.3, 5.4 and 5.5 respectively.

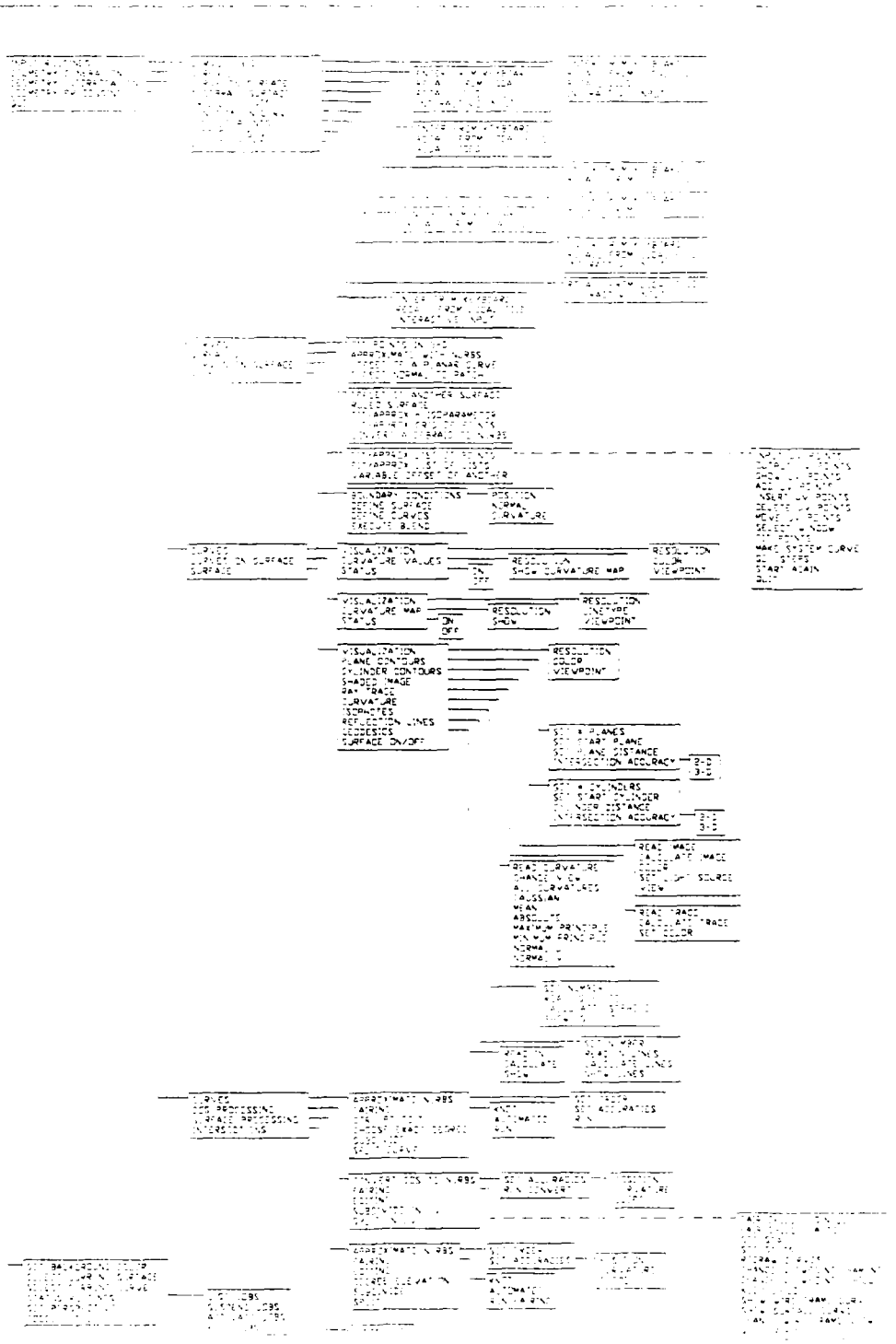


Figure 5.1 - Complete Editor Menu Tree

5.1 CHANGING A VIEW POINT

All routines which set a specific view of an object call the **change_view_aziy** program. This program will be explained in detail here and all routines which call this routine will be listed with the object they use as the input object.

The user is given a display showing the current azimuthal and incident angles along with the distance of the viewpoint from the object. The azimuthal angle is with respect to the negative **y** axis in a counter clockwise rotation in the **XY** plane. The incident angle is measured down from the positive **z** axis. The distance value is in screen units. The user can change either of these angles and the distance from the object.

The distance from the object and translation of the object (in x, y and z) are controlled by selecting plus and minus boxes in the editing window. All of the quantities start changing slowly and change more rapidly the longer the change is selected. Also, each of the quantities can be reset to their original values (those values set when the routine was called) by selecting the zero option square on the screen.

The incident and azimuthal angles are set by moving the mouse while depressing the left mouse button. Care must be taken to ensure that the wire frame is actually in the desired orientation. Once the object has been positioned correctly, using the middle mouse button accepts the entered values.

5.2 MAIN MENU

5.2.1 INPUT ROUTINES

5.2.1.1 CURVE (3-D)

5.2.1.1.1 ENTER FORM KEYBOARD

5.2.1.1.2 RECALL FRO LOCAL FILE

5.2.1.1.3 RECALL IGES FILE

5.2.1.1.4 INTERACTIVE INPUT

5.2.1.2 SURFACE

5.2.1.2.1 ENTER FROM KEYBOARD

Not currently interfaced.

5.2.1.2.2 RECALL FROM LOCAL FILE

This allows surface data from a local file to be read into the system. The user is prompted for a file name to use. The file is checked to ensure it is a valid file name and that the file is readable. If the file name fails either of these tests, the user is prompted for another name. This loop is repeated until either a valid file name is entered or the user selects the abort option.

The entered file name is first used to call the library routine needed to read in a parametric surface, **ReadParSurf**. After the basic surface data has been read, the system is checked to see if an auxiliary file with the same name exists. If such a file does exist

(it would have been created by previously saving data from the editor) the auxiliary data for the surface is read in. If such a file has not been created, the auxiliary surface variables are assigned default values as follows:

1. All flags are set to NULL.

2. The shading light source is positioned at 135 degrees azimuthal (relative to **-x** axis), 45 degrees incidence (relative to **+z** axis) and intensity of 8.00 (relative to an ambient intensity of 1.00).

3. The shading and curvature segmentation is set to 5 in both parametric directions.

4. The wire frame segmentation is set to 10 in both parametric directions.

5. The translation values are set to the center of a box that holds all of the surface control points.

6. The viewpoint is from a distance of 200.0, an azimuthal angle of 160 degrees and an incident angle of 45 degrees (in relation to the same positions as in 2.).

7. The color is set to green.

5.2.1.2.3 RECALL IGES FILE

Not currently interfaced.

5.2.1.2.4 INTERACTIVE INPUT

Not currently interfaced.

5.2.1.3 CURVE ON SURFACE

5.2.1.3.1 ENTER FROM KEYBOARD

5.2.1.3.2 RECALL FROM LOCAL FILE

5.2.1.3.3 RECALL IGES FILE

5.2.1.4 ALGEBRAIC SURFACE

5.2.1.4.1 ENTER FROM KEYBOARD

5.2.1.4.2 RECALL FROM LOCAL FILE

5.2.1.5 GRID OF POINTS

5.2.1.5.1 ENTER FROM KEYBOARD

5.2.1.5.2 RECALL FROM LOCAL FILE

5.2.1.6 FUNCTION ON CURVE

5.2.1.6.1 ENTER FROM KEYBOARD

5.2.1.6.2 RECALL FROM LOCAL FILE

5.2.1.7 LIST OF POINTS

5.2.1.7.1 ENTER FROM KEYBOARD

5.2.1.7.2 RECALL FROM LOCAL FILE

5.2.1.7.3 INTERACTIVE INPUT

5.2.1.8 LIST OF LISTS

5.2.1.8.1 RECALL FROM LOCAL FILE

5.2.1.8.2 INTERACTIVE INPUT

5.2.1.9 LIST OF POINTS (3-D)

5.2.1.9.1 ENTER FROM KEYBOARD

5.2.1.9.2 RECALL FROM LOCAL FILE

5.2.1.9.3 INTERACTIVE INPUT

5.2.2 GEOMETRY GENERATION

5.2.2.1 CURVES

5.2.2.1.1 FIT POINTS IN 3-D

5.2.2.1.2 APPROXIMATE WITH NURBS

5.2.2.1.3 OFFSET OF A PLANAR CURVE

5.2.2.1.4 OFFSET NORMAL TO PATCH

5.2.2.2 SURFACES

5.2.2.2.1 OFFSET OF ANOTHER SURFACE

5.2.2.2.2 RULED SURFACE

5.2.2.2.3 FIT/APPROXIMATE n ISOPARAMETER LINES

5.2.2.2.4 FIT/APPROXIMATE GRID OF POINTS

5.2.2.2.5 CONVERT ALGEBRAIC TO NURBS

5.2.2.3 CURVES ON SURFACE

5.2.2.3.1 FIT/APPROXIMATE LIST OF POINTS -->
calls submenu (see Chapter 5.3)

5.2.2.3.2 FIT/APPROXIMATE LIST OF LISTS

5.2.2.3.3 VARIABLE OFFSET OF ANOTHER CURVE ON SURFACE

5.2.2.4 BLEND

5.2.2.4.1 BOUNDARY CONDITIONS

5.2.2.4.1.1 POSITION

5.2.2.4.1.2 NORMAL

5.2.2.4.1.3 CURVATURE

5.2.2.4.2 DEFINE SURFACE

5.2.2.4.3 DEFINE CURVES

5.2.2.4.4 EXECUTE BLEND

5.2.3 GEOMETRY INTERROGATION

5.2.3.1 CURVES

5.2.3.1.1 VISUALIZATION

5.2.3.1.1.1 RESOLUTION

5.2.3.1.1.2 COLOR

5.2.3.1.1.3 VIEWPOINT

5.2.3.1.2 CURVATURE VALUES

5.2.3.1.2.1 RESOLUTION

5.2.3.1.2.2 SHOW CURVATURE MAP

5.2.3.1.3 STATUS

5.2.3.1.3.1 ON

5.2.3.1.3.2 OF

5.2.3.2 CURVES ON SURFACE

5.2.3.2.1 VISUALIZATION

5.2.3.2.1.1 RESOLUTION

5.2.3.2.1.2 LINETYPE

5.2.3.2.1.3 VIEWPOINT

5.2.3.2.2 CURVATURE MAP

5.2.3.2.2.1 RESOLUTION

5.2.3.2.2.2 SHOW

5.2.3.2.3 STATUS

5.2.3.2.3.1 ON

5.2.3.2.3.2 OFF

5.2.3.3 SURFACE

5.2.3.3.1 VISUALIZATION

5.2.3.3.1.1 RESOLUTION

With this routine the user sets the number of segments to be used when generating the shaded image, curvature plots and wire frame presentations. The shaded image and curvature plots use the same segmentation values. The user can select between 2 and 32 segments and the U and V segmentation can be set separately. The more segments chosen the finer the presentation of a surface. The cost of this improved definition is a decrease in speed of presentation.

5.2.3.3.1.2 COLOR

Not currently interfaced.

5.2.3.3.1.3 VIEWPOINT

This routine calls the basic view setting program discussed in Chapter 5.1. The object used is the wire frame. Changing the view at this level changes the view of both the curvature plots and the shaded surface. The input values for this routine come from the curvature settings.

This routine gives the user a method to easily set the curvature and shaded image views to the same values so that when all curvatures are shown to the screen along with the wire frame and shaded image, all plots will have the same view.

5.2.3.3.2 PLANE CONTOURS

5.2.3.3.2.1 SET # PLANES

5.2.3.3.2.2 SET START PLANE

5.2.3.3.2.3 SET PLANE DISTANCE

5.2.3.3.2.4 INTERSECTION ACCURACY

5.2.3.3.2.4.1 2-D

5.2.3.3.2.4.2 3-D

5.2.3.3.3 CYLINDER CONTOURS

5.2.3.3.3.1 SET # CYLINDERS

5.2.3.3.3.2 SET START CYLINDER

5.2.3.3.3.3 CYLINDER DISTANCE

5.2.3.3.3.4 INTERSECTION ACCURACY

5.2.3.3.3.4.1 2-D

5.2.3.3.3.4.2 3-D

5.2.3.3.4 SHADED IMAGE

5.2.3.3.4.1 READ IMAGE

Not currently interfaced.

5.2.3.3.4.2 CALCULATE IMAGE

This routine forces the calculation and presentation of the shaded image. The speed with which

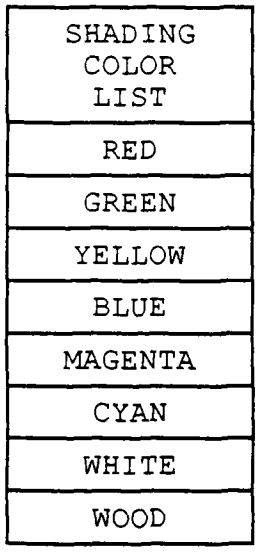
this routine operates depends upon the order of the surface and the number of segments it has be separated into.

Once the image is calculated, it is presented. The object is saved for immediate viewing at a later time. The image does not need to be recalculated as long as the shaded image/curvature segments are not reset and the position and intensity of the light source are not changed.

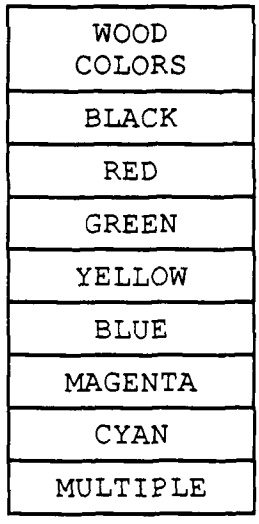
5.2.3.3.4.3 COLOR

This allows the user to set the color to be used for the wire frame and the shaded image. The routine uses the popup menu capabilities of the editor. This means that once the routine is selected, even though the menus look like regular menus, no help screens are available.

The user is presented with the following color options



If any but the last selection is chosen, the routine is finished. Selecting the last entry will give the following display



and the user selects the wood color to use. This option is useful to show a representation of constant intensity bands on the surface rather than a smooth transition of intensity values over a single color

range. When the color is chosen, the color map is called and the colors are changed immediately. This means that if a surface is being displayed using the GREEN color and the CYAN color is selected, the screen display will change to CYAN as soon as the new color is selected.

5.2.3.3.4.4 SET LIGHT SOURCE

This routine calls the standard view setting routine (Chapter 5.1). In this case rather than changing a viewing location, the light source location is changed. Since the light source always points towards the origin, there is no need for translation capabilities in this routine so they are omitted. Also, the distance variable is changed to an intensity variable. The source intensity is in relation to an ambient intensity of 1.00.

Changing the light source placement or intensity will force the shaded image to be recalculated.

5.2.3.3.4.5 SET VIEWPOINT

This routine calls the standard viewing routine (Chapter 5.1) with the wire frame as the input object. Changing the view at this level only changes the shaded image viewpoint. Since the shading of the surface depends only upon the surface normals and light source placement, neither of which is changed

in this routine, the shaded image does not need to be recalculated. However, because of the **z** buffering of the editor (a type of hidden surface removal) the surface presentation will change.

5.2.3.3.5 RAY TRACE

5.2.3.3.5.1 READ TRACE

5.2.3.3.5.2 CALCULATE TRACE

5.2.3.3.5.3 SET COLOR

5.2.3.3.6 CURVATURE

5.2.3.3.6.1 READ CURVATURE

Not currently interfaced.

5.2.3.3.6.2 CHANGE VIEW

This routine calls the standard viewing routine discussed in Chapter 5.1. The surface wire frame is used as the input object. Only the view of the curvature plots is changed at this level of the editor.

5.2.3.3.6.3 ALL CURVATURES

This routine shows the curvature plots discussed in Chapters 5.2.3.3.6.4 through 5.2.3.3.6.10 at one time along with the wire frame surface and the shaded image (Chapter 5.2.3.3.4). The graphics screen is separated into nine sections as diagrammed below in Table 5.1

GAUSSIAN	MEAN	ABSOLUTE
MAXIMUM PRINCIPAL	WIRE FRAME	NORMAL U
MINIMUM PRINCIPAL	SHADED IMAGE	NORMAL V

Table 5.1 - All Curvatures Screen Arrangement

5.2.3.3.6.4 GAUSSIAN

All curvature values are calculated as discussed in **[7]**. If this is the first curvature routine called the curvature array must be calculated first. All of the different curvature plots are combinations of the values calculated for this array. This means the first curvature plot calculated will be much slower than all of the others.

The curvature array contains the following values for each point calculated on the surface

K1 - Maximum Principal Curvature

K2 - Minimum Principal Curvature

Normal U - Curvature in U direction

Normal V - Curvature in V direction

The GAUSSIAN curvature is defined as follows:

$$K = K1 \cdot K2$$

5.2.3.3.6.5 MEAN

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The MEAN curvature is defined as follows:

$$H = \frac{1}{2}(K1 + K2)$$

5.2.3.3.6.6 ABSOLUTE

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The ABSOLUTE curvature is defined as follows:

$$|K1| + |K2|$$

5.2.3.3.6.7 MAXIMUM PRINCIPAL

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The MAXIMUM PRINCIPAL curvature is the K1 value of the array.

5.2.3.3.6.8 MINIMUM PRINCIPAL

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The MINIMUM PRINCIPAL curvature is the K2 value of the array.

5.2.3.3.6.9 NORMAL U

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The NORMAL U curvature is the third entry of the four in the array.

5.2.3.3.6.10 NORMAL V

This routine uses the curvature array calculated as discussed in Chapter 5.2.3.3.6.4. The NORMAL V curvature is the fourth entry of the four in the array.

5.2.3.3.7 ISOPHOTES

5.2.3.3.7.1 SET NUMBER

Not currently interfaced.

5.2.3.3.7.2 READ ISOPHOTE

Not currently interfaced.

5.2.3.3.7.3 CALCULATE ISOPHOTES

This routine spawns an external job that calculates a user selected number of constant intensity lines, or isophotes. When the routine is selected, the user is prompted

Enter the number of isophotes desired :

and the user enters the number of isophote lines to be drawn. When the number is selected and external job is spawned to calculate the isophotes. When the job is complete a message will be sent to the parent system and the user will be prompted that the job is complete.

5.2.3.3.7.4 SHOW ISOPHOTES

Not currently interfaced.

5.2.3.3.8 REFLECTION LINES

5.2.3.3.8.1 SET NUMBER

5.2.3.3.8.2 READ IN LINES

5.2.3.3.8.3 CALCULATE LINES

5.2.3.3.8.4 SHOW LINES

5.2.3.3.9 GEODESICS

5.2.3.3.9.1 READ IN

5.2.3.3.9.2 CALCULATE

5.2.3.3.9.3 SHOW

5.2.3.3.10 SURFACE ON/OFF

5.2.4 GEOMETRY PROCESSING

5.2.4.1 CURVES

5.2.4.1.1 APPROXIMATE NURBS

5.2.4.1.1.1 SET ORDER

5.2.4.1.1.2 SET ACCURACIES

5.2.4.1.1.3 RUN

5.2.4.1.2 FAIRING

5.2.4.1.2.1 KNOT

5.2.4.1.2.2 AUTOMATED

5.2.4.1.2.3 RUN

5.2.4.1.3 CONTROL POINT EDIT

5.2.4.1.4 CHOOSE EXACT DEGREE

5.2.4.1.5 SUBDIVIDE

5.2.4.1.6 SPLIT CURVE

5.2.4.2 CURVE ON SURFACE

5.2.4.2.1 CONVERT COS TO NURBS

5.2.4.2.1.1 SET ACCURACIES

5.2.4.2.1.1.1 POSITION

5.2.4.2.1.1.2 CURVATURE

5.2.4.2.1.1.3 SLOPE

5.2.4.2.1.2 RUN CONVERT

5.2.4.2.2 FAIRING --> calls submenu (see Chapter 5.4)

5.2.4.2.3 EDITING

5.2.4.2.4 SUBDIVIDE IN UV

5.2.4.2.5 SPLIT IN UV

5.2.4.3 SURFACE

5.2.4.3.1 APPROXIMATE NURBS

5.2.4.3.1.1 SET ORDER

5.2.4.3.1.2 SET ACCURACIES

5.2.4.3.1.2.1 POSITION

5.2.4.3.1.2.2 CURVATURE

5.2.4.3.1.2.3 SLOPE

5.2.4.3.1.3 RUN

5.2.4.3.2 FAIRING

5.2.4.3.2.1 KNOT

5.2.4.3.2.2 AUTOMATED

5.2.4.3.2.3 RUN FAIRING

5.2.4.3.3 EDITING

5.2.4.3.4 DEGREE ELEVATION

5.2.4.3.5 SUBDIVIDE

5.2.4.3.6 SPLIT

5.2.4.4 INTERSECTIONS

5.2.4.4.1 LISTS 2-D

5.2.4.4.2 LISTS 3-D

5.2.5 QUIT

5.3 UV_MENU

5.3.1 INPUT U-V POINTS

This allows the user to recall U-V points previously saved. The user is given the following prompt

Enter complete path and name of file to read from (or ! to abort) :

The entered file name is checked for validity. If valid, the data is read in, the points are plotted along with connecting lines to show the progression of the data points, and the user is asked to confirm that the data file shown is the one that is wanted. The user can accept the data and continue or reject the data and enter another file name. This loop continues until the user either enters a file name and accepts it or selects the abort option.

5.3.2 OUTPUT U-V POINTS

This allows entered U-V points to be saved for later use. When the routine is selected, the user is prompted for the output file name. This file name is checked against the system. If the file does not already exist, the user is given the following prompt:

****
Entered file does not exist : create it? (Y or N) :

An 'n' response asks for a file name again, a 'y' response creates the file. This check is performed because the most frequent use of this routine is to store updated points to an existing file not to create new files. The only data fields stored with this routine are the minimum and maximum U-V values of the surface these points were generated for, the number of points being saved and the actual U and V values of the points.

5.3.3 SHOW U-V POINTS

This shows the data points connected by lines in the currer window setting. This allows the user to get a quick look at the point set up before selecting the next editing operation. Also shown is the minimum and maximum U and V values of the current window.

5.3.4 ADD U-V POINTS

This allows U-V points to be added at the end of the existing data base. When chosen, the current window is given with the existing points drawn connected by permanent lines.

Since the user is allowed to set the window used for displaying the data points, it is possible that the current window will have been set such that the last point entered in the data base is not shown. IF THE CURRENT WINDOW DOES NOT CONTAIN THE LAST POINT IN THE DATA FILE, the window layers are stepped through until a window is found that does hold the last point.

The cursor is drawn in the selected window and a temporary line is drawn from the cursor to the last point in the data base. Pressing the left mouse button enters a new point at the cursor position. The cursor position coordinates are printed at the bottom of the screen (U on the left, V on the right) to assist the user in point placement.

When a point is entered, a permanent line is drawn from it to the previously entered point. This gives the user an updated presentation of the order of the data points. When adding is complete the user quits the routine by selecting either the end periodic or end non-periodic option by using the appropriate mouse button.

Although there is essentially no limit to the number of points that can be entered into the system overall (except for the finite size of the computer memory), only 50 points can be added with a single call to this routine. When 45 points have been added, the user is given the following prompt:

ONLY {50-n} ENTRIES LEFT

When 50 points have been entered, the user is given the prompt:

DATA ARRAY IS FULL - USE MOUSE BUTTON TO CHOOSE TYPE OF CURVE TO END.

and is forced to select one of the ending options.

If the INSERT U-V POINTS routine is called and either there are no points in the system or the last point is chosen as the insertion point, the editor transfers to this routine.

5.3.5 INSERT U-V POINTS

This allows points to be entered into the interior of a set of data points. When the routine is first called the user is presented the current window with points connected by permanent lines and is prompted to select the point AFTER WHICH the insertion is to be done. When the insertion point is selected, the line joining the point to the next point is made temporary and the cursor is placed between the two points.

The left mouse button is used to insert points. As the points are inserted the line from the inserted point to the previous point will be made permanent. This is done so that the user is constantly aware of the progression of points in the data base. When inserting is complete the user ends the routine by selecting the middle or right mouse button. As with the adding of points routine (Chapter 5.3.4), only 50 points can be inserted with a single call to the routine.

If it is desired to insert points before the first point in the system the following procedure must be followed. Use the MOVE U-V POINTS routine (Chapter 5.3.7) and select the first point in the system. Move this point to the position of the desired first point. Then choose the INSERT U-V POINTS routine and select the first point as the insertion point. Enter new points as desired, remembering to insert a new point at the old first point location if it is still needed.

If this routine is called and either there are no points in the system or the last point in the system is chosen as the insertion point, the editor transfers to the ADD U-V POINTS routine, Chapter 5.3.4.

5.3.6 DELETE U-V POINTS

This allows the user to remove previously entered points from the data base. When the routine is called the current window is shown with the points connected with permanent lines. The user is prompted to select the point to be deleted.

When the point has been selected (the selection is not made until the mouse button has been released), a box is drawn around the point and the user is prompted to be sure the correct point has been selected. If the user answers that the correct point was selected, it is deleted. When the point is deleted the points on either side of it are joined with a permanent line.

Any number of points can be deleted with one call to this routine. If all points are deleted the user is returned to the main U-V menu upon the deletion of the last point. When the user has deleted all the desired points, the square labeled SELECT TO EXIT must be selected to quit the routine. To sel ct this square, move the cursor point to the interior of the square and press and release the left mouse button.

Depending upon the network set up being used and the load on the network, the user may experience a lag between when the mouse button is depressed and released to select a point and when the point is actually selected. The time lag is of no consequence as long as the mouse is not moved before the point is picked.

5.3.7 MOVE U-V POINTS

This allows the user to move previously entered data points. The user is prompted to select a point to be moved. The point is selected with the left mouse button **AND THE BUTTON MUST BE HELD DOWN WHILE THE MOUSE IS MOVED** to move

the point. Once the point has been moved to the desired point, the left mouse button is released and the new placement is accepted.

The initial screen presentation has the current window with all points plotted and permanent lines between them. When a point has been selected for moving it is connected to its neighbors with temporary lines. As the point is moved these temporary lines are maintained and the point position is printed at the bottom of the screen. This helps the user correctly place the point.

When all points have been moved the SELECT TO EXIT square is used to quit the routine as discussed in the DELETE U-V POINTS write up, Chapter 5.3.6.

5.3.8 SELECT WINDOW

This allows the user to zoom in on a specific area of the parameter space. This is done by selecting a smaller viewing box around the area of interest. When the routine is selected, the user is prompted to place the cursor at one corner of the desired viewing box. Once the cursor has been positioned, the left mouse button is depressed and KEPT DEPRESSED while the mouse is used to drag the cursor to the opposite corner of the desired viewing box. As the mouse is moved, the new viewing box is drawn to give the user feedback on the selected

area. Once the second corner placement has been set correctly, the left mouse button is released and the viewing area has been changed.

There are nine levels available in the window queue. To go back to the previous level, the middle mouse button is used. To go all the way back to the first window from any level, the right mouse button is used.

If when called there are no previous window layers set up (this is the first time the routine has been called for the given data), the user is allowed to abort the routine by using the middle mouse button.

Another way for the user to leave the routine without making a new window is to release the left mouse button with the same U or V value used when the button was pressed (the screen display will appear as a horizontal or vertical 'line' rather than a 'box').

The only other time that the user is allowed to abort the routine with no action is if the window queue is full. If this is the case, the left mouse button can be used to abort rather than to select a new window.

5.3.9 FIT POINTS

This allows the user to have a B-spline curve fit through the entered points. This curve will interpolate all of the entered points. The type of curve used to fit these points depends upon the type of curve selected when the data points

were entered (see Chapter 5.3.4 or 5.3.5), i.e. periodic (closed) or non-periodic (open). The curve is drawn to the screen along with the points after the fit has been performed. The curve is broken into the number of steps set in the SET STEPS routine, Chapter 5.3.11. Depending upon the step size and curvature values, the drawn curve may appear not to pass through all of the entered points.

The editor uses a curve of order four (degree three) to interpolate the actual points. With this fourth order curve there must be at least four data points in the system to use this routine. If fewer than four points exist in the data base, the user is given the following message:

Must have at least 4 data points.
Press <RETURN> to continue.

The library routine used to interpolate the points is capable of handling higher order curves. However, if an order higher than four is used the curve will be a least squares fit of the entered points, not an interpolation. Because it is desired to have the curve actually pass through all of the entered points, the fourth order curve was selected for the editor.

5.3.10 MAKE SYSTEM CURVE

Before this routine can be used the FIT POINTS routine (Chapter 5.3.9) must have been run. If this has not been done, the user is given the following message:

Do not have a current curve to keep.
Press <RETURN> to continue.

Up until this routine is called, the points entered in the system and the curve fit (if done) have used a dummy data structure. Using this routine, the user can bring this dummy data into the system so that other editor routines can operate with it. When the routine is selected, the user is prompted

Enter Curve Description :

and a small description of the file should be entered so that it can be easily distinguished from other files in the system.

5.3.11 SET STEPS

This routine is used to set the number of steps used to draw a curve. Any time the curve is drawn this number of steps is used. Each curve has its own number of steps. The default number of steps is 50 and the number chosen must be at least 25. There is no upper limit to the number of steps chosen. However, the larger the number of steps, the longer many routines that operate on curves will take to run. The following prompt is given

Current step setting is : xxx
Enter steps desired (>= 25) :

5.3.12 START AGAIN

This routine is used to initialize the dummy curve so that new points can be entered into the system. This is not needed if points are to be entered into the system by recalling data points from a file

5.3.13 QUIT

This routine cleans up the dummy data structure. If the data structure has not been saved, the user is prompted:

Current data not saved: Save it? (USE MOUSEBUTTONS)

Answering with the left mouse button (YES - SAVE IT) transfers the user to the OUTPUT U-V POINTS routine, Chapter 5.3.2. Answering with the middle mouse button (NO - DON'T SAVE) completes the clean up and returns the user to the COS Generation Menu.

5.4 COSFAIR_MENU

5.4.1 FAIR CHILD - SINGLE

This routine prompts for the starting and ending knot for the fairing operation. The user is prompted

Enter starting knot (>=xx , <=yy) :

where xx and yy are identification numbers of the lower and upper knots available for fairing and will vary depending on the number of knots and the type of curve (periodic or non-periodic). Once the starting knot has been entered, the user is prompted

Enter ending knot (>=xx , <=yy) :

The starting knot (ks) is compared to the ending knot (ke) to determine the direction of fairing since even on a closed curve the user can not fair across the "ends". This means that if the starting knot for fairing is 10 and the ending knot is 3 (assuming 12 knots), the order of fairing will be {10,9,8,7,6,5,4,3} **NOT** {10,11,0,1,2,3}.

The fairing operation is performed as discussed in **[7]**. When completed, the starting and ending knots are listed along with the knot where the largest curvature discontinuity occurs. The initial, previous and current global curvature discontinuities are also printed to judge the progression of the fairing. When the fairing is complete a new child object is drawn in the bottom window of the fairing screen.

5.4.2 FAIR CHILD - AUTO

This routine performs the same function as the FAIR CHILD - SINGLE routine (Chapter 5.4.1). However, the user is not asked for a starting and ending knot. Instead, the previously selected ending knot becomes the new starting knot and the previously selected starting knot becomes the new ending knot. If this is the first time a fairing operation has been selected, the starting and ending knots default to the minimum and maximum values possible respectively. The same output is given to the user as in the SINGLE option.

5.4.3 SET SCALE

When the curvature plots are placed on the screen the curvature spines may be too large to be kept on the screen. If this happens, the user can set the scale with this routine. The user is shown the current scale and prompted for the new scale to be used. The initial scale is 1.0.

5.4.4 SET STEPS

See Chapter 5.3.11 for a discussion of this routine.

5.4.5 REDRAW CURVES

There are occasions when the parent curve will be erased from the screen. To get this curve and the child redrawn to the screen select this routine.

5.4.6 CHANGE VIEW, USE PARENT

This routine uses the program discussed in Chapter 5.1 with the parent curve as the input object. Although only the parent curve is the input object for this view change, the view of both the parent and the child curves will be changed when the routine is completed.

5.4.7 CHANGE VIEW, USE CHILD

This routine uses the program discussed in Chapter 5.1 with the child curve as the input object. Although only the child curve is the input object for this view change, the view of both the child and the parent curves will be changed when the routine is completed.

5.4.8 KEEP CHILD

This allows the user to save the child as a system curve and then continue with fairing. The following prompt is given

Choices : Replace Parent with Child (R)
Add Child To System (A)
Abort (!)

Enter choice (R , A or !) :

Choosing the **R** will replace the parent data with the child data. Selecting **A** will make a new system curve with the description of **copy of {parent description}**. If it is decided not to save the child, choosing the abort option is the proper action.

5.4.9 SHOW WIRE FRAME/CURVE

This routine takes the child curve and draws it onto the wire frame of the current system surface. There must be a system surface already set up prior to calling this routine. If there is no surface, the user receives the following message

There is no surface in the system!
Press <RETURN> to continue.

5.4.10 SHOW SURFACE/CURVE

Not currently interfaced.

5.4.11 SET VIEW OF COS

This routine uses the program discussed in Chapter 5.1 with the wire frame of the current surface as the input object.

5.4.12 QUIT FAIRING

If the child has not been saved (see Chapter 5.4.8) prior to choosing this routine, the user is prompted

Do you want to save child? (Y or N) :

If the child is to be saved the user is transferred to the KEEP CHILD routine, Chapter 5.4.8. Once saved, the quit routine continues with no other user interfacing.

5.5 WORLD_MENU

5.5.1 SET BACKGROUND COLOR

This allows the user to set the graphics background color to either white or black. Black is better for taking photographs of the screen and white is better for doing screen dumps to a printer. If the user is only interested in on screen graphs, it is a matter of individual preference in deciding which color to use. The mouse buttons are used to select the color, left for black and right for white. The middle button quits the routine. When a specific color is chosen, an axis is drawn to the screen with the chosen background as a demonstration.

5.5.2 SELECT CURRENT SURFACE

This routine allows the user to select the current surface from a list of system surfaces. The surface and curve routines are identical in their user interface so only the surface routine will be described in detail.

If the routine is called with no surfaces in the system, the user is given the message

There are no surfaces in system.
Press <RETURN> to continue.

If there is only one surface in the system, the user is given the message

Only one surface in system -
{ description }

Press <RETURN> to continue.

If more than one surface has been entered into the system, the user is given the following display and prompt

```
   0    {surface 0 file name}    {surface 0 description}
--> 1    {surface 1 file name}    {surface 1 description}
                      .
                      .
                      .
   n    {surface n file name}    {surface n description}

  PRESS LEFT MOUSE ON FILE NUMBER DESIRED (current -->)
```

and the cursor must be placed on the file number desired and the left mouse button used to select the surface. The current

surface is shown with a --> to the left of the file number. If no change is desired, the middle mouse button can be used to quit.

Once a surface has been selected, the user is prompted

YOU HAVE SELECTED xx
IS THIS CORRECT ?

and the mouse buttons are used to answer the question (left mouse button YES, middle mouse button NO). If NO is selected, the user is again prompted to select a surface. If YES is selected, the surface is selected and the routine ends.

5.5.3 SELECT CURRENT CURVE

The operation of this routine is identical to the SELECT CURRENT SURFACE routine except that it works on curves rather than surfaces. See Chapter 5.5.2 for an explanation of the surface routine.

5.5.4 STATUS ROUTINES

5.5.4.1 LIST JOBS

5.5.4.2 SUSPEND JOBS

5.5.4.3 ACTIVATE JOBS

5.5.4.4 KILL JOBS

5.5.5 SET PERSPECTIVE

This allows the user to set the field of view, near plane and far plane values used for perspective views of three dimensional figures. The field of view (fov) is an integer

in tens of degrees (entering 40 means four degrees) and the near and far planes are floating point values in screen units. The values should be entered in the following format (where the symbol Δ denotes a blank space and an **x** is a decimal digit)

xxxΔxxxx.xΔxxxx.x

where the order is fov, near, far.

For good surface presentation, the near and far planes should be as close to the extents of the surface as possible. Good starting values for surfaces for the above three numbers are 40, 0.0 and 2000.0. These are the default values. Sometimes better results can be gotten by moving the near plane to the positive side. If at any time a surface or curve has been sent to the screen but nothing is visible, there is a good chance the perspective values have been set incorrectly. Returning to the default values will usually solve the problem.

5.5.6 TOGGLE BELL

This allows the user to turn the bell prompt used in many of the prompts off or on. No further prompt is given, the bell is just toggled.

CHAPTER 6
NEW MODULES - EDITOR EXPANSION

When the editor is to be expanded, the programmer must decide which of the categories the expansion belongs in. It may be a routine that is already called from another place in the editor and is to be used again for the same or a related purpose. Similar to this would be a routine that does not already exist in the editor but can run directly from the editor with no arguments. The most involved addition is a set of routines that require an extensive and identical initial set up. This type of addition is best handled with its own menu structure. Finally, a routine that is to be run as an external job will require special treatment. Each of these expansions is discussed in the following sections. There are also certain common actions that must be taken for all of the types of additions and these are explained first.

6.1 Help File Additions

The most important step in the whole interfacing process is the help file maintenance. The only time that a change to the help file is not required is when calling an existing routine with no changes. This is because, as discussed in Chapter 3, modules are really called by their help file identifier and if an existing routine is being called, it already has an identifier.

All routines must be interfaced through the help file. One important thing to remember is that regardless of where in the actual editor the new routine is to be called from, additions to the help file are always made at the end of the file. This is because adding a new routine will require a new identifier and the identifiers must be in sequential order in the help file. The following is an example help file entry -

---

```
Vxxx
adding_new_routine
7

New Routine Title
    This routine allows the new function to be
    performed by the editor. When the help file
    has had this addition made it will be able to
    interface the routine.

!
```

- The Vxxx signifies that the routine being added, **adding_new_routine**, returns type VOID. The xxx is the next number available for VOID routines. It is one higher than the last V entry in the help file (**helpfile.m**).

- The routine name must appear exactly as it appears in the program and it does not include any parentheses.

- The number 6 signifies the number of help message lines that are part of this entry. While the blank lines are not required, they help format the screen presentation and are suggested. If included, they must be counted.

- The trailing exclamation point signifies that the entry is complete.

This has shown an entry for only one new routine. There must be a help file entry for each menu entry that will be added to the editor. For example, if a menu entry **EDIT** was to be added to some menu and it was going to call a menu with four subroutines **ADD**, **DELETE**, **MOVE** and **CHANGE**, then five entries would have to be made to the help file, probably an N entry and four V entries. As discussed in Chapter 3, the N entry has no routine name included and is only used for help messages and heading a submenu.

It is suggested that complete help entries be generated initially rather than adding skeleton entries to be filled in later. Formalizing the text that is to be given to a user on a help screen acts as a good flow chart for programming.

IF ANY CHANGES ARE MADE TO THE FILE **helpfile.m**, THE PROGRAM **makemsg** MUST BE RERUN TO HAVE THESE CHANGES APPLIED TO THE APPLICABLE FILES.

6.2 Menu File Additions

Next, the programmer must decide on which menu, and where on that menu, the new routine will be located. This placement

can easily be changed if it turns out that the initial set up is not the best for the user. As mentioned earlier, there are four menu files currently being used. They are the main menu, the two-dimensional parameter space menu, the curve on a surface fairing menu and the world item selection menu. If it is not suitable to add the new routine to any of these menus, a new menu can be created as discussed in the examples below.

6.2.1 Menu Entry Formats

Figure 6.1 shows the two formats available for new menu entries. If a compound entry is added, each of the m entries can be either a simple or compound entry, and so on. The new entry can then be put into a menu structure either as an addition or a replacement.

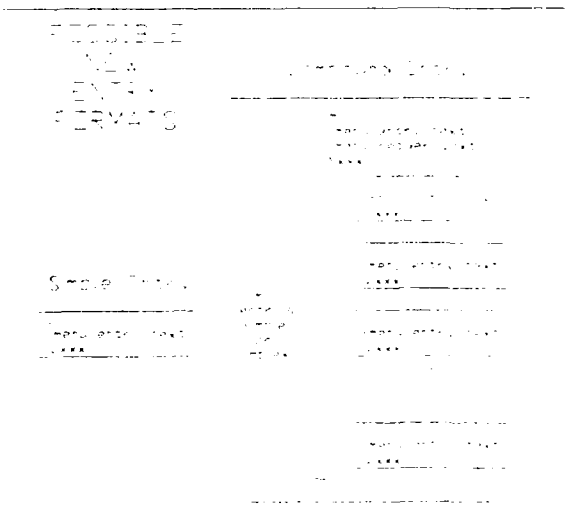


Figure 6.1 - New Menu Entry Format

6.2.2 Menu Addition Examples

Going back to the previous example with the EDIT menu entry and four "sub" entries, they could be added to the menu structure shown in the left column of Table 6.1 below to make the new structure in the center column. Note that the number of entries directly above **POINTS** is now a 5. Except for the simple word processing function of inserting the new menu entries, changing the 4 to a 5 is the only change needed to the menu data file to add these new functions to the menu.

6.2.3 Menu Replacement Examples

Another frequent operation will be to replace an existing entry with another, either a simple or complex entry. If the desire was to replace the GRAPH DATA simple entry above with the new compound entry, the third column would be the result. One of the important points here is that the number of entries in the **POINTS** menu is still 4 so the lead number has not changed. This example also shows how an N entry can be nested under another N entry.

These are the only operations needed to add functions to the menu structure. Removal of functions is the exact opposite and will not be addressed further.

EXISTING MENU	ADD NEW ENTRY TO EXISTING MENU	REPLACE GRAPH DATA WITH NEW ENTRY
. . . 4 POINTS Point Capabili- ties N27 1 INPUT DATA V51 1 OUTPUT DATA V53 1 GRAPH DATA V54 1 QUIT V55 . . .	. . . 5 POINTS Point Capabili- ties N27 1 INPUT DATA V51 1 OUTPUT DATA V53 1 EDIT DATA N62 1 ADD V100 1 DELETE V101 1 MOVE V102 1 CHANGE V103 1 GRAPH DATA V54 1 QUIT V55 . . .	. . . 4 POINTS Point Capabili- ties N27 1 INPUT DATA V51 1 OUTPUT DATA V53 1 EDIT DATA N62 1 ADD V100 1 DELETE V101 1 MOVE V102 1 CHANGE V103 1 QUIT V55 . . .

Table 6.1 - Menu Replacement Examples

6.3 Interface Examples

The following examples will take the existing menu data file, **main_menu.dat**, as the base file. This file is included in Appendix 10.1 for reference. THE CHANGES MADE IN THIS SECTION ARE INDICATED BY BULLETS ON THE APPENDIX AS TO WHERE THEY WOULD GO. The examples below will be cumulative. Any routines added will be added "in function" only - the routine will not actually be written but the function will be listed. Changes mentioned to the file **Makefile** are shown in Appendix 10.3 by bullets also.

6.3.1 Add Call To Existing Routine

It is possible a user could want to set the wire frame and surface resolutions from inside the [GEOMETRY INTERROGATION], [SURFACES], [SHADED IMAGE] menu. To add this function, do the following:

1. Add the following lines at position (A) in Appendix 10.1:

```
1
SET SEGMENTATION
V57
```

2. Change the number pointed to by (B) to a 6.

This completes the adding of a repetitive call of an existing routine.

6.3.2 Add New Simple Entry

The current color of the curve or a surface is always red. If, with future system development, it is possible to

show more than one curve on a surface at one time, it will be necessary to be able to change the color of these curves. To add this capability, do the following:

1. Add the following lines at position (C) in Appendix 10.1:

```
1
SET COS COLOR
V164
```

2. Change the number pointed to by (D) to 4.

3. Add the following to the end of the file **helpfile.m** (where <- b -> indicates a blank line):

```
V164
change_cos_color
5
<- b ->
Change Curve On Surface Color
   This routine allows the user to set the curve
on surface color to any of the primary colors.
<- b ->
!
```

4. Generate the file **change_cos_color.c** that actually changes this color for a specific curve.

5. Add **change_cos_color.o** to Appendix 10.3, the makefile for the editor, as indicated by (a).

NOTE: Anytime directions are given to change the makefile entries it has been assumed that the added routine has been put in a separate file. If

instead the new routine has been added to an existing file, the changing of the makefile will not be necessary.

6. Perform system call **make -k** to generate new executable code.

6.3.3 Add New Compound Entry

This is essentially the same as adding many simple entries except for the addition of an N entry. The editor is designed to have many curves and surfaces in the system at one time but only one curve or surface is "active" at any given time and only the active curve or surface can be shown at one time. It will be necessary to be able to show more than one curve or surface when the editor has the capability to do blending operations. The functions to perform the multiple presentation could be interfaced as follows by adding a new compound entry.

1. Add the following lines to Appendix 10.1 at the point (E) :

```
8
N53
MULTIPLE ENTITIES
Multiple Entities Routines
    1
    PICK SURFACES
    V165
    1
    PICK CURVES
    V166
    1
    REMOVE SURFACES
    V167
    1
    REMOVE CURVES
    V168
    1
    SET VIEW
    V169
    1
    SHOW SURFACES
    V170
    1
    SHOW CURVES
    V171
    1
    SHOW CURVES/SURACES
    V172
```

2. Change the 5 pointed to by (F) in Appendix 10.1 to 6.

3. Add the following lines to the end of **helpfile.m**:

```
N53
m0
 m0 help lines
!
    V165
    pick_surface_to_show
    m1
     m1 help lines
    !
    V166
    pick_curves_to_show
    m2
     m2 help lines
    !
    V167
```

```
remove_surfaces_to_show
m3
 m3 help lines
!
V168
remove_curves_to_show
m4
 m4 help lines
!
V169
set_compound_view
m5
 m5 help lines
!
V170
show_compound_surfaces
m6
 m6 help lines
!
V171
show_compound_curves
m7
 m7 help lines
!
V172
show_compound_curves_surfaces
m8
 m8 help lines
!
```

4. Add the following program calls to Appendix 10.3 at the point indicated by (b).

```
pick_surfaces_to_show.o
pick_curves_to_show.o
remove_surfaces_to_show.o
remove_curves_to_show.o
set_compound_view.o
show_compound_surfaces.o
show_compound_curves.o
show_compound_curves_surfaces.o
```

5. Generate the programs and code necessary to perform the options listed. Each routine must be a different file to go with the makefile entries listed above.

Although this example has been explained in detail, it is only a long version of the previous example. The only new item is the N entry at the top of the added items and at the top of the help file insertion.

6.3.4 Consolidation Of Routines

There is currently a limit of fourteen entries for any menu of the editor. This is due to the screen layout, not to any program limitation. As more functions are needed for a specific area of the editor this limit may be reached or exceeded. One remedy to this problem is to group some of the menu items into a submenu with a new N entry. This keeps all of the functions in essentially the same place in the editor and also frees up some menu entries. As an example, the curvature maps will be made into one submenu under the GEOMETRY INTERROGATION , SURFACES , CURVATURE menu.

1. Add the following lines to Appendix 10.1 where the G is:

   ```
   8
   DRAW CURVATURES
   Curvature Plots
   N54
   ```

2. Change the 10 pointed to by the H in Appendix 10.1 to a 3.

3. Although not required, it is suggested that the lines indicated by I in Appendix 10.1 be further indented. While this is not needed for correct

program operation, it makes the menu file much easier to read and understand if more changes are required in the future.

4. Add the following lines to the end of **helpfile.m**:

```
N54
9
<- b ->
The routines under this entry allow various
    curvature maps to be shown either
    individually or all at one time.  The
    curvature types available are Gaussian,
    Mean, Absolute, Maximum Principal,
    Minimum Principal, Normal U and Normal
    V.
<- b ->
!
```

6.3.5 Multiple Related Routines Requiring Identical Setup

The type of addition to be discussed here was used to interface the fairing of the curves on a surface section and the entering of points in parameter space. An important difference for this type of interface when compared to the regular type of routine is that, when routines are added using this method, THERE MUST BE A SPECIFIC QUIT ROUTINE ASSIGNED AS PART OF THE MENU. Using the right mouse button will not quit from the menu.

The new routine to be added is called **run_new_entry**. The C program code needed for this program to interface

correctly with the editor is shown below. The code needed for the specific application is called required common setup in the listing.

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

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/errno.h>

#include "struct.h"
#include "defines.h"
#include "external_variables.h"

    application include files

#define NEW_HELP_FILE MAIN_HELP_FILE

    application define lines

extern Menu_Entry *linked_menu();
extern void message_handling();
extern Menu_Entry *menu_allocate();
extern void menu_deallocate();

    application extern files
    application extern variables

void run_new_entry()
  {
    Menu_Entry *wherefrom,*pnew_entry,*pptr,*pnptr;
    int moreentries,number_of_subs,is_message,type=0;
    FILE *fptr
    Message msg;

    other variable declaration

    wherefrom = pnull; /*  an external variable  */
    moreentries = 0;

    fptr = fopen("new_menu.dat","r");
    pnew_entry =
         menu_allocate(moreentries,wherefrom,fptr,0);
    fclose(fptr);
```

```
pptr = pnew_entry;

 required common setup

ccontinue = 1; /*  an external variable  */
var_quit_routine = 0; /*  an external variable  */

while ( ccontinue || !var_quit_routine )
  {
    pnptr =linked_menu(pptr,MENULETTERS,MENUBORDER,
                       MENUBACKGROUND,MENUHIGHLIGHT,
                       MENUMASK,port[MENUAREA][0],
                       port[MENUAREA][3],1,
                       NEW_HELP_FILE);

    is_message = msgrcv(status_msgqid,&msg,
                        sizeof(msg.mtext),type,
                        MSG_NOERROR | IPC_NOWAIT);
    if ( is_message > 0 )
       message_handling(msg);
    if ( pnptr == pnew_entry )
      {
       ccontinue = 0;
       pptr = pnptr;
      }
    else if ( pnptr == pptr )
      {
       pptr = pnptr->head;
       if ( pptr != pworld ) in_world = 0;
      }
    else
      {
       number_of_subs = pnptr->num_subs;
       if ( number_of_subs > 1 )
         pptr = pnptr;
       else
         subs(pnptr->menu_title);
      }
  } /*  end of while  */

var_quit_routine = 0;

menu_deallocate(pnew_entry);

free((char *)pnew_entry);
} /*  end routine run_new_entry()  */
```

In order to process messages that may be sent from externally running jobs while the program is inside this new menu setup, the portion of the code including **is_message = msgrcv(...** and **if (is_message > 0)...** is needed **[8]**. The check **if (pptr != pworld)...** is needed to ensure that if the user is not in the world menu structure it will be possible to get into it later. The flag in_world is used for this purpose. If it is a 1, the user is in the world menu structure and cannot call it again from inside itself.

The file used for storing the new menu data structure has been called **new_menu.dat**. This data file must be prepared and will have the same format at that listed in Appendix 10.2 for the uv_menu.dat file. One thing to remember here is that the numbers used in the new menu data file must be referenced in **helpfile.m**. The editor is not currently set up to use more than one help file. However, if future development should allow this option, the defined variable NEW_HELP_FILE is used and set to be equal to the MAIN_HELP_FILE so that it can be set to another file name.

Once this file has been generated, the programmer must decide where in the editor the routines should be interfaced. It will be assumed that the new routines have something to do with geometry generation for purposes of demonstration. The following steps must now be taken to fully integrate the above routine into the editor:

1. Insert the following lines into Appendix 10.1 where the (J) is located:

```
1
NEW ROUTINE ADDED
V173
```

   Note that because these routines will have their own menu structure, they only need one entry in the main menu structure

2. Change the 4 pointed to by (K) in Appendix 10.1 to a 5.

3. Generate the new_menu.dat file with the desired menu structure. The specific layout of this file is up to the programmer but the general format must be the same as that used for the main_menu.dat or uv_menu.dat files shown in Appendices 10.1 and 10.2 for examples. One important thing to remember: THE NEW MENU STRUCTURE MUST HAVE A SPECIFIC QUIT ROUTINE CALL AVAILABLE AND THIS ROUTINE MUST **AS A MINIMUM** SET THE SYSTEM VARIABLES ccontinue TO 0 AND var_quit_routine TO 1. Without this, the program will be unable to exit the new menu structure. It is expected that along with these requirements, the quit routine will also have to perform a general cleanup after the common routines.

4. Generate the routines called by the new_menu.dat menu structure.

5. Add references to all of the added routines to the end of **helpfile.m** as discussed in previous examples.

There are advantages and disadvantages to setting up a new menu structure such as that explained here. The major advantage is that the routines can be developed external to the editor and, when running correctly on their own, interfaced to the editor through a single routine call. As shown, very few changes need to be made to the main portion of the editor to interface a very extensive module if this type of addition is performed.

Another advantage is the complete menu structure does not have to be in memory at any given time. However, this can also be a disadvantage since every time the routine with the separate menu structure is called the menu structure will have to be generated. Although not of major concern, large menu structures can take a noticeable amount of time to be made and any unnecessary slow down in an interactive program should be eliminated if possible.

6.3.6 Add Routine To Run External Job

There is currently only one routine interfaced to the editor that allows for running an external program. This routine is **isophotes** and it calls the program **hiso**. From the beginning it was known that many portions of the final editor would need this capability, for example when interfacing contouring or other general intersection modules. The

decision was made to develop the structure needed to run external jobs even though initial editor development would not need the function except to show its possibilities.

The following C code is necessary to interface a program that is to be used to run an external job **[8][9]**.

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

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/errno.h>

#include "struct.h"
#include "defines.h"
#include "external_variables.h"

   application include files

   application define lines

extern void sta_box();
extern char *strdup();
extern void exit();
extern void sleep();

   application extern files
   application extern variables

void new_external_program()
  {
    int ii,j,job_number,message_type;
    FILE *ifptr;
    char *file_name,*argv[2],inline;

      other variable declaration

    j = (-1);
    while ( j == -1 )
      {
        for ( ii=0 ; ii<20 ; ii++ )
          {
            if ( status[ii].colors == 64 )
```

```
            {
              j = ii;
              ii = 20;
            }
        }
      if ( j == (-1) )
        printf("\n \r Waiting for open external pouch");
    }

  job_number = j;

  prompt for needed info

  ifptr = fopen("indummy.appl","w");

  print needed info to file

  fprintf(ifptr,"%d ",job_number);

  message_type = 10;
  fprintf(ifptr,"%d ",message_type);

  sprintf(inline,"STAT ");
  fprintf(ifptr,"%s",inline);

  sprintf(inline,"outdummy.appl");
  fprintf(ifptr,"%s",inline);

  fclose(ifptr);

  status[j].colors = GREEN;
  sta_box();

  if ( fork() ) /*  starting child process  */
    {
      sleep(1);
    }
  else
    {
      file_name = strdup("   full path of program to run   ");
      argv[0] = strdup("   program name to run only   ");
      argv[1] = (char *)NULL;

      execv(file_name,argv);

    }
} /*  end routine new_external_program()  */
```

The file named **indummy.appl** is a dummy file set up to be read in by the application program for the information needed to run correctly. The file named **outdummy.appl** is the output file from the application that the editor will need to use to bring the information into the editor.

The programmer must also generate the application program to use the data stored above. This application program must include the following lines of code in addition to the code needed to perform the application.

```
/*  application_program  */
main()
{
  int job_number,msgqid,length,i;
  char inline[LEN];
  FILE *ifptr;
  long message_type;
  key_t key;
  Message msg;

  ifptr = fopen("indummy.appl","r");

  read needed info
  fscanf(ifptr,"%d",&job_number);

  fscanf(ifptr,"%d",&message_type);

  fscanf(ifptr,"%s",inline);
  message_id = strdup(inline);

  fscanf(ifptr,"%s",inline);
  output_file = strdup(inline);

  fclose(ifptr);

    .
    .
    .
  application specific routines
    .
```

```
   .
   .
  key = getkey(message_id);

  msgqid = msgget(key,0);

  msg.mtype = message_type;

  sprintf(inline," { message to user } ");
  sprintf(msg.mtext,"%2dCOMPLETE%s",job_number,inline);

  length = strlen(msg.mtext);

  i = msgsnd(msgqid,&msg,length,0);

  if ( i = -1 ) printf("\n \r Error in sending message.");
} /*  end of application  */
```

The relative order of this code is important. The message function should be done last to ensure the application program has been completed prior to sending the message. Note that only the code needed to properly make and send the message has been shown.

The only remaining operation is to add the necessary lines to the menu file and to **helpfile.m**. Since this is a single entry addition that was not previously used, these additions will be the same as described in the Chapter 6.3.2, Add New Simple Entry and will not be discussed further here.

CHAPTER 7
DEMONSTRATION OF EDITOR

In this chapter an example of how the editor can be used as a design tool will be given. The screen displays given to the user as the design progresses will also be shown. A detailed description of any specific routine discussed in this chapter can be found in the applicable sub section of Chapter 5, EDITOR PROGRAMS AND LAYOUT.

7.1 Screen Layout

Figure 7.1 shows the different ways in which the screen is sectioned for the major applications of the editor. The mouse icon is always in the upper left hand corner. This gives the user direction as to the function of the mouse buttons. As the functions of these buttons change, the user is shown the changes. This makes the editor easier to use for novice and experienced users alike. Immediately below the mouse icon is the menu. While the number of lines in the menu is variable, the menu placement on the screen is constant. This allows the user to know exactly where the menu will be throughout the use of the editor. Added to this, the cursor is always placed inside the menu whenever a menu selection is required. These two features make menu item selection easier and quicker for the user.

The area at the lower left of the screen is the external job status box. The editor has the capability of starting external jobs so that the designer is free to use the editor

for interactive design work while slower, computation intensive programs can be run in the background. When the status of these external jobs changes, the status box is used to inform the designer of the changes. As shown, this area is sometimes overwritten for other data presentations. However, whenever the auxiliary data presentation has been completed, the status box is again drawn to the screen.

The text port used throughout the editor is along the bottom right of the screen. Although small, this area is adequate for the editor because all routines in the editor have been optimized for visual presentation and mouse interaction.

The main graphics presentation is in the area above the textport and to the right of the mouse icon and menu. This area is also used for help screen displays but it is refreshed after the help item has been cleared.

The areas discussed above are the main uses of the screen. During some portions of the editor there will be other uses of some of these areas.

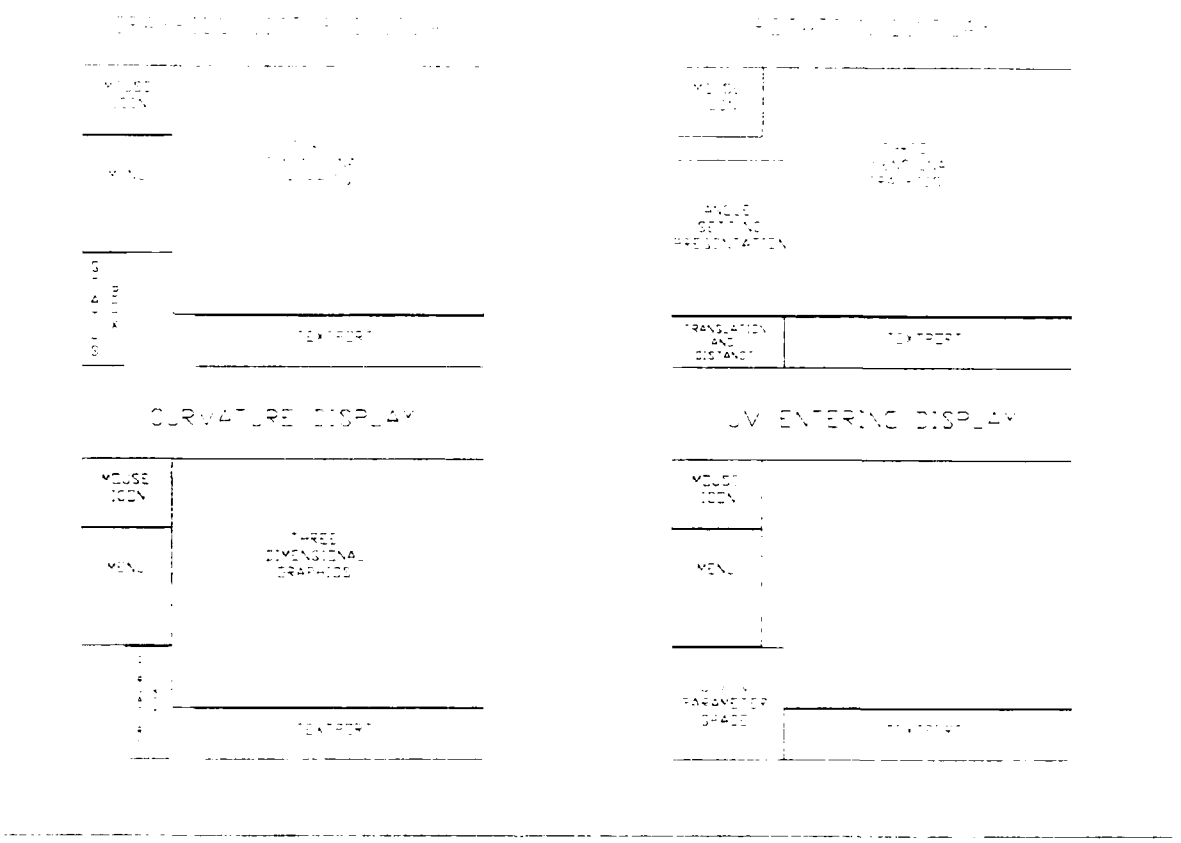


Figure 7.1 - Editor Screen Presentations

7.2 Starting A Design - Input Data

To start the design some sort of data must be entered into the editor. For this demonstration, the input data will be a surface geometry from a local file. This data is the minimum information necessary to define a B-spline surface. The surface for this demonstration resembles a sine wave in both parametric U and V directions. This surface was chosen because it easily shows most of the capabilities of the editor.

7.3 Surface Operations

Once the data has been entered, the curvature of the surface will be investigated by selecting the GEOMETRY INTERROGATION , SURFACE , CURVATURE menu items (the boxes are actual menu item selections made). The first curvature operation to be done is to view the Gaussian curvature map of the entered surface. Figure 7.2 shows the results of the selection. Because only the bare minimum surface information was entered into the system, the view and segmentation of the surface was assigned default values. From the figure it appears that these default values are inadequate so they must be changed.

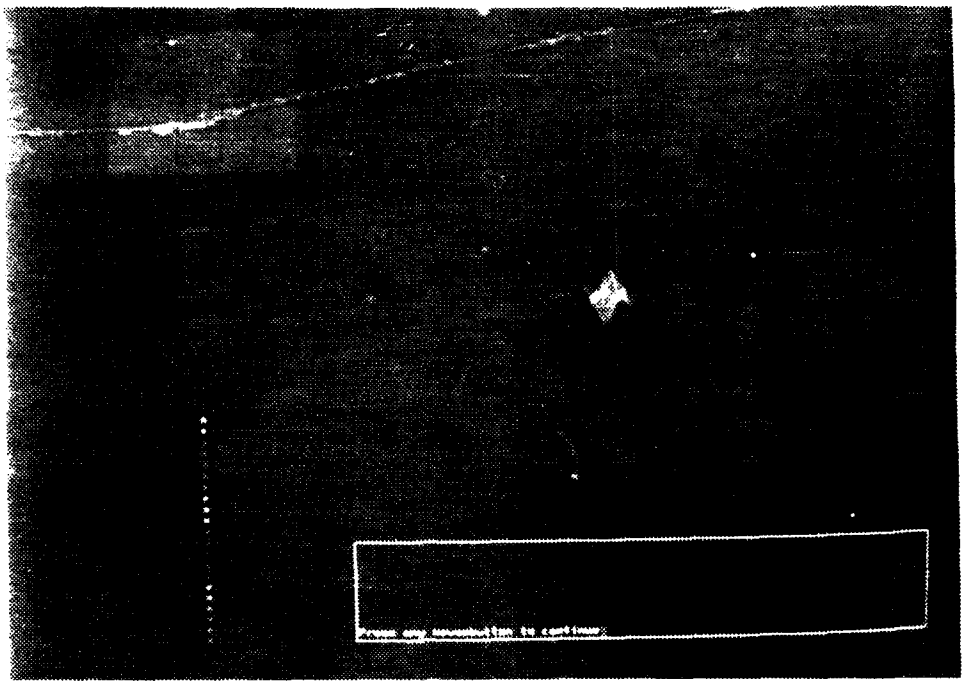


Figure 7.2 - Initial Detection Invasion Map

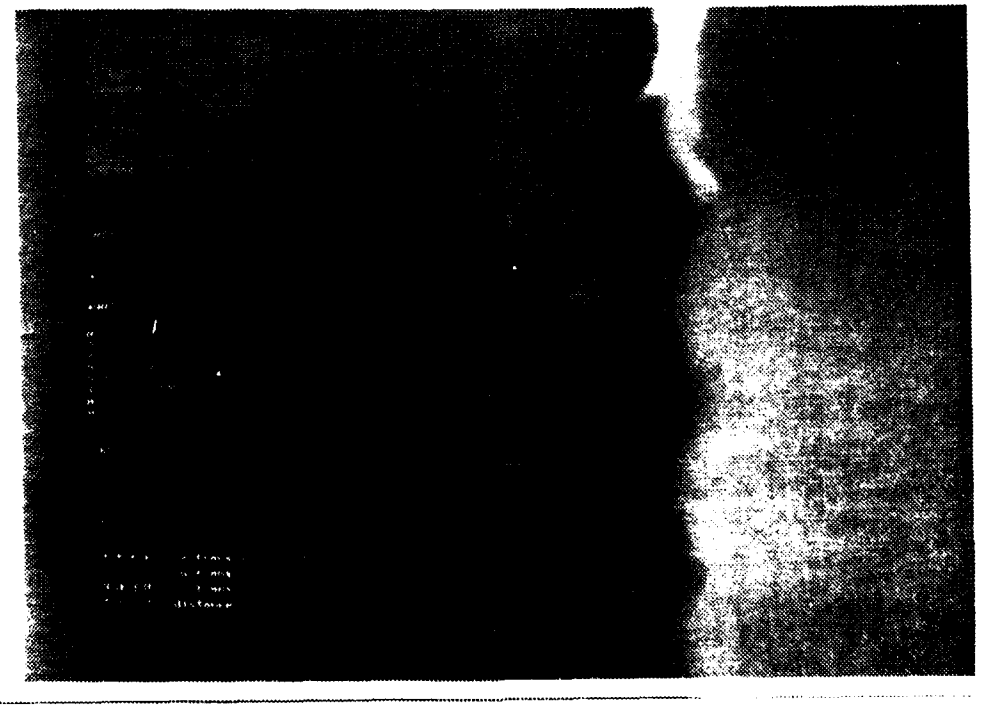


Figure 7.3 - View Detection Invasion

7.3.1 View Changing

First the view of the surface will be changed. Figure 7.3 shows the view selection screen with a wire frame of the surface in the initial orientation. It is sometimes difficult to tell from looking at a wire frame which portion of the figure is closest to the point of view. By using the rotation features of the view setting routine, it is easier to visualize the actual surface orientation.

Once the orientation of and distance from the surface has been set to the values as shown in Figure 7.4, the view changing routine can be quit and the curvature routine for Gaussian curvature recalled. Since the curvature values were previously calculated and these values are independent of surface orientation, the new curvature presentation occurs immediately. Figure 7.5 shows the Gaussian with the new orientation.

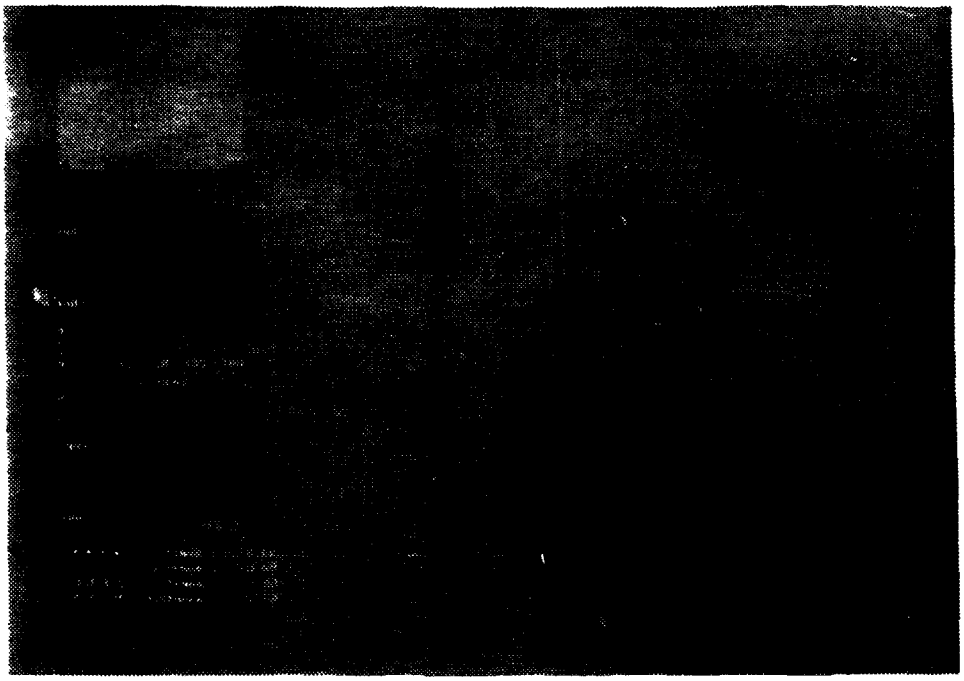


Figure 7.4 - View Selection Display : After Change

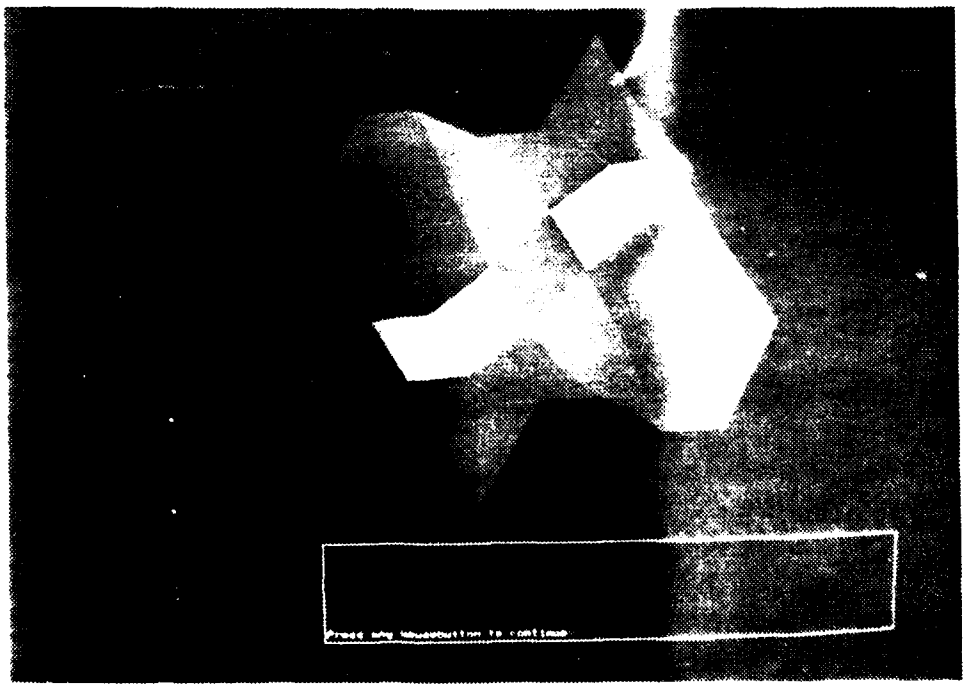


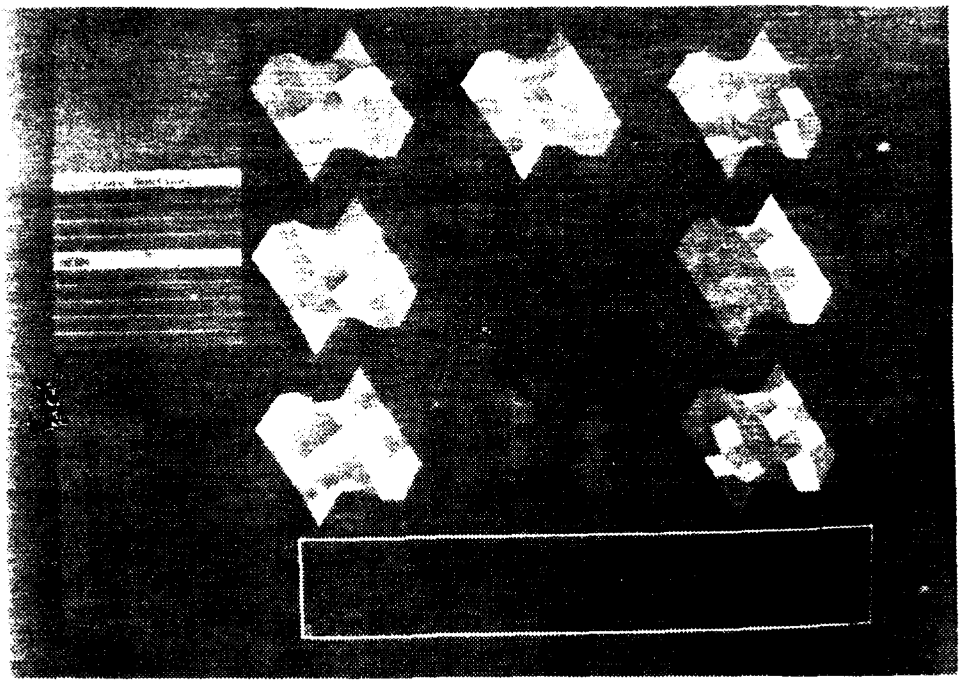
Figure 7.5 - Gaussian Curvature Map

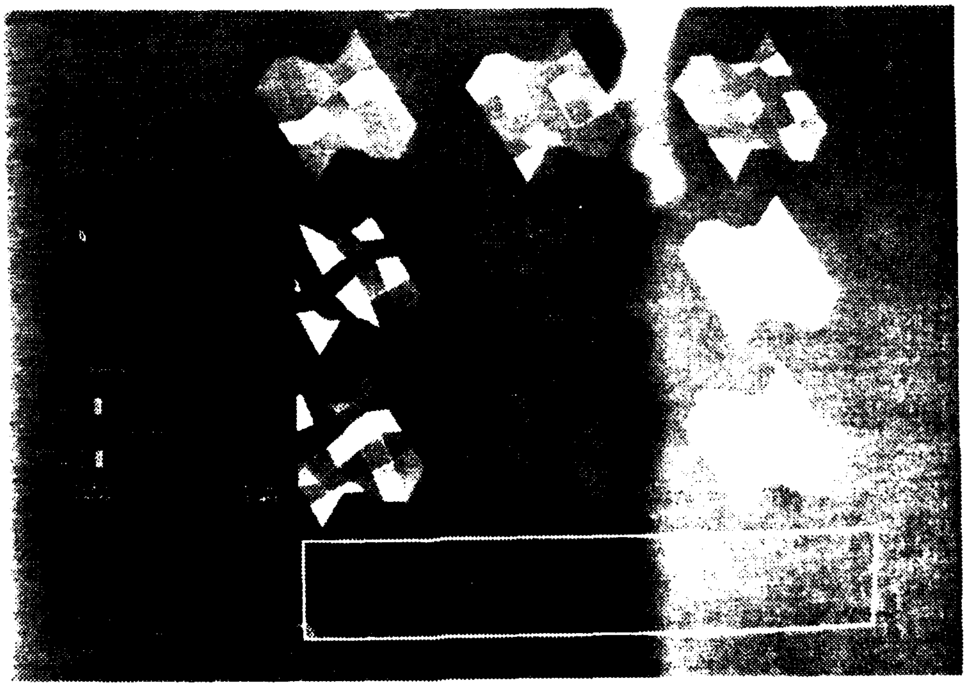
7.3.2 All Curvatures

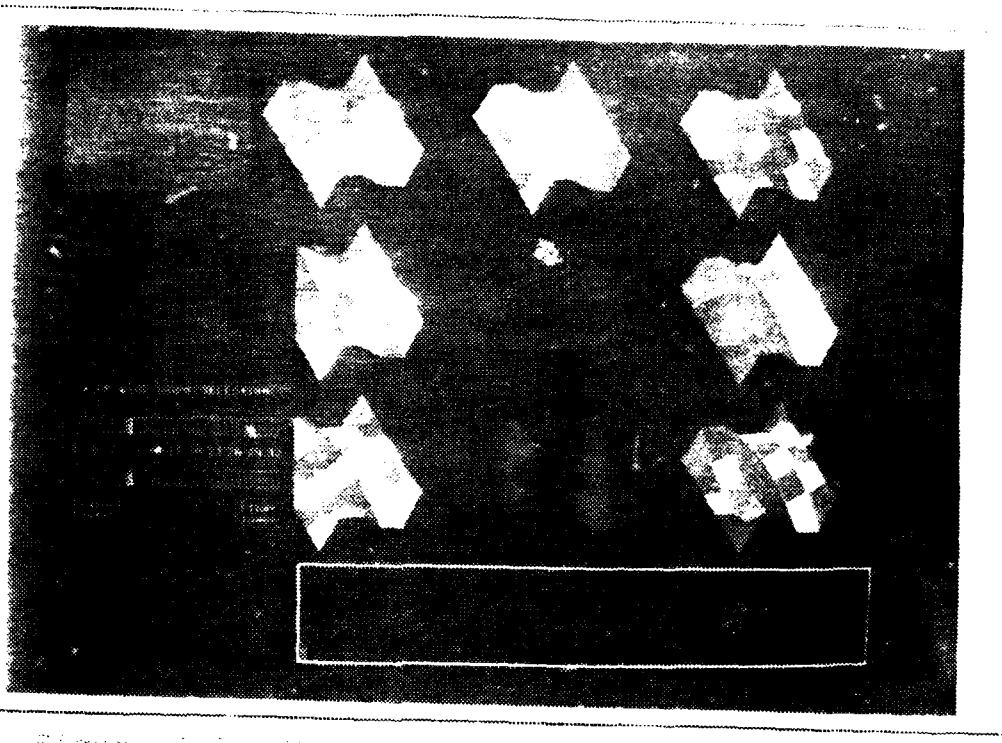
Next all curvatures will be presented at one time on the screen as shown in Figure 7.6. This feature allows the designer to easily see all of the useful curvature presentations at one time. This will make it easier to compare the different curvatures. Although it is possible to see the general differences in the areas of interest of the various curvatures with the default segmentation, the presentation does not give sufficient detail for use in moving to a specific area for further investigation.

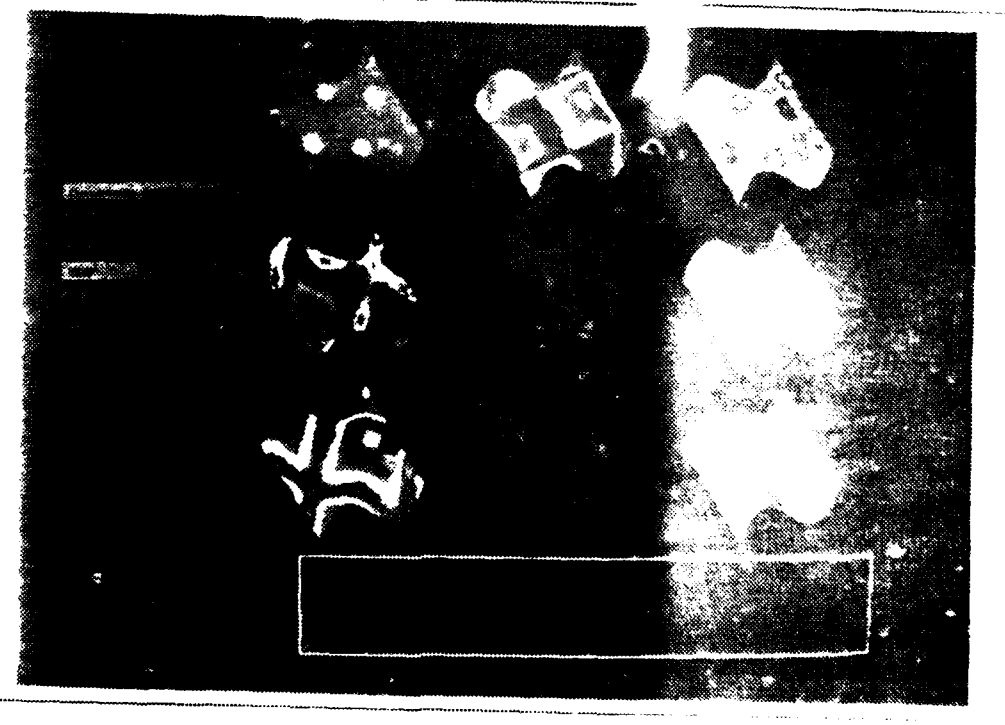
7.3.3 Segmentation Selection

Selecting the segmentation routine, the number of segments to be used in surface rendering can be selected as shown in Figure 7.7 or the number of segments for wire frame drawing can be selected as shown in Figure 7.8. The maximum possible value will be chosen in each case for the U and V directions. Once these selections are made, the routine can be exited. As the number of segments is increased, the time to calculate the various values needed for surface rendering increases. A large portion of these calculations is performed before the segmentation routine is completed so there will be a noticeable time lag between when the segmentation routine is quit and when the menu is placed on the screen.







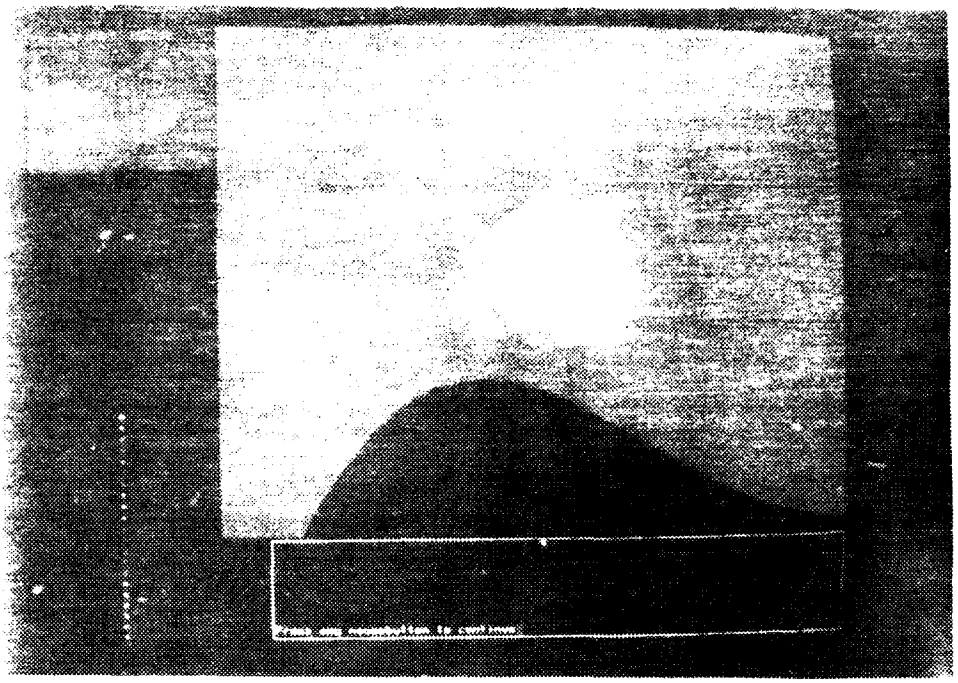


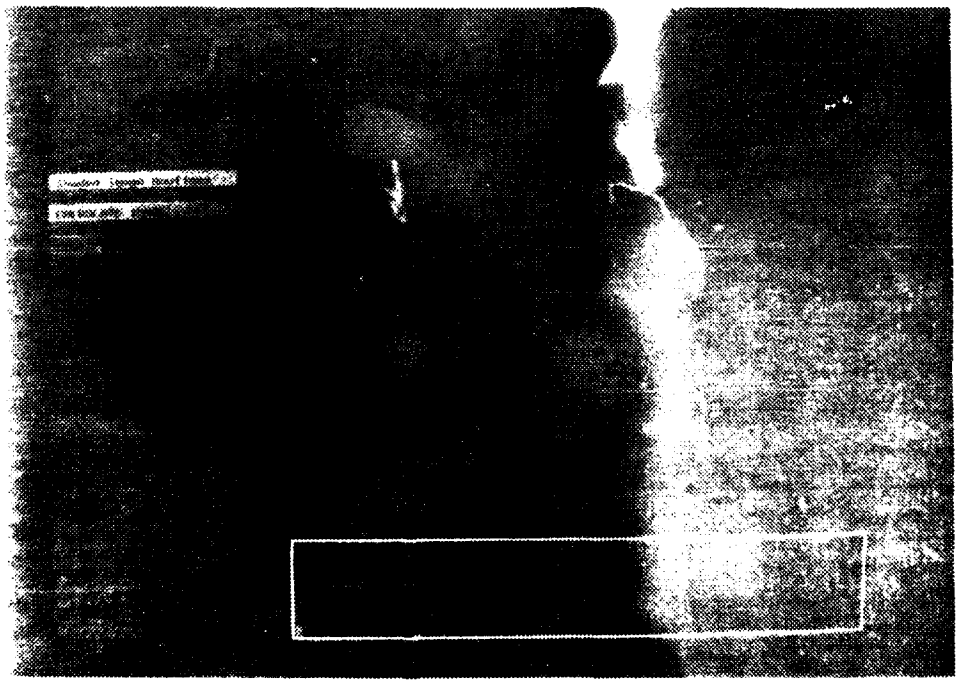
Once these calculations are complete, all curvatures can again be shown as Figure 7.9. A more specific area of interest can now be seen on the Gaussian curvature plot and the area can be further zoomed in on for investigation as is shown in Figure 7.10.

7.3.4 Shaded Image

The bottom of Figure 7.9 shows the shaded image that is available. Going to the shaded image portion of the menu, this object can be displayed on the full screen, as shown in Figure 7.11, for a better presentation.

The highlights of a surface are best realized when the light source is placed to give good definition to these highlights. The user has the capability to position the light source and change its intensity. The screen given to the user during this positioning is shown in Figure 7.12. The ambient intensity is always set at 1.0 and the light source intensity can be varied as desired. Changing the light source changes the presentation given in Figure 7.11 to that of Figure 7.13.





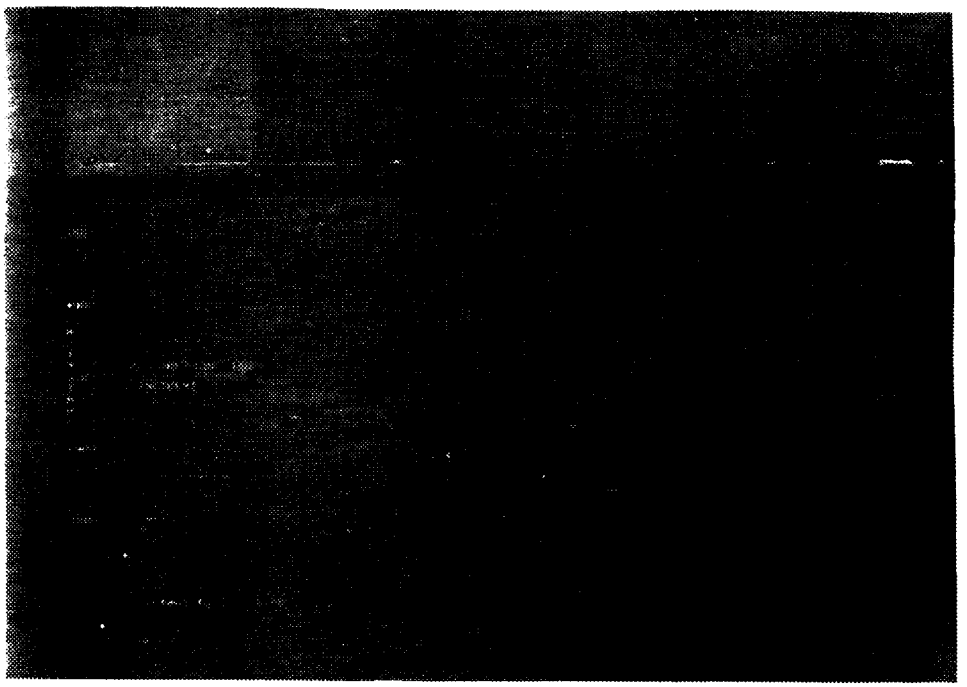


Figure 7.12 - Light Source Scaling Presentation

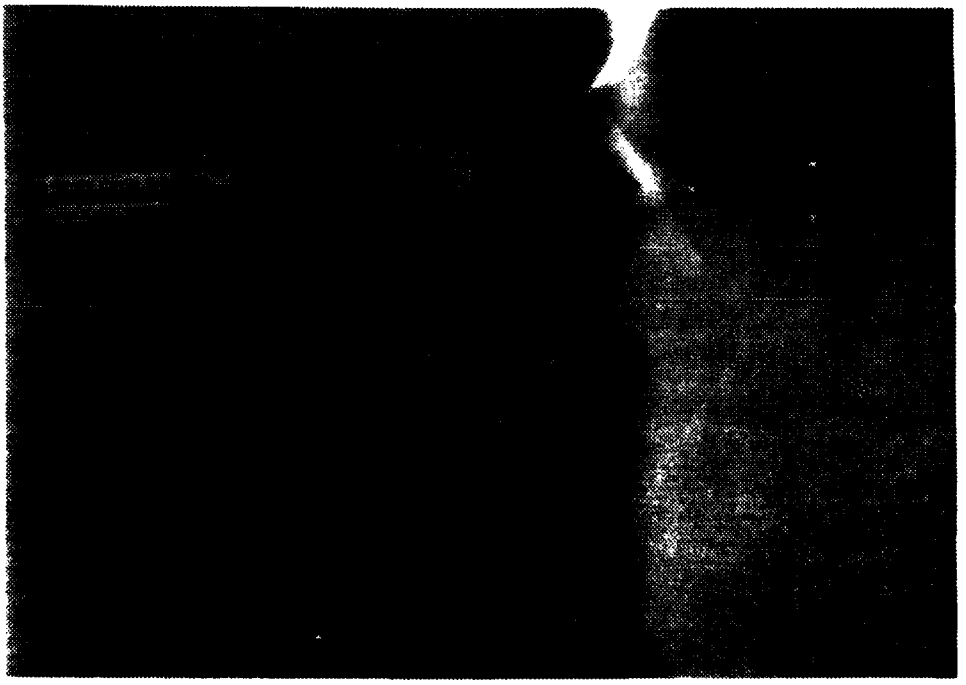


Figure 7.13 - Skewed Image : Mo

Because some colors show detail better for different people, the editor gives the user the capability to change the colors used when drawing the wire frame and when shading the surface. Figure 7.14 shows the green color changed to cyan and in Figure 7.15 the color has been set to yellow. Another option for setting the color is to use the "wood" color choice. This gives a very rapid presentation of constant intensity bands on the surface. This type of coloring is shown in Figure 7.16 using the cyan shaded wood pattern. Figure 7.17 shows the multicolored wood pattern.

7.3.5 System Level Routines

There are various system level routines available. These are accessed by moving to the menu header within any menu. Figure 7.18 shows the menu header will change to SYSTEM SELECTIONS and become highlighted. This system menu can be used to change the background color used during surface presentation. This could be needed if a hard copy of the screen presentation was needed and the method to be used required a white background. Also, some users may find the white background preferable to the black. Selecting the white background and returning to the shaded image menu to change the surface color to white will give the presentation shown in Figure 7.19.

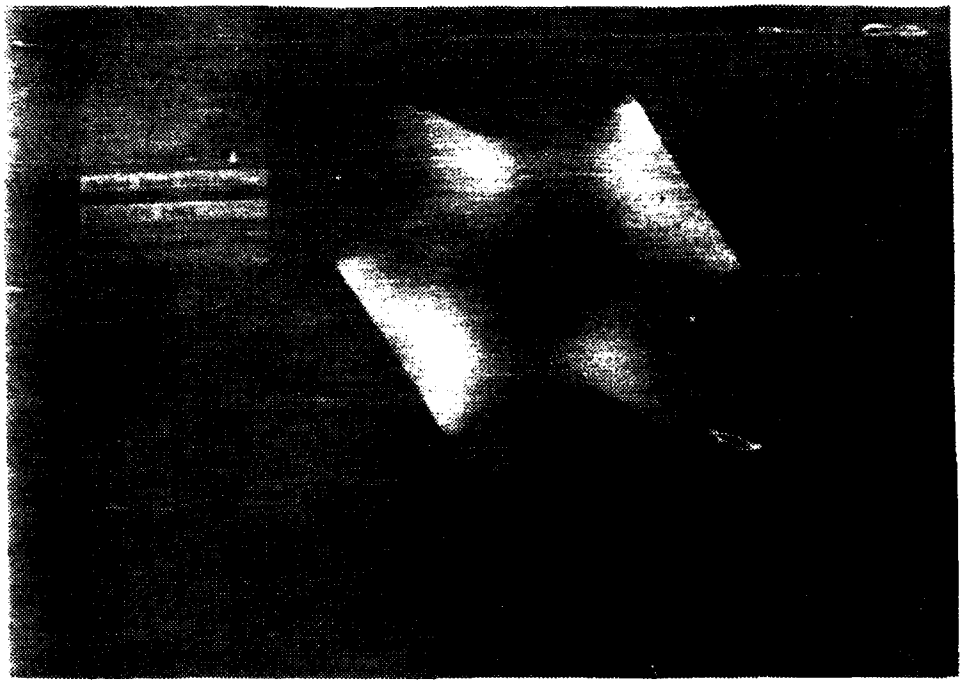


Figure 7.14 - Shaded Image : Color Changed (CYAN)

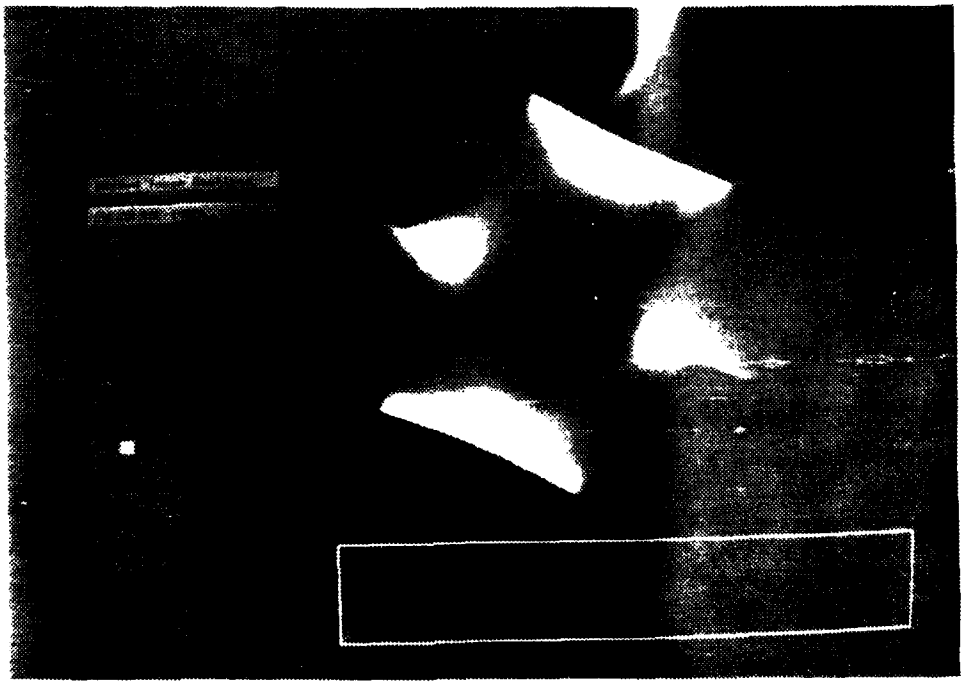


Figure 7.15 - Shaded Image : Color Changed (BLUE)

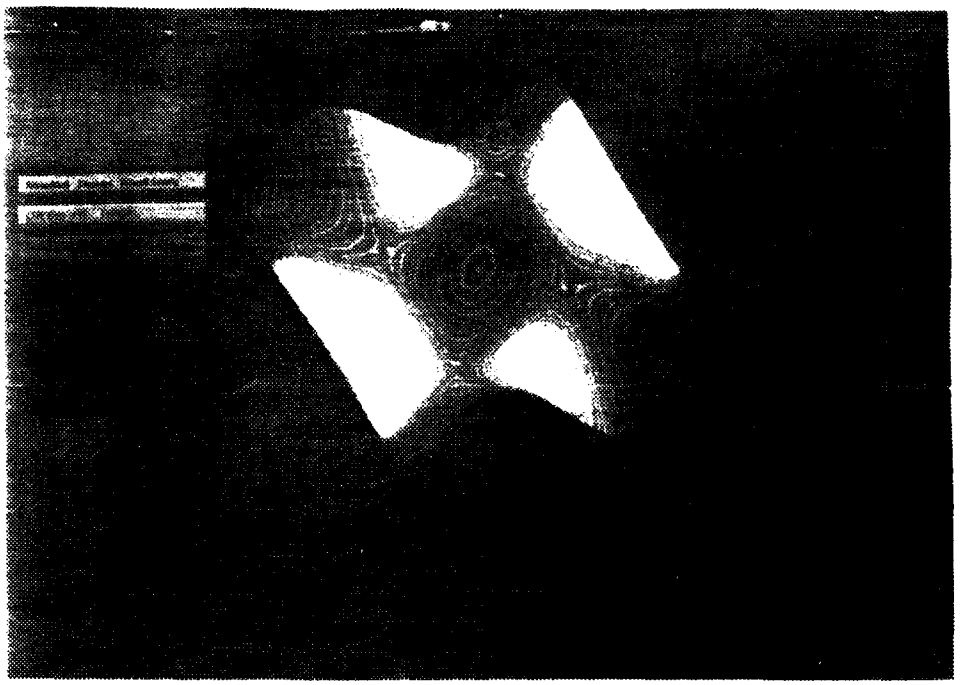


Figure 7.16 - Shaded Image : [illegible] coloring (CYAN)

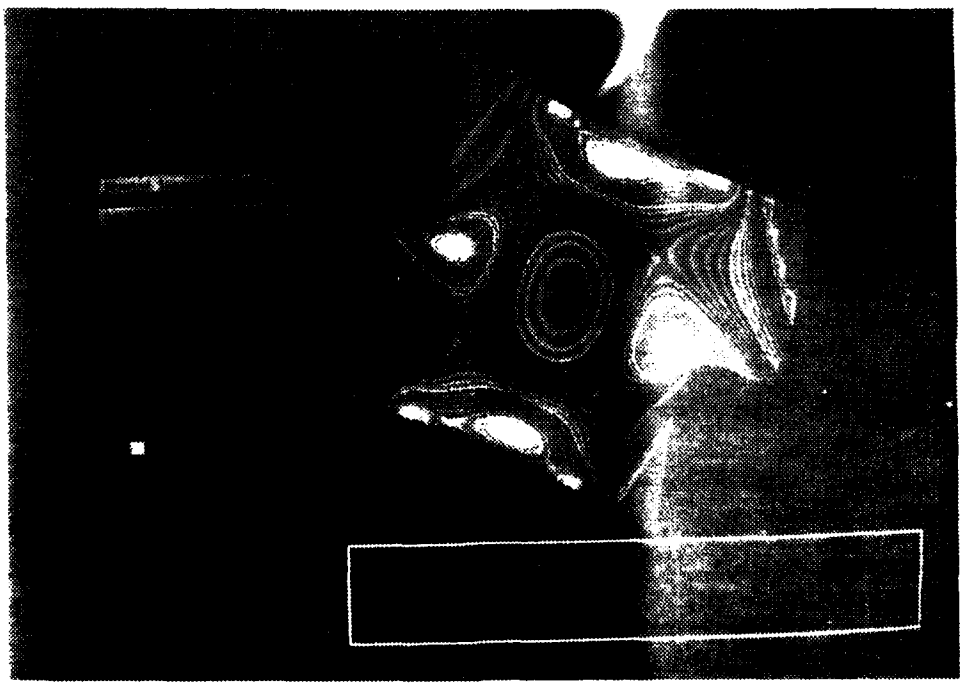


Figure 7.17 - Shaded Image : [illegible]

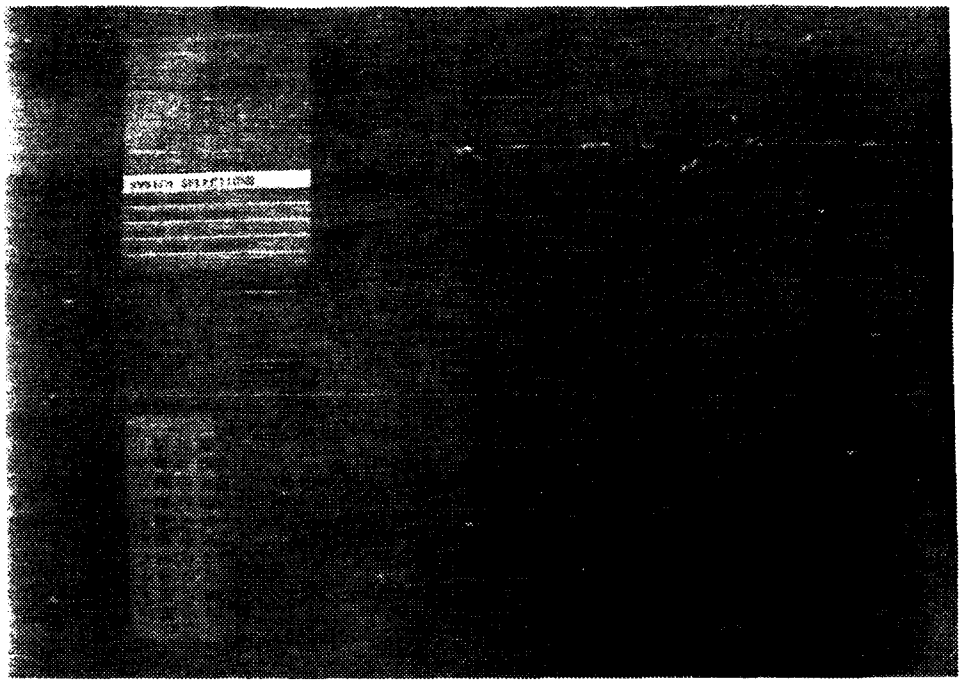


Figure 7.12 - Test Harness SYSTEM SELECTION Screen

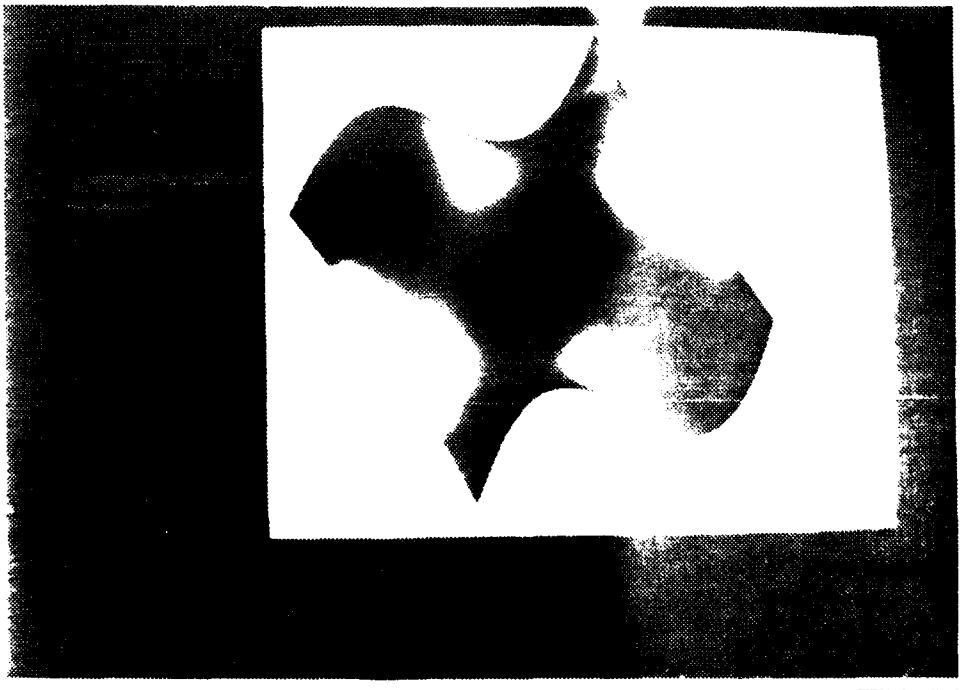


Figure 7.13 - Shaded Image Out

7.3.6 External Jobs

As mentioned previously, the editor has the capability to run external jobs. This capability is used to calculate constant intensity lines called isophotes. Selecting the isophote option of the surface menu and then selecting the calculate isophotes item will start this external process. Figure 7.20 shows the editor screen just after this job has started. Notice the change in the status area of the screen. Job 1 status is now green indicating that an external job has started and is running. When the job completes a message will be sent to the editor and the green color will change to cyan signifying the job has completed and input is ready for the system. Figure 7.21 shows what this will look like (note that this screen would not be seen this soon after the isophote job was started).

7.3.7 Further Development In Surface Area

The routines shown up to this point mainly have dealt with presentation of a surface. There are routines available for editing surfaces, for example tweaking of control points and subdividing and fairing of surfaces, but they have not as yet been interfaced into the editor. Since all functions of the editor will need the capability to visualize curves and surfaces, it was more important to ensure the presentation

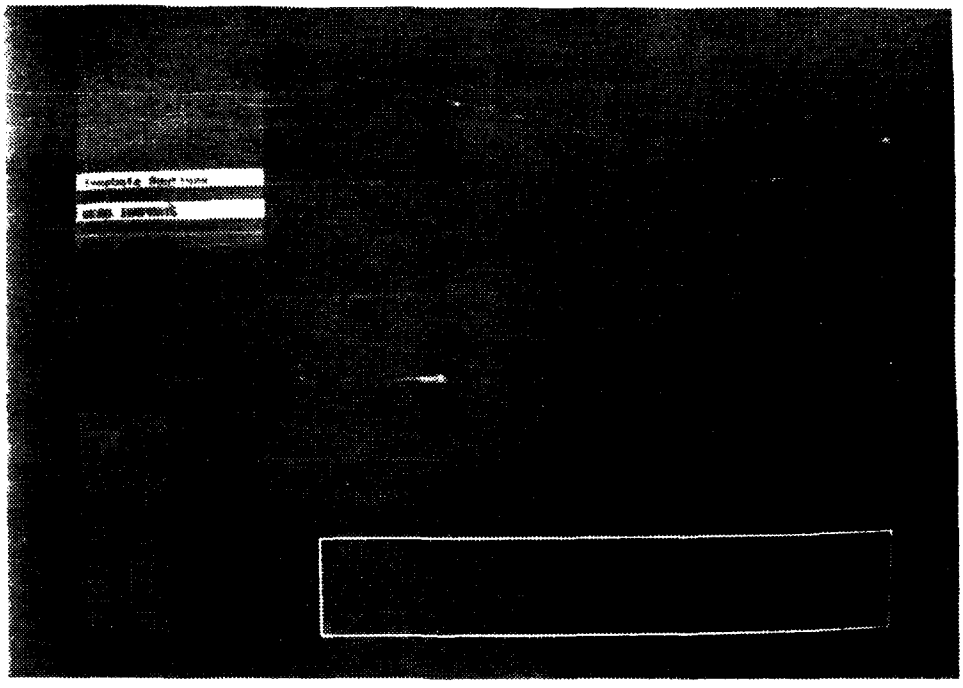


Figure 7.20 - External Job : Status Box

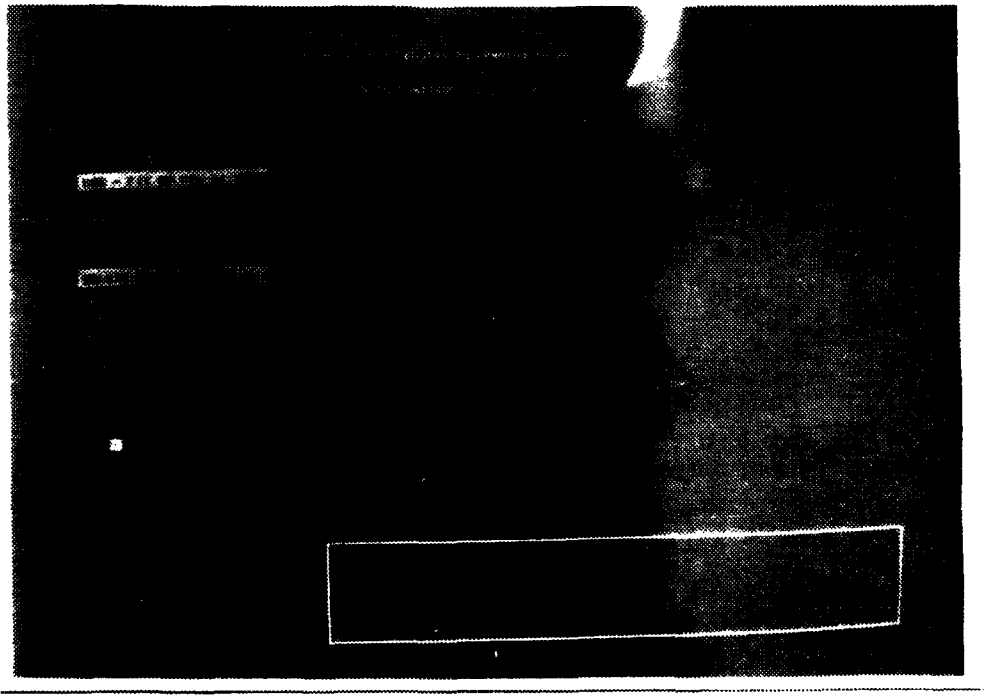


Figure 7.21 - External Job :

modules were complete before application interfacing was done. Chapter 6, NEW MODULES - EDITOR EXPANSION, discusses how the editing function can be easily added to the editor.

7.4 Curve Operations - Entering and Editing

One editing section that has been interfaced to the editor is entering points in the parameter space of a parametric surface, in this case a NURBS surface patch. This allows the user to design a curve on a surface for further processing. One routine that will require this type of interface is blending operations between two curves on surfaces. The presentations given during the input and editing of the parameter points will be discussed next.

7.4.1 Input Of Data Points From File

By selecting the GEOMETRY GENERATION , CURVE ON A SURFACE , FIT LIST OF POINTS items, the user will be in the U-V entry submenu. By selecting the input option, the prompt shown in Figure 7.22 is gotten. When a valid file name is entered, the data is read in and a simple point to point line presentation is given as shown in Figure 7.23. The user is allowed to select this curve if it is correct or select another if some mistake has been made. Once the correct file has been entered, all of the editing routines can be used.

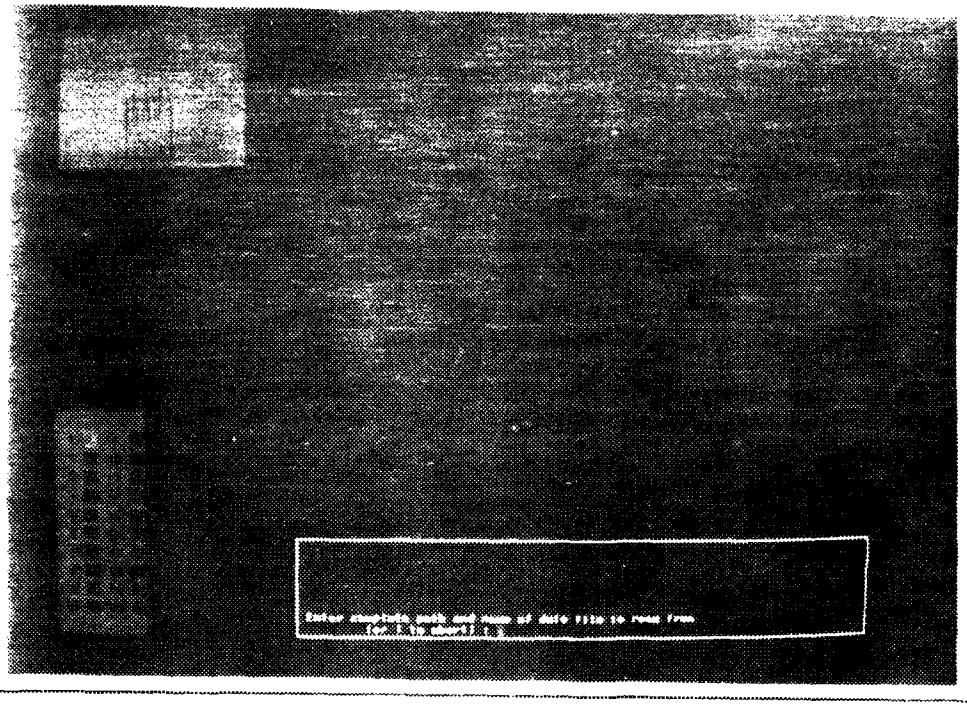


Figure 7.22 - [illegible]

Figure 7.23 - [illegible]

7.4.2 Windowing

It is probable that the designer would want to narrow in on a smaller area of the curve. To do this, the window option is selected. The current window with the curve is given as shown in Figure 7.24. Once the first point has been selected, the presentation will change to that of Figure 7.25. Releasing the left mouse button will fix the selected window.

7.4.3 Adding New Points

Once the area of interest has been selected, the editing functions can be used. First is the addition of points. This is done to the end of the data point array. As shown in Figure 7.26, the last point in the data is connected to the cursor with a temporary line. When a point is actually chosen, this line becomes permanent and a new line is drawn from here to the new cursor position as shown in Figure 7.27. Once all points have been added, the middle or right mouse button is used to quit the add routine.

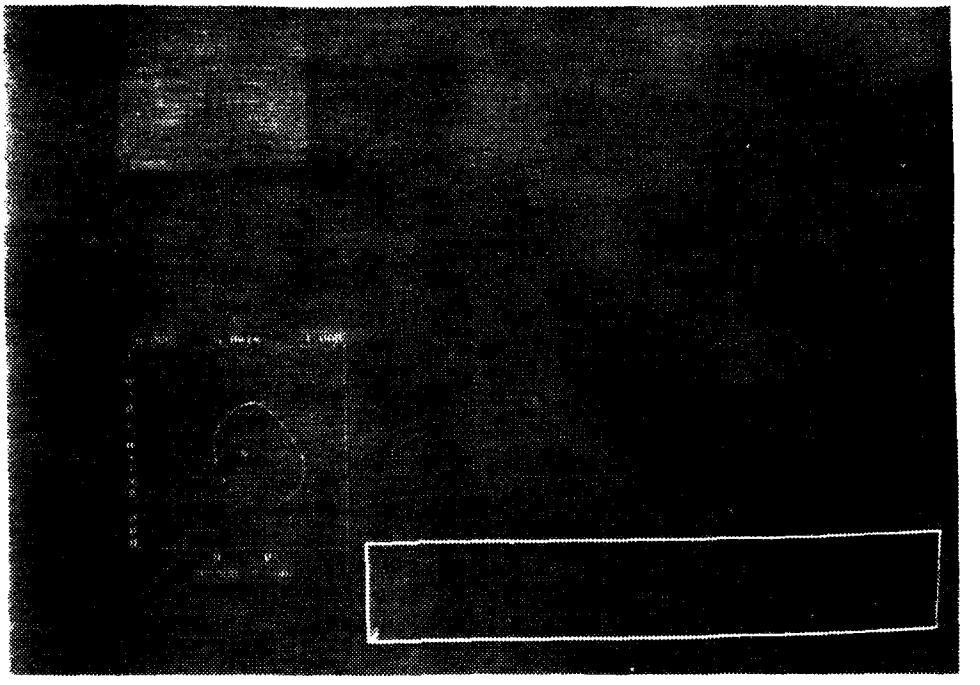


Figure 7.24 - [illegible] presentation

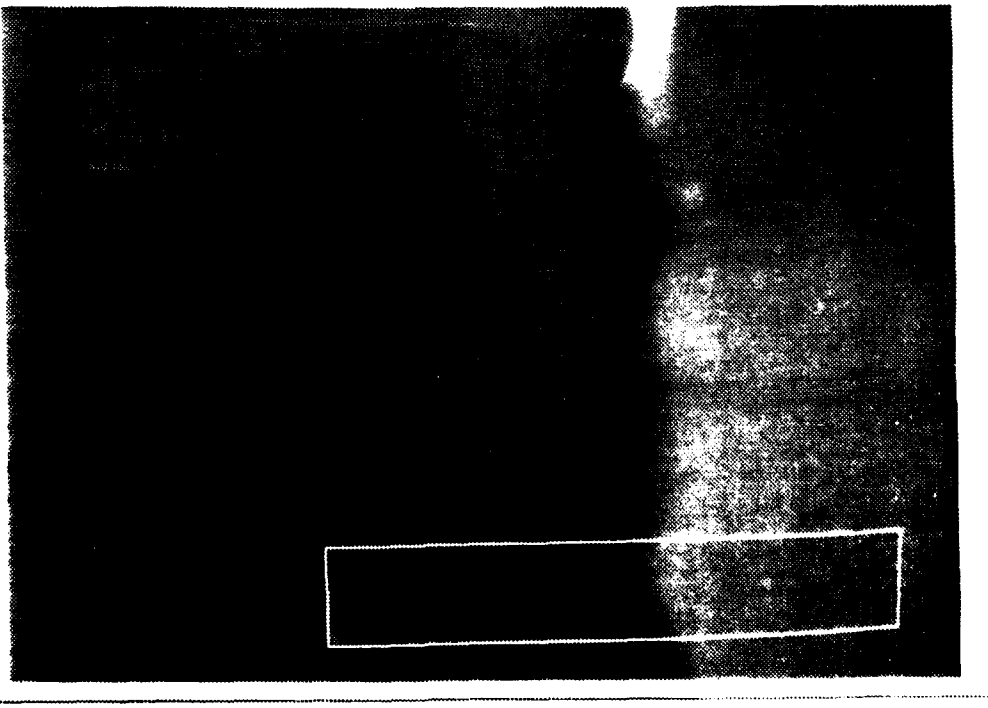
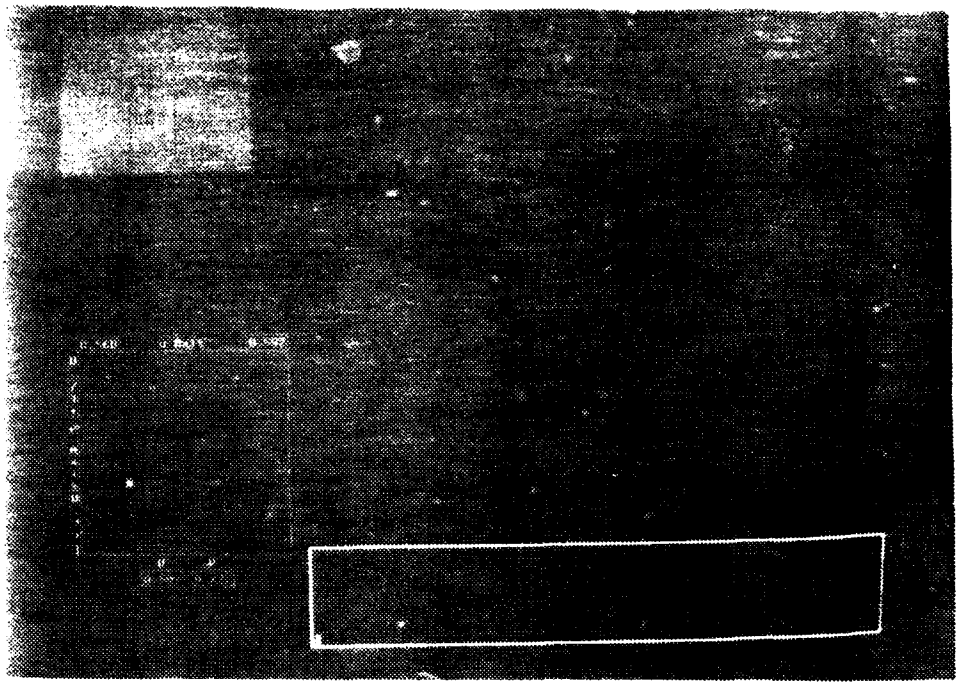
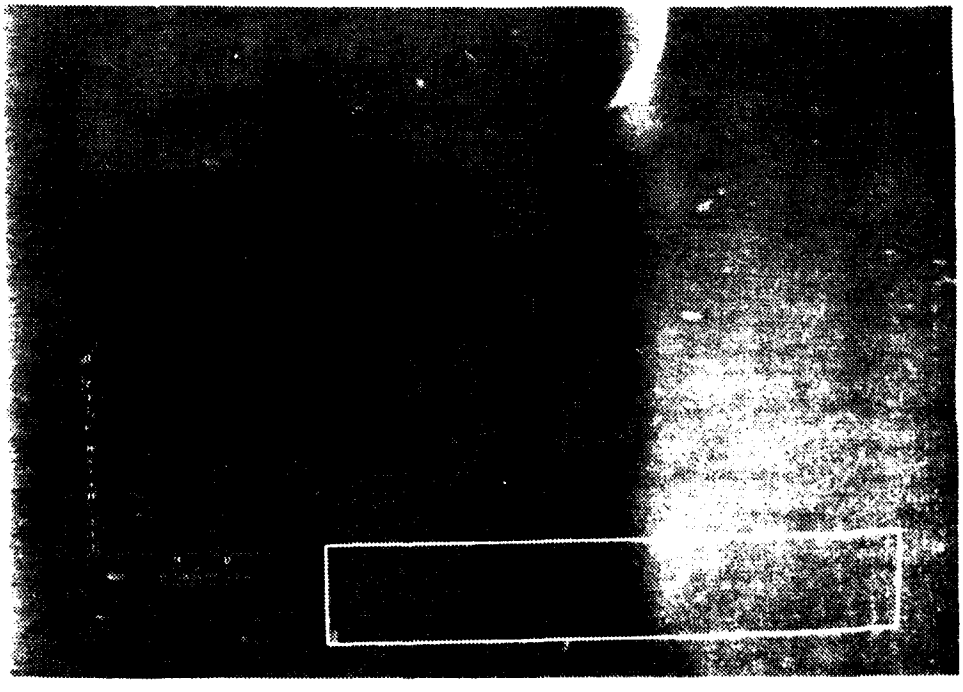


Figure 7.25 - [illegible]



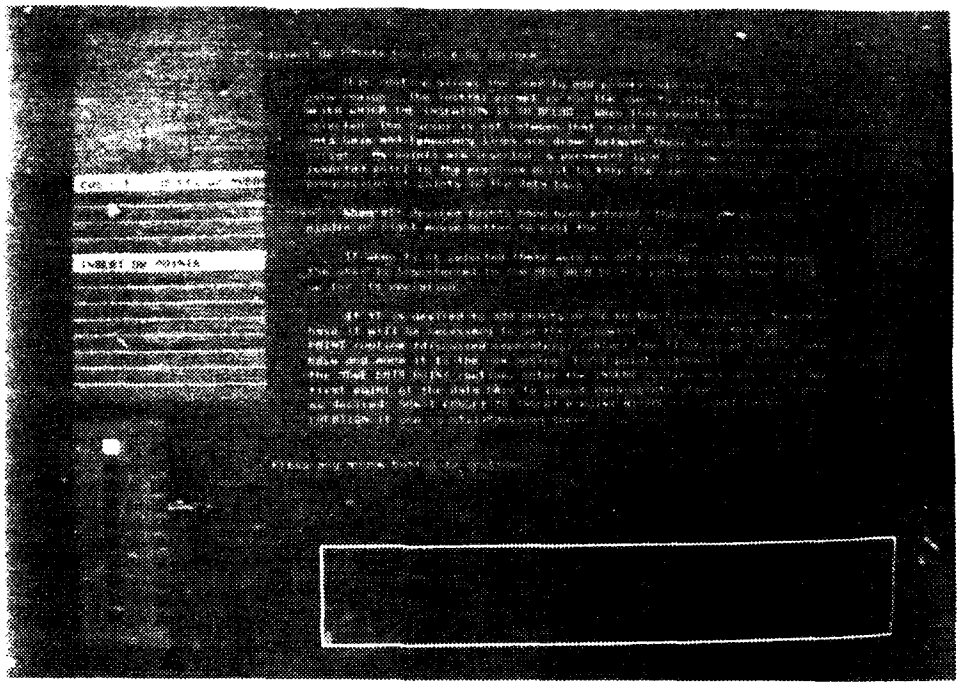


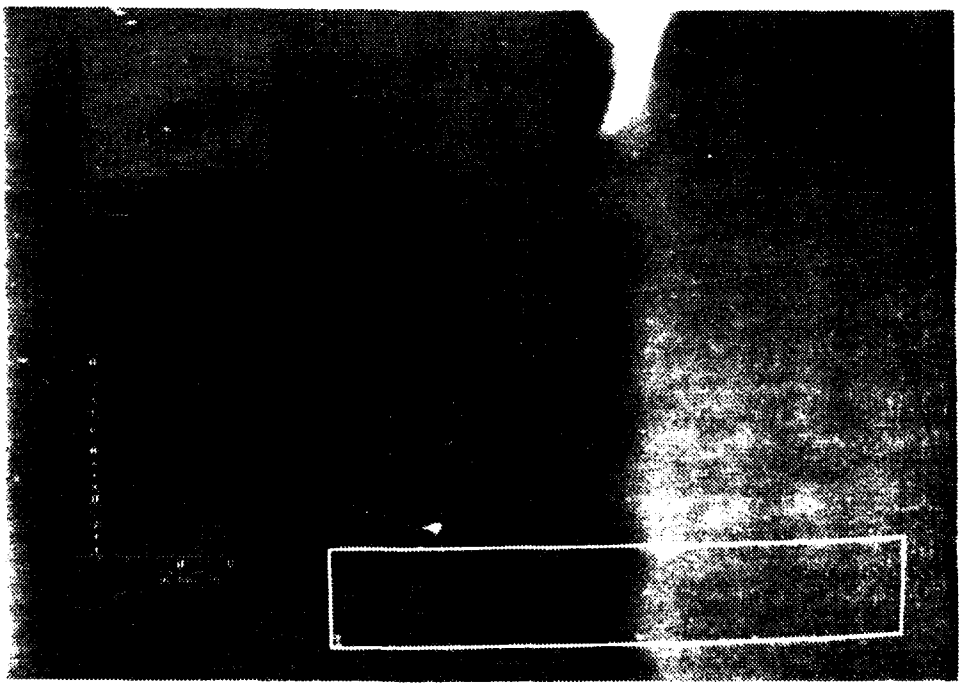
7.4.4 Help Screens

Each option of the editor has a help entry associated with it. Many of the decisions of how to organize the editor and present information to the user were based upon having this help feature available. To access this help feature, the user needs only to place the cursor on the desired menu item and press the middle mouse button. Figure 7.28 shows the help screen available for the insert routine.

7.4.5 Insert New Points

When the insert routine is selected, the user is prompted to select the point after which the insertion is to occur. Once this point is selected, the cursor is placed between the two points and the previously permanent connecting line becomes temporary. Figure 7.29 shows this arrangement. Inserting a point will cause a permanent line to be drawn from the initial point to the inserted point and another point can be inserted. Figure 7.30 shows the situation after three points have been inserted.





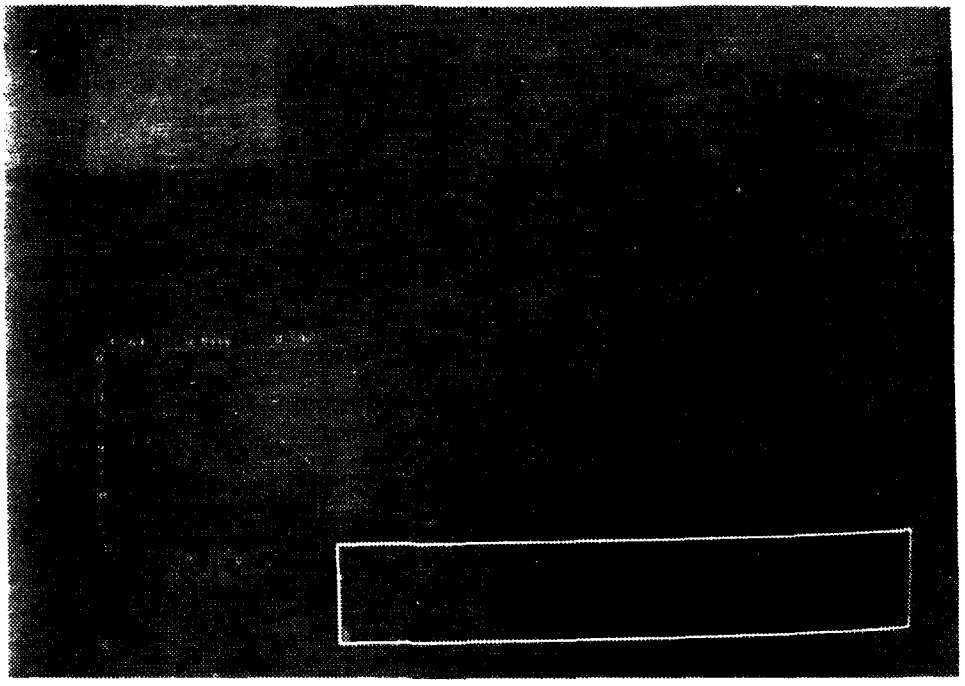


Figure [illegible] - Insert sound [illegible] fiber insertion

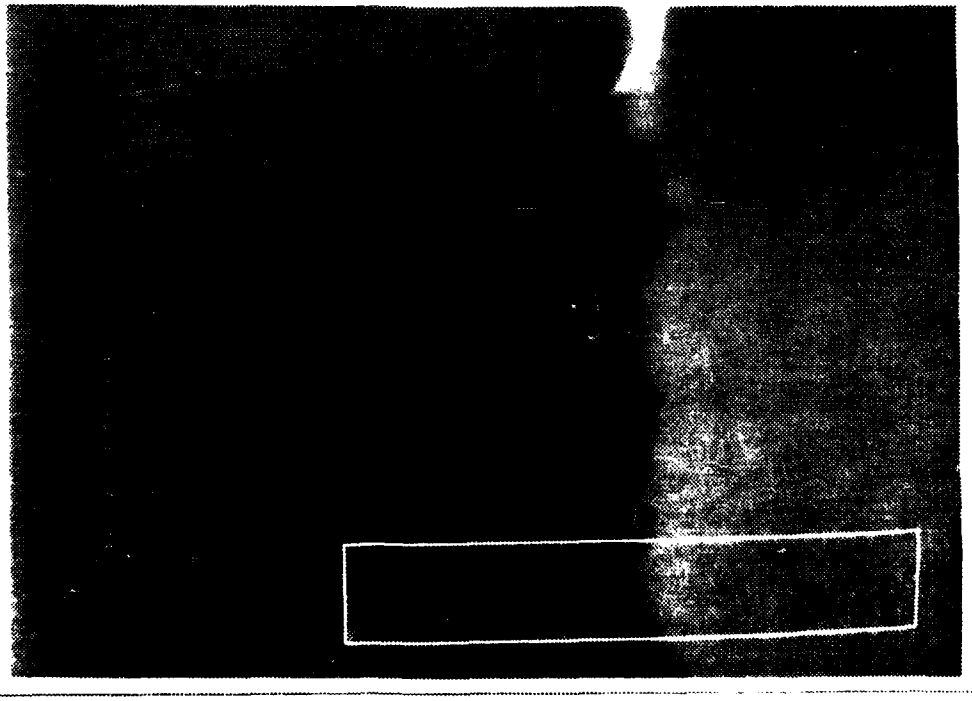


Figure [illegible] - [illegible] [illegible]

7.4.6 Data Point Corrections

If one of the points has been entered incorrectly the user has two ways to fix the problem - move and delete.

7.4.6.1 Move A Point

When the move routine is entered the user is prompted to select the desired point. When the point is selected the permanent joining line to the points on either side is changed to a temporary joining line as shown in Figure 7.31. Once the final position is selected the mouse button is released and Figure 7.32 shows the result.

7.4.6.2 Delete A Point

The other way to correct mistakes is to delete a point. The user is prompted to select a point and when this is done, the prompt and screen shown in Figure 7.33 is given to the user. If the response is to delete the point, it is deleted immediately and Figure 7.34 is the result. If the point is not to be deleted, it is released unchanged.

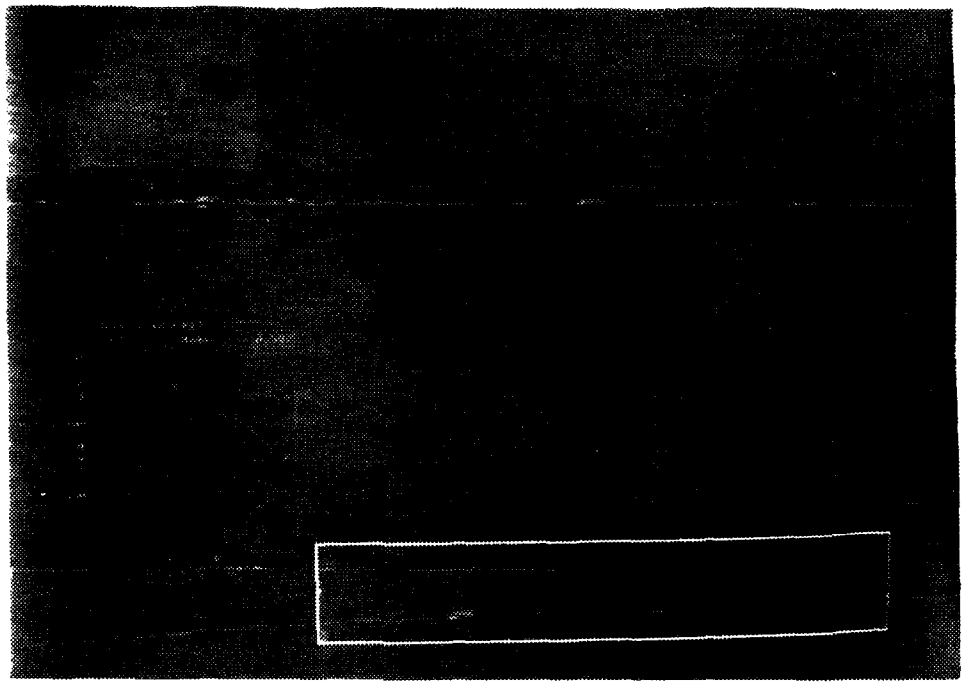


Figure 7.32 - Move : Near Move

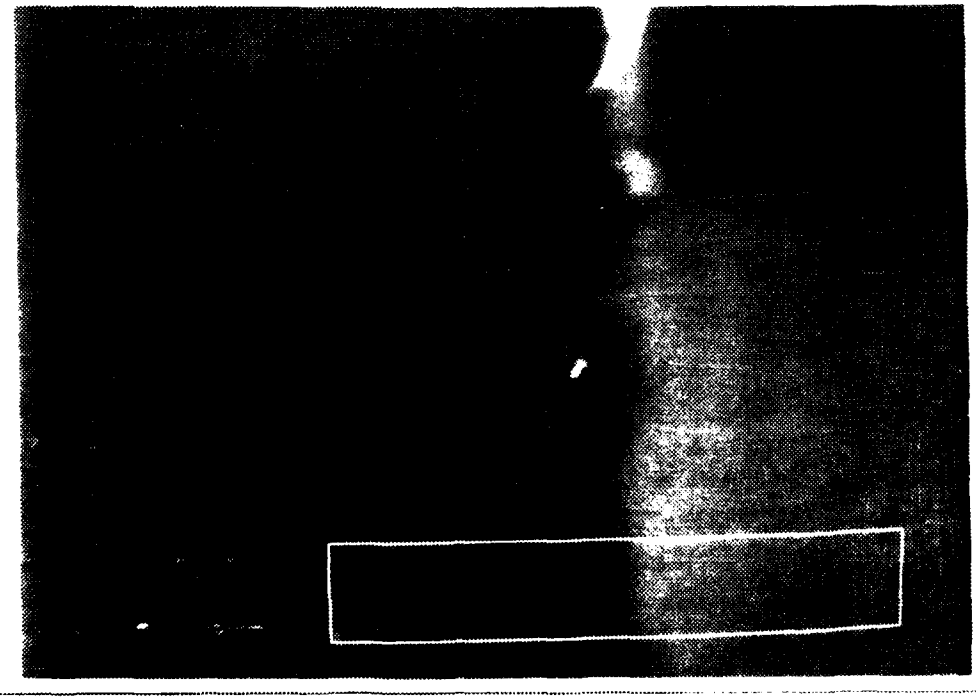
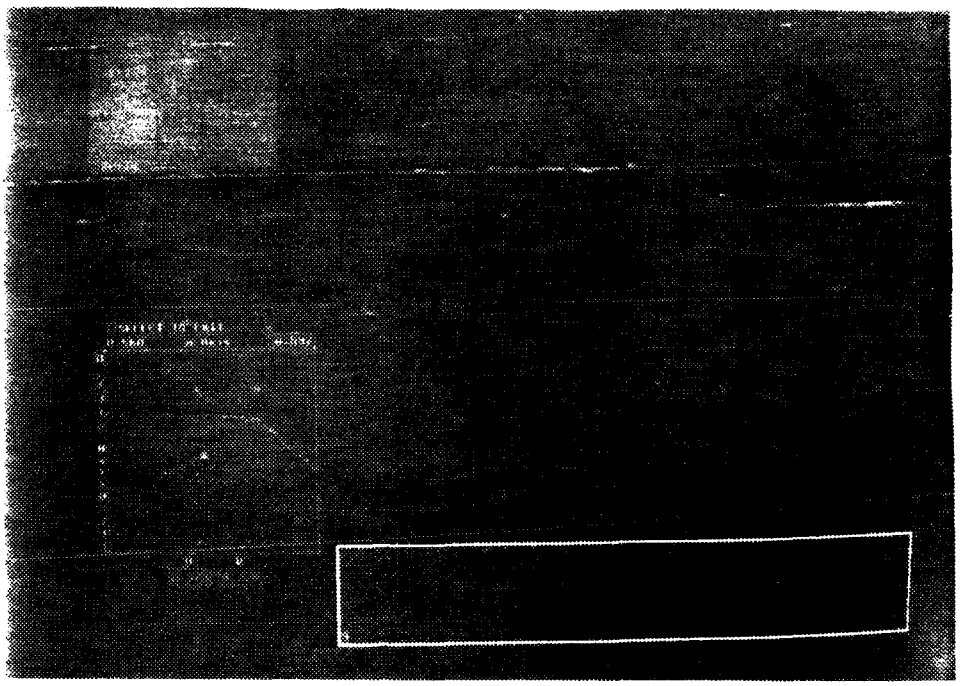
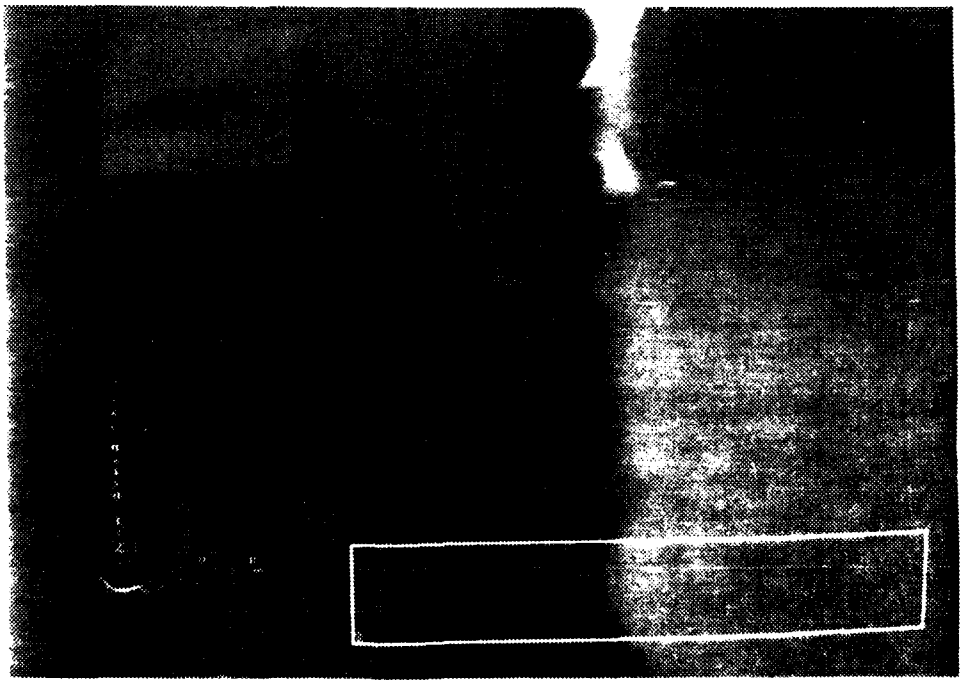


Figure 7.33 - Delete : After De





7.4.7 Fit Of Points and Step Size

Once all points have been entered and edited as desired, the user can fit a fourth order B-spline curve through the points. This curve interpolates all the data points. If this does not appear to be the case it is because the number of steps chosen to display the curve is too small. Setting the number of steps to a larger value will fix this visualization problem. Figures 7.35 and 7.36 show the same curve using 25 and 100 steps.

7.4.8 Making a System Curve and Quitting

When the fit operation is complete, the user could make a system curve and then quit the menu. Care has been taken to ensure data entered into the system is not lost accidentally. If the quit routine is chosen before saving the current data set, the user is prompted as shown in Figure 7.37 and positive action must be taken to continue on in the editor.

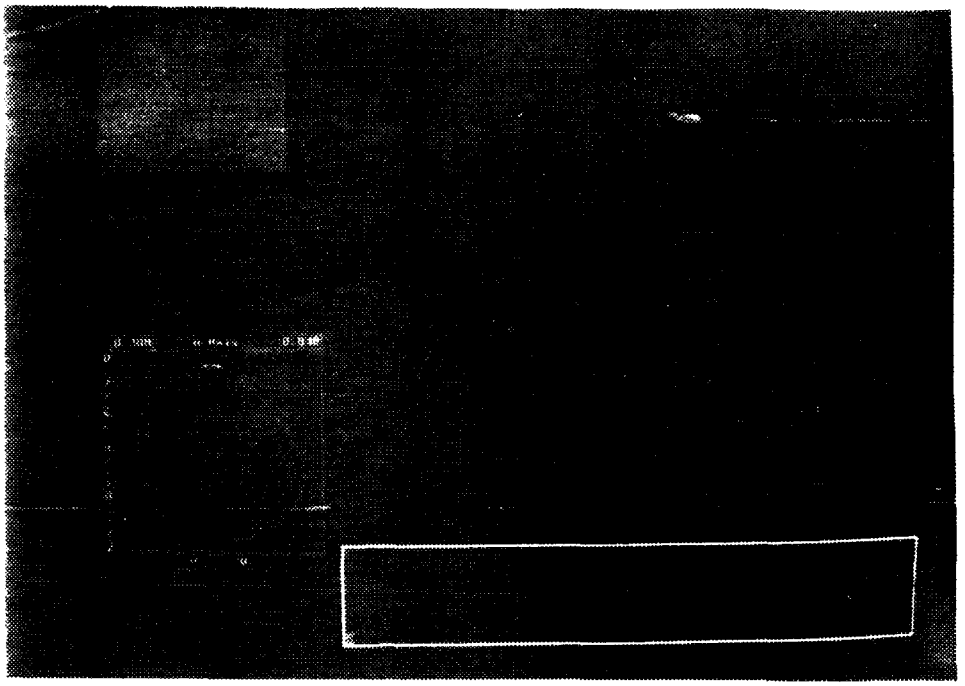


Figure 7.36 - Curve Fit Drawn With 100 Steps

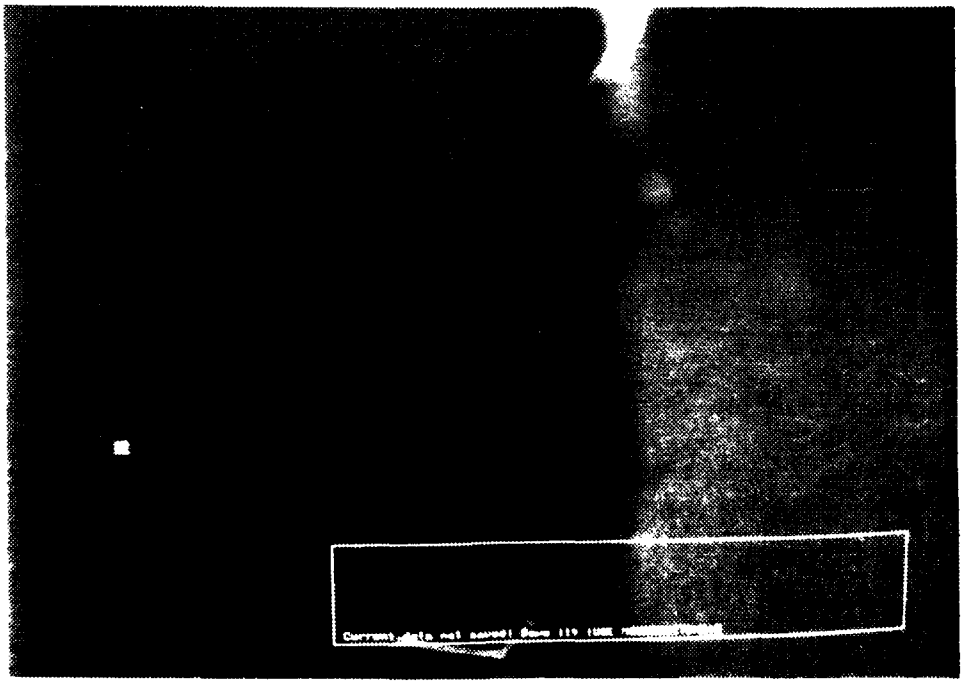


Figure 7.37 - Curve Fit Drawn With 1000 Steps

7.5 Curve Operations - Fairing

Once the designer is satisfied with the points entered for the curve, the GEOMETRY PROCESSING , CURVE ON A SURFACE , FAIRING routine would be called to ensure the curve is sufficiently smooth for use. The various options in the area are discussed below.

7.5.1 Actual Fairing

When the fairing routine is first entered a copy is made of the curve and a curvature map of both curves is given on a split screen presentation. These maps are called porcupines for obvious reasons. The designer can fair the curve on the lower screen. Fairing can be done manually or automatically.

7.5.2 Setting the Scale

The scale of the porcupines can be set to bring all curvatures into view. Figure 7.38 shows the screen upon first entering the fairing routine and Figure 7.39 shows the same presentation but with a scale of 0.1 .

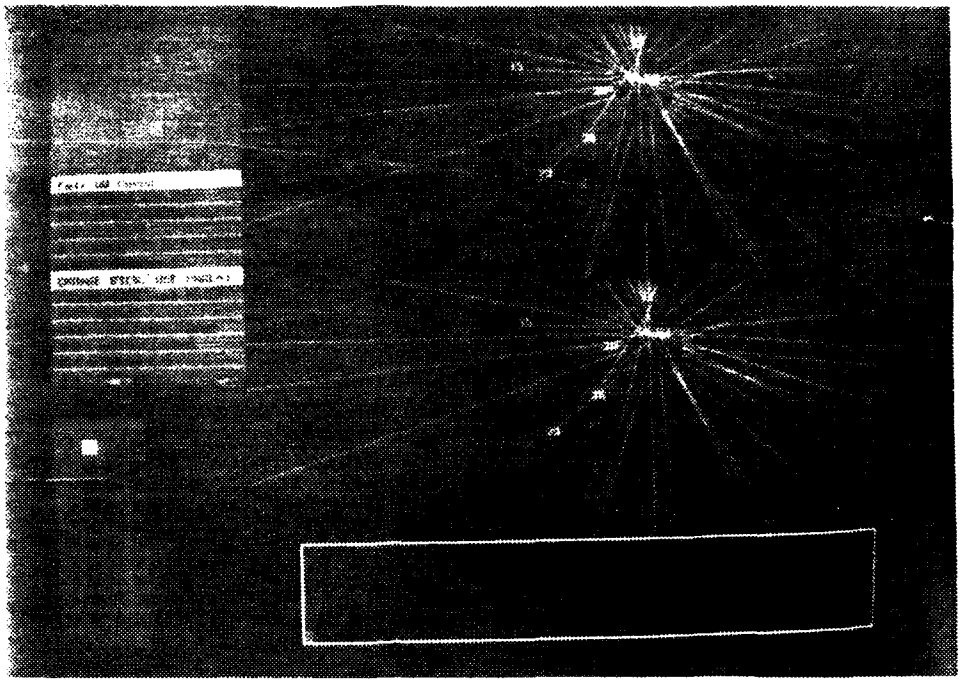


Figure 7.12 – Initial Sailing Ship Presentation

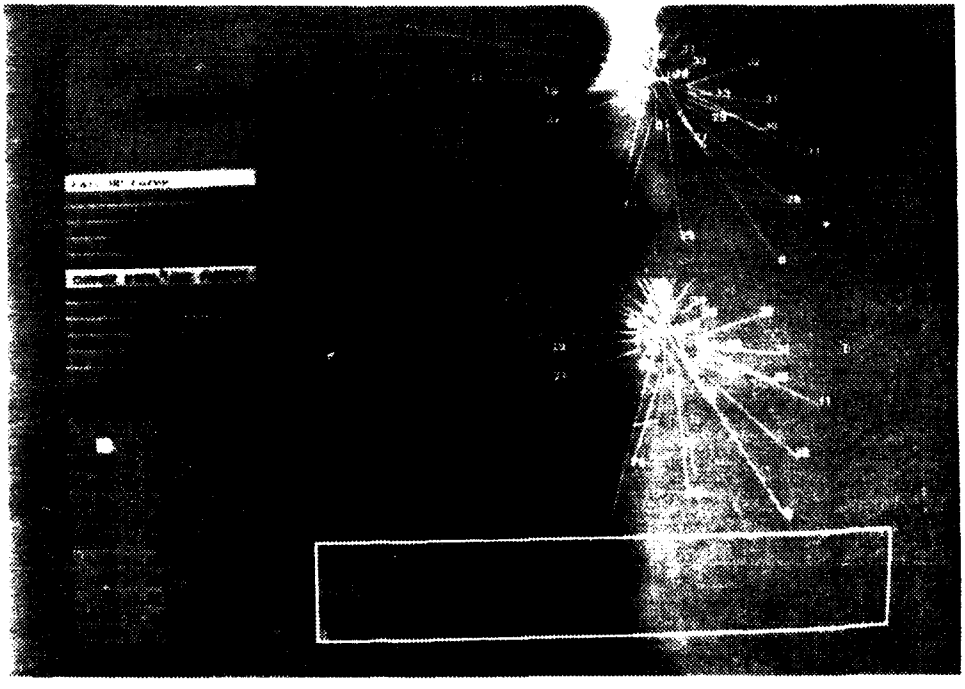


Figure 7.13 – Initial Sailing Ship Presentation

7.5.3 Porcupine View

As the designer fairs the lower curve, the curvature values will change causing the porcupine to change. After fairing has progressed the user may see the changed presentation of Figure 7.40. To get a better idea of the porcupine details, the view of the parent and child porcupines can be changed as desired. The routine for changing the view works as previously discussed in Chapter 7.3.1. Figure 7.41 shows the screen after changing the view.

7.5.4 Curve on Wire Frame

Once the curve is sufficiently fair, the designer can have a wire frame of the surface of interest and the curve displayed at the same time. This is shown in Figure 7.42. As with other presentations, the wire frame and curve viewing position can be changed to suit the designer and Figure 7.43 shows the curve and surface in Figure 7.42 after such a view change.

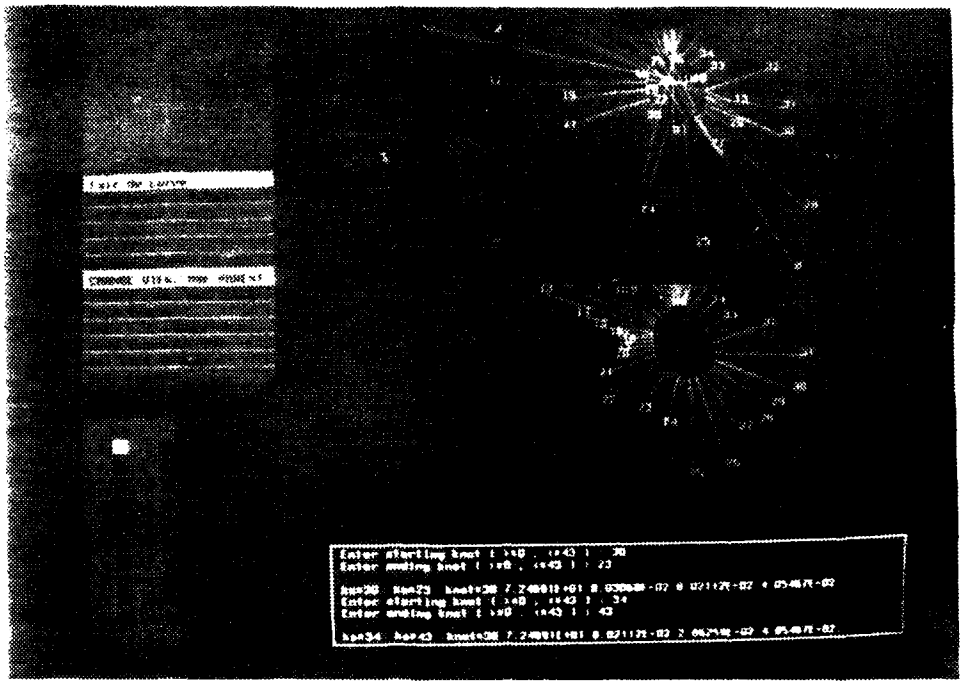


Figure 7.40 - Parent and Child Cursors After Pairing

Figure 7.41 - View Change of Parent and Child

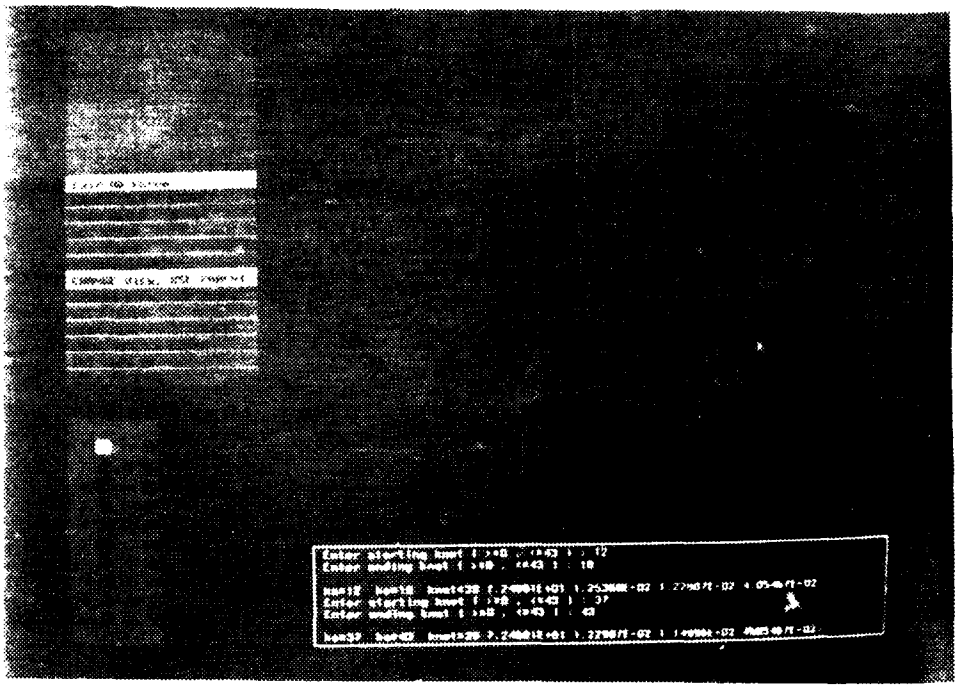
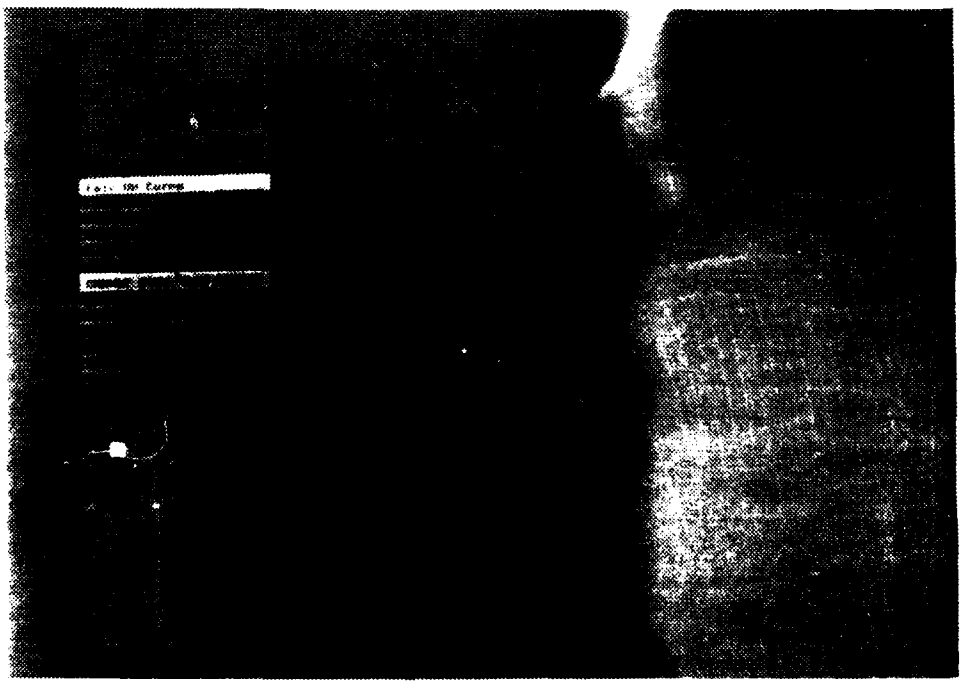


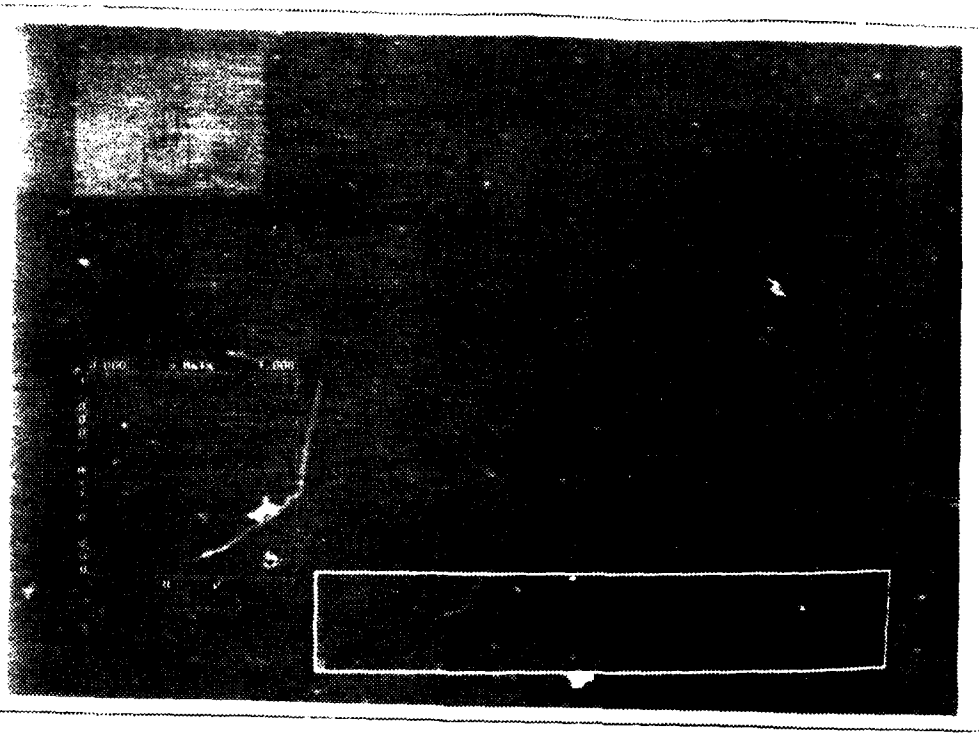
Figure 7.42 - Fluid Flow in Narrow Slot Film.

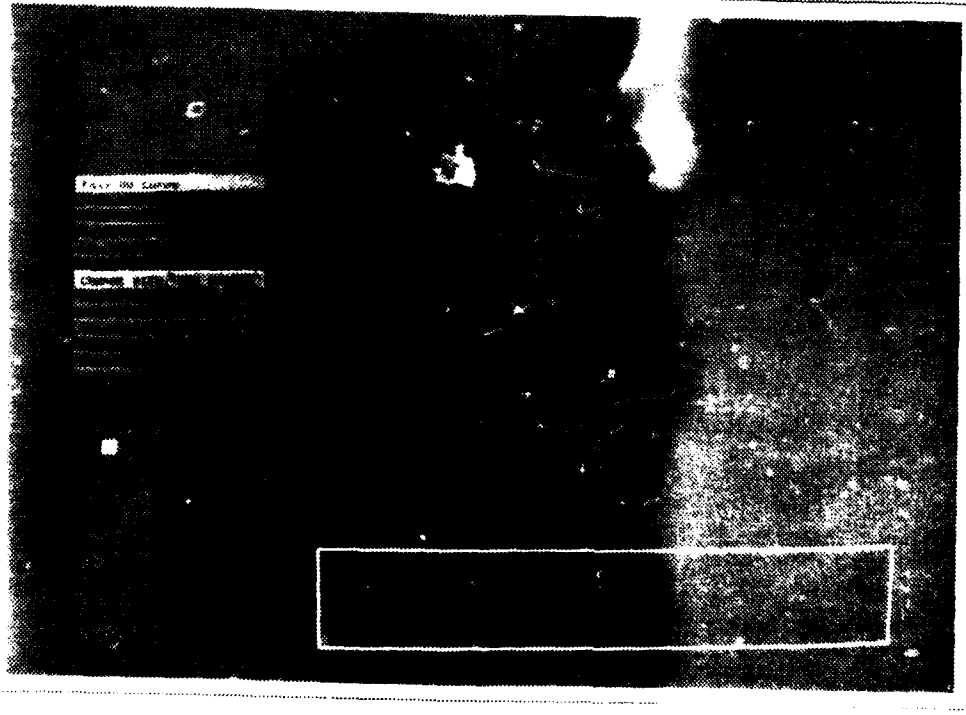


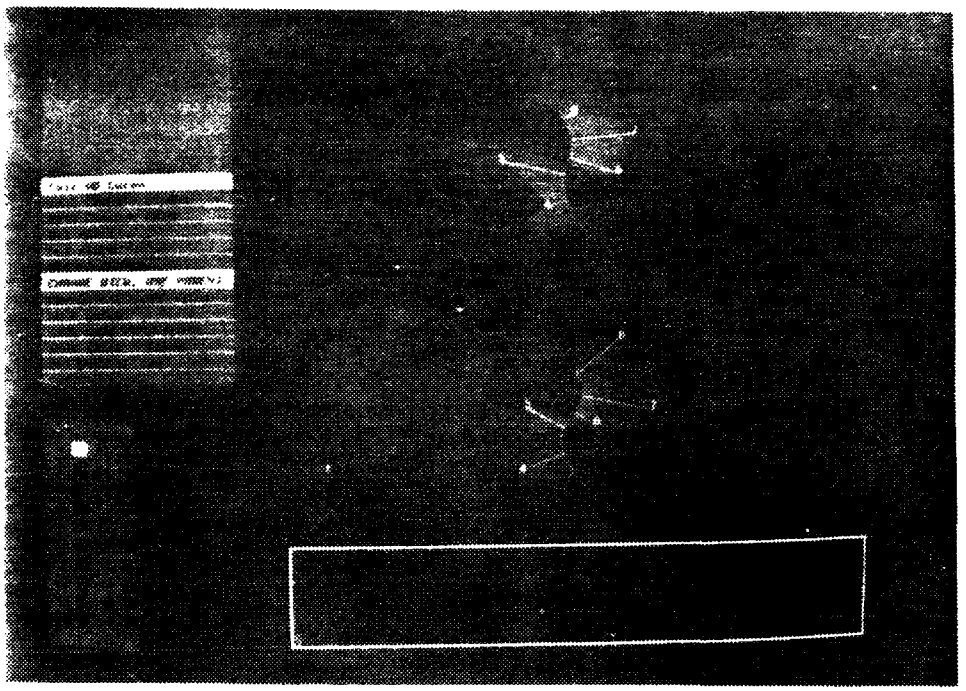
If the curve can not be faired to suit the designer, the fairing routine can be exited and the points in the U-V space changed as necessary. This type of movement through the editor is to be expected during the design process. Once the curve is sufficiently smooth and the presentation of the curve on the surface is acceptable, the child curve can be saved and the editor session can continue with another problem.

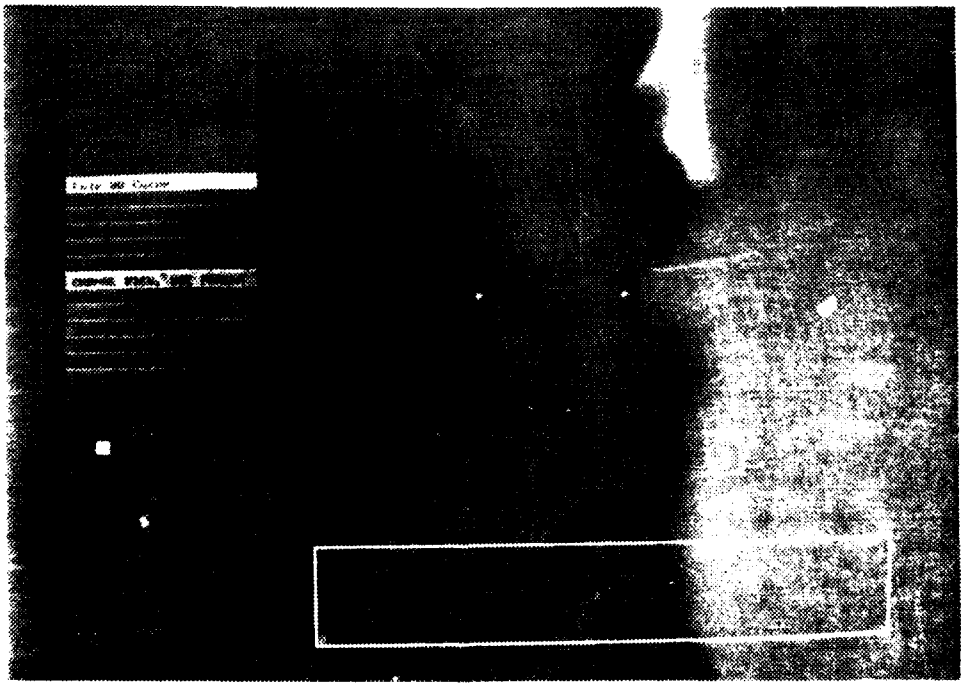
7.6 Open Parametric Curve

The example given above for curve fairing used a periodic curve. The editor will also accept open curves. Figure 7.44 shows an example of an open curve in the parametric space. Figures 7.45 and 7.46 show this curve before and after fairing. Figure 7.47 shows this curve with the same surface and viewport as used in Figure 7.43.









CHAPTER 8
SUMMARY

The previous chapters have discussed the general idea behind the development of the editor, including the need for such an editor and the structure of both the data files and menu arrangements. Examples were given on how to expand the editor to include all the functions listed in the menus. Finally, a pictorial example of the use of the current editor was given.

Future development on the editor should address the continued interfacing of new modules. In addition to this expansion, there are a few enhancements to the modules already interfaced from which the user would benefit. The following are some of these enhancements:

1. Allow the user to enter specific segmentation values greater than 32.
2. Add a call to segmentation routine from within the shading and curvature menus.
3. Allow the step size in the translation and distance portion of the view setting routine to be user selectable and variable.
4. Allow the user to set the color of the background to values other than black and white.
5. Increase the size of the parameter space data input to the larger, 3-D portion of the screen.

6. List applicable data files and allow user to select from the list with the mouse or enter name manually if desired.

These are but some of the enhancements that can be implemented to make the editor even more user friendly. As the editor is used by more designers there will be many more additions and enhancements that need to be made. This is the nature of any program - the more it is used, the more the user will want. It is because of this that the editor was designed for easy additions and changes.

CHAPTER 9
REFERENCES

[1] Patrikalakis, N. M., Bardis, L. and Kriezis, G. A. Approximate Conversion Of Rational B-Spline Curves and Surfaces Patches. Design Laboratory Memorandum No. 88-5, July, 1988.

[2] Curry, H. B., and Schoenberg, I. J. On the Polya Frequency Functions IV: The Fundamental Spline Functions and their Limits. *Journal d' Analyse Mathematique*, 17:71-107, 1966.

[3] De Boor, C. On Calculating with B-Splines. *Journal of Approximation Theory*, 6:50-62, 1972.

[4] Cox, M. G. The Numerical Evaluation of B-Splines. *Journal of the Institute for Mathematics Applications*, 10:134-149, 1972.

[5] Tiller, W. Rational B-Splines for Curve and Surface Representation. *IEEE Computer Graphics and Applications*, 3(6):61-69, September, 1983.

[6] IRIS User's Guide, Volume 1, Programming Guide, Version 4.0, Document Number 007-1101-040, Silicon Graphics, Inc., Mountain View, CA, 1987.

[7] Alourdas, P. G.

Shape Creation, Interrogation and Fairing Using B-Splines. Naval Engineer's Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, May, 1989.

[8] Thomas, R., Rogers, L. R., and Yates, J. L.

Advanced Programmer's Guide To UNIX System V. Osborne McGraw-Hill, Berkeley, CA, 1986.

[9] Kernighan, B. W., and Ritchie, D. M.

The C Programming Language, 2nd Edition. Prentice-Hall, Englewood Cliffs, NJ, 1988.

CHAPTER 10
APPENDICES

10.1 MAIN MENU DATA FILE

The following file is used to set up the main menu of the editor. Additions and changes discussed in Chapter 6 are indicated by bullets, ⌒. As discussed previously the indentation scheme is not required but it is suggested that it be used to add clarity in the presentation of long data files. Also, for a clearer presentation the data file has larger spacing between the different menu items that would be the actual case in the data file. THERE CAN BE NO BLANK LINES IN THE DATA FILE.

```
5 ————————————————————————————————————F
MAIN MENU
Main Menu Routines
N50
   ————————————————————————————————————E
   9
   INPUT ROUTINES
   Input Menu Routines
   N9
      4
      CURVE (3-D)
      3-D Curve Input Routines
      N0
         1
         ENTER FROM KEYBOARD
         V0
         1
         RECALL FROM LOCAL FILE
         V1
         1
         RECALL IGES FILE
         V2
         1
         INTERACTIVE INPUT
         V3
```

```
4
SURFACE
Surface Input Routines
N1

    1
    ENTER FROM KEYBOARD
    V4

    1
    RECALL FROM LOCAL FILE
    V5

    1
    RECALL IGES FILE
    V6

    1
    INTERACTIVE INPUT
    V7

3
CURVE ON SURFACE
Curve On A Surface Data
N2

    1
    ENTER FROM KEYBOARD
    V8

    1
    RECALL FROM LOCAL FILE
    V9

    1
    RECALL IGES FILE
    V10

2
ALGEBRAIC SURFACE
Algebraic Surface Inputs
N3

    1
    ENTER FROM KEYBOARD
    V11

    1
    RECALL FROM LOCAL FILE
    V12

2
ID OF POINTS
Grid Of Points Input
N4
```

```
    1
    ENTER FROM KEYBOARD
    V13
    1
    RECALL FROM LOCAL FILE
    V14
2
FUNCTION ON CURVE
Function On A Curve Menu
N5
    1
    ENTER FROM KEYBOARD
    V15
    1
    RECALL FROM LOCAL FILE
    V16
3
LIST OF POINTS
List Of Points Input Menu
N6
    1
    ENTER FROM KEYBOARD
    V17
    1
    RECALL FROM LOCAL FILE
    V18
    1
    INTERACTIVE INPUT
    V19
2
LIST OF LISTS
List Of Lists Input Menu
N7
    1
    RECALL FROM LOCAL FILE
    V20
    1
    INTERACTIVE INPUT
    V21
3
LIST OF POINTS (3-D)
List Of Points (3-D)
N8
```

```
        1
        ENTER FROM KEYBOARD
        V22

        1
        RECALL FROM LOCAL FILE
        V23

        1
        INTERACTIVE INPUT
        V24

4 ————————————————————————————————————(K)
GEOMETRY GENERATION
GEO Generation Routines
N15
    ————————————————————————————————————(J)
    4
    CURVES
    Curve Generation Menu
    N10

        1
        FIT POINTS IN 3-D
        V25

        1
        APPROXIMATE WITH NURBS
        V26

        1
        OFFSET OF A PLANAR CURVE
        V27

        1
        OFFSET NORMAL TO PATCH
        V28

    5
    SURFACES
    SUR Generation Routines
    N11

        1
        OFFSET OF ANOTHER SURFACE
        V29

        1
        RULED SURFACE
        V30

        1
        FIT/APPROX n ISOPARAMETER
        V31
```

```
    1
    FIT/APPROX GRID OF POINTS
    V32

    1
    CONVERT ALG TO NURBS
    V33

3
CURVE ON SURFACE
COS Generation
N12

    1
    FIT/APPROX LIST OF POINTS
    V34

    1
    FIT/APPROX LIST OF LISTS
    V35

    1
    VAR OFFSET OF ANOTHER
    V36

4
BLEND
Blend Generation
N14

    3
    BOUNDARY CONDITIONS
    Blend Boundary Conditions
    N13

        1
        POSITION
        V37

        1
        NORMAL
        V38

        1
        CURVATURE
        V39

    1
    DEFINE SURFACE
    V40

    1
    DEFINE CURVES
    V41
```

```
        1
        EXECUTE BLEND
        V42
3
GEOMETRY INTERROGATION
Geometry Interrogation
N36
    3
    CURVES
    Curve Interrogation
    N19
        3
        VISUALIZATION
        Curve Visualization
        N16
            1
            RESOLUTION
            V43
            1
            COLOR
            V44
            1
            VIEWPOINT
            V45
        2
        CURVATURE VALUES
        Curvature Map Values
        N17
            1
            RESOLUTION
            V46
            1
            SHOW CURVATURE MAP
            V47
    2
    STATUS
    Curve Status
    N18
            1
            ON
            V48
            1
            OFF
            49
```

```
3
CURVES ON SURFACE
Curves On Surface
N23

    3
    VISUALIZATION
    Visualization Routines
    N20

        1
        RESOLUTION
        V50
          ------------------------------------(C)
        1
        LINETYPE
        V51

        1
        VIEWPOINT
        V52

    2
    CURVATURE MAP
    Curvature Map Routines
    N21

        1
        RESOLUTION
        V53

        1
        SHOW
        V54

    2
    STATUS
    Status
    N22

        1
        ON
        V55

        1
        OFF
        V56

10
SURFACES
Surface Routines
N35
```

```
3
VISUALIZATION
Visualization Routines
N24

     1
     RESOLUTION
     V57

     1
     COLOR
     V58

     1
     VIEWPOINT
     V59

4
PLANE CONTOURS
Plane Contours Menu
N25

    1
    SET # PLANES
    V60

    1
    SET START PLANE
    V61

    1
    SET PLANE DISTANCE
    V62

    2
    INTERSECTION ACCURACY
    Intersection Accuracy
    N26

        1
        2_D
        V63

        1
        3_D
        V64

4
CYLINDER CONTOURS
Cylinder Contours
N27

    1
    SET # CYLINDERS
    V65
```

```
    1
    SET START CYLINDER
    V66
    1
    CYLINDER DISTANCE
    V67
    2
    INTERSECTION ACCURACY
    Intersection Accuracy
    N28
        1
        2_D
        V68
        1
        3_D
        V69
5   ------------------------------------------------------- B
SHADED IMAGE
Shaded Image Routines
N29
    1
    READ IMAGE
    V70
    1
    CALCULATE IMAGE
    V71
    1
    COLOR
    V72
    1
    SET LIGHT SOURCE
    V73
    ------------------------------------------------------- A
    1
    VIEW
    V74
3
RAY TRACE
Ray Trace Routines
N30
    1
    READ TRACE
    V75
```

```
    1
    CALCULATE TRACE
     V76

    1
    SET COLOR
    V77

10 ------------------------------------------------------------ H
CURVATURE
Curvature Routines
N31

    1
    READ CURVATURE
    V78

    1
    CHANGE VIEW
    V87
    ------------------------------------------------------------ G
```

```
    1
    ALL CURVATURES
    V79
    1
    GAUSSIAN
    V80
    1
    MEAN
    V81
    1
    ABSOLUTE
    V82                                  I
    1
    MAXIMUM PRINCIPLE
    V83
    1
    MINIMUM PRINCIPLE
    V84
    1
    NORMAL U
    V85
    1
    NORMAL V
    V86
```

```
4
ISOPHOTES
Isophote Routines
N32
```

```
      1
      SET NUMBER
      V88

      1
      READ ISOPHOTE
      V89

      1
      CALCULATE ISOPHOTE
      V90

      1
      SHOW ISOPHOTE
      V91

   4
   REFLECTION LINES
   Reflection Lines
   N33

      1
      SET NUMBER
      V92

      1
      READ IN LINES
      V93

      1
      CALCULATE LINES
      V94

      1
      SHOW LINES
      V95

   3
   GEODESICS
   Geodesics Routines
   N34

      1
      READ IN
      V96

      1
      CALCULATE
      V97

      1
      SHOW
      V98

1
SURFACE ON/OFF
V99
```

```
4
GEOMETRY PROCESSING
Geometry Processing
N49
    6
    CURVES
    Curves Processing
    N39
        3
        APPROXIMATE NURBS
        Approximate NURBS
        N37
            1
            SET ORDER
            V100
            1
            SET ACCURACIES
            V101
            1
            RUN
            V102
        3
        FAIRING
        Fair Curve
        N39
            1
            KNOT
            V103
            1
            AUTOMATED
            V104
            1
            RUN
            V105
        1
        CTRL PT EDIT
        V106
        1
        EXACT DEGREE
        V107
        1
        SUBDIVIDE
        V108
```

```
    1
    SPLIT CURVE
    V109
5
COS PROCESSING
Curve On Surface
N43
    2
    CONVERT COS TO NURBS
    Convert Curve On Surface
    N41
        3
        SET ACCURACIES
        Accuracy Setting
        N40
            1
            POSITION
            V110
            1
            CURVATURE
            V111
            1
            SLOPE
            V112
        1
        RUN CONVERT
        V113
    1
    FAIRING
    V114
    1
    EDITING
    V117
    1
    SUBDIVIDE IN UV
    V118
    1
    SPLIT IN UV
    V119
6
SURFACE PROCESSING
Surface Processing
N47
```

```
2
APPROXIMATE NURBS
Approximate With NURBS
N45
    1
    SET ORDER
    V120
    3
    SET ACCURACIES
    Set Accuracies
    N44
        1
        POSITION
        V121
        1
        CURVATURE
        V122
        1
        SLOPE
        V123
3
FAIRING
Surface Fairing
N46
    1
    KNOT
    V124
    1
    AUTOMATED
    V125
    1
    RUN FAIRING
    V126
1
EDITING
V127
1
DEGREE ELEVATION
V128
1
SUBDIVIDE
V129
```

```
            1
            SPLIT
            V130
        2
        INTERSECTIONS
        Intersections
        N48
            1
            LISTS 2_D
            V131
            1
            LISTS 3_D
            V132
    1
    QUIT
    V133
0
END MENU
```

10.2 UV_MENU.DAT DATA FILE

```
13
FIT/APPROX LIST OF POINTS
COS - Fit UV Pts w/ NURBS
N53
   1
   INPUT UV POINTS
   V141
   1
   OUTPUT UV POINTS
   V142
   1
   SHOW UV POINTS
   V150
   1
   ADD UV POINTS
   V143
   1
   INSERT UV POINTS
   V144
   1
   DELETE UV POINTS
   V145
   1
   MOVE UV POINTS
   V146
   1
   SELECT WINDOW
   V147
   1
   FIT POINTS
   V148
   1
   MAKE SYSTEM CURVE
   V151
   1
   SET STEPS
   V158
   1
   START AGAIN
   V161
```

```
   1
   QUIT
   V149
0
END MENU
```

10.3 PROGRAM MAKEFILE

The make file listed in this appendix was used to compile the editor program during development. The additions discussed in Chapter 6 are indicated by ⌒.

```
ROOT = /u1/deslab/hottel
BINDIR = ../bin

INCLUDE = /usr/local/einclude
INCLUDE2 = $(ROOT)/thesis1

OBJECTS2 = getsubs.o menu_allocate.o linked_menu.o \
        read_help.o sta_box.o subs.o change_view_aziy.o \
        surface_input_local.o segments.o surface_vis.o \
        hdraw.o all_curvature.o shaded.o set_world.o \
        pick_surf.o isophotes.o message_handling.o \
        run_uv_entry.o delete_uv_points.o move_uv_points.o \
        insert_uv_points.o window_uv_points.o \
        input_uv_points.o output_uv_points.o \
        add_uv_points.o toggle_bell.o set_steps.o \
        pick_uv_points.o quit_uv_points.o view_uv_points.o \
        fit_uv.o pick_curv.o run_cos_fairing.o \
        copy_FulCurv.o make_child_from_parent.o \
        fair_cos_child.o keep_cos_child.o fair_per_knot.o \
        calc_desc.o fair_knot.o quit_cos_fairing.o \
        find_deviation.o porcupine.o get_new_uv_points.o \
        curve_input_local.o put_curve_on_surf.o \
        cos_and_wire_frame.o cos_and_shaded_surface.o \
        print_FulCurv.o popup.o ⌒a ⌒b

OBJECTS3 = getkey.o strdup.o setmapcolor.o show_mouse.o \
        mousewords.o vp.o check_file.o cleararea.o

OBJECTS1 = mainmenu.o

MESSAGES = linked_menu.o mainmenu.o message_handling.o \
        run_cos_fairing.o run_uv_entry.o

CFLAGS = -g -p -I$(INCLUDE) -I$(INCLUDE1)
FFLAGS = -g -p -I$(INCLUDE) -I$(INCLUDE1)

LIB = /usr/local/elib/libbspl.a /usr/local/elib/libgen.a \
     /usr/local/elib/libgraph.z
```

```
LIB1 = rgl2
LIB2 = oegl50

OBJECTS = main.o $(OBJECTS1) $(OBJECTS2) $(OBJECTS3)

menu: Makefile $(OBJECTS)
       f77 -g -o mainmenu $(OBJECTS) $(LIB) -l$(LIB1) -lm -lnag

$(OBJECTS1): $(ROOT)/thesis1/struct.h
$(OBJECTS1): $(ROOT)/thesis1/hottel.h

$(OBJECTS2): $(ROOT)/thesis1/struct.h

$(OBJECTS): $(ROOT)/thesis1/defines.h

$(MESSAGES): $(ROOT)thesis1/msg.h
```