AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 2

SAND94-1100
Unlimited Release
Printed May 1994

Distribution
Category UC-705

CUBIT Mesh Generation Environment Volume 1: Users Manual

Cubit Development Team*
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

The CUBIT mesh generation environment is a two- and three-dimensional finite element mesh generation tool which is being developed to pursue the goal of robust and unattended mesh generation—effectively automating the generation of quadrilateral and hexahedral elements. It is a solid-modeler based pre-processor that meshes volume and surface solid models for finite element analysis. A combination of techniques including paving, mapping, sweeping, and various other algorithms being developed are available for discretizing the geometry into a finite element mesh. CUBIT also features boundary layer meshing specifically designed for fluid flow problems. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes. This manual is designed to serve as a reference and guide to creating finite element models in the CUBIT environment.

This manual documents CUBIT Version 1.8.1.

*. See the next page for the members of the CUBIT Development Team.

Exceptional Service in the National Interest

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

CHAPTER

Cubit Development Team Membership

Sandia National Laboratories, Albuquerque New Mexico	
Ted D. Blacker	Computational Mechanics & Visualization
William J. Bohnhoff	Computational Mechanics & Visualization
Tony L. Edwards	Advanced Engineering & Manufacturing Software
James R. Hipp	Computational Mechanics & Visualization
Randy R. Lober	Advanced Engineering & Manufacturing Software
Scott A. Mitchell	Applied & Numerical Mathematics
Gregory D. Sjaardema	Solid & Structural Mechanics
Timothy J. Tautges	Computational Mechanics & Visualization
Tammy J. Wilson	Computational Mechanics & Visualization
Los Alamos National Laboratories, Los Alamos, New Mexico	
William R. Oakes	Technology Modeling and Analysis Group
Brigham Young University, Salt Lake City, Utah	
Steve Benzley	Civil Engineering Department
Jan C. Clements	
Leandro Lopez-Buriek	
Scott Parker	
Mark Whitely	
David White	
Consultants	
Eric Trimble	Consultant, Denver, Colorado

▼ Table of Contents

▼ Cubit Development Team Membership	4
▼ Table of Contents	5
▼ List of Figures	11
▼ List of Tables	13
 Chapter 1: Getting Started	 15
▼ How to Use This Manual	15
▼ CUBIT Mailing List	16
▼ Problem Reports and Enhancement Requests	16
▼ Executing CUBIT	17
Execution Command Syntax	17
Initialization File	18
User Environment Settings	18
Graphics Customization	19
Command Preprocessing	19
▼ Command Syntax	20
▼ Features	21
Geometry Creation	22
Geometry Consolidation	22
Supported Element Types	22
Mesh Creation	22
Boundary Condition Application	22
Graphical Display Capabilities	23
Hardware Platforms	23
▼ Future Releases	23
 Chapter 2: Tutorial	 25
▼ The Tutorial	25
▼ Step 1: Beginning Execution	27
▼ Step 2: Creating the Brick	28
▼ Step 3: Create the Cylinder	30
▼ Step 4: Adjusting the Graphics Display	31
▼ Step 5: Forming the Hole	34
▼ Step 6: Setting Body Interval Size	36
▼ Step 7: Setting Specific Surface Intervals	38
▼ Step 8: Setting Specific Curve Intervals	40
▼ Step 9: Surface Meshing	41
▼ Step 10: Volume Meshing	43

Table of Contents

▼ Congratulations!	47
Chapter 3: Environment	49
▼ Interface Choices	49
Overview	49
Command Line Version	49
Batch Interface	50
Graphical User Interface	51
GUI Advantages	51
GUI Design and Terminology	51
Main GUI Window	53
The Menu Bar	53
Command Line Scroll Window	54
Command Line Text Field	54
Commonly Used Buttons	54
Picker Window	54
Picker Button	55
Console Window	55
▼ Session Control	56
General Execution Commands	56
▼ Journal Files	56
CUBIT Journal File Generation	57
Replaying Journal Files	58
▼ Graphics	58
Graphics Window Control	59
Image Rendering Control	59
Viewing the Model	61
Controlling the Set of Displayed Entities	64
Setting Visibility	64
Global Settings	64
Individual Geometric Entity Settings	65
Immediate Mode Drawing	66
Color	66
Entity Labeling	67
Hardcopy Output	68
Video Animations	69
▼ Model Interrogation	70
▼ Help Facility	74
Chapter 4: Geometry	75
▼ Geometry Definition	75
Geometric Topology	75
Vertex	75

Curve	76
Surface	76
Volume	76
Body	76
Cellular Topology	76
▼ Geometry Creation	77
Geometry Primitives	77
Brick	78
Cylinder	78
Prism	79
Frustum	80
Pyramid	81
Sphere	81
Torus	81
Sketchpad Geometry	82
Sketch Overview	83
Polygonal Outline	84
Outline Refinement	84
Importing Geometry	84
Importing ACIS Files	84
ACIS Test Harness	84
PRO/Engineer	85
FASTQ	85
▼ Geometry Manipulation	85
Transform Operations	86
Copy	86
Move	87
Scale	87
Rotate	88
Reflect	88
Restore	88
Boolean Operations	89
Intersect	89
Subtract	89
Unite	89
▼ Geometry Decomposition	91
Web Cutting	91
▼ Geometry Consolidation	92
General Geometry Consolidation	93
Selective Geometry Consolidation	94
Chapter 5: Mesh Generation	95
▼ Mesh Definition	95
Mesh Hierarchy	95

Table of Contents

Node	95
Edge	96
Face	96
Brick	96
Mesh Generation	96
▼ Mesh Attributes	96
Meshing Schemes	96
Interval Specification	97
Element Types	97
▼ Curve Meshing	98
Node Density	98
Relative Element Edge Lengths	99
Curvature Based Node Placement	100
Field Function Based Node Placement	100
Meshing the Curve	100
▼ Surface Meshing	101
Scheme Designation	101
Mapping	101
Paving	102
Primitives	102
Triangle Primitive	102
Curvature Based Surface Meshing	103
Field Function Based Surface Meshing	104
Boundary Layer Tool	105
Meshing the Surface	106
▼ Volume Meshing	107
Scheme Designation	108
Mapping	108
Sweeping (Project, Translate, and Rotate)	108
Project	109
Translate	110
Rotate	111
Plastering	111
Whisker Weaving	111
Meshing the Volume	112
▼ Mesh Duplication	112
▼ Mesh Editing	112
Smoothing	112
Surface	112
Volume	113
Accessing Smooth Functions in the GUI	114
Node Repositioning	114
Mesh Deletion	115
Import Mesh	116

Chapter 6: Finite Element Model Definition and Output	117
▼ Finite Element Model Definition	117
Element Blocks	117
Nodesets	117
Sidesets	118
▼ Element Block Specification	118
Default Element Types, Block IDs, and Attributes	119
Element Block Definition Examples	119
Multiple Element Blocks	119
Surface Mesh Only	120
Two-Dimensional Mesh	120
▼ Boundary Conditions—Nodesets and Sidesets	120
▼ Setting the Title	121
▼ Exporting the Finite Element Model	121
 Appendix A: Command Index	 123
▼ Command Syntax	123
▼ Commands	123
 Appendix B: Examples	 137
▼ General Comments	137
▼ Simple Internal Geometry Generation	138
▼ Octant of Sphere	139
▼ Airfoil	141
▼ The Box Beam	142
▼ Thunderbird 3D Shell	145
▼ Assembly Components	148
 Appendix C: Fsqacs: A FASTQ to ACIS Command Interpreter	 153
▼ Description	153
▼ Program Execution	153
▼ Limitations	154
 Appendix D: CUBIT Installation	 155
▼ Licensing	155
▼ Distribution Contents	156

CHAPTER

▼ Installation	156
Appendix E: Available Colors	157
Appendix F: CUBIT Application Defaults File	159
References	161
Glossary	163
Index	167

▼ List of Figures

Figure 2-1	Geometry for Cube with Cylindrical Hole.....	26
Figure 2-2	Generated Mesh for Cube with Cylindrical Hole	26
Figure 3-1	MainWindow for Interaction within the GUI	53
Figure 3-2	Main Window Menu Items	54
Figure 3-3	Minimum Configuration of a Picking Window	55
Figure 3-4	Console Window.....	55
Figure 3-5	Main GUI Window Showing File Menu	56
Figure 3-6	Journal Record/Play Dialog Box	57
Figure 3-7	File Selection Dialog Box	58
Figure 3-8	Graphics Mode Dialog Box	60
Figure 3-9	Schematic of From, At, Up, and Perspective Angle	62
Figure 3-10	The Graphics View Dialog Box.....	62
Figure 3-11	Visibility Dialog Box.....	65
Figure 3-12	Graphics Draw Dialog Box.....	66
Figure 3-13	Color Dialog Box.....	67
Figure 3-14	Label Dialog Box.....	68
Figure 3-15	Hardcopy Output Dialog Box	69
Figure 3-16	List Window.....	70
Figure 4-1	Cellular Topology Between Volumes.....	76
Figure 4-2	Dangling Faces & Edges.....	77
Figure 4-3	CUBIT Geometry Primitives	78
Figure 4-4	Brick Creation Dialog Box	79
Figure 4-5	Cylinder Creation Dialog Box	79
Figure 4-6	Prism Creation Dialog Box.....	80
Figure 4-7	Frustum Creation Dialog Box.....	81
Figure 4-8	Pyramid Creation Dialog Box.....	82
Figure 4-9	Sphere Creation Dialog Box	82
Figure 4-10	Torus Creation Dialog Box.....	83
Figure 4-11	Sketch Creation Dialog Box	83
Figure 4-12	Body Copy Dialog Box.....	86
Figure 4-13	Body Move Dialog Box	87
Figure 4-14	Body Scale Dialog Box.....	87
Figure 4-15	Body Rotate Dialog Box.....	88
Figure 4-16	Body Reflect Dialog Box.....	89
Figure 4-17	Intersect Boolean Dialog Box	90
Figure 4-18	Subtract Boolean Dialog Box	90
Figure 4-19	Unite Boolean Dialog Box	90
Figure 4-20	Web Cutting Dialog	91
Figure 4-21	Solid Model Prior to Decomposition	93
Figure 4-22	Solid Model After Decomposition.....	93
Figure 4-23	Geometry Consolidation Dialog Box.....	94

CHAPTER

Figure 5-1	Curve Meshing With The GUI Mesh Dialog Box	99
Figure 5-2	Equal and biased curve meshing	101
Figure 5-3	Surface Meshing with the GUI Mesh Dialog	102
Figure 5-4	Mapped and paved surface meshing	103
Figure 5-5	Triangle primitive mesh	103
Figure 5-6	Plastic strain metric	104
Figure 5-7	Adaptively generated mesh	104
Figure 5-8	Boundary Layer Parameters	105
Figure 5-9	Boundary Layer Dialog Box	106
Figure 5-10	Volume Meshing with the GUI Mesh Dialog	107
Figure 5-11	Project Volume Meshing	109
Figure 5-12	Multiple Surface Project Volume Meshing	110
Figure 5-13	Plastering Examples	111
Figure 5-14	Smooth Surface and Smooth Volume Dialog Boxes	114
Figure 5-15	Mesh Delete and Mesh Delete Warning Dialog Boxes	115
Figure 6-1	Block Identifier Dialog	118
Figure 6-2	Nodeset and Sideset Dialogs	121
Figure B-1	Geometry for Cube with Cylindrical Hole	139
Figure B-2	Generated Mesh for Cube with Cylindrical Hole	140
Figure B-3	Generated Mesh for Octant of Sphere	141
Figure B-4	Airfoil mesh generated using the boundary layer tool and paving.	142
Figure B-5	Box Beam example	143
Figure B-6	Sandia Thunderbird 3D shell	145
Figure B-7	Components in electronics assembly package	148
Figure B-8	Generated mesh for the electronics assembly package	149

▼ List of Tables

Table 3-1	Command Line Interface Line Editing Keys	50
Table 5-1	Default Meshing Attributes	96
Table 5-2	Valid Meshing Schemes for Curves, Surfaces, and Volumes	97
Table B-1	CUBIT Features Exercised by Examples.	138
Table E-1	Available Colors	157

CHAPTER

Chapter 1: Getting Started

- ▼ How to Use This Manual...15
- ▼ CUBIT Mailing List...16
- ▼ Problem Reports and Enhancement Requests...16
- ▼ Executing CUBIT...17
- ▼ Command Syntax...20
- ▼ Features...21
- ▼ Future Releases...23

Welcome to CUBIT, the Sandia National Laboratory automated mesh generation environment. With CUBIT the geometry of a part can be imported, created, and/or modified using an embedded solid modelling engine. The geometry can then be discretized into a finite element mesh using a combination of techniques including paving [1], mapping, sweeping, and various other algorithms being developed. CUBIT also features boundary layer meshing specifically designed for fluid flow problems. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes.

▼ How to Use This Manual

This manual provides specific information about the commands and features of CUBIT. It is divided into chapters which roughly follow the process in which a finite element model is designed, from geometry creation to mesh generation to boundary condition application. An example is provided in a tutorial chapter to illustrate some of the capabilities and uses of CUBIT. Appendices containing complete command usages, examples, installation instructions, and a list of available colors are included.

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. As such, the code is constantly being updated and improved. Feedback from our users indicates that new meshing tools are often needed and/or desired before they have been completely tested and debugged. As a service to the user, these tools are integrated and made available as quickly as possible, but in a *user beware* state. As they are further tested (often with the assistance of users) and

improved, the state of the particular tool becomes more stable, and thus the risk to the user is lowered. Since documentation of the tool is necessary for actual use, we have included the documentation of all available tools in the manual. However, to warn the user, a “hammer” icon is placed in the document next to those features that are only minimally tested or are in a state of work-in-progress (See “hammer” icon in left margin). In other words, “proceed with caution.” Certain portions of this manual contain information that is vital to understand in order to run CUBIT effectively. In order to highlight these portions, a “key” icon is positioned in the document next to these sections. In other words, “this is a key point”.

This manual is Volume 1 of the CUBIT documentation set. The companion document is *CUBIT Mesh Generation Environment, Volume 2: Developers Manual* [3] which contains internal programming-related details of the CUBIT mesh generation environment.

This manual documents CUBIT Version 1.8.1, May 1, 1994.

▼ CUBIT Mailing List

A mailing list has been created to keep interested users informed of new features, bug-fixes, and other pertinent information about CUBIT. The list can also be used by users for general discussions about CUBIT. Users can subscribe to the mailing list by sending a mail message to **listserv@sahp046.jal.sandia.gov** with the body (not the subject) of the mail message containing the line:

subscribe cubit Your Full Name

The user would then receive a message confirming the subscription to the CUBIT mailing list. More information about the use of the mailing lists can be obtained by sending the message **help** to the above mail address. Messages are sent to the list by sending mail to the address:

cubit@sahp046.jal.sandia.gov

The CUBIT developers will be sending announcements of new CUBIT capabilities, enhancements, and user-visible bug fixes to this list on a regular basis. In addition, this list may be used for general questions regarding CUBIT that may be solvable by other subscribers to the list.

▼ Problem Reports and Enhancement Requests

The CUBIT project is using GNATS [17], the GNU Problem Report Management System, for tracking questions, problem reports, and enhancement requests. The GNATS system is designed to allow users who have problems with the CUBIT code, or who have requests for new features to be added to the CUBIT code, to submit reports of these problems or requests to the CUBIT developers. The GNATS system provides a program called *send-pr* which can be used to submit problem reports and enhancement requests in a standard, defined format which can be read by an electronically managed database which automatically notifies responsible parties of the existence of the problem. It also provides a mechanism for keeping everyone involved in the problem report informed of the current state of the problem.

In general, any editor and mailer can be used to submit valid Problem Reports (PR), as long as the format required by GNATS is preserved. However, *send-pr* automates the process and ensures that certain fields necessary for automatic processing are present. The use of *send-pr* is strongly recommended for all initial problem-oriented correspondence with the CUBIT developers. The user documentation for the GNATS system is supplied with the CUBIT program. On the Sandia Internal Restricted Network, the documentation is available from `sahp046.jal.sandia.gov:/usr/local/doc/gnats.ps`. On the Sandia External Open Network, the documentation is available from `sass577.endo.sandia.gov:/home/cubit/gnats.ps`. Contact your CUBIT code sponsor, or if this fails, Greg Sjaardema (`gdsjaar@sandia.gov`) if you cannot access either of these locations or if you have problems using the system.

Note: The existence and recommended use of an electronic bug reporting mechanism is not an attempt by the CUBIT developers to ignore and or discourage face-to-face discussion of problems with, or enhancements to the CUBIT code with users. The use of GNATS is intended to help the developers manage, prioritize, and track the tasks required to produce a usable "state-of-the-art" production mesh generation package.

▼ Executing CUBIT

Execution Command Syntax

Three versions of CUBIT are currently supported: 1) a Motif-based Graphical User Interface version, 2) a basic command line version which output graphics to a standard X Window System graphics window, and 3) a batch command line version with no graphics. The commands to execute these versions of CUBIT on most systems are simply:

cubit Graphical User Interface version.

cubitx Command line version with X Window system graphics.

cubitb Batch command line version, no graphics¹.

Throughout this manual, "CUBIT" will be used as a generic term that applies to all of the executables. If it is necessary to specify a specific version of CUBIT, one of above three names will be used.

The command syntax recognized by CUBIT is:

```
{cubit|cubitx|cubitb} [-help] [-solidmodel <val>] [-initfile <val>]
[-batch] [-noinitfile] [-nojournal] <input_file_list...>
```

where the quantities in square brackets **[-options]** are optional parameters that are used to modify the default behavior of CUBIT and the quantities in angle brackets **<values>** are values supplied to the option. The effect of these parameters are:

-help Print a short usage summary of the command syntax to the terminal and exit.

-initfile <val> Use the file specified by **<val>** as the initialization file instead of the default initialization file `$HOME/.cubit`.

¹ The `cubitb` executable may be eliminated in the future and its functionality duplicated using the `'-batch -nographics'` command line options to the `cubitx` executable.

-solidmodel <val> Read the ACIS solid model geometry information from the file specified by **<val>** prior to prompting for interactive input.

-batch Specify that there will be no interactive input in this execution of CUBIT. CUBIT will terminate after reading the initialization file, the geometry file, and the file specified by the **-input** option.

-noinitfile Do not read any initialization file. The default behavior is to read the initialization file **\$HOME/.cubit** if it exists.

-nojournal Do not create a journal file for this execution of CUBIT. This option performs the same function as the **Journal Off** command. The default behavior is to create a journal file.

<input_file_list...> All files specified on the command line following the last option are processed prior to prompting for interactive command input.

An example of the use of the command line options is:

```
cubitb -batch -nojournal final_mesh.jou
```

which specifies that **cubitb** will execute the commands in the file **final_mesh.jou** unattended. This mode is typically used to recreate a previously generated mesh with no user interaction.

The command options can also be specified through the **CUBIT_OPT** environment variable. See the "User Environment Settings" section below for more information.

Initialization File

If the file **\$HOME/.cubit** or the file specified by the optional **-initfile <val>** option exists when CUBIT begins executing, it is read prior to beginning interactive command input. This file is typically used to perform initialization commands that do not change from one execution to the next, such as turning off journal file output, setting geometric and mesh entity colors, and setting the size of the graphics window.

User Environment Settings

To execute CUBIT several environment variables must be set. In particular the "**HOME**", "**PATH**", "**HOOPS_PICTURE**" and "**DISPLAY**" variables. The **HOME** environment variable is typically set automatically when you login to a system. Its purpose is to provide a pointer to your login directory. The **PATH**, on a Unix system, is a list of directories that are searched for commands to be executed. Proper setting of the path is system-dependent; if CUBIT does not execute correctly, contact your system manager or another CUBIT user for the correct setting of the **PATH** specification.

The Motif-based GUI version of CUBIT (**cubit**) and the X Window System-based command line input version of CUBIT (**cubitx**) both require the specification of the **DISPLAY** and **HOOPS_PICTURE** environment variables which are used by the application to determine where the graphics window should be displayed (and which screen should be used on displays with multiple monitors). For multiple monitors, the environment variable **CUBITM_GUI** can be established to tell the system that the GUI window should be displayed on a different screen than the graphics window. C Shell users can set these environment variables by typing:

```
setenv HOOPS_PICTURE      my_display:0.1
setenv CUBITM_GUI        my_display:0.0
```

This will make the Graphics Window show up on display number 0.1 and the GUI windows show up on display number 0.0.

CUBIT also requires the environment variable **CUBIT_HELP_DIR** if the online hypertext help system will be used. This variable should be set to the pathname specifying where the CUBIT help file **cubitHelp.hlp** is located. The person responsible for installing CUBIT on the system should be contacted for this information.

Another useful environment variable is **CUBIT_OPT** which can be used to set execution command line parameter options. For example, if journalling of commands is never wanted, then setting **CUBIT_OPT** to **-nojournal** will turn off journalling for all CUBIT executions¹.

Graphics Customization

Settings for the default CUBIT window sizes, locations, colors, and fonts can be set in the **.Xdefaults** or **.Xresources** resource files in the user's home directory. This file is a text file that can be edited with any standard UNIX text editor. In the resource file, each resource must be on a separate line. The resource setting consists of a resource label, a colon, one or more spaces or tabs, and the resource value. For additional general information on resources, see X Window System documentation; a readily available documentation source is the RESOURCES section of the X(1) manual page which is usually installed on most systems.

A CUBIT resource label begins with the word "cubit" followed by an asterisk or period, followed by the specification of the resource. For example, to specify that the CUBIT graphics window should be 700 pixels square rather than the default size, the following line should be added to the resource file:

```
cubit*CUBIT.geometry: 700x700+445+0
```

Another file, similar to the resource file is the Application Defaults file which is used to customize CUBIT's graphical user interface. This file, called **CUBIT.ad** is distributed with the CUBIT executables. This file must be renamed to **CUBIT** and then installed in either **/usr/lib/X11/app-defaults** for system-wide defaults and/or in a user's home directory for per-user defaults. The contents of this file are reproduced in "CUBIT Application Defaults File" on page 159. The format of this file is the same as the resource file format.

For example:

```
cubit*XmTextField*background: LightBlue
```

This line states that the background of all text fields in the **cubit** application will be the color LightBlue. Colors can normally be found in the **/usr/lib/X11/rgb.txt** file on your system. For additional information about application default files, see the application defaults section in any X Window user's book.

Command Preprocessing

Many analysts use the Aprepro [13] program to preprocess commands and journal files which contain algebraic expressions. Future plans for CUBIT include the addition of an algebraic preprocessing capability directly in the CUBIT parsing code; however this capability does not

1. Journalling could then be turned back on with the "Record" command.

yet exist. Aprepro can however be used with **cubitx** and **cubittb** by “piping” CUBIT command line input through Aprepro using a command similar to¹:

```
aprepro --exit_on --interactive | cubitx
aprepro cubit_input_file | cubittb
```

▼ Command Syntax

The execution of CUBIT is controlled either by inputting commands from the command line (or in the command line field in the main window of the Motif-based version of CUBIT [See Figure 3-1 on page 53]), or through the use of the Graphical User Interface (GUI). The Graphical User Interface parses the user input and generates an equivalent command which is recorded in a journal file and actually executes the action requested. Throughout this document, each function or process will have a description of both the command required to perform the function or process and the steps required to perform the same function through the graphical user interface. In this section, the command syntax used in this manual will be described. Although knowledge of the command syntax is not necessary for the use of the graphical user interface version of CUBIT, it may be helpful to skim this section anyway since the journal file written by CUBIT uses these commands. The user can obtain a quick guide to proper command format by issuing the **<keyword> help** command. This help message will indicate the full command syntax expected by the keywords. For example, entering **view help** results in the following output:

```
View At <X_coord> <Y_coord> <Z_coord>
View From <X_coord> <Y_coord> <Z_coord>
View List
View Up <X_coord> <Y_coord> <Z_coord>
```

The words that begin with an uppercase letter are keywords which must be entered (case is not significant) and the bracketed words are user supplied parameters.

The commands recognized by CUBIT are free-format and must adhere to the following syntax rules.

- Either lowercase or uppercase letters are acceptable.
- The “#” character in any command line begins a comment. The “#” and any characters following it on the same line are ignored.

Each command typically has either:

- an action keyword or “verb” followed by a variable number of parameters, for example

```
Mesh Volume 1
```

- or a selector keyword or “noun” followed by a “verb” or “selector keyword and a variable number of parameters, for example

```
Volume 1 Scheme Project Source 1 Target 2
```

The action or selector keyword is a character string matching one of the valid commands. It may be abbreviated as long as enough characters are used to distinguish it from other commands. The meaning and type of the parameters depend on the keyword. Valid entries for parameters are:

1. The double dashes are required for the Aprepro options.

- A numeric parameter may be a real number or an integer. A real number may be in any legal C or FORTRAN numeric format (for example, 1, 0.2, -1e-2). An integer parameter may be in any legal decimal integer format (for example, 1, 100, 1000, but not 1.5, 1.0, 0x1F).
- A string parameter is a literal character string contained within single quotes. For example, **'This is a string'**.
- A filename parameter must specify a legal filename on the system that CUBIT is running. Environment variables and aliases may not be used in the filename specification. For example, the C-Shell shorthand of referring to a file relative to the user's login directory (**~jdoe/cubit/mesh.jou**) is not valid. The filename must be specified using either a relative path (**../cubit/mesh.jou**), or a fully-qualified path (**/home/jdoe/cubit/mesh.jou**). Like a string, it also must be contained within single quotes.
- Several commands permit a range of values. A range is one of the following forms:
 - **"n1"** selects a single value **n1**,
 - **"n1 to n2"** selects all values from **n1** to **n2**. The value **n2** must be greater than **n1**,
 - **"n1 to n2 by n3"** selects all values from **n1** to **n2** stepping by **n3**, where **n3** may be positive or negative.
 The keywords **"through"** and **"thru"** may be used instead of **"to,"** and **"step"** may be used instead of **"by."**
- Some commands require a **"toggle"** keyword to enable or disable a setting or option. Valid toggle keywords are **"on"**, **"yes"**, and **"true"** to enable the option; and **"off"**, **"no"**, and **"false"** to disable the option.

The notation conventions used in the command descriptions in this document are:

- The command will be shown in a format that **looks like this**.
- A word enclosed in angle brackets (**<parameter>**) signifies a user-specified value. The value can be an integer, a range of integers, a real number, or a string. The valid value types should be evident from the command or the command description.
- A series of words in braces and delimited by a vertical bar (**{choice1 | choice2 | choice3}**) signifies that one of the words within the brackets must be entered.
- A word enclosed in square brackets (**[optional]**) signifies an optional parameter which can be entered to modify the default behavior of the command, but is not required.

An example of this command syntax is shown below.

{volume surface curve} <range> size <size>	
volume 1 size 0.5	Valid
surface 1 to 10 by 3 size 0.05	Valid
volume 10 to 1 size 0.05	Invalid — negative increment
volume 10 to 1 by -1 size 0.05	Valid
surface 1 10 size 1.0	Invalid — not a valid range specification
surface 1 to 10 interval 5.0	Invalid — "interval" requires an integer

▼ Features

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. CUBIT is based on a solid

CHAPTER 1

Getting Started

modeler that provides it with a precise geometric representation. The *paving* algorithm [1] has been extended to mesh complex three dimensional surfaces based on the solid modeler. Volumetric meshing is provided by mapping transformations and sweeping algorithms. Multiple user interfaces are supported as are several quadrilateral and hexahedral element types. The following sections provide a brief overview of the CUBIT meshing toolkit.

Geometry Creation

Geometry creation is accomplished using the geometric primitives and boolean operations in CUBIT or by reading an external solid model file into the CUBIT meshing toolkit. External solid model files can be created from any of several environments that support the ACIS[®] solid model format: a rudimentary command line system, referred to as the "test harness [4]," is useful for building quick and straightforward models. Other more advanced environments include the Aries[®] ConceptStation and PRO/Engineer via a PRO/Engineer/ACIS translator. A specialized software translator has been designed to translate sheet solid models from FASTQ [5] input files into an ACIS format, which can be further modified using the ACIS test harness. The resulting ACIS models can then be imported into CUBIT and meshed.

Geometry Consolidation

When assembly solid models are imported into the CUBIT environment, many surface, curve, and vertex entities will be redundant. To resolve this issue, the automated geometry consolidation or "merge" routines will identify matching entities and make database modifications to remove the redundancy. Geometry consolidation can also be interactive, in case certain redundant features need to be retained to represent slide surfaces or slide lines. The geometry merge capability eliminates the generation of non-contiguous elements between adjacent surfaces and curves which would have to be removed after meshing.

Supported Element Types

Element types supported in CUBIT include 2 and 3 node bars and beams; 4, 8, and 9 node quads; 4, 8, and 9 node shells; and 8, 20, and 27 node hex elements. Element types must be set before mesh generation is initiated.

Mesh Creation

Mesh generation in CUBIT is designed to be highly automated although numerous control mechanisms are provided to allow the user to guide the meshing process. Meshing is controlled through scheme choice, and interval number or node density specification. Curve meshing schemes include equally spaced and biased spaced intervals. Surface meshing schemes include mapping transformations, paving, boundary layers, and primitives. Volume meshing schemes include mapping transformations and mesh sweeping or projecting. Other automated volume meshing algorithms are being added.

Boundary Condition Application

Once a suitable mesh has been generated, elements can be grouped into sets using three control classes: element blocks, nodesets, and sidesets. Numeric flags are associated with these sets allowing analysis codes to apply appropriate boundary conditions to the correct mesh entities.

Element blocks are used for efficient storage of a finite element model. Within an element block, all elements are of the same type (basic geometry and number of nodes) and have the same material definition.

Nodesets provide a means to reference a group of nodes with a single identification number rather than by each node's identification number. Nodesets are typically used to specify load or boundary conditions, or to identify a set of nodes for a special processing within CUBIT. A node may appear in multiple nodesets, but will only appear once in any single nodeset.

Sidesets provide an additional means of applying load and boundary conditions. Unlike nodesets, sidesets group sides or faces of elements rather than simply a list of nodes. For example, a pressure load must be associated with elements rather than nodes to apply it properly.

Nodes, element edges, and element faces can belong to multiple nodesets and sidesets. Nodesets and sidesets can be individually displayed for visual inspection. See reference [6] for more information.

Graphical Display Capabilities

CUBIT uses the Hoops graphic display environment to render images. CUBIT can display a wireframe, hiddenline, or shaded representation of geometric and mesh entities. CUBIT can also generate a PostScript file of any displayed image (see "Hardcopy Output" on page 68). Complete control over the viewport parameters and the zoom magnification provide the user with an intuitive modeling environment.

When operating the GUI, users can perform screen picking and point-and-click zoom operations. All of the user-defined options are represented inside option windows by colored status buttons. This gives the user an easy to read description of the current settings. The GUI also gives the user control over the display of the geometry and mesh.

Hardware Platforms

CUBIT is written in "standard" C++ and should execute on any Unix operating system. To date, it has been compiled and used on Sun, Hewlett-Packard, and Silicon Graphics workstations.

▼ Future Releases

CUBIT is currently on a 4-month major release cycle. The capabilities of CUBIT will be expanded and enhanced on a regular basis as dictated by user needs and the developmental progress of new meshing algorithms. Areas of concentration will include

- full-featured automatic hexahedral meshing using a combination of *plastering* and *whisker-weaving*,
- quadrilateral and hexahedral adaptivity,
- enhanced geometric functionality such as overlapping geometry consolidation and more robust geometry decomposition.
- improved control of the naming of geometric and mesh entities through the use of persistent identification numbers.
- improved internal geometry generation through the use of an interactive sketch pad.
- improved usability, robustness, and functionality.

CHAPTER 1 Getting Started

Extension of the CUBIT environment to new platforms will also be pursued according to user needs.

Chapter 2: Tutorial

▼ The Tutorial...	25
▼ Step 1: Beginning Execution...	27
▼ Step 2: Creating the Brick...	28
▼ Step 3: Create the Cylinder...	30
▼ Step 4: Adjusting the Graphics Display...	31
▼ Step 5: Forming the Hole...	34
▼ Step 6: Setting Body Interval Size...	36
▼ Step 7: Setting Specific Surface Intervals...	38
▼ Step 8: Setting Specific Curve Intervals...	40
▼ Step 9: Surface Meshing...	41
▼ Step 10: Volume Meshing...	43
▼ Congratulations!...	47

The purpose of this chapter is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a brief tutorial on the use of the software package. This chapter is designed to demonstrate step-by-step instructions on generating a simple mesh on a perforated block.

▼ The Tutorial

The following is a sample of the basics of using CUBIT to generate and mesh a geometry. By following this tutorial, you will become familiar with the Graphical User Interface (GUI) window environment and with as much of the CUBIT environment as possible without stopping for detailed explanations. All the commands introduced in this tutorial are thoroughly documented in subsequent chapters. Here are a few tips in following the example in the tutorial

- Focus on instructions preceded with “Step” numbers. These step you through a series of explicit activities that describe exactly what to do to complete the task.
- Refer to screen shots and other pictures that show you what you should see on your own display as you progress through the tutorial.

- The equivalent command line will be shown with each step-by-step instruction. These commands can be used in the "command line" version to replicate the action depicted through the GUI window interface. An example of the command line is shown below:

Command: This is a Command Line

This simple example demonstrates the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. The **brick**, **cylinder** and **subtract** commands are used to create solid model geometry with primitives and boolean operations. The block is then meshed using paving and translation. The geometry to be generated is shown in Figure 2-1. This figure also shows the curve

Figure 2-1 Geometry for Cube with Cylindrical Hole

and surface ids of the geometry. These id's are used in the command lines shown with each step. The final meshed body is shown in Figure 2-2 and also at the end of this chapter.

Figure 2-2 Generated Mesh for Cube with Cylindrical Hole

▼ Step 1: Beginning Execution

- Type "cubit" to begin execution of CUBIT. If you have not yet installed CUBIT, see instructions for doing so in the "CUBIT Installation" Appendix. A CUBIT console window will appear which tells the user which version is being run and the most recent revision date. (See the following screen shot for example of window). This window relays information about the success/failure of attempted actions.

- Next you will see the main GUI window (shown below) which contains four parts listed from top to bottom:.

Main Graphical User Interface (GUI) Window

- 1) The menu bar across the top of the window. These menus are used to execute actions in CUBIT
- 2) The command line scroll window where equivalent commands are listed.
- 3) The command line text field where the user may type commands
- 4) Commonly used buttons for control of the graphics window

▼ Step 2: Creating the Brick

Now you may begin generating the geometry with the creation of a brick. From the main menu:

- Pull down the **Geometry** menu

- Choose **Primitives** which will then bring up the primitive creation window.

Primitive Creation Window

- Choose "**Brick**" to specify that a cuboid is to be generated
- Specify "10" in each of the **Width [x]**, **Height [z]**, and **Depth [y]** options (you may double click in these boxes to specify numbers). If the brick is a true cube, only the width needs to be specified--the depth and height options will default as indicated by the (**opt**) label.
- Click on **Apply** to form the cube. The cube will appear in the CUBIT display window (see example below).

Brick Display

Command: Brick Width 10. Depth 10. Height 10

▼ Step 3: Create the Cylinder

Now you must form the cylinder which will be used to cut the hole from the brick:

- Choose **Cylinder** from the primitive creation window
- Specify **Height [Z]** as “12” (slightly larger than the cube)
- Specify **Radius[X/Y]** as “3”

Note: You may double-click over the existing numbers that may appear in the option box and over-ride with the numbers you type.

- Click on **Apply** to form the cylinder. At this point you will see a dimensional cube with a cylinder appear in the CUBIT display window. You will also see each command that you have specified thus far as it appears in the command line scroll section of the Main Menu Window.

Brick with Cylinder Display

Command: Cylinder Height 12. Radius 3.

▼ Step 4: Adjusting the Graphics Display

The picture on the graphics display can now be adjusted to verify that what you expected to happen has indeed occurred:

- From the Main window, pull down **Graphics** menu
- Choose **View**. A Viewing Options Window will appear.

Main Window Graphics Option

CHAPTER 2 Tutorial

- Under “View From” specify **X** as “3”, **Y** as “4”, and **Z** as “5” as shown below. This tells CUBIT that the viewing location is along the vector (3,4,5).

View At	View From	View Up-Vector
0.000000 X	3.000000 X	-0.291044 X
0.000000 Y	4.000000 Y	0.624625 Y
0.000000 Z	5.000000 Z	-0.465073 Z

VIEW ROTATION	Pin	VIEWPORT
<div><div>- + X</div><div>- + Y</div><div>- + Z</div></div> <div>Increment <input type="text"/></div>	<div><div>▲</div><div>◀ ▶</div><div>▼</div></div> <div>Increment <input type="text"/></div>	<div>VIEWPORT</div> <div>RESET</div>

APPLY and UPDATE	Get Current Values	CANCEL
------------------	--------------------	--------

- Click **Apply and Update**. This will change the display box to reflect a three-dimensional wireframe brick with cylinder. The word “Display” will appear in the Command Line Scroll Window each time **UPDATE DISPLAY** or **APPLY and UPDATE** are clicked.

Command: From 3 4 5

Note: In the display, the wireframe picture shows the relative locations of the bodies. Turning the image to smooth shaded (as will be described in following steps) improves the perspective.

Note: At this point you may rotate the model to view its validity. To rotate the model, select the **View** menu item from the **Graphics** main menu. Add a degree variable in the **Increment** option under **View Rotation** and click on the plus (+) or minus (-) X, Y, or Z buttons to watch the display rotate. HINT: If you double-click the plus or minus boxes twice, the display will continually rotate. Click again to stop continual rotation. See examples below for various rotations of the geometrics created with the preceding commands.

▼ Step 5: Forming the Hole

Now the cylinder can be subtracted from the brick to form the hole in the block. From the main window:

- Pull down the **Geometry** menu
- Choose **Booleans**. The boolean window will appear. The **subtract** command subtracts one body from another.

Main Menu Geometry Option

- Click the **Subtract** option
- Click in **Body1 Name**, specify "2" to indicate the body to be subtracted (the cylinder)

Boolean Window - Subtract Option

- Click In **Body2 Name**, specify "1" to indicate the body from which it is to be subtracted (the brick)
- Click **APPLY** to perform the subtraction. This will update the display box to reflect the removal of the cylinder from the brick.

Note: Note that both original bodies are deleted in the boolean operation and replaced with a new body (3) which is the result of the boolean.

Command: Subtract 2 From 1

Command: Display

▼ Step 6: Setting Body Interval Size

The next step is to generate the surface mesh on one of the surfaces to be swept.

- Pull down the **Mesh** option

- Choose **Generate**. The mesh generation dialog box will appear. From this dialog box, the number of increments along edges and the schemes, and the actual meshing can be performed.
- The number of increments must be set first in order to mesh any geometry. This is done first for the entire body by specifying a desired size of increment. The **Geometry Type** should remain as "**Body**", change **Interval** to "**Size**", and set the **Value** to **1.0**.
- The identifiers of the entities (in this case bodies) on which to operate are specified at the bottom of the dialog box, under **Current Selection(s)**. In this case there is only one body (it will be body "3"). Click the **Get All** button to select the body. This will automatically place a "3" in the selection box.

- Click **Apply** to issue the command for interval sizing on the body. This size is propagated downward to all the volumes, surfaces, and curves in the body.

Mesh Generation Dialog Box

Geometry Type:

Interval:

Value (real)

Scheme:

- ☒ Blas
- ☒ Equal
- ☒ Max
- ☒ Pave
- ☒ Flaster
- ☒ Sweep
- ☒ Rotate
- ☒ Translate
- ☒ Triangle

☒ Center
☒ End

Growth Factor

Source ID

Target ID

Current Selection(s):

Enter a selection: To

Command: body 3 interval size 1.0

▼ Step 7: Setting Specific Surface Intervals

The cylindrical surface (the inside of the hole) must be mapped in order for the sweeping type tools to work. Since this surface is periodic (contains no edge along the side of the cylinder) the mapping algorithm uses the surface interval setting to determine how many elements are to be mapped along the axis of the cylinder. This is done using the mesh generation dialog box already activated

- Change **Geometry Type** to "Surface".
- Change **Interval** to "Number", and the **Value (Integer)** option should be "10" specifying the use of 10 intervals for the length of the cylinder.

The screenshot shows the CUBIT mesh generation dialog box with the following settings:

- Geometry Type:** Surface
- Interval:** Number
- Value (Integer):** 10
- Scheme:**
 - ☒ Bias
 - ☒ Equal
 - ☒ Map
 - ☒ Pave
 - ☒ Plaster
 - ☒ Sweep
 - ☒ Rotate
 - ☒ Translate
 - ☒ Triangle
- Center** (checked)
- Growth Factor**
- Exit** (checked)
- Source ID** (empty)
- Target ID** (empty)
- Current Selection(s):**
 - Clear All**
 - Get All**
 - Mesh Now**
 - Rev. Bias**
- Enter a selection:** 10 **To** (empty)
- Buttons:** APPLY, Pick, HELP, CANCEL

- The surface must now be identified. Since we only want a specific surface to have this number of intervals, we cannot use the **Get All** button. One way to identify the surface desired is to turn the labels for surfaces "on". To do this pull down the **Graphics** menu from the Main Window, choose **Label**. The labelling dialog box will appear.

- Choose **Surface** and then push the **Update Display** button from the Main Window.

Command: label surface on

Command: display

- The labels will now be shown for the surfaces. The labels are positioned in the center of the geometric bounding box of the surface. From the display it is evident that the cylindrical surface is surface 10 (see previous page for display)
- Now return to the mesh generation window. In the **Enter a Selection** box, enter "10" and then click the **Apply** button to set these intervals.

Command: surface 10 interval 10

▼ Step 8: Setting Specific Curve Intervals

The surface interval command also propagates downward to the edges. In this case we want the cylindrical ends of the surface to have not 10 intervals, but 20 intervals. This is accomplished using the mesh control dialog box again.

- From the mesh generation dialog box already activated, change **Geometry Type** to "Curve", **Interval** to "Number", **Value (integer)** to "20"
- Go back to the label window (from the Main Menu, pull down the **Graphics** option and choose **label** again) and click the **Surface** button to turn "off" surface labels, and click the **Curve** button to turn "on". This will specify the curves to be applied to the intervals. Then push the **Update Display** button from the **Main Menu** window to get the following picture

Note: From this picture it is evident that curves 15 and 16 are the correct ID's.

Command: Label curve on

- Curves 15 and 16 can be entered in the **Enter a Selection** box. Place a "15" to "16" in the **Enter a Selection** option and then push the **Apply** button.

Geometry Type:

Interval:

Value (Integer):

Scheme:

- ☒ Bias
- ☒ Equal
- ☒ Map
- ☒ Pave
- ☒ Plaster
- ☒ Sweep
- ☒ Rotate
- ☒ Translate
- ☒ Triangle

Growth Factor:

Source ID:

Target ID:

Current Selection(s):

Enter a selection: To

Command: curve 15 to 16 interval 20

▼ Step 9: Surface Meshing

Now all necessary intervals have been set, and the meshing can proceed. First begin by meshing the front surface (with the hole) using the paving algorithm. The mesh control dialog box is used to accomplish this.

- Change **Geometry Type** to **Surface** from the Mesh Generation dialog box already activated
- Go to **Scheme** and click **Pave**

CHAPTER 2 Tutorial

- Now use the "picking" to choose the surface. Click the **Pick** button and then click in the center of the hole of the cube in the display window (the center of the front surface). This will highlight the surface and enter the number "11" automatically into the selection box.

Geometry Type:	Surface	
Interval:	Number	
Value (Integer)	10	
Scheme:	<div><div>☒ Bias ☒ Equal ☒ Map ☒ Pave ☒ Plaster ☒ Sweep ☒ Rotate ☒ Translate ☒ Triangle</div><div><div>☒ Center ☒ EBN</div><div>Growth Factor</div><div>Source ID <input type="text"/></div><div>Target ID <input type="text"/></div></div></div>	
Current Selection(s):	<div><div>Clear All</div><div>Get All</div><div> </div></div>	<div><div>Mesh Now</div><div>Rev. Bias</div></div>
Enter a selection:	<input type="text" value="11"/>	To <input type="text"/>
<div><div>APPLY</div><div>Pick</div><div>HELP</div><div>CANCEL</div></div>		

- Click the **Mesh Now** button which will apply the current settings (set the scheme to pave) and then mesh the surface. Pushing the **Update Display** button at the Main Menu window will produce the following picture:

Command: surface 11 scheme pave

Command: mesh surface 11

▼ Step 10: Volume Meshing

The volume mesh can now be generated again using the mesh control dialog box.

- From the Mesh Generation dialog box already activated, change **Geometry Type** to **Volume**.
- Under **Scheme**, click **Translate** to indicate that a translated surface mesh is to be used to generate the volume mesh.
- Go to the **Source ID** box and click the arrow beside the box to indicate that picking is desired. Click on the center of the meshed surface in the display window as before. The number **11** should appear in the box. This indicates that this surface will be the start of the translation.

CHAPTER 2 Tutorial

- Go to **Target ID** and click the arrow beside the box. With this “pick arrow”, go to the display and choose the center of the back circle. The number “12” should appear in the box. This indicates that this surface will be the conclusion of the translation.
- Go to **Enter a Selection** and click **Get All** since there is only one volume. The number 3 should appear in the box.
- Now click **Mesh Now** to apply the current settings (the translate scheme) and then generate the volume mesh.

The screenshot shows the CUBIT software interface with the following elements:

- Geometry Type:** A dropdown menu set to "Volume".
- Interval:** A dropdown menu set to "Number".
- Value (integer):** An empty text input box.
- Scheme:** A list of meshing schemes with checkboxes:
 - Blas (checked)
 - Equi (checked)
 - Map (checked)
 - Prva (checked)
 - Plaster (checked)
 - Sweep (checked)
 - Rotate (checked)
 - Translate (checked)
 - Triangle (checked)
- Growth Factor:** A text input box.
- Center:** A checkbox (checked).
- End:** A checkbox (checked).
- Source ID:** A text input box containing "11".
- Target ID:** A text input box containing "12".
- Current Selection(s):** A list box showing "1".
- Clear All:** A button.
- Get All:** A button.
- Mesh Now:** A button.
- Rev. Blas:** A button.
- Enter a selection:** A text input box containing "3".
- To:** A text input box.
- APPLY:** A button.
- Pick:** A button.
- HELP:** A button.
- CANCEL:** A button.

The final meshed body will then appear in the display window

Command: volume 3 scheme translate source 11 target 12

Command: mesh volume 3

Note: The type, quality, and speed of the rendered image can be controlled in CUBIT using several graphics mode types and rendering options. The **Graphics** options from the main menu, **Mode** option dialog box is used to set these options in the GUI version of CUBIT. The graphics mode type is set by using the **Graphics Mode Type** option pull-down menu. The Viewing Options dialog box (shown below) displays examples of "Wireframe", "Hiddenline", and "Smoothshade". (See Chapter 3, Environment, "Image Rendering Control" and "Viewing the Model" for detailed information)

Graphics Mode Type dialog box

1. Wireframe Display

2. Hiddenline Display

3. Smoothshade Display

▼ Congratulations!

You have created your first CUBIT mesh. The following chapters contain more detailed information about using CUBIT and an in depth description of the meshing algorithms available.

CHAPTER 2 Tutorial

Chapter 3: Environment

- ▼ Interface Choices...49
- ▼ Session Control...56
- ▼ Journal Files...56
- ▼ Graphics...58
- ▼ Model Interrogation...70
- ▼ Help Facility...74

The CUBIT user interface is designed to fulfill multiple meshing needs throughout the analysis process. The user interface options include a GUI based environment, a traditional command line interface, and batch mode operation. This chapter covers the interface options as well as the use of journal files, control of the graphics, a description of ways to interrogate the model for specific information, and an overview of the help facility.

▼ Interface Choices

Overview

The user interface options for CUBIT are: 1) command line, 2) batch mode, and 3) a graphical user interface (GUI). The command line version requires the user to type commands in order to interact with CUBIT. The GUI allows a user to interact with CUBIT by pressing buttons and selecting menu choices, as well as entering commands using a more traditional command line. The batch mode version allows a CUBIT process to be run in the background. All commands are stored in a journal file regardless of which version is used. These commands may then be replayed in the command line, batch mode or GUI versions.

For information on the commands and options used to execute CUBIT, see "Execution Command Syntax" on page 17.

Command Line Version

The command line interface provides the user access to all CUBIT commands via keyboard entry. When the command line version is executed, the command prompt (CUBIT>) appears in the UNIX shell window or terminal. A graphics window pops up once a display-related command is executed.

The CUBIT command line interface allows the execution of any command at any time. In contrast, the GUI presents context driven hierarchical menus. The absence of a strict hierarchy enables the command parser to recognize any command entered at the CUBIT prompt. Commands may be abbreviated as long as they remain unique from other commands.

The command line interface provides an EMACS-style line editing input package with command history¹. It allows the user to edit the current line and move through the history list of lines previously typed. Commands replayed from a journal file are not saved in the history list. The available editing commands are defined in Table 3-1.

Table 3-1 Command Line Interface Line Editing Keys

Key ^a	Function
^A, ^E	Move to beginning or end of line, respectively
^F, ^B	Move forward or backward one position in the current line.
^D	Delete the character under the cursor. Sends end-of-file if no characters on the current line.
^H, DEL ^b	Delete the character to the left of the cursor.
^K	Delete from the current cursor position to the end of the line
^P, ^N	Move to the previous or next line in the history buffer.
^L	Redraw the current line.
^U	Delete the entire line.
NL, CR ^c	Places current input on the history list, appends a newline and returns that line to the CUBIT program for parsing.

- The notation ^X refers to holding down the control key and then typing the letter X. Case is not significant.
- See the documentation for your keyboard/workstation to determine which key sends the DEL character.
- NL is a newline, typically ^J, CR is a carriage return entered the normal way you end a line of text.

Batch Interface

The CUBIT environment is available as a separate executable, typically called **cubitb**, that contains no graphical display capability. This implementation will operate as any other version of CUBIT, but is intended to perform unattended mesh generation. The batch implementation of CUBIT is invoked by entering '**cubitb -batch <journal_file>**' at the UNIX prompt². To initiate unattended operation, a journal file playback must be started. The journal file should

- The command line interface package used in CUBIT is Copyright 1991 by Chris Thewalt. The following copyright notice appears in the source code: "Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notices appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty".
- See "Executing CUBIT" on page 17 for more information.

contain the **Export** command to enable CUBIT to write out the model to a Genesis database file. Any graphics commands issued during a batch run are ignored.

Graphical User Interface

The graphical user interface (GUI) allows a user to interact with CUBIT using the mouse and a series of menus rather than strictly through typing commands. The GUI was designed to reduce the learning curve for the new user and provide some functionality that the command line version of CUBIT does not offer. There is still a command line available in the GUI version for those users who wish to continue using the command line but still take advantage of some of the extended GUI features. It is also important to note that new commands can be implemented into CUBIT faster than they can be implemented into the GUI. Thus some commands may only be available from the command line. Also, as the user becomes proficient with the available commands in the CUBIT environment, it may become more efficient to enter some commands in the command line rather than through the use of the GUI windows.

This section will discuss advantages of the GUI interface, review commonly used terminology, describe the main GUI window and its functionality, overview a typical picking window, and briefly describe the console window where output from CUBIT is written.

GUI Advantages

The GUI implementation provides the following advantages:

- **Screen picking.** The screen picking capability allows the user to pick the geometry for a command by using the mouse. Selected geometry will be temporarily highlighted.
- **Mouse Controlled Zooms.** The Zoom and ViewPort commands are implemented by using the mouse to draw a rectangular region on the screen to determine the X and Y coordinates for the commands. This implementation of the Zoom command makes viewing small entities much more efficient than a command line implementation.
- **Command Recall.** As the user applies commands through the GUI, the actual commands sent to the CUBIT parser will be displayed in a list window in the center of the main GUI window (See Figure 3-1). This also allows commands executed in the current CUBIT session to be scrolled back and re-executed. The command editing capability explained in "Command Line Version" on page 49 is currently not available in the GUI command window.

GUI Design and Terminology

The graphical user interface has been designed to abide by the commercial Open Software Foundation (OSF)/Motif standard. This section assumes a familiarity with the X Window system. For a detailed description of OSF/Motif, please refer to Reference [16]. Users will find that the buttons, toggles, menus, etc. in CUBIT operate similar to those offered by many of today's common applications using Motif graphical user interfaces.

The following terms are used throughout this manual when a reference is made to a GUI window:

- **Button.** A button is a rectangular 3-dimensional object that appears to be raised. When you click on a button, a single action is usually performed.

- **Cascade Button.** Cascade Buttons are only present in menus. These buttons always have an arrow following the text description on the button. When the user moves the cursor over the arrow, another series of menus normally "pops up" or "cascades."
- **Click.** Click means to press and release the mouse button once. Unless otherwise specified, the mouse button to press will be Mouse Button 1 (this is usually the left-most button on a three button mouse).
- **Dialog Box.** A dialog box is a window used to acquire user input or to display a message.

Note: All Dialog Boxes have a button labeled **CANCEL** near the bottom of the window. This button is used to dismiss or get rid of the dialog box. This is the only method that should be used to dismiss a window. **DO NOT** select any buttons provided by the window manager for this process or by sending a "delete" or "kill" message to the window. Doing this will cause CUBIT to crash.

- **Drag.** Drag means to click and hold a mouse button and move the cursor across the screen.
- **Double-click.** Double-clicking means to quickly press and release the mouse button twice. The period of time between clicks to qualify a double-click is dependent upon user specified preferences established at the X Window system level.
- **File Selection Dialog Box.** File selection dialog boxes allow the user to browse through the directory tree of the users system, filter specific files and file types, and to explicitly input a filename. Click **OK** to accept the selection or **Cancel** to cancel the file selection process. The **Filter** button is used to refresh the list of files after the **Filter** text field is modified. A typical file selection dialog box is shown in Figure 3-7.
- **Menu Bar.** This is an object that is present in the main window. A menu bar contains multiple text descriptions of menus.
- **Menu Item.** A menu item is a button under one of the text descriptions of the menu bar or a series of buttons within an option menu. A menu item may be selected by either bringing up the menu and then clicking on a menu item or clicking and dragging the cursor to the desired menu item and then releasing the mouse button. Menu items usually "popup" dialog box windows that require further user interaction.
- **Option Menu.** An option menu consists of multiple menu items. To the user, an option menu looks like a button until it is clicked. Once an option menu is clicked, it reveals a series of menu items that may be selected. The current selection is then displayed once the user selects an item.
- **Radio Button.** A radio button is a diamond shaped button that is highlighted in a noticeable color if selected. A radio button is usually one of many buttons grouped together. Only one button may be active at a time. To activate a radio button, click the button once.
- **Scrolled List.** In CUBIT, a scrolled list usually contains selected geometry or mesh entities. The scrolled list will appear as only a depressed window until enough information is contained in the list to activate scroll bars on the sides of the list. The scroll bars appear as arrows. To delete an item from a scrolled list, double-click the item. The command list in the Main Window is an exception to this. See the Main Window description for details.
- **Text Field.** A text field is a rectangular object that has a depressed, 3-dimensional appearance. Click on a text field to activate it and then type the text required. Text in

CUBIT is usually an integer or a real value. In certain critical fields, the user-inputted text will be checked to see if it is valid input. If the input is not valid, the inputted character or number will not be displayed in the text field. You will not hear a “beep” since this can be annoying.

- **Toggle Button.** A toggle button is a rectangular shaped button that is highlighted in a noticeable color if selected. To activate a toggle button click once.

Main GUI Window

The GUI main window shown in Figure 3-1 contains four parts listed from top to bottom: 1) the menu bar across the top of the window, 2) the command line scroll window, 3) the command line text field, and 4) commonly used buttons for control of the graphics window.

Figure 3-1 MainWindow for Interaction within the GUI

• The Menu Bar

The menu bar at the top of this window reveals additional menu items when the user clicks a menu bar item. Once a menu item is selected, a dialog box prompts the user for more input. The menu items include:

- **File.** This item allows access to commands for importing geometry, playing back journal files, exporting Exodus files, and exiting CUBIT.
- **Geometry.** Geometry creation and manipulation is available through this item.
- **Mesh.** This menu item provides access to meshing commands and settings.
- **Constraints.** Boundary conditions are added through this menu with the use of sidesets and nodesets.
- **Graphics.** Control of the graphics window and display parameters is provided with this item.
- **Special.** This menu item contains interfaces for feature consolidation, journal file recording, and
- **Help.** On-line help is available through this menu item.

The items contained in each menu item are shown in Figure 3-2.

File	Geometry	Mesh	Graphics	Constraints	Special
Import	Primitives...	Color...	Generate...	Nodesets...	Geometry Consolidation...
Export	Sketch...	Visibility...	ElementBlock...	Sidesets...	List Info Window...
Reset	Transform...	Draw...	Smooth...		Journal Record/Play...
Playback...	Booleans...	Labels...	Delete...		
Record...	Sweep...	View...	Delete All		
Exit	Decompose	Mode...	Boundary Layer		
		Hardcopy...			

Figure 3-2 Main Window Menu Items

• Command Line Scroll Window

As the user applies commands through the GUI or through the command line text field, the actual commands sent to the CUBIT parser will be displayed in a list window in the middle of the main GUI window. This also allows commands executed in the current CUBIT session to be scrolled back and re-executed. A double-click of the item will execute that command. A single click will place the command in the command line text field, but will not execute the command.

• Command Line Text Field

The user may enter commands in the text field under the text label **Enter Command Here**. After the user enters the command, the command will be displayed in the scrolled list above the command line. As mentioned above, the user can recall commands to the command line text field by clicking on the item in the scrolled list. The command editing capability explained in "Command Line Version" on page 37 is currently not available in the GUI command window.

• Commonly Used Buttons

The buttons at the bottom of the main GUI window are

- **Clear Display** button which executes a **Clear** command.
- **Update Display** button which executes a **Display** command.
- **Zoom** button which allows a zoom window to be defined on the screen using the mouse. **Zoom Reset** button issues a **Zoom Reset** command which updates the zoom limits to a size appropriate to capture all currently defined entities.

Picker Window

The picker window is used as a part of many GUI windows, and is present at the bottom of most dialog boxes. Figure 3-3 shows the minimum configuration of a picker window. To pick a geometric entity using the picker window, first select a type of geometry to be picked using the **Geometry Type** option menu. Next, click the **Pick** button to activate picking for this dialog box. Then, using the mouse select the *centroid* of the desired geometry entity in the Graphics Window with the left mouse button. The entity selected will be highlighted and the ID of the entity will be written in the scrolled window beneath the **Current Selection** label. To select all entities of the selected geometry type click the

Figure 3-3 Minimum Configuration of a Picking Window

Get All button. To un-select an item, double-click the item in the scrolled list. When un-selected, the item will also be un-highlighted in the Graphics Window. To un-select all items in the scrolled list, click the **Clear All** button. Entities can be added to the scroll list using the text fields across from **Enter Selection** by typing a single ID into the left field, or a range in the left and right field. To immediately have the scrolled list update with the ID numbers entered, click return after the ID number is typed in the text field(s). Otherwise they will be updated when the **Apply** button or another button is clicked. The **Apply** button executes the command associated with the window only if there is an ID present in the scrolled list. The text fields are cleared of information when a command is executed.

Picker Button

The picker button (shown to the left) is present in many of the windows. Use this button to activate a single picking action to select geometry. The ID of the geometry entity will then be placed in the text field to the left of the picker button.

Console Window

The GUI version of CUBIT also uses a console window (Figure 3-4) where all information about the success of a command, any errors or warnings generated, and information requested when querying the database are written. Since this window will contain essential information for the successful use of CUBIT, it should be placed on the screen in a readily accessible location.

Figure 3-4 Console Window

▼ Session Control

Several commands are available to control the overall CUBIT environment. Most of these commands are available in the GUI version by selecting the appropriate menu items in the **File** menu of the Main GUI window (Figure 3-5)

- **Exit.** The CUBIT session can be discontinued with either of the following commands

Exit

Quit

- **Reset.** A reset of CUBIT will clear the CUBIT database of the current geometry and mesh model, essentially allowing the user to begin a new session without exiting CUBIT. This is accomplished with the command

Reset.

Figure 3-5 Main GUI Window Showing File Menu

General Execution Commands

CUBIT contains a few commands which control the executable in general. To determine the software version number, execute the **version** command which reports the version number, the revision date of CUBIT and the version number of the ACIS solid modeler. This information is useful when discussing available capabilities or software problems with CUBIT developers.

Command echoing is controlled with the **echo {on | off}** command. By default, commands entered by the user will be echoed to the terminal in the command line version of CUBIT or to the console window in the GUI version of CUBIT.

▼ Journal Files

Journal files are used as a means to control CUBIT from simple text files and as insurance against lost work during execution. These files are created in various ways within CUBIT, or can be generated by any ASCII text editor by the user. They also serve as a means of error logging and user support. If a bug occurs and a journal file was being written, in many cases a support person can reproduce the error by simply playing the journal file.

CUBIT Journal File Generation

The CUBIT journaling facility can record all commands entered during the current CUBIT modeling session. By default, the journaling facility is on—if no journaling is desired, the user may issue the command **Journal Off** or run CUBIT with the **-nojournal** command line option.

Turning journaling off should be done with care, as the journal file can save model recreation time when errors occur during a long session. Once off, the default journaling cannot be turned on again, although the record option described below is always available.

Unless turned off, the journal file is automatically created in the current directory. All commands entered (except for **Quit** and **Help**) during the current modeling session are saved. The name of the journal file begins with the word "cubit" followed by a number between 1 and 1000 followed by the characters ".jou.", for example `cubit45.jou`. The number following "cubit" will increment as more journal files are generated in that directory.

Note: The command **Help** should never be placed inside a journal file; this is why these commands are excluded from the journaling described above. The relationship of the command parser and the help system is such that unrecoverable recursion would occur if a help session were requested when a journal file is being played back.

In addition to the default journaling, specific portions of the CUBIT session can be saved to user assigned files. If running the GUI version, selecting the **Journal Record/Play** menu item from the **Special** menu will display the dialog box shown in Figure 3-6

Figure 3-6 Journal Record/Play Dialog Box

To start recording a journal file, click the **Record** button. A File Selection dialog box will prompt the user for a filename. A typical file selection dialog box is shown in Figure 3-7. Recording can also be accomplished with the command

Record '<filename>'

Once initiated, all commands issued in CUBIT are copied to this file, as well as to the default journal files (if on). This journal file can be closed and recording to this file terminated by either pushing the Stop button shown in Figure 3-6, or with the command

Record Stop

The record command is particularly useful when a new finite element model is being built and alternate meshing strategies are being experimented with. Once the geometry has been defined, the record option can be used to record initial meshing controls and subsequent meshing commands. The mesh can be deleted, the recording terminated, and the process repeated to test alternate meshing strategies. To compare trial results, the user need only delete the current mesh and replay the journal file of the trial being considered.

Figure 3-7 File Selection Dialog Box

Replaying Journal Files

In the GUI version, selecting the **Playback** menu item from the **File** menu, or clicking on the **Play** button of the dialog box shown in Figure 3-6 will display a File Selection dialog box that allows you to select a journal file name. To replay a journal file using the command line, issue the command

Playback '<filename>'

The file will be read and commands in the file executed. **Pause** commands can be inserted in the journal file to cause the command execution to pause at that point. Typing a return if running the command line version or clicking on the continue button in the pause dialog box will continue execution. Playback commands can be nested. Note that the filename must be enclosed in single quotes.

▼ Graphics

The graphics display window displays a graphical representation of the geometry and/or the mesh. This display is used in either the command line version or the GUI version of CUBIT. The quality and speed of rendering the graphics, the visibility, location and orientation of objects in the window, and the labeling of entities can all be controlled by the user.

The geometric model can be viewed from any point and shaded before mesh generation has occurred. The shaded representation of the part does not represent the actual surface of the geometry, but a faceted approximation of it computed by the ACIS® solid modeler. The

wireframe representation only shows edges of the model, and can be misleading for geometry that has no edges (i.e. a sphere).

Similar to the geometric model, the mesh model can be viewed from any angle and displayed in either wireframe, hiddenline, polygonfill, flatshaded, painters or smoothshaded modes. The shaded representation of the part is generated directly from the quadrilateral faces which exist after the meshing process.

This section will discuss: 1) the control of the graphics window itself, 2) the control of the rendering parameters which affect the type, quality and/or speed of rendering of the image, 3) the control of which objects to draw and the color of drawn objects, 4) the desired labeling of objects on the image, 5) obtaining hard copy (e.g. postscript files) of the image, and 6) video animation generation.

As a general help to this section, when running the GUI version of CUBIT, most of the graphics controls are available under the **Graphics** menu item in the main window. To update the screen, a **Display** command must be issued.

Graphics Window Control

The graphics window is where the meshing graphics will be displayed and is the default viewport. The following attributes of the window can be controlled:

- **Window Size.** The graphics window may be resized with the mouse, or with the commands

Graphics Maximum

Graphics Windowsize <x_dimension> <y_dimension>

where **Maximum** will make the graphics window as big as the screen and the **x_dimension** and **y_dimension** are given in screen coordinates (pixels).

- **Background Color.** The window background color defaults to black but can be changed at any time using the command

Color Background <color_name>

Color Background <color_number>

where **color_name** is one of the colors listed in Appendix E, and **color_number** is an integer ID identifying the color. The background color can also be set using the **Graphics** menu **Color** dialog box as explained in section "Color" on page 66.

Image Rendering Control

The type, quality, and speed of the rendered image can be controlled in CUBIT using several graphics mode types and rendering options. The **Graphics** menu **Mode** dialog box shown in Figure 3-8 is used to set these options in the GUI version of CUBIT. The graphics mode type is set by using the **Graphics Mode Type** option menu. The available graphics mode types are:

- **Wireframe.** Wireframe drawing is the quickest mode, but it also can be the most confusing if the mesh or the geometry is very complex. No hiddenline processing is done. The command to set this mode is

Graphics Mode Wireframe

Figure 3-8 Graphics Mode Dialog Box

- **Hiddenline.** This option produces an accurate hidden line representation of the mesh or geometry. The command to set this mode is

Graphics Mode Hiddenline

- **Polygonfill.** This option is similar to flat shading, but uses a slightly different algorithm. The command to set this mode is

Graphics Mode Polygonfill

- **Painters¹.** This option produces a shaded image where each polygon is drawn in a single shade. The polygons are drawn in a depth-sorted order. Although a correct rendering is produced for most images, there are cases where an incorrect image may be rendered. This mode is usually faster than the Flatshade and Smoothshade modes. The command to set this mode is

Graphics Mode Painters

- **Flatshade.** This option produces images where each polygon is drawn in a single shade. The image is slightly degraded with this option, but the speed of rendering is improved. The command to set this mode is

Graphics Mode Flatshade

- **Smoothshade.** A smoothshaded image produces the highest quality picture, but at the most expense. Colors are blended continuously over the drawn surfaces. The command to set this mode is

Graphics Mode Smoothshade

The **Graphics** menu **Mode** dialog box shown in Figure 3-8 is also used to set the graphics mode options. These options control details of how the image is controlled between displays, and the type of enhancements added to the regular drawing modes. All options default to **On** at the start of execution. The graphics mode options are chosen by pushing the appropriate radio

1. The terminology "painters" is used since it draws the scene similar to the method used by a painter who might paint closer objects over more distant objects.

buttons in the **Graphics Mode Options** region of this window. The graphics mode options available in CUBIT are:

- **Autocenter.** This option automatically centers the model in the viewport. The command to set this option is

Graphics Autocenter {On | Off}

- **Autoclear.** This option automatically clears the graphics window between displays, or updates. The command to set this option is

Graphics Autoclear {On | Off}

- **Border.** This option draws a border around the current viewport. The command to set this option is

Graphics Border {On | Off}

- **Axis.** This option controls the display of the axis or coordinate triad. The command to set this option is

Graphics Axis {On | Off}

- **Line Width.** This option controls the width of the lines used in the wireframe and hiddenline displays. The command to set the line width is

Graphics LineWidth <width>

All option settings take affect at the time they are selected, it is not necessary to apply the changes. The **Set View Parameters** is a short-cut method for popping-up the Graphics View Dialog Box described in the next section.

Two additional commands,

graphics clear

graphics center

are available from the command line to perform a one-time only clear of the graphics window or centering of the model in the viewport. They do not affect the setting of the autoclear and autocenter toggles.

Viewing the Model

Figure 3-9.shows a schematic of the variables that effect the view of the object. Adjusting these variables will effect the way the three-dimensional model is projected onto the two-dimensional screen. These adjustments require you to update the display to see the results. To change the view parameters in the GUI version, select the **View** menu item from the **Graphics** menu. The dialog box shown in Figure 3-10 will be displayed. This dialog box will show the current values for the 'at' point, the 'from' point and the up vector. The following adjustments can be made by the user:

- **View At Point.** The point you are viewing or looking 'at' can be set using the **X**, **Y**, and **Z** coordinates of the **View 'At'** text fields. To set the looking 'at' point using the command line, issue the command

[View] At <x> <y> <z>

- **View From Vector.** The point you are viewing 'from' can be set using the **X**, **Y**, and **Z** coordinates of the **View 'From'** text fields. If automatic centering (see "Image Rendering Control" on page 59) is on, the input 'from' vector defines a relative viewpoint away from the

Figure 3-9 Schematic of From, At, Up, and Perspective Angle

View At	View From	View Up-Vector
0.0 X	0.0 X	0.0 X
0.0 Y	0.0 Y	1.0 Y
0.0 Z	100.00 Z	0.0 Z

VIEW ROTATION	Pan	VIEWPORT
<div> <div>-</div><div>+</div><div>X</div> </div> <div> <div>-</div><div>+</div><div>Y</div> </div> <div> <div>-</div><div>+</div><div>Z</div> </div> <div>Increment</div>	<div> <div>▲</div><div>▼</div><div>◀</div><div>▶</div> </div> <div>Increment</div>	<div>VIEWPORT</div> <div>RESET</div>

APPLY and UPDATE	Get Current Values	CANCEL
------------------	--------------------	--------

Figure 3-10 The Graphics View Dialog Box

'at' point, in the direction of the 'from' vector. The magnitude of the 'from' vector is computed so that the picture fits nicely on the screen. When automatic centering is off, the 'from' vector defines an absolute viewpoint. To set the viewing 'from' vector using the command line, issue the command

[View] From <x> <y> <z>

- **Up Vector.** The up vector sets the orientation for the graphical display. In other words, along the line which connects the 'from' and the 'at' point, the up vector specifies which direction is displayed as up on the screen. This can be set using the **X**, **Y**, and **Z** coordinates of the **Up Vector** text fields in Figure 3-10 or using the command

[View] Up <x> <y> <z>

- **Rotate.** The rotation of the view can be specified by an angle about the global axis, or about a vector specified by two vertices. The *right hand rule* is used in all rotations. To rotate about an axis using the GUI interface, place a rotation angle (in degrees) in the **Increment** text field shown in Figure 3-10. Rotation is performed by clicking on either the + or - button by the desired axis (**X**, **Y**, or **Z**). The command to accomplish such a rotation is

[Graphics] Rotate <angle> About {X | Y | Z}

Continuous rotations about any axis can be performed by double clicking one of the + or - buttons. The view is changed in sequential steps by the rotation increment specified. The screen is updated as fast as the picture can be processed. When the desired view is obtained the continuous rotations can be stopped by clicking the mouse somewhere in the View dialog box. These continuous rotations are not actually sent as commands to the parser, and as such are not stored in the journal file. To save the final state of the viewing angle, push the **Get Current Values** button in Figure 3-6 followed by the **Apply and Update** button. This will store the viewing parameters in the journal file. Continuous rotations are not available in the command line version.

Rotations can also be performed about the line joining the two vertices of a curve in the model, or a line connecting two vertices in the model. This is done with the commands¹

[Graphics] Rotate <angle> About Curve <curve>

[Graphics] Rotate <angle> About Vertex <vertex_1> Vertex <vertex_2>

- **Perspective.** The perspective angle can be set to adjust the relative perspective distortion of the view. A value of 0.0 will produce no distortion as if the viewing "from" location was at infinity. A larger value will produce more distortion. Values of about 15.0 degrees are normal. The perspective angle is set using the command

Graphics Perspective Angle <angle>

A more convenient method of adjusting the perspective is with a simple on/off toggle. This toggles the perspective angle between 0.0 and the current setting. The command to toggle perspective on and off is

Graphics Perspective {On | Off}

- **Zooming.** The image can be zoomed to provide a close-up view of portions of the image. When using the GUI, the **Zoom** and **Zoom Reset** buttons on the main GUI window (Figure 3-1) are used. The **Zoom** button allows a zoom window to be defined on the screen using the mouse. When the **Zoom** button is clicked the cursor will move to the Graphics Window and become a cross-hair cursor. Click the left mouse button and drag the cursor over the desired zoom area and then release the mouse button. The third mouse button will cancel a zoom if clicked before the user clicks the first mouse button. The command for performing a zoom is

[Graphics] Zoom <x_min> <y_min> <x_max> <y_max>

1. See "Geometry Definition" on page 75 for definitions of Curve and Vertex

where the values specified are in screen coordinates (between 0 & 1). The **Zoom Reset** button updates the zoom limits to a size appropriate to capture all currently defined entities. The equivalent command is

[Graphics] Zoom Reset

The **Get Current Values** button in Figure 3-10 displays the current 'at', 'from', and 'up vector' in the appropriate text fields. This is useful after doing multiple rotations. To journal a viewing angle after doing a continuous rotate command, select this button then click the **Apply and Update** button. The commands

List View

View List

will list the current values of the At point, From point, Up vector, and Perspective angle in the command line version.

Controlling the Set of Displayed Entities

The entities to be drawn in the current image can be controlled to limit the amount of information presented on the screen. There are two distinct modes of getting entities to the screen. The first is to set the visibility of the entities desired to be on and to turn the rest off. This visibility control establishes a "display list" of items that will be included in the image every time it is redrawn with a **display** command. The second method is an *immediate mode* drawing capability. Using a number of **draw** commands, individual items can be drawn onto the current picture. A draw command does not put the designated entities on the "display list" - it simply draws their wire frame image over the top of the current image. This immediate mode drawing is useful in highlighting specific nodes, faces, etc., but will not change the picture that is displayed when the image is updated using the **display** command. This section uses geometry and mesh terms defined in "Geometry Definition" on page 75, "Mesh Definition" on page 95, and "Finite Element Model Definition" on page 117. The reader may want to read those sections prior to reading the following discussion.

Setting Visibility

Visibility of the geometry and the mesh is controlled by the use of global settings as well as through the use of individual (selective) geometric entity settings. In the GUI version, selecting the **Visibility** menu item from the **Graphics** menu will display the Visibility dialog box shown in Figure 3-7. The **Global/Selective Mode** can be set to either **Global** or **Selective**. The **Visibility Type** will change to the proper list based on the mode. The **Visibility Mode** radio buttons control whether visibility is set to **on** or **off**.

• Global Settings

Choosing the **Global Mode** will change the **Visibility Type** to include the appropriate global items when using the GUI (See Figure 3-11). The following global settings can be used to adjust the display list:

- **Geometry**. This sets the visibility of all geometric entities.
- **Mesh**. This sets all mesh entities to be set to on or off.
- **Node**. This sets the drawing of mesh nodes (small dots).
- **Vertex**. This sets the drawing of geometric vertices (small dots).

Figure 3-11 Visibility Dialog Box

- **NodeSet.** This sets the drawing of all nodes (small dots) in a NodeSet to be either on or off.
- **SideSet.** This sets the drawing of all element faces (or edges) in a SideSet to be either on or off.

The command line equivalent for setting global visibility flags is:

```
{block | geometry | mesh | node | vertex | nodeset | sideset} visibility {on | off}
```

• Individual Geometric Entity Settings

Two visibility flags are attached to individual geometric entities: 1) a flag indicating whether the geometry itself is to be included in the display list (visible), and 2) a flag to indicate if the mesh attached to the geometry is to be visible. Choosing the **Selective Mode** will change the **Visibility Type** to include the appropriate selective items when using the GUI. For each geometric entity, the visibility of the item and any owned mesh can be set, or just the geometry visibility or the mesh visibility can be set. The visibility for bodies, volumes and surfaces can also be set with this interface.

The command line equivalents for the selective visibility flags are:

```
{body | volume | surface} visibility <range> {on | off}
```

```
{body | volume | surface} geometry visibility <range> {on | off}
```

```
{body | volume | surface} mesh visibility <range> {on | off}
```

Immediate Mode Drawing

The series of **Draw** commands allow inspection of individual geometric and mesh entities. Individual entities or ranges of entities can be displayed. The draw commands affect the graphics system only temporarily and updating the display will show only those items actually in the display list.

When using the GUI version, select the **Draw** menu item from the **Graphics** menu to display the dialog box shown in Figure 3-12. Select an entity type to be drawn, enter an ID or a range of IDs for that entity type, and then click **Apply** to draw the entities.

Figure 3-12 Graphics Draw Dialog Box

The command line equivalent to draw selective entities is:

```
draw {body | curve | edge | face | hex | volume |
      node | nodeset | sideset | surface | vertex} <id_range>
```

If **autoclear** mode is enabled, each **draw** command will clear the screen prior to updating the display. If **autoclear** mode is disabled, the specified entities will be added to the current set of displayed entities. An explicit **clear** command may be issued at any time to clear the display.

Color

The **Color** commands give the user customization control of the screen appearance of any geometric entity and its owned mesh entities. The default color used for an entity is the color of the owning entity¹. For example, if the color of a curve is not specifically set, it inherits the color of the owning surface. Mesh entity colors are determined by the owning geometry entity, unless set specifically according to the nodeset, sideset, or mesh entity color commands. The user can also control the color of the screen background.

1. See "Geometric Topology" on page 75 and "Mesh Definition" on page 95 for a description of the ownership of geometric and mesh entities.

The colors available at this time are listed in Appendix E. To change the color of an entity in the GUI version select the **Color** menu item from the **Graphics** menu. A Color dialog box will be displayed (see Figure 3-13) from which the geometry type of the entity to be changed can be selected. Choose a color from the list. The color chosen will be displayed in the swatch area to the right of the list. Pick the entities to be modified and click **Apply**.

Figure 3-13 Color Dialog Box

The command line equivalents to change colors are:

```
color {body | volume | surface | nodeset | sideset | block} <id_range>
    <color name>
color {body | volume | surface} <id_range> mesh <color name>
color {body | volume | surface} <id_range> geometry <color name>
color {node | background} <color_name>
```

Entity Labeling

All geometric entities can be labelled with unique (to their geometric type) labels to enable specific entity identification. All mesh entities can also be labelled with unique (to their mesh entity type) labels to enable specific entity identification. The labels are turned on or off by using the **Label** commands. The labels will be displayed on the entity's centroid, which is helpful in the screen picking operations which are used in the GUI version of CUBIT. The screen picks try to locate the entity with the closest centroid to the actual screen pick. Thus by turning entity labelling on, the user knows exactly where to click the pointing device in order to pick a specific

entity. Labels are also useful in determining which entities were merged during a Feature Consolidation operation.

In the GUI version, selecting the **Label** menu item from the **Graphics** menu will display the Label dialog box shown in Figure 3-14. The Label dialog box shows the currently defined values for labels upon selecting this menu item. It will be necessary to update the display to see the results of the executed commands. The **All**, **Geometry** and **Mesh** buttons will turn labels on or off for all the related entities. The remaining buttons turn labels on or off for the selected entity type.

Figure 3-14 Label Dialog Box

The equivalent command line commands are:

label all {on | off}

label geometry {on | off}

label mesh {on | off}

**label {body | volume | surface | curve | vertex | face | edge | hex | node}
{on | off}**

Hardcopy Output

The **hardcopy** command is used to capture graphics output to a PostScript file. Color or monochrome PostScript and encapsulated PostScript files are available.

In the GUI version, selecting the **Hardcopy** menu item from the **Graphics** menu displays the dialog box shown in Figure 3-15. This dialog box is used to define PostScript capture options. Input a filename to be used for the PostScript capture in the filename text field. Select a radio

button to define either encapsulated PostScript (**EPS**) or normal **PostScript** and a **radio** button to define either **Color** or **Monochrome** PostScript. Click the **Apply** button to capture the current display or **Cancel** to abort the output. Only one display can be written to a file at the current time. A new filename must be specified for each display or the file will be overwritten. The output defaults to non-encapsulated color PostScript.

The command line equivalent is:

hardcopy '<filename>' [encapsulated|postscript|eps] [color|monochrome].

Figure 3-15 Hardcopy Output Dialog Box

Video Animations

Several commands are available for generating a video display of the graphics on the screen. The actual video initialization and recording has been set to work with the system in the graphics lab of the computational mechanics department. However, the animation commands can be used without actually recording the pictures to produce a smooth rotation of various aspects of the model being displayed. To initialize the video system, the command

Video Initialize

is used. This will set up the recording devices, move the recorder to the first available frame, and set the system ready to record the picture displayed on the workstation screen next to the video recording equipment. It has been found that resizing the window with the following commands positions the graphics display in the proper position for optimal recording:

Windowsize Maximum

Windowsize 1170 820

The video animation can now be generated by taking sequential snapshots of the screen. Each snapshot is captured by issuing the command:

Video Snap

Many of the meshing algorithms have been imbedded with flags that allow incremental snaps during the meshing process. These flags are activated by the initialization of the video device. This allows the generation of video animations of the meshing process rather easily. Often it is useful during an animation sequence to spin the picture around for the viewer to see the geometry and/or the mesh. These spins will appear choppy on the resulting video unless the object starts and stops rotation *smoothly*. Several **Animate** commands have been implemented which will rotate the geometry, mesh, or mesh overlaid on a smooth rendering of the object in a specified number of steps in this *smooth* manner. These commands are also useful for showing the model to others during demonstrations. For video animations, 30 to 60 steps are needed

when making a full revolution. If the video system has been initialized, the animation command will take a snapshot at each step of the rotation. The respective command syntax for displaying the geometry, the mesh, and the overlay is:

Animate Geometry Volume <volume_id_range> **Rotate** <degree>
About <axis> **Steps** <number>

Animate Mesh Volume <volume_id_range> **Rotate** <degree>
About <axis> **Steps** <number>

Animate Overlay Volume <volume_id_range> **Rotate** <degree>
About <axis> **Steps** <number>

▼ Model Interrogation

Mesh and geometric model information can be obtained through the **List** commands. For example, a global tally of all model entities can be determined with the **List Totals** command.

In the GUI version, selecting the **List Info** menu item from the **Special** menu displays a dialog box to interrogate a model (see Figure 3-16). Select either the total or a specific geometry type to get information on the **Geometry Type** option menu. Input an ID for the geometry type selected (not used if getting total) or click the pick button (↖) to select the geometry from the graphics window. The output from the interrogation will be displayed in the scrolled window.

Figure 3-16 List Window

Note: Currently **Hex** and **Face** are not available from a GUI window; however, they can be accessed from the command line.

The command line equivalents are:

list totals

list {body | volume | surface | curve | vertex | hex | face | node} <id_range>

A sample of these commands and a brief description of the output follow:

```
CUBIT> list totals
Executing Command: List Totals
Model Entity totals:
    Geometric Entities:
        Total bodies:1
        Total wires:0
        Total volumes:1
        Total surfaces:6
        Total curves:12
        Total vertices:8
    Mesh Entities:
        Total Hex Elements:64
        Total mesh faces:228
        Total mesh edges:294
        Total mesh nodes:125
    Boundary Condition Entities:
        Total nodesets:1
        Total sidesets:0
    Special Entities:
        Total BoundaryLayers:0
```

Specific geometric information can be printed for bodies, volumes, surfaces, curves, and vertices. The **List Body** command will indicate the meshed status (whether the body has been meshed yet) and the volumes owned by the body and their respective meshed status.

```
CUBIT> list body 1
Executing Command: List Body 1
Body Entity 1:
    Owned Volumes:      Id:      Meshed:
                        Volume    1      Yes
```

The **List Volume** command will indicate meshed status, the current meshing scheme for the Volume, the current interval count, interval size, and element block id, and the surfaces owned by the Volume and their respective meshed status.

CHAPTER 3 Environment

```
CUBIT > list volume 1
Executing Command: List Volume 1
Volume Entity 1:
    Meshed:    Yes
    Scheme:    MAP
    Interval Count:4
    Interval Size:0.00
    Block Id: 1
    Owned Surfaces:    Id:    Meshed:
        Surface 1      Yes
        Surface 2      Yes
        Surface 3      Yes
        Surface 4      Yes
        Surface 5      Yes
        Surface 6      Yes
```

The **List Surface** command will indicate meshed status, total number of nodes if meshed (including nodes on the bounding curves), the current meshing scheme for the surface, the interval count, interval size, and element block id, and the curves owned by the surface and their respective meshed status.

```
CUBIT > List Surface 1
Executing Command: List Surface 1
Surface Entity 1:
    Meshed:    Yes
    Total nodes (all inclusive)16
    Scheme:    MAP
    Interval Count:4
    Interval Size:0.00
    Block Id: 0
    Total number of curves:4
    Owned Curves:    Id:    Meshed:
        Curve 1      Yes
        Curve 2      Yes
        Curve 3      Yes
        Curve 4      Yes
```

The **List Curve** command will indicate meshed status, the current meshing scheme for the curve, the arc length of the curve, the interval count, interval size, and the vertices owned by the curve and their respective meshed status.

```
CUBIT > List Curve 1
Executing Command: List Curve 1
Curve Entity 1:
    Meshed:    Yes
    Scheme:    MAP
    Curve length:100.00
    Interval Count:4
    Interval Size:25.00
    Owned Vertices:
        Start Vertex    Id:    Meshed:
        End Vertex      1      Yes
                        2      Yes
```

The **List Vertex** command will indicate the meshed status and the coordinates of the vertex.

```
CUBIT > List Vertex 1
Executing Command: List Vertex 1
Vertex Entity 1:
    Meshed:    Yes
    X coordinate:    50.00
    Y coordinate:    -50.00
    Z coordinate:    50.00
```

The **List Hex** command will indicate the Exodus Id (id of hex when placed in the exodus database, not the CUBIT id, default value is -1 before writing Exodus file), the volume which owns the hex element, and the eight corner nodes (in standard Exodus order).

```
CUBIT > List Hex 1
Executing Command: List Hex 1
Hex Entity 1:
    Exodus Id = -1
    Owned by Volume 1
    Contains nodes:
        6      13      59      47
        9      25      99      75
```

The **List Face** command will indicate the geometric entity (volume or surface) which owns the element face, the hexes which share the face, and the four corner nodes (in standard Exodus order).

```
CUBIT > List Face 1
Executing Command: List Face 1
MeshFace Entity 1:
    Owned by Surface 6
    Shared by:
        hex element 52
    Contains nodes:
        2      5      17      14
```

The **List Node** command will indicate the coordinates of the node.

```
CUBIT > Executing Command: List Node 1
MeshNode Entity 1:
  x coordinate: 50.000000
  y coordinate: -50.000000
  z coordinate: 50.000000
```

▼ Help Facility

Two help systems are available in CUBIT: an window-based hypertext help facility and a text-based help facility. The user can access the hypertext help facility by typing **hyperhelp** and the text-based help facility by entering **help**. To attain further information about specific commands, the user can enter a specific keyword name after the term **help** or **hyperhelp**.

The text-based help facility prints only the usage syntax for the specified keyword. It is accessed by typing **help <keyword>**. For example, typing **help volume** would output the syntax for all volume commands to the standard output window. If no keyword is entered, a list of all valid keywords is output. The text-based help facility is accessed from the GUI version of CUBIT by typing help in the command line text field.

The window-based hypertext help facility is accessed by typing **hyperhelp <keyword>**. A window should pop-up containing a representation of the command index portion of this manual. Additional keyword parameters can be entered following the keyword to obtain more specific help. For example: **hyperhelp view at** would provide help on the **view at** command rather than general help on the **view** command.

The hypertext help facility can be accessed from the GUI version of CUBIT by selecting the **Help** menu item on the main menu or by clicking the **help** button on specific windows for help specific to those windows.

Chapter 4: Geometry

- ▼ Geometry Definition...75
- ▼ Geometry Creation...77
- ▼ Geometry Manipulation...85
- ▼ Geometry Decomposition...91
- ▼ Geometry Consolidation...92

This chapter describes methods available in CUBIT to produce and manipulate the geometry needed for meshing. Definitions of geometric entities and the structure of the nonmanifold geometry represented by CUBIT are given. This is followed by sections describing geometry importation, creation and modification. Geometry consolidation, the process of generating cellular topology from a manifold model, is also described.

▼ Geometry Definition

All geometric entities that exist in the CUBIT environment are represented by a solid model. In two dimensional modeling systems, a list of connected directional line segments is sufficient to provide a complete and unambiguous geometric definition. In three dimensions, only a solid model representation can guarantee complete and unambiguous geometry. All meshing tools available in CUBIT use this solid model geometry when generating the discretized representation of the geometry, i.e. the mesh. ACIS[®] is the solid modeling engine currently used by CUBIT. However, CUBIT uses its own geometric overlay to represent a non-manifold cellular topology for meshing.

Geometric Topology

Topology refers to the manner in which geometric entities are connected within the solid model. Within CUBIT, the geometric entities consist of *vertices*, *curves*, *surfaces*, *volumes*, and *bodies*. They are defined as follows:

Vertex

A vertex occupies a single point in space. A vertex is used to bound a curve and/or to specify a specific location for a node. A vertex which is located in the interior of a surface is called a hard point. It is used to force a node location to that specific geometric location.

Curve

A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. An example of a single vertex curve (both ends of the curve meet the same vertex) might be the curve which bounds an end cap surface of a cylinder. A curve is used to bound a surface. Curves may be generated independent of surfaces and as such used to specify the geometry for a sequence of bar or beam elements.

Surface

A surface in CUBIT is a finite bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the volume in a volume. A surface is bounded by a set of curves. Surfaces must have at least one bounding curve to be meshed in CUBIT. Surfaces may be generated independent of volumes. Such free surfaces may be used to specify the geometry for shell elements. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction.

Volume

Volumes are volumetric regions and are always bounded by one or more surfaces. For practical consideration, volumes will always be bounded by two or more surfaces. CUBIT currently cannot mesh a volume bounded by only one surface (e.g. a sphere) since such a surface has no bounding curves.

Body

A body is simply a collection or set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume.

Cellular Topology

Cellular topology (non-manifold topology) allows the connection of any number of surfaces to curves. A typical manifold model (or 2-manifold topology) only allows two surfaces to be bounded by a single curve. Cellular topology, because of its more general nature, allows two adjacent volumes to share a common surface between them as shown in Figure 4-1. It also allows the formation of dangling faces and edges (Figure 4-2). These geometric constructs are sometimes useful in generating complex meshes.

Lump 1 Web Lump 2

Figure 4-1 Cellular Topology Between Volumes

Figure 4-2 Dangling Faces & Edges

Cellular topology's advantage for mesh generation is that, when used properly, it eliminates the problem of equivalencing the mesh entities that are supposed to be shared between adjacent geometric entities (for instance, the common surface (Web) shown in Figure 4-1). It also supports the formation of dangling faces and dangling edges as shown in Figure 4-2. This allows the proper connection of surface, beam, and/or solid mesh elements.

▼ Geometry Creation

The examples in "Examples" on page 137 show solid model construction using several standard techniques. One technique is to use primitives (e.g. bricks, cylinders, spheres, etc.) and perform booleans (e.g. subtractions, intersections, and unions) on the primitives to generate the desired geometric shape. Another approach is to construct two-dimensional wireframe geometry and then perform a sweep (either along a path or about an axis) to create the three-dimensional solid model. As the examples indicate, it is necessary to save the solid model to a disk file (using a .sat filename extension) for subsequent meshing in CUBIT.

CUBIT geometry may be created within CUBIT using basic primitives and/or the sketchpad in combination with the boolean operators. The generation of geometric primitives and their manipulation and positioning will be discussed first in this section, with the boolean operations covered afterwards. Geometry may also be produced using several programs and imported into CUBIT through ACIS® format files. Several of these packages are discussed later in this section.

Since the geometry acts as the template for discretization into a mesh, care must be taken to insure an appropriate representation for the problem being analyzed. Users will find that building solid models of "real parts" will likely consist of a combination of the approaches mentioned above.

Geometry Primitives

Geometry primitives are classes of general geometric shapes which are differentiated by basic parameters. For example, a cylinder is a basic shape that can be specified by the parameters *height* and *radius*. Primitives available in CUBIT include the brick, cylinder, torus, prism, frustum, pyramid, and sphere. These primitives can be generated and used in boolean operations to produce very complex shapes. The geometric primitives can also be used in boolean operations with geometry generated through other means (e.g. using the sketch pad, or geometry read in from other sources). When using the GUI, the geometric primitives can be generated

using the **Primitives** suboption in the **Geometry** menu. Figure 4-3 shows a sample of the available primitives.

Figure 4-3 CUBIT Geometry Primitives

Brick

The brick is a cuboid where all the surfaces intersect at right angles. Figure 4-4 shows the **Geometry Primitives** dialog box when **Brick** is chosen. There are three parameters that may be specified, **Width** (x-dimension), **Depth** (y-dimension), and **Height** (z-dimension). Only **Width** is required. If only a **Width** value is specified, the other two values default to that value, thus producing a cube. The **Apply** button will generate the brick. The equivalent command to generate a brick using the command line is

[create] brick width <x-dimension> [depth <y-dim> height <z-dim>]

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin and aligned with the coordinate axes.

Cylinder

The cylinder is a constant radius tube with right circular ends. Figure 4-5 shows the relevant portion of the **Geometry Primitives** dialog box when **Cylinder** is chosen. There are two parameters that must be specified, **Height** (z-dimension), and **Radius** (x/y-dimension). The **Apply** button will generate the cylinder. The equivalent command to generate a cylinder is

[create] cylinder height <z-height> radius <x/y-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin and aligned with the length of the cylinder along the z-axis.

Figure 4-4 Brick Creation Dialog Box

Figure 4-5 Cylinder Creation Dialog Box

Note: A cylinder may also be created using the **frustum** command with all radii set to the same value.

Prism

The prism is an n-sided, constant radius tube with n-sided planar faces around the tube. Figure 4-5 shows the relevant portion of the **Geometry Primitives** dialog box when **Prism** is chosen. There are three parameters that must be specified, **Height** (z-dimension), **Sides** (number of sides) and **Radius** (x/y-dimension). The radius defines the circle circumscribing the prism.

cross-section. The **Apply** button will generate the prism. The equivalent commands to generate prisms are:

[create] prism height <z-height> sides <nsides> radius <x/y-radius>

**[create] prism height <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius>**

The new body contains one volume which will be given the next highest body ID number. The prism will be centered about the origin and aligned with the height of the cylinder along the z-axis. One of the planar sides will be perpendicular with the X-axis. The number of sides must be greater than or equal to three.

Figure 4-6 Prism Creation Dialog Box

Note: A prism may also be created using the **pyramid** command with all radii set to the same value.

Frustum

A frustum is a general elliptical right frustum. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing a different radius in the two principle directions of the cone. Figure 4-7 shows the relevant portion of the **Geometry Primitives** dialog box when **frustum** is chosen. There are four parameters that may be specified, **Height** (z-dimension), **Major Radius** (x-radius), **Minor Radius** (y-radius) and **Top Radius** (x-radius at the top). The top y radius is calculated based on the ratio of the major and minor radii given. If only **Height** and **Major Radius** are specified, the other two radii are defaulted to the major radius value. If all radii are equal, a frustum defaults to a simple cylinder. The **Apply** button will generate the frustum. The command to generate a frustum using the command line is

**[create] frustum height <z-height> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>**

The new body contains one volume which will be given the next highest body ID number. The frustum will be centered about the origin with the central frustum axis aligned with the z-axis.

Figure 4-7 Frustum Creation Dialog Box

Pyramid

A pyramid is a general n -sided prism. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing different circumscribing radii in the two principle directions of the pyramid. Figure 4-7 shows the relevant portion of the **Geometry Primitives** dialog box when **Pyramid** is chosen. There are five parameters that may be specified, **Height** (z -dimension), **Major Radius** (x -radius), **Minor Radius** (y -radius) and **Top Radius** (x -radius at the top). The top y radius is calculated based on the ratio of the major and minor radii given. If only **Height**, **Sides**, and **Major Radius** are specified, the other two radii default to the major radius value. If all radii are equal, a pyramid defaults to a simple n -sided prism. The **Apply** button will generate the pyramid. The command to generate a pyramid using the command line is

```
[create] pyramid height <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius> top <top-x-radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the central pyramid axis aligned with the z -axis.

Sphere

The **sphere** command generates a simple sphere. Figure 4-9 shows the relevant portion **Geometry Primitives** dialog box when **Sphere** is chosen. Only one parameter may be specified, **Radius**. The **Apply** button will generate the sphere. The command to generate a sphere from the command line is

```
[create] sphere radius <radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin.

Torus

The torus command generates a simple torus. Figure 4-10 shows the relevant portion of the **Geometry Primitives** dialog when **Torus** is chosen. Two parameters must be specified, **Major Radius**, or the radius of the spine of the desired torus, and **Minor Radius**, or the radius

Figure 4-8 Pyramid Creation Dialog Box

Figure 4-9 Sphere Creation Dialog Box

of the cross-section of the ring. The minor radius must be less than the major radius. The **Apply** button will generate the torus. The command to generate a torus from the command line is

[create] torus major [radius] <major-radius> minor [radius] <minor-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the spline of the torus aligned perpendicular to the z-axis.

Sketchpad Geometry

The sketchpad is used to construct two-dimensional wireframe geometry profiles. The profiles can be swept (either along a path or about an axis) to form more general solid bodies than geometry primitives. The swept bodies created with the sketchpad can be used in boolean operations with other geometry (e.g. primitives, other swept bodies, or geometry read in from other sources). The sketchpad is activated by selecting the **Sketch** suboption in the **Geometry** menu. The sketchpad is active when the sketch creation dialog box shown in Figure 4-11 is visible. The sketchpad is only available from the GUI version of CUBIT at this time.

Figure 4-10 Torus Creation Dialog Box

Figure 4-11 Sketch Creation Dialog Box

Sketch Overview

Creating a sketched profile is a two step operation: first define a polygonal outline and then refine the outline by chamfering or filleting corners or by pulling edges of the polygon into arc shape. The sketch primitives (line, box, round, chamfer, or arc) are created by clicking the corresponding button in the left column of the sketch creation dialog box. Then, in the graphics window, press the middle mouse button, drag the mouse to create the desired shape, and then

release the mouse button. While dragging the mouse, the sketchpad provides rubber band feedback. When the sketched profile is complete, accept it by pressing the **Apply** button. The sketchpad will close the profile and create a two-dimensional wireframe geometry profile. The sketch can be aborted by pressing the **Cancel** button.

Polygonal Outline

The primary method for creating the polygonal outline is to use the **line** primitive. The first mouse press defines a starting point for the outline. Mouse release defines a straight **line**. If the sketched line is nearly horizontal or nearly vertical, the sketchpad will adjust the line. The other method for creating part of the polygonal outline is to use the **box** primitive. The box is a shortcut for sketching three lines in order to create a horizontal or vertical slot or tab. If the line created before the box is more closely horizontal than vertical, then the slot or tab created will be vertical. Similarly, if the box is attached to a roughly vertical face, the resulting slot or tab will be horizontal.

Outline Refinement

The polygonal outline can be refined by chamfering or rounding any corner. After selecting the appropriate option, **chamfer** or **round**, press the middle mouse button near the corner to be modified and drag the profile to see the effect of the modification. During drag, feedback is the same for chamfer and round. When the profile is finally created rounds will be correct.

Another way to refine the outline is to bend any line into an arc. After selecting the **arc** option, press the middle mouse button near the middle of a line to be modified and drag the profile to see the effect of the modification.

Importing Geometry

Capabilities exist to use geometry files generated using the ACIS[®] Test Harness, Pro/Engineer, or FASTQ. All geometry read into CUBIT must be in the form of ACIS[®] files. The ACIS[®] Test Harness can generate ACIS[®] files directly. Pro/Engineer and FASTQ files must first be translated into the ACIS[®] format before being usable with CUBIT.

Importing ACIS Files

Externally-generated ACIS[®] files can be read into CUBIT by selecting the **Import > Acis** options under the **File** menu in the GUI. The command line syntax for this command is:

```
Import Acis '<acis_filename>'
```

Note that the filename must be enclosed in single quotes.

ACIS Test Harness

The ACIS[®] test harness is a convenient method to generate geometry, since it is available on every platform which maintains an ACIS[®] license which includes all systems on which CUBIT is available. The test harness can be used to create geometric models via boolean operations (unions, differences, and intersections) and sweeps of wire profiles.

The ACIS test harness is a solid model construction program that supports English-like commands entered via the keyboard, and provides simple graphic output in a graphics window. Users should find it similar to FASTQ [5] and GEN3D [10] in that all commands entered during a session are saved in a monitor file (.mon extension) that can be subsequently replayed at a later time to re-generate the model. The graphics window is only displayed when a draw command

is entered, thus simplifying batch processing (draw commands may be commented out with a '#' character). Additionally, Aprepro [13] may be used to evaluate any algebraic expression in the ACIS monitor file, allowing for parameterized geometry construction.

PRO/Engineer

The PRO/Engineer® product can also be used to create CUBIT geometry, although the support for this package is still being developed. Converters are being implemented which manage the translation of PRO/Engineer® assembly data into ACIS® format.

This solution is being pursued to address user needs and requirements regarding standard geometry formats at Sandia National Laboratories. Advantages to this creation method include the availability of the large number of parts being designed under the PRO/Engineer® format. PRO/Engineer® documentation and training is available at Sandia including consulting on an as-needed basis. Disadvantages include the single direction translation through which these parts must be sent to convert them to ACIS®. Some difficulties with numerical accuracy involving PRO/Engineer® have been cited although no negative impact of this on CUBIT meshing functionality has yet been observed.

FASTQ

Limited support for reading a FASTQ file directly into CUBIT is available. The limitations of this reader are due to the incompatibility of some FASTQ constructs and assumptions with the representation of three dimensional solids used in CUBIT. Some FASTQ geometric entities, such as sides, cannot be read directly into CUBIT. Users are encouraged to pursue the construction of new geometry using the internal sketch pad and primitives, the ACIS® test harness, or other geometry creation environments that are either compatible with ACIS or can be translated into ACIS format rather than using the FASTQ reader exclusively.

FASTQ files are imported into CUBIT using the **Import Fastq** command or the **Import > Fastq** option under the File menu in the GUI. The command line command syntax is:

Import Fastq '<fastq_filename>'

Since a FASTQ file by nature will be defined in a plane, it must be projected or swept to generate three dimensional geometry. Since this sweeping operation cannot currently be executed within CUBIT, only two-dimensional meshing of the FASTQ geometry can be performed. If three-dimensional geometry generated from the FASTQ geometry is required, the **fsqacs** translator should be used. It will generate ACIS files which can be used in the ACIS Test Harness and projected or swept to generate three-dimensional geometry which can be imported into CUBIT.

▼ Geometry Manipulation

Bodies can be translated, rotated, reflected, scaled, and copied in order to position them correctly before performing other tasks associated with generating a model. Boolean operations can also be performed between bodies. The transform operations which translate, reflect, scale, rotate and copy are described first in this section, followed by a description of the boolean operations intersect, subtract and unite.

Transform Operations

The translate, reflect, scale, and rotate functions do not create new geometry, whereas the copy, intersect, subtract and unite functions do create new geometry. If running the GUI, the **Transform** submenu of the **Geometry** menu is used for these operations.

Copy

The copy command copies an existing body to a new body without modifying the existing body. Figure 4-12 shows the dialog box that appears when the **Copy** option is chosen. Simply inserting a body ID in the **Body Name** and then pressing the **Apply** button will generate a new copy at the same location as the old one. When using the command interface, a copy can be made of several bodies at once, and the group can be translated with a specified offset, or rotated about a given vector. The commands for copying a body from the command line are

body <range> copy [move <x-offset> <y-offset> <z-offset>]

body <range> copy [rotate <angle> about {x | y | z}]

body <range> copy [rotate <angle> about <x-comp> <y-comp> <z-comp>]

If the **copy** command is used to generate new bodies, a copy of the original mesh generated in the original body can also be copied directly into the new body. This is currently limited to copies that do not interact with adjacent geometry. For details on mesh copies, see “Mesh Duplication” on page 112.

Figure 4-12 Body Copy Dialog Box

Move

The move command moves a body without adding any new geometry. Figure 4-13 shows the relevant portion of the dialog box that appears when the **Move** option is chosen. Inserting a body ID in the **Body Name** slot and filling in any of the offsets desired will move the body once the **Apply** button is pushed. Any **Offset** buttons not filled in are assumed to be zero. When using the command interface, a range of bodies to move can be specified

body <range> move <x-offset> <y-offset> <z-offset>.

Figure 4-13 Body Move Dialog Box

Scale

The scale command resizes the body without adding any new geometry. Figure 4-14 shows the relevant portion of the dialog box that appears when the **Scale** option is chosen. Inserting a body ID in the **Body Name** slot and filling in the desired scale will resize the body once the **Apply** button is pushed. The body will be scaled about its centroid. When using the command interface, a range of bodies to scale can be specified

body <range> scale <scale>

Figure 4-14 Body Scale Dialog Box

Rotate

The rotate command rotates a body about a given axis without adding any new geometry. Figure 4-15 shows the relevant portion of the dialog box that appears when the **Rotate** option is chosen. Inserting a body ID in the **Body Name** slot and filling in an **Angle** as well as any of the vector **Components** desired will rotate the body once the **Apply** button is pushed. If the **Angle** or any **Components** are not filled in they are defaulted to be zero. When using the command interface, a range of bodies to be rotated can be specified, as well as a rotation about one of the cartesian axes:

body <range> rotate <angle> about {x | y | z}

body <range> rotate <angle> about <x-comp> <y-comp> <z-comp>

Figure 4-15 Body Rotate Dialog Box

Reflect

The reflect command mirrors a body about a plane normal to an arbitrary vector without adding any new geometry. Figure 4-13 shows the relevant portion of the dialog box that appears when the **Reflect** option is chosen. Inserting a body ID in the **Body Name** slot and filling in the components of the reflection vector reflects the body about a plane normal to this vector once the **Apply** button is pushed. Any vector **Component** buttons not filled in are assumed to be zero. When using the command interface, a range of bodies to be reflected can be specified as well as directly specifying a plane normal to one of the coordinate axes.

body <range> reflect <x-comp> <y-comp> <z-comp>

body <range> reflect {x | y | z}

Restore

The restore command removes all previous geometry transformations from the specified body. Execution of this command is similar to the copy command except that the Restore button is selected in the Dialog Box. When using the command interface, a range of bodies to be restored can be specified.

body <range> restore

Figure 4-16 Body Reflect Dialog Box

Boolean Operations

Boolean operators allow boolean interactions (e.g. intersection, union, etc.) between bodies to produce a new body. The boolean operators available in CUBIT for modifying bodies are intersect, subtract and unite. When using the GUI, the boolean operators are available under the **Geometry** menu's **Booleans** option.

Intersect

The intersect command generates a new body composed of the portions of the geometry that are shared by both. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. Figure 4-17 shows the **Geometry Booleans** dialog when the **Intersect** option is chosen. The command line command syntax is

Intersect <body_id> With <body_id>

Subtract

The subtract operator will subtract one body from another. The order of subtraction is significant. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. Figure 4-18 shows the relevant portion of the **Geometry Booleans** dialog box pertaining to subtract when the **Subtract** option is chosen. When the **Apply** button is pressed, Body2 is subtracted from Body1. The command line command syntax is

Subtract <body_id> From <body_id>.

Unite

The unite operator will combine two bodies into one. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. Figure 4-19 shows the relevant portion of the **Geometry Booleans** dialog pertaining to this operation when the **Unite** option is chosen. The command line command syntax is

Unite <body_id> With <body_id>

Figure 4-17 Intersect Boolean Dialog Box

Figure 4-18 Subtract Boolean Dialog Box

Figure 4-19 Unite Boolean Dialog Box

▼ Geometry Decomposition

The capability to perform geometry decomposition within CUBIT is currently under active development. At the current time, geometry decomposition can only be performed crudely using boolean operations. A more robust and capable method called Web Cutting is not yet functional, but should be available in the near future.

Web Cutting

Web cutting allows the insertion of webs or faces between pieces of the geometry to facilitate meshing. Web cutting, although still being developed, should allow a user to decompose the geometry without the generation and use of boolean operations. This should greatly simplify the decomposition task.

When using the GUI, web cutting may be accomplished by choosing the **Decompose** option from the **Geometry** menu. A submenu will appear with the **Web Cutting** button. The dialog shown in Figure 4-20 will appear. A web is inserted by selecting a body, designating a plane.

Figure 4-20 Web Cutting Dialog

and optionally specifying a vector. The body is supplied in the **Body ID to Cut** box (picking is optional). With the radio buttons in the area of **Select A Plane Choice** the desired mechanism for defining a plane can be designated. Selecting **Face** will require a face ID. Selecting **3 Vertices** will require 3 vertex IDs to define the plane. The **Optional Vertex** option allows the designation of a vector in the direction the cut is desired. Without this designation, the program will fire rays in multiple directions tangent to the plane to calculate an intersection loop. The ID of the vertex at the origin of the vector is entered in **From Vertex**. The ID of the vertex in the direction of the cut is entered in **To Vertex**. The **Help** button will

provide a brief help message if the help system is installed on the system. Web cutting is initiated by pressing the **Apply** button. The commands to perform web cutting using the command line are:

Webcut Body <body_id> Face <face_id> [Vector <from_vertex> <to_vertex>]

**Webcut Body <body_id> Vertex <vertex_1> Vertex <vertex_2>
Vertex <vertex_3> [Vector <from_vertex> <to_vertex>]**

The first command corresponds to using a face to specify the cutting plane and the second command uses three vertices to define the cutting plane. An optional vector defined by two vertices can be used to designate the direction of the cut.

▼ Geometry Consolidation

Geometry consolidation is a means of enforcing mesh continuity and avoiding mesh node equivalencing. When a (manifold) solid model is constructed, it may contain any number of volumes which can belong to a single body. This does not imply that these volumes are "aware" of each other; CUBIT has no record or history of how the volumes from a body were created and what their spatial relationship may be. To determine this spatial relationship, some simple feature recognition concepts have been implemented to detect proximity between geometric entities, and to resolve any redundant entities.

Redundant geometry is not really redundant in a solid modeling sense, yet from a meshing standpoint (in which a contiguous mesh is required between volumes) if two surfaces exist where a single surface mesh must be generated, a redundancy must be resolved. This requirement can be illustrated in Figure 4-21. This figure depicts an "L" block (Figure 4-21), which could easily be meshed with a sweep operation, yet for our purposes here will be decomposed into two volumes (Figure 4-22) to demonstrate solid model surface redundancy.

For a contiguous mesh, CUBIT requires adjacent volumes to maintain one-to-one matching surfaces. The geometry consolidation tool in CUBIT is designed to recognize the spatial proximity between the surfaces and force them to use one surface mesh between them. This mesh sharing approach avoids the need to equivalence nodes and preserves mesh continuity. After the model in Figure 4-21 is decomposed into the model in Figure 4-22, surfaces 1 and 2 meet the one-to-one requirement. They can now be consolidated and treated as a single surface for meshing purposes. By extension, curves and vertices of adjacent volumes need to adhere to the one-to-one requirement also.

Assemblies which are imported from external solid modelers (e.g., Aries[®] ConceptStation, PRO/Engineer, ACIS[®] Test harness) will need to be inspected to insure that the adjacent volume surface to surface and curve to curve matching requirement has been properly adhered to. Some of these external modelers will use other internal geometry engines instead of ACIS[®] (PRO/Engineer does not run on ACIS[®]); when using geometry which was created in PRO/Engineer, it is critical to set the accuracy within PRO/Engineer to the highest level possible. The standard level of accuracy within ACIS[®] is much higher than that of PRO/Engineer (10^{-6} vs. 10^{-3}). CUBIT relies on geometric reasoning to identify matching surfaces and curves, and this technique is sensitive to the accuracy setting of the model being processed. Since Aries[®] ConceptStation is based on ACIS[®], it does not suffer from accuracy problems due to varying data formats and is probably the most attractive commercial modeler to use for creating geometry outside of CUBIT.

Figure 4-21 Solid Model Prior to Decomposition

General Geometry Consolidation

When a model fully meets the one-to-one topology requirement, it is ready for full geometry consolidation of redundant geometric entities. The **Merge All** command searches first for matching surfaces, then for matching curves, and finally for matching vertices between adjacent volumes in the current CUBIT model. When a match is found, the consolidator merges the two

Figure 4-22 Solid Model After Decomposition

matching entities inside the CUBIT database and reconnects all the affected neighboring entities (such as now disconnected curves and surfaces) with the appropriate replacement entity. It is critical to complete geometry consolidation before any meshing of the affected geometry is initiated. If geometric entities are merged subsequent to meshing operations, some of the associated mesh entities may be lost. When using the GUI, selecting the **Geometry Consolidation** menu item from the **Special** menu will display the Geometry Consolidation Dialog shown in Figure 4-23. This dialog contains a **Geometry Consolidate** option menu which is used to describe the type of consolidation to be performed. If **All** is chosen from this option menu, general geometry consolidation is performed when the **Apply** button is pushed. General geometry consolidation is performed with the following command from the command line interface:

Merge All

Selective Geometry Consolidation

When analyses may require certain redundant geometric entities (such as slide lines requiring redundant curves), a more selective mode of geometry consolidation is available. Selective geometry consolidation can be initiated between any two user-specified surfaces, curves, or vertices. The consolidation process will be limited to the specified entities and their affected neighbor entities. The command syntax for selective geometry consolidation is as follows: **Only Surfaces**, **Only Curves**, **Surface**, and **Curve**. Below this are two text fields which are used to select specific geometric entities to be merged if **Surface** or **Curve** are selected in the option menu. The ID of the desired geometric entities can either be typed into the text fields, or the user can click on the arrows to the right of the text field to enable picking of the geometric entities. At this time, click the **Apply** button to execute the geometry consolidation process or click the **Cancel** button to exit from the dialog without executing any commands

Merge Only Curves

Merge Only Surfaces

Merge Curve <int_curve_id> With <int_curve_id>

Merge Surface <int_surface_id> With <int_surface_id>

Merge Vertex <int_vertex_id> With <int_vertex_id>.

Figure 4-23 Geometry Consolidation Dialog Box

Chapter 5: Mesh Generation

- ▼ Mesh Definition...95
- ▼ Mesh Attributes...96
- ▼ Curve Meshing...98
- ▼ Surface Meshing...101
- ▼ Volume Meshing...107
- ▼ Mesh Duplication...112
- ▼ Mesh Editing...112

The methods used to generate a mesh using existing geometry are discussed in this chapter. The definitions used to describe the process are first presented, followed by descriptions of curve, surface, and volume meshing techniques. The chapter concludes with a description of the mesh editing capabilities.

▼ Mesh Definition

The mesh consists of entities similar in hierarchy to the geometry described in the previous chapter. The mesh entities include *nodes*, *edges* (bar elements), *faces* (quadrilateral elements), and *bricks* (hexahedral elements). Each mesh entity is associated with a geometry entity which owns it. This associativity allows the user to mesh, display, color, and attach attributes to the mesh through the geometry. For example, setting a mesh attribute on a surface affects all mesh entities owned by that surface.

Mesh Hierarchy

Mesh hierarchy refers to the manner in which mesh entities are connected within the mesh. Once a mesh is formed, the mesh entities define a discretized version of the geometry. The nodes required by higher-order elements are generated subsequent to the initial discretization; however, they are generated in the correct position based on the underlying geometry.

Node

A node is a single point in space. A node at a vertex is owned by that geometric vertex, a node on a curve is owned by that curve, a node on the interior of a surface is owned by that surface, and a node in the interior of a volume is owned by that volume.

Edge

An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge on a curve is owned by that curve, an edge in the interior of a surface is owned by that surface, and an edge in the interior of a volume is owned by that volume.

Face

A face is defined by four connected edges. A face on a surface is owned by that surface, a face in the interior of a volume is owned by that volume.

Brick

A brick is a hexahedral element defined by six connected faces. A brick is owned by the enclosing volume.

Mesh Generation

The mesh for any given geometry is generated hierarchically. For example, if the user issues a command to mesh a volume, first any unmeshed vertices are meshed with nodes. Next, any unmeshed curves are meshed, followed by the meshing of unmeshed surfaces. Finally the mesh within the volume is generated. Vertex meshing is of course trivial and thus the user is given no control over this process. However, curve, surface, and volume meshing can be directly controlled by the user. The **scheme** command specifies the meshing algorithm which will be used in meshing each of these geometric entities and the block command specifies the type of elements which will be generated by the meshing algorithm.

▼ Mesh Attributes

Each geometric entity has mesh attributes which specify information such as meshing scheme, discretization density, discretization distribution (equal or biased), and element type. Unless otherwise specified by the user, the default meshing attributes listed in Table 5-1 will be used.:

Table 5-1 Default Meshing Attributes

<i>Geometric Entity</i>	<i>Default Attributes</i>		
	<i>Scheme</i>	<i>Element Type</i>	<i>Intervals</i>
Curve	Equal	2-node Bar	1
Surface	Map	4-node Quadrilateral	—
Volume	Map	8-node Hexahedron	—

Meshing Schemes

The mesh scheme attribute specifies the meshing algorithm that will be used to generate the mesh on each geometric entity. Note that the meshing scheme can be specified independently for curves, surfaces, and volumes; however, the meshing schemes for all surfaces on a volume must be compatible with the meshing scheme specified for that volume. For example, the

Project, Translate, and Rotate volume meshing schemes require that some of the volume surfaces be meshed using the Mapping scheme. The currently supported meshing schemes are listed in Table 5-2.

Table 5-2 Valid Meshing Schemes for Curves, Surfaces, and Volumes

<i>Scheme</i>	<i>Description</i>
Curve Meshing Schemes	
Equal	linear distribution of nodes along a curve based on arc length of the curve (default)
Biased	the distribution of nodes along a curve by biasing the nodal positions toward one of the curve ends given a growth factor
Surface Meshing Schemes	
Map	standard surface mapping transformation [7] (default)
Pave	advancing front algorithm for general surfaces including those with holes [1]
Triangle	meshing primitive for three-sided regions
Volume Meshing Schemes	
Map	standard volumetric mapping transformations [7] (default)
Project	2&1/2D Sweeping Algorithm—general purpose
Translate	2&1/2D Sweeping Algorithm—along a vector
Rotate	2&1/2D Sweeping Algorithm—about a central axis
Plaster	Research algorithm for automatic hexahedral volume meshing
Weave	Research algorithm for automatic hexahedral volume meshing

Interval Specification

Interval settings control the discretization density of the generated mesh. The number of intervals, or discretizations, can be set on a body, volume, surface, or curve. A related setting, the interval size, specifies the *length* of element edges on a curve, rather than the number of intervals. An advantage of the interval size option is that consistent element sizes can be specified throughout the mesh.

Element Types

CUBIT supports several element types, including bars, beams, quadrilaterals, shells, and hexahedrons. Two- and three-node bar and beam elements; four-, eight-, and nine-node quadrilateral and shell elements; and eight-, twenty-, and twenty-seven node hexahedral elements are available. Multiple element types can be used in a single CUBIT model. The **Block** commands are used to set element types. These are described in “Element Block Specification” on page 118.

AIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202

Centimeter

Inches

MANUFACTURED TO AIM STANDARDS
BY APPLIED IMAGE, INC.

2 of 2

Note: The type of elements to be generated on a geometric entity must be specified prior to meshing the geometric entity unless the defaults listed in Table 5-1 are desired.

▼ Curve Meshing

Curve meshing discretizes the curve, creating nodes and edges using the Mesh Dialog Box shown in Figure 5-1. For curve meshing, the **Geometry Type** must be set to **curve**. During curve meshing the user controls the density of nodes/edges (or intervals) along the curve and the relative spacing or bias of the nodes along the edge.

Node Density

The density of edges along curves is specified by setting the actual number of intervals or by specifying a desired average interval size. When the actual *number* of intervals is specifically set (curve by curve) by the user, these intervals are designated as being “hard set” and are never adjusted by the meshing algorithms—even when such an adjustment may produce a better mesh. It is assumed that when the interval *size* rather than an actual *number* of intervals is specified, the user is specifying an approximate number of edges to be placed on a curve rather than an absolute number. The meshing algorithms in CUBIT will then make minor adjustments to the number of intervals to insure an even number around the edge of a surface¹ and may also modify intervals so the chosen algorithm will have a higher probability of success. The number of intervals on a curve will not be adjusted after that curve has been meshed, either explicitly or as the result of meshing a surface or volume containing that curve. Intervals can be changed if the existing mesh is first deleted. The number of intervals or interval size can be explicitly set curve by curve, or implicitly set by specifying the intervals or interval size on a surface or volume containing that edge. For example, setting the intervals for a volume sets the intervals on all curves in that volume. The default number of intervals is 1. Note that if higher-order elements are being generated, the number of intervals and interval size refer to the edge of an element, not necessarily to the spacing of nodes along that edge.

When using the GUI to specify the number of intervals, the **Interval** button must be set to **Number** and an integer supplied in the **Value** dialog box. *To actually apply this interval setting to the geometry, the **Apply** button must be pushed.* When setting the interval size, the **Interval** button must be set to **Size** and an appropriate size entered into the **Value** dialog box. *Again, to actually apply this interval size to the curve, the **Apply** button must be pushed.*

The equivalent commands to specify the number of intervals in the command line interface are:

{curve|surface|volume|body} <range> interval <intervals>

where **range** may be a single integer or a range of integers. Interval size may be specified in the command line interface using similar commands:

{curve|surface|volume|body} <range> size <interval_size>

1. Meshing a surface or volume with an all-quadrilateral or all-hexahedral mesh requires an even number of intervals around the boundary of that surface or volume.

Figure 5-1 Curve Meshing With The GUI Mesh Dialog Box

Relative Element Edge Lengths

The relative length of element edges along a curve is specified using the curve scheme. Two curve schemes are currently supported: **equal** and **bias**. The **equal** scheme generates elements with equal length edges along the curve. The **bias** scheme requires the specification of a *bias factor* which designates a geometric progression of element edge lengths along the edge. For

example, a bias factor of 0.9 will make (as much as possible) each edge along the curve 0.9 times the length of the previous edge, starting at the first vertex¹. The default bias factor is 1.0.

When using the GUI to specify the scheme, either the **Equal** or **Bias** radio button must be chosen. The curve scheme Map is equivalent to equal. If **Bias** is chosen, a bias factor is supplied in the **Growth Factor** dialog box. To actually apply this scheme setting to the geometry, the **Apply** button must be pushed. The command used to specify the curve scheme in the command line interface is:

```
curve <range> scheme {equal | bias} <factor>
```

The **factor** must be provided if using the bias scheme. To reverse the bias direction of a curve in the GUI, use the **Rev. Bias** (reverse bias) button on the curve meshing dialog. The command used to reverse the bias in the command line interface is:

```
curve <range> reversebias
```

Reversing the curve bias using this command is equivalent to setting a bias factor equal to the inverse of the original bias factor.

Curvature Based Node Placement

A density of nodes based on the curvature of the curve is also available in CUBIT, but currently has not been released to end users. This capability provides more nodes where the curve is changing shape significantly, thus allowing a better capture of the geometry of the curve. When available, the command syntax for this capability will be:

```
{curve | surface | volume} <range> Scheme Curvature
```

Field Function Based Node Placement

The ability to specify the number and location of nodes based on a general field function is also available in CUBIT, but has not been released to end users. With this capability the node locations along a curve can be determined by some field variable (e.g. an error measure). This provides a means of using CUBIT in adaptive analyses. See "Field Function Based Surface Meshing" on page 104 for an example of this capability.

Meshing the Curve

Once the appropriate interval and scheme settings have been made, the curve can be meshed. If using the GUI, the **Mesh Now** button is pushed to mesh the curve.

Note: The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.

If using the command interface, the appropriate command is:

```
mesh curve <range>
```

1. The first vertex of a curve can be determined with the list curve <id> command.

The resulting mesh will be drawn on the screen in the mesh color designated for the curve. Figure 5-2 shows the result of meshing two edges with equal and bias schemes.

Figure 5-2 Equal and biased curve meshing

▼ Surface Meshing

Surface meshing discretizes a surface into nodes, edges, and faces. When meshing a surface, the bounding curves of the surface are first meshed (if not already meshed). The nodes on those curves are then used as the initial data for the surface meshing. Surface meshing algorithms include mapping, primitives, and paving, and a technique to apply boundary layers, or rows of aspect-controlled elements, to the surface before using the chosen algorithm.

When using the GUI, the portion of the **Mesh** dialog, from the **Generate** option shown in Figure 5-3 is used for surface meshing. The **Geometry Type** must be set to **Surface**. The Node density and relative spacing along the curves which bound the surface can be set as explained in “Curve Meshing” on page 98.

Scheme Designation

The algorithm to be used for surface meshing is designated as the scheme of the surface. Currently supported surface mesh schemes are **Map** (mapping algorithm), **Pave** (paving algorithm), and **Triangle** (triangle primitive). Each of these algorithms are briefly described below. The default scheme for a surface is **Map**.

When using the GUI, the **Scheme** can be set by pushing one of the highlighted radio buttons, **Map**, **Pave**, or **Triangle**. When using the command line, the scheme is set with the command

```
surface <range> scheme {map | pave | triangle}
```

Mapping

The surface mapping capability in CUBIT is based on standard mapping transformations [7]. The transformations work well and yield high quality meshes in regions with roughly parallel

Figure 5-3 Surface Meshing with the GUI Mesh Dialog

opposing sides. The surface may have any number of curves defining the sides and still produce a nice mapped mesh. Figure 5-4a illustrates a mapped mesh on a Bezier surface using two biased and two equal curve meshes on the region's boundary.

Paving

Paving (see reference [1]) allows the meshing of an arbitrary three-dimensional region with quadrilateral elements. The paver supports interior holes and arbitrary boundaries. It also allows for easy transitions between dissimilar sizes of elements. Figure 5-4 illustrates a paved mesh on the same surface paved with mapping in the same figure, using the same discretization of the boundary curves.

Primitives

Several basic shapes can be meshed as primitives using an internal decomposition technique to decompose the shape into mappable segments. The triangle primitive is currently the only surface meshing primitive available in CUBIT.

• Triangle Primitive

The **triangle** scheme indicates that the region should be meshed as a triangle. The definition of the triangle is general in that surfaces containing 3 natural corners can often be meshed successfully with this algorithm. For instance, the surface of a sphere octant is handled nicely by the triangle primitive. The algorithm requires that there be at least 6 intervals (2 per side) specified on the curves representing the perimeter of the surface and that the sum of the intervals on any two of the triangle's sides be at least two greater than the number of intervals on the remaining side. Figure 5-5 illustrates a triangle mesh on a 3D surface.

Figure 5-4 Mapped and paved surface meshing

Figure 5-5 Triangle primitive mesh

Curvature Based Surface Meshing

Meshing a surface with element sizes based on the curvature of the surface is available in CUBIT, but currently has not been released to end users. This capability provides smaller faces where the surface is curving significantly, thus allowing a better capture of the geometric shape of the surface. When available, the command syntax for this capability will be:

{curve | surface | volume} <range> Scheme Curvature

Field Function Based Surface Meshing

The ability to specify the size of elements based on a general field function is also available in CUBIT, but has not been released to end users. With this capability the desired element size can be determined using a field variable (e.g. an error measure) and paving. This provides a means of using CUBIT in adaptive analyses. Figure 5-6 depicts a plastic strain metric which was

Figure 5-6 Plastic strain metric

generated by PRONTO-3D [18], a transient solid dynamics solver, and recorded into an ExodusII data file. When the file is read back into CUBIT, the paving algorithm can be driven by the function values at the original node locations, resulting in an adaptively generated mesh [19]. Figure 5-7 depicts the resulting mesh from this plastic strain objective function.

Figure 5-7 Adaptively generated mesh

Boundary Layer Tool

The Boundary Layer meshing scheme is an algorithm designed to insert rows of elements around the boundary of a surface before meshing the interior. The aspect ratios of these elements can be carefully controlled. This capability is specifically designed for fluid simulations involving a boundary layer. With this tool, the total boundary layer thickness and relative thicknesses of each of the rows can be specified.

A boundary layer is specified as a set of parameters, with a boundary layer ID. This set of parameters is then attached to curve/surface pairs in the geometry. Thus a curve may have a row of boundary layer elements next to it on one of the surfaces it bounds, but not on another. A boundary layer is currently defined using a combination of four of the five possible parameters. These parameters are shown in Figure 5-8.

Figure 5-8 Boundary Layer Parameters

- **Sublayer Depth.** This parameter specifies the physical depth of the sublayer of elements. This is an optional parameter. If it is specified a sublayer of elements (not counted in the “number of layers” parameter) is added.
- **First Layer Depth.** This parameter specifies the physical depth of the first layer of elements. This is a required parameter.
- **Growth Factor.** This parameter specifies the relative difference of each subsequent layer’s depth. For instance, a factor of 1.2 makes each layer 1.2 times the preceding layer’s depth. This is an required parameter unless the second layer depth is specified.
- **Total Depth.** This parameter specifies the total depth of all the boundary layers. Either the total depth or the number of layers must be specified.
- **Number of Layers.** This parameter specifies the total number of layers in the boundary layer. Either the number of layers or the total depth must be specified.

When using the GUI, the **Boundary Layer** menu item under the Mesh menu will display the dialog shown in Figure 5-9. The **Boundary Layer ID** sets the boundary layer either to be defined or attached to a curve/surface pair. Specifying the **Surface ID** and **Curve ID** sets the curve/surface pair for the application of the boundary layer. The **Curve ID** is set in the picker window at the bottom of the dialog. If the user only wishes to define a boundary layer, and not apply it to any particular curve/surface pair, these two parameters need not be filled in. The **SubLayer Depth**, **First Layer Depth**, **Growth Factor**, **Total Depth**, and **Number of Layers** are the parameters that may be filled in to define a new boundary layer. These must be supplied in the combinations described above. If the user is simply attaching an existing boundary layer to a curve/surface pair, these parameters should not be used. Pushing the **Apply** button performs the definition and/or attachment of the boundary layer to the curve/surface pair.

Geometry Type:

Surface ID

Boundary Layer ID

SubLayer Depth

First Layer Depth

Growth Factor

Total Depth

Number of Layers

Current Selection(s):

☐ Picking Enable

Enter a selection: **To**

Figure 5-9 Boundary Layer Dialog Box

When using the command line, a boundary layer is set with the command

```
BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Total Depth <depth> [Sublayer <depth>]
```

```
BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Layers <count> [Sublayer <depth>]
```

The boundary layer is attached to curve/surface pairs with the command:

```
BoundaryLayer <layer_id> Surface <range> Curve <range>
```

Meshing the Surface

Once the desired scheme has been chosen, and any boundary layers for the surface defined and attached, the surface can be meshed. If the user wishes to receive a different element type than the default (four node quads) for surface meshing, this specification needs to be set prior to

creating any surface meshes. If using the GUI, the **Mesh Now** button on the **Mesh Control** dialog box shown in Figure 5-3 is pushed to mesh the surface(s).

Note: *The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.*

If using the command interface, the appropriate command is:

mesh surface <range>

The resulting mesh will be drawn on the screen in the mesh color designated for the surface.

▼ Volume Meshing

Volume meshing discretizes the volume into nodes, edges, faces, and hexahedral elements. When meshing a volume, the bounding surfaces of the volume are first meshed (if not already meshed). Available volume meshing algorithms are **mapping**, **project**, **translate**, **rotate**, **plaster**, and **whisker weaving** (**plaster** and **whisker weaving** are currently under development and as such are not recommended for production use). When using the GUI, the Mesh Control option of the **Mesh** menu button will produce the dialog box shown in Figure 5-10, which when **Geometry Type** is set to **Volume** can be used to control volume meshing. Node density and relative spacing along the curves which bound the surfaces of the volume can be set as explained in “Curve Meshing” on page 66.

Figure 5-10 Volume Meshing with the GUI Mesh Dialog

Scheme Designation

The algorithm to be used for volume meshing is designated as the scheme of the volume. Currently, valid schemes are:

Map Create mesh using mapping transformations.

Project Create mesh by projecting the mesh from one surface to another.

Translate Create mesh by translating the mesh from a source surface along the vector from the source surface to the target surface.

Rotate Create mesh by rotating about an axis from the source surface to the target surface.

Plaster Fill the volume in a free meshing inward approach—currently being researched.

Weave Attempt to generate whisker weaving sheets to fill the volume—currently being researched.

Each of these algorithms are briefly described below. The default scheme for a volume is **Map**. When using the GUI, the **Scheme** can be set by pushing one of the highlighted radio buttons, **Map**, **Project**, **Translate**, **Rotate**, **Plaster**, or **Weave**. When using the command line, the scheme is set with the commands

```
volume <range> scheme {map | weave | plaster}
```

```
volume <range> scheme {project | translate | rotate} Source <id> Target <id>
```

Mapping

The volume mapping capability in CUBIT is based on the standard mapping transformations [7] discussed in the surface meshing section. The volume transformations work well and yield high quality meshes unless the volume exhibits large concave sections on its bounding surfaces. This tendency is discussed in detail in the mapping transformation reference.

Sweeping (Project, Translate, and Rotate)

The CUBIT volume sweeping capability is divided into three algorithms (Project, Translate, and Rotate) which each generate a volume mesh by extruding hexahedrons from a previously meshed source surface to a topologically similar target surface. Topologically similar includes relationships such as a rectangular surface being extruded into an elliptical surface as long as the two surfaces contain the same number of boundaries or loops¹.

The geometric requirements for a sweeping operation are that the volume be “2 and 1/2 D,” or extrudable. This requirement is typically satisfied if the surfaces linking the source surface and target surface can be meshed with compatible mapping transformations (See “Mapping” on page 101.), where the “compatible” qualifier means that the edges linking the source surface to the target surface have the same number of intervals. The source surface may be meshed using any of the meshing methods described in “Surface Meshing” on page 101. The smoothed topology of the source surface mesh will be reproduced on the target surface unless the target surface is already meshed. In this case, the target surface mesh must have the same topology and connectivity as the source target mesh. It is much more efficient to let the sweeping meshing algorithms mesh the target source face if at all possible.

1. The number of loops on a surface refers to the number of boundaries it has. A surface always has at least one boundary, the set of curves which bound it externally. Some surfaces also have internal boundaries, or loops, in the form of holes.

The procedure for the sweeping volume mesh generation algorithms is as follows: first the attributes (interval settings, element type, etc.) of the volume should be set, and then the surfaces which will act as the source and target must be selected. When the command to mesh the volume is executed, the sweeper will mesh the source surface, and then the linking surfaces. The sweeping algorithms will then project a layer at a time, progressing through the unmeshed volume. The difference between the three sweeping algorithms is the method used to project the nodes and elements from one layer to the next. These details will be discussed in the following sections.

• Project

The **project** sweeping algorithm is a modified version of the plastering hex element projection. The sweep path can be completely general. The nature of the swept region can also be general in that it can contain draft angles and non-symmetric transformations. Figure 5-11 displays swept meshes involving mapped and paved source surfaces. The project algorithm can also

Figure 5-11 Project Volume Meshing

handle multiple surfaces linking the source surface and the target surfaces. An example of this is shown in Figure 5-12. Note that for the multiple surface meshing case, the interval requirement is that the total number of intervals along each multiple edge path from the source surface to the target surface must be the same for each path.

The project algorithm proceeds by determining a "projection node" for each node on the boundary of the current "layer." A node's projection node is the node directly "above" (in the sense of up being closer to the target surface than the source surface). An approximate planar surface is then generated through these nodes. For each node interior to the boundary of the volume, an average "projection vector" is calculated by determining which neighboring nodes have existing projection nodes and averaging the vector from these nodes to their projection nodes. The interior nodes are then calculated by projecting from the interior nodes along the average projection vector to the approximate planar surface calculated earlier. The interior nodes are ordered in a manner to maximize the number of neighboring nodes with existing

Figure 5-12 Multiple Surface Project Volume Meshing

projection nodes. This process is repeated for each interior node on the current layer. After all projection nodes have been created, a new layer of hexes is created and smoothed. The process then repeats for the next layer.

If the approximate planar surface does not closely match the surface defined by the boundary projection nodes, the interior projection nodes are created simply by projecting along the average projection vector; the intersection with the planar surface is not calculated.

The project algorithm is very general in that it can create a mesh on almost any extrudable volume; however, this generality has some disadvantages in that it does not use any global information about the actual generation of the volume. It simply projects a layer at a time in moving from the source surface to the target surface. Because of this, and the smoothing that is done after each layer is created, features that are present in the source surface mesh sometimes tend to get smoothed out or smeared by the time the mesh reaches the target surface mesh. The project algorithm is also slower and requires more memory than the translate and rotate algorithms since it must calculate a local projection for each node and maintain the information required by the smoothing algorithms.

• **Translate**

The **translate** sweeping algorithm is a more restricted version of the **project** algorithm. It is used when the source surface and target surface have exactly the same geometry and are parallel. If it is possible to translate the source surface along a vector and have it completely overlay the target surface, this algorithm can be used.

The **translate** algorithm proceeds by calculating the vector from the source surface to the target surface. The thickness of each layer is then calculated as the distance from a boundary node on that layer to that node's projection node. This distance is the same for each node on the current layer since the source surface and the target surface (and therefore, each layer) are parallel. Each interior node is then projected that thickness along the vector. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

• Rotate

The **rotate** sweeping algorithm is also a restricted version of the **project** algorithm. It is used when the source surface and target surface are exactly the same and are connected by a conic or toroidal surface. If it is possible to rotate the source surface about a single axis and have it completely overlay the target surface, this algorithm can be used. This algorithm cannot be used if the rotation axis contacts either the source surface or the target surface, that is, there must be a hole through the center of the generated mesh.

The **rotate** algorithm proceeds by calculating the axis of rotation from the source surface to the target surface. The thickness of each layer is calculated from the amount of rotation from a boundary node on that layer to that node's projection node. This rotational distance is the same for each node on the current layer. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

Plastering

Plastering uses the discretized surface and begins to lay elements into the interior of the volume. This continues until the volume fills, with adjustments made to the exterior surface mesh as deemed necessary. This algorithm is currently under development and not suggested for use although it may be tested if desired. It should currently perform well for blocky structures where the surface mesh will form a valid boundary for an interior hex mesh. Some examples of these structures are shown in Figure 5-13. These structures allow very straightforward hex element

Figure 5-13 Plastering Examples

connectivity and do not contain any irregular nodes (nodes that are shared by other than four element edges in a given layer).

Whisker Weaving

This meshing algorithm is in a very preliminary stage of development and will not be described in this version of the manual.

Meshing the Volume

Once the desired scheme has been chosen, the volume can be meshed. If the user wishes to receive a different element type than the default (eight-node hexahedrons) for volume meshing, this specification needs to be set prior to creating any volume meshes. If using the GUI, the **Mesh Now** button is pushed to mesh the curve(s).

Note: *The **Apply** button must be pushed before the settings shown in the dialog are stored on the geometry and used when the **Mesh Now** button is pushed.*

If using the command interface, the appropriate command is:

mesh volume <range> | All

mesh body <range> | All

The resulting mesh will be drawn on the screen in the mesh color designated for the volume.

▼ Mesh Duplication

If the geometry to be meshed was generated using the body copy command explained in “Copy Bodies” on page 52, then the mesh from the original geometry can be copied directly to the new geometry using the command

copy mesh {volume|surface} <id1> to {volume|surface} <id2>

This functionality is not yet available in the GUI version of CUBIT.

▼ Mesh Editing

A limited capability to modify portions of a mesh is provided to allow the user to make judgements about the level of smoothing required for volumetric meshes. Since the models to be meshed vary widely, and since CUBIT does not contain a sophisticated geometry recognizer, CUBIT is unable to make decisions regarding the type of smoothing algorithm to employ. Certain algorithms work well for classes of problems and fail for others. There seems to be no perfect smoothing algorithm, therefore the decision of which type of smoother to use is left to the user.

Smoothing

Surface

Surface smoothing algorithms currently consist of a variety of equipotential stencils. The nature of equipotential smoothers is one of weight equalization between adjacent nodes. For a generic, area, or Jacobian-based weighted smooth, this is roughly similar to equalizing areas between adjacent elements. The techniques behave well for regular or irregular grids on non-periodic surfaces, but are not yet released for periodic surfaces such as cylinders, spheres, (some) nurbs, and tori. Smoothing is implemented like the meshing, where the scheme is set first and the

action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a surface is as follows:

**Surface <range> Smooth Scheme Equipotential [Fixed]
[Weight {Jacobian | Area | Inverse [Area]}]**

If the **Weight** keyword is not specified, a **Generic** weighting is used by default. The **Fixed** keyword forces the nodes lying on the bounding curves of a surface to remain stationary instead of “floating” along the equation of the curve until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the surface mesh.

To smooth a surface based on the previously set scheme, the following command is used:

Smooth Surface <range>

If no scheme has been set, the Equipotential scheme is used by default.

Volume

Two volume smoothing algorithms are currently available in CUBIT. The first type of user controlled smoother is the length-weighted Laplacian. This smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement [8]. Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions⁸.

The second type of smoother is a variation of the equipotential [8] algorithm that has been extended to manage non-regular grids [9]. This method tends to equalize element volumes as it adjusts nodal locations. The advantage of the equipotential method is its tendency to “pull in” badly shaped meshes. This capability is not without cost: the equipotential method may take longer to converge or may be divergent. To impose an equipotential smooth on a volume, each element must be smoothed in every iteration—a typically expensive computation. While a Laplacian method can complete smoothing operations with only local nodal calculations, the equipotential method requires complete domain information to operate.

Smoothing is implemented like the meshing, where the scheme is set first and the action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a volume is as follows:

**Volume <range> Smooth Scheme
{Laplacian | Equipotential} [Fixed]**

The **Fixed** keyword force the nodes lying on the bounding surfaces of a volume to remain stationary instead of “floating” along the equation of the surface until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the volume mesh.

To smooth a volume based on the previously set scheme, the following command is used:

Smooth Volume <range>

If no scheme has been set, the **Equipotential** scheme is used by default.

While future objectives include investigation into weighting schemes to explicitly control mesh flow according to user-defined field or element functions, a simple hex weighting function exists which demonstrates the potential of the equipotential smoothers. This command applies a user specified weight to a group of hex elements, in this case, the elements which contain a mesh face

which belongs to a specified geometric surface. By adjusting the weight and running the smoother, one can expand or compact the elements being weighted. This capability may be used eventually to control adaptive hex element meshing, and to assist the free-form three-dimensional volumetric algorithms in their efforts to maintain element quality. The command syntax for applying the hex element weights is as follows:

Weight Hexes Surface <range> <weight>

Accessing Smooth Functions in the GUI

The **Smooth** menu item is located in the **Mesh** menu. The SmoothDialog for both surface smoothing and volume smoothing is shown in Figure 5-14. Each radio button corresponds to a smoothing command described in the previous two sections. The bottom portion of the SmoothDialog is a standard picker configuration described in “Picker Window” on page 54. The geometry type, either **Surface** or **Volume**, is specified in the **Geometry Type** option menu, and specific entities of this type are selected in the PickerWindow. After selecting the **Smoothing Method**, click **Apply** to perform the smoothing or **Cancel** to abort the smoothing.

Figure 5-14 Smooth Surface and Smooth Volume Dialog Boxes

Node Repositioning

The user can reposition nodes appearing in the same nodeset using the **NodeSet Move** command. Moves can be specified using either a relative displacement or an absolute position. Nodes cannot be moved separately by the user, except as part of the smoothing process. The command to reposition nodes in a nodeset is:

```
nodeset <id> move <x_distance> <y_distance> <z_distance>
nodeset <id> move to <x_position> <y_position> <z_position>
```


The first form of the command specifies a relative movement of the nodes by the specified distances and the second form of the command specifies absolute movement to the specified position.

Mesh Deletion

The **Delete All** menu item in the **Mesh** menu permanently removes all mesh entities from the model and resets the mesh entity identification counters. A warning dialog will appear requesting a confirmation of this action. The command line syntax is:

delete mesh

Several additional commands are available through the command line version of CUBIT. Partial delete capabilities exist for volumes, surfaces, curves, and vertices. Only the mesh which is owned by or is dependent on the specified geometry will be removed.

The routines are intelligent in the respect that if a mesh delete command is executed for a surface, curve, or vertex which belongs to one (or more) fully meshed volumes, CUBIT will delete all mesh entities which must be deleted as a result: for example, if one node on the vertex is deleted, then the mesh which lies on the connected curves to that vertex is incomplete and will be deleted, and then the surface mesh which relied on the curves must also be deleted, and finally the interior hex elements within the volume. The command syntax for these commands is as follows:

delete mesh volume <range>

delete mesh surface <range>

delete mesh curve <range>

delete mesh vertex <range>

In the GUI version, selecting the **Delete** option from the **Mesh** menu will display the Mesh Delete dialog box shown in Figure 5-15. This dialog box contains the standard picker window. After the correct geometry type is specified (Vertex, Curve, Surface, or Volume), and the entity IDs are selected, either by picking or by entering the values, selecting the Apply button will display the mesh delete warning dialog. The deletion will be performed if OK is selected.

Figure 5-15 Mesh Delete and Mesh Delete Warning Dialog Boxes

Import Mesh

A limited capability exists to read in a previously created mesh from an existing ExodusII file and associate the mesh (retaining full functionality for additional meshing or smoothing operations which depend on the imported mesh) with matching geometry in the CUBIT model. This command is useful for continuing a previous meshing problem at the point it was terminated rather than regenerating the entire mesh from the beginning. For example, if one is trying to create a difficult volumetric mesh, which can only be attempted after complex surface meshes are created (such as paving a large nurb surface), then the surface can be meshed once and saved to an ExodusII file created with a surface element block. Then if multiple volumetric meshing runs are required to attain the best possible mesh, the user need only read in or create the geometry (through a journal file from the original session), read in the surface mesh, and immediately commence volume meshing. The geometry (volume) which matches the original mesh must be present in the model for the command to work. The command line syntax is:

Import Mesh '<exodusII_filename>' Block <block_id> Volume <volume_id>

This command is also accessible in the GUI version as the **Import > Mesh** option under the **File** menu.

Chapter 6: Finite Element Model Definition and Output

- ▼ Finite Element Model Definition...117
 - ▼ Element Block Specification...118
- ▼ Boundary Conditions—Nodesets and Sidesets...120
 - ▼ Setting the Title...121
- ▼ Exporting the Finite Element Model...121

Chapter 6 describes the techniques used to complete the definition of the finite element model and the commands to export the finite element mesh to an Exodus database file. The definitions of the basic items in an Exodus database are briefly presented, followed by a description of the commands a user would typically enter to produce a customized finite element problem description.

▼ Finite Element Model Definition

Sandia's finite element analysis codes have been written to transfer mesh definition data in the Exodus [6] file format. CUBIT is one code in a suite of computer codes that supports the Exodus format for the pre- and post-processing of finite element analyses [14]. Since CUBIT is dedicated to mesh generation, the resulting database exported during a CUBIT session is actually a Genesis database file. A Genesis file is a subset of an Exodus file containing the problem definition only, i.e., no analysis results are included in the database.

A Genesis database consists of the following basic entity types: Element Blocks, Nodesets, and Sidesets.

Element Blocks

Element Blocks (also referred to as simply, *Blocks*) are a logical grouping of *elements* all having the same basic geometry and number of nodes. All elements within an Element Block are required to have the same element type. Access to an Element Block is accomplished through the use of a single integer ID known as the Block ID. Typically, Element Blocks are used by analysis codes to associate material properties and/or body forces with a group of elements.

Nodesets

Nodesets are a logical grouping of *nodes* also accessed through a single ID known as the Nodeset ID. Nodesets provide a means to specify load or boundary conditions on the CUBIT model.

Sidesets

Sidesets are another mechanism by which constraints may be applied to the model. Sidesets represent a grouping of *element sides* and are also referenced using an integer Sideset ID. They are typically used in situations where a constraint must be associated with element sides to satisfactorily represent the physics (e.g. a contact surface).

▼ Element Block Specification

Element blocks are the method CUBIT uses to group related sets of elements into a single entity. Each element in an element block must have the same dimensionality, type, number of nodes, and attributes. Element Blocks may be defined for volumes, surfaces, and curves. Multiple volumes, surfaces, and curves can be contained in a single element block. Element blocks are defined in the Block Identifier Dialog (Figure 6-1) which is accessed from the Block Identifier menu item. The **Block ID** and the **Geometry Type** to which this block ID is to be applied is

The dialog box is titled 'Block Identifier Dialog'. It contains the following elements:

- Geometry Type:** A dropdown menu currently set to 'Volume'.
- Block ID:** A text input field.
- Element Type Selection:** A grid of checkboxes for various element types:
 - Row 1: ☐ Shell4, ☒ Hex8, ☐ Quad4, ☐ Bar2
 - Row 2: ☐ Shell8, ☐ Hex20, ☐ Quad8, ☐ Bar3
 - Row 3: ☐ Shell9, ☐ Hex27, ☐ Quad9
- Attribute:** A text input field.
- Current Selection(s):** A section containing:
 - Buttons: 'Clear All' and 'Get All'.
 - A list box for selecting elements.
 - A label 'Pickings Element'.
- Enter a selection:** Two input fields labeled 'Enter a selection:' and 'To'.
- Buttons:** 'APPLY', 'Pick', 'HELP', and 'CANCEL' at the bottom.

Figure 6-1 Block Identifier Dialog

CHAPTER 6 Finite Element Model Definition and Output

specified in the top section of the dialog. The entities of this geometry type are specified using the normal picker window in the bottom section of the dialog. The center section of the dialog is used to specify the characteristics (element type and element attributes) to be applied to this element block. The element type desired is specified by selecting one of the radio buttons. The elements along the top row are basic linear elements and the subsequent rows are higher-order elements. The number following the element name denotes the number of nodes in the element. For example, the **Hex27** element is a 27-noded hexahedral element with mid-side, mid-face, and mid-volume nodes. The Shell and Bar elements require the specification of an **Attribute**¹ value which defines the thickness or cross-sectional area of the element for use in the finite element code². The attribute defaults to 1.0 if not specified. The commands to perform these functions using the command line are:

Block <block_id> {Curve | Surface | Volume} <range>

Block <block_range> elementtype <type>

Block <block_range> attribute <value>

Where the first command defines a **block_id** containing the specified geometric entities, the second command sets the **elementtype** for that block and the third command sets the **Attribute** for those elements.

Note: Higher order element blocks **must** be specified prior to meshing since additional nodes are inserted as part of the meshing process *only* if an Element Block's element type calls for them.

Default Element Types, Block IDs, and Attributes

The following defaults will be used unless otherwise specified or modified:

Volume: The default block ID will be set to the Volume ID and 8-node hexahedral elements will be generated.

Surface: The block ID will be set to 0 and 4-node shell elements will be generated.

Curve: The block ID will be set to 0 and 2-node bar elements will be generated.

Meshing could then be accomplished and the desired finite element model exported to the Genesis database.

Element Block Definition Examples

Multiple Element Blocks

Multiple element blocks can and almost always are combined when generating a finite element mesh. For example if the finite element model consists of a block which has a thin shell encasing the volume mesh, the following block commands would be used:

```
Block 100 Volume 1
Block 100 ElementType Hex8
```

1. Only zero or one attributes can be defined at the current time. This limitation will be removed in a future version.
2. The thickness and cross-section attribute values are not used internally in CUBIT, they are merely flags which are written to the EXODUS file to be used by subsequent codes. The documentation for the code which will be reading the EXODUS file should be consulted to determine the correct specification and use of the attribute value for the Shell and Bar elements.

CHAPTER 6 Finite Element Model Definition and Output

```
Block 200 Surface 1 to 6
Block 200 ElementType Shell4
Block 200 Attribute 0.01
Mesh Volume 1
Export Genesis 'block.g'
```

Which defines two element blocks (100 and 200). Element block 100 is composed of 8-node hexahedral elements and element block 200 is composed of 4-node shell elements on the surface of the block. The “thickness” of the shell elements is 0.01. The finite element code which reads the Genesis file (block.g) would refer to these blocks using the element block IDs 100 and 200. Note that the second line of the example is not required since **Hex8** elements are the default element type for a Volume element block.

Surface Mesh Only

If a mesh containing only the surface of the block is wanted, the first two lines of the example would be omitted and the **Mesh Volume 1** line would be changed to, for example, **Mesh Surfaces 1 to 6**.

Two-Dimensional Mesh

CUBIT also provides the capability of writing two-dimensional Genesis databases similar to FASTQ. The user must define element blocks that use the **Quad*** type elements and assign those element blocks to the appropriate surfaces in the model. For example

```
block 1 surface 1 to 4
block 1 elementtype quad4
```

In this case, it is important for users to note that a two-dimensional Genesis database will result. In writing a two-dimensional Genesis database, CUBIT *ignores* all z-coordinate data. Therefore, the user must ensure that the Element Block is assigned to a planar surface lying in a plane parallel to the x-y plane. Currently, the **Quad*** element types are the only supported two-dimensional elements. Two-dimensional shell elements will be added in the near future if required.

▼ Boundary Conditions—Nodesets and Sidesets

Boundary conditions such as constraints and loads are applied to the finite element model through nodesets and sidesets. Nodesets can be created from groups of nodes categorized by their owning volumes, surfaces, or curves. Nodes can belong to more than one nodeset. Sidesets can be created from groups of element sides or faces categorized by their owning surfaces or curves. Element sides and faces can belong to more than one sideset. Nodesets and Sidesets can be viewed individually through CUBIT by employing the **Draw Nodeset** and **Draw Sideset** commands.

Nodesets and Sidesets may be assigned to the appropriate geometric entities in the model using the following commands in the command line:

```
nodeset <nodeset_id> {curve | surface | volume | vertex} <range>
sideset <sideset_id> {curve | surface} <range>
```

When using the GUI version of CUBIT, nodesets and sidesets are specified by accessing their respective dialog boxes from the **Constraints** menu. The Nodesets menu item will display the

CHAPTER 6 Finite Element Model Definition and Output

Nodeset Dialog and the Sidesets menu item will display the Sideset Dialog. The top portion of both of these are shown in Figure 6-2; the bottom portion is a standard picker window. The

Figure 6-2 Nodeset and Sideset Dialogs

Geometry Type option menu can be set to **Body**, **Volume**, **Surface**, or **Curve**. The user-specified output identification number for the nodeset or sideset is entered in the **Nodeset ID** or **Sideset ID** text field. This is the number which will be used to identify this boundary condition in the exported EXODUSII file. The geometric entities to which this boundary condition is to be applied is then specified using the normal picking syntax.

▼ Setting the Title

The GUI version of CUBIT does not currently provide a means to set a title for the Genesis database. However, it may be accomplished by typing the command:

```
title '<title_string>'
```

CUBIT's parser requires that strings be enclosed in single quotes, for example **'This is a string'**.

▼ Exporting the Finite Element Model

A Genesis database can be output using the **Export** option of the **File** Menu. Only Exodus II format is currently supported. A file can also be output using the following command:

```
export genesis '<filename>'
```

The Export Genesis command automatically creates a *unique* Element Block for every volume that is meshed (assuming the user has not entered any Block commands overriding this default behavior). Users can selectively control which blocks are output to the Genesis file since Element Blocks will *not* be created for any volumes that are not meshed.

CHAPTER **6** Finite Element Model Definition and Output

Appendix A: Command Index

▼ Command Syntax...123

▼ Commands...123

In this section the commands available in CUBIT are listed in alphabetical order. There may be more than one command available under a given command definition, since some commands can be executed singly or over a range of objects.

▼ Command Syntax

Each command will be first listed by a command heading, which will be phrased in standard terminology and will typically be very close to the computer syntax for the particular command. Underneath the command heading, each variation of the command will be listed, to document commands which can be applied to single objects as well as a range of objects. At the end of each command heading is a page number cross-reference to the location in the main document where the command is documented.

Command listings will ask for four types of command arguments: arguments include integers (6, 1, 3, etc., these are typically id's), reals (1.4, 2.35, etc., typically floating point quantities such as angles, spatial coordinates), strings (jjrome.jou, part.g, etc.), and logicals (1, 0, on, off, etc.).

Detailed descriptions of most commands and their usage can be found within the main sections of this document. A few of the special purpose commands are not documented elsewhere and only their syntax is shown below. These commands will be fully-documented after the functionality and generality are improved.

▼ Commands

Animate Geometry page 70

Animate Geometry Volume <volume_id_range> Rotate <degree> About <axis>
Steps <number>

Animate Mesh page 70

Animate Mesh Volume <volume_id_range> Rotate <degree> About <axis>
Steps <number>

Animate Overlay page 70

Animate Overlay Volume <volume_id_range> Rotate <degree> About <axis>
Steps <number>

At	page 61
[View] At <X_coord> <Y_coord> <Z_coord>	
Block Attribute	page 119
BlockId <block_id_range> Attribute <attribute>	
Block Element Type	page 119
BlockId <block_id_range> ElementType <element_type_name>	
Block Geometry Type	page 119
BlockId <id> { volume surface curve } <block_id_range>	
Block Visibility	page 65
BlockId <block_id_range> Visibility { on off }	
Body Copy	page 86
Body <body_id_range> Copy [Move <x> <y> <z>]	
Body <body_id_range> Copy [Rotate <angle> About {x y z}]	
Body <body_id_range> Copy [Rotate <angle> About <x> <y> <z>]	
Body Geometry Visibility	page 65
Body <body_id_range> Geometry { on off }	
Body <body_id_range> Geometry Visibility { on off }	
Body Interval	page 98
Body <body_id_range> Interval <interval>	
Body Mesh Visibility	page 65
Body <body_id_range> Mesh { on off }	
Body <body_id_range> Mesh Visibility { on off }	
Body Move	page 87
Body <body_id_range> Move <x> <y> <z>	
Body Reflect	page 88
Body <body_id_range> Reflect {x y z}	
Body <body_id_range> Reflect<x> <y> <z>	
Body Rotate	page 88
Body <body_id_range> Rotate <angle> About {x y z}	
Body <body_id_range> Rotate <angle> About <x> <y> <z>	
Body Scale	page 87
Body <body_id_range> Scale <scale>	
Body Size	page 98
Body <body_id_range> Size <size>	
Body Visibility	page 65
Body <body_id_range> { on off }	
Body <body_id_range> Visibility { on off }	

BoundaryLayer	page 106
BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Total Depth <depth> [Sublayer <depth>]	
BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Layers <count> [Sublayer <depth>]	
BoundaryLayer Surface	page 106
BoundaryLayer <layer_id> Curve <curve_id_range> Surface <surface_id>	
Brick	page 78
[Create] Brick Width <width> [Depth <depth> Height <height>]	
Color Background	page 59
Color Background <color_name> Color Background <color_id>	
Color Block	page 67
Color Block <block_id_range> <color_name>	
Color Body	page 67
Color Body <body_id_range> <color_name> Color Body <body_id_range> <color_id>	
Color Body Geometry	page 67
Color Body <body_id> Geometry <color_name> Color Body <body_id> Geometry <color_id>	
Color Body Mesh	page 67
Color Body <body_id_range> Mesh <color_name> Color Body <body_id_range> Mesh <color_id>	
Color Node	page 67
Color Node <color>	
Color NodeSet	page 67
Color NodeSet <nodeset_id_range> <color>	
Color SideSet	page 67
Color SideSet <sideset_id_range> <color>	
Color Surface	page 67
Color Surface <surface_id_range> <color> Color Surface <surface_id_range> <color_id>	
Color Surface Geometry	page 67
Color Surface <surface_id_range> Geometry <color> Color Surface <surface_id_range> Geometry <color_id>	
Color Surface Mesh	page 67
Color Surface <surface_id_range> Mesh <color> Color Surface <surface_id_range> Mesh <color_id>	

Color Volume	page 67
Color Volume <volume_id_range> <color>	
Color Volume <volume_id_range> <color_id>	
Color Volume Geometry	page 67
Color Volume <volume_id_range> Geometry <color>	
Color Volume <volume_id_range> Geometry <color_id>	
Color Volume Mesh	page 67
Color Volume <volume_id_range> Mesh <color>	
Color Volume <volume_id_range> Mesh <color_id>	
Copy Mesh	page 112
Copy Mesh Surface <surface_id> To Surface <surface_id>	
Copy Mesh Volume <volume_id> To Volume <volume_id>	
Create Brick	page 78
[Create] Brick Width <width> [Depth <depth> Height <height>]	
Create Cylinder	page 78
[Create] Cylinder Height <height> Radius <radius>	
[Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>	
Create Frustum	page 80
[Create] Frustum Height <height> Major Radius <radius>	
Minor Radius <radius> [Top <top_radius>]	
[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
Create Prism	page 80
[Create] Prism Height <height> Sides <sides> Major <radius>	
Minor <radius>	
[Create] Prism Height <height> Sides <sides> Radius <radius>	
Create Pyramid	page 81
[Create] Pyramid Height <height> Sides <sides> Major <radius>	
Minor <radius> Top <radius>	
[Create] Pyramid Height <height> Sides <sides> Radius <radius>	
Create Sphere	page 81
[Create] Sphere Radius <radius>	
Create Torus	page 82
[Create] Torus Rad1 <R1> Rad2 <R2>
Curve	Undocumented
Curve <curve_id> [Vertex] <vertex_id> [Vertex] <vertex_id>	
Curve Interval	page 98
Curve <curve_id_range> Interval <interval>	
Curve Reverse Bias	page 100
Curve <curve_id_range> ReverseBias

Curve Scheme Curvature page 100

Curve <curve_id_range> Scheme Curvature

Curve Scheme Bias page 100

Curve <curve_id_range> Scheme Bias <growth_factor>

Curve Scheme Equal page 100

Curve <curve_id_range> Scheme Equal

Curve Size page 98

Curve <curve_id_range> Size <size>

Curve Type Undocumented

Curve <curve_id> Type <type> <vertex_id> <vertex_id>

Curve <curve_id> Type <type>

From <id> To <id> About <id>

Curve <curve_id> Type <type>

From <id> To <id> About <id> Sense <dir>

Cylinder page 78

[Create] Cylinder Height <height> Radius <radius>

[Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>

Delete Mesh page 115

Delete Mesh

Delete Mesh Vertex <vertex_id_range>

Delete Mesh Curve <curve_id_range>

Delete Mesh Surface <surface_id_range>

Delete Mesh Volume <volume_id_range>

Display page 58

Display

Draw Block page 66

Draw Block <block_id_range>

Draw Body page 66

Draw Body <body_id_range>

Draw Curve page 66

Draw Curve <curve_id_range>

Draw Edge page 66

Draw Edge <edge_id_range>

Draw Face page 66

Draw Face <face_id_range>

Draw Hex page 66

Draw Hex <hex_id_range>

Draw Loop Undocumented

Draw Loop <loop_id_range>

Draw Loop Circle	Undocumented
Draw Lcircle <lcircle_id_range>	
Draw Lcircle <lcircle_id_range> Color <colorName>	
Draw Node	page 66
Draw Node <node_id_range>	
Draw NodeSet	page 66
Draw NodeSet <nodeset_id_range>	
Draw SideSet	page 66
Draw SideSet <sideset_id_range>	
Draw Skeleton	Undocumented
Draw Skeleton	
Draw Surface	page 66
Draw Surface <surface_id_range>	
Draw Vertex	page 66
Draw Vertex <vertex_id_range>	
Draw Volume	page 66
Draw Volume <volume_id_range>	
Echo	page 56
Echo {on off}	
Exit	page 56
Exit	
Quit	
Export Genesis	page 121
Export Genesis <filename>	
From	page 62
[View] From <X_coord> <Y_coord> <Z_coord>	
Frustum	page 80
[Create] Frustum Height <height> Major Radius <radius>	
Minor Radius <radius> [Top <top_radius>]	
[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
Geometry Visibility	page 65
Geometry { on off }	
Geometry Visibility { on off }	
Graphics Autocenter	page 61
Graphics Autocenter {on off}	
Graphics Autoclear	page 61
Graphics Autoclear {on off }

Graphics Axis	page 61
Graphics Axis {on off}	
Graphics Border	page 61
Graphics Border {on off}	
Graphics Center	page 61
Graphics Center	
Graphics Clear	page 61
Graphics Clear	
Graphics Line Width	page 61
Graphics LineWidth <width>	
Graphics Mode	page 59
Graphics Mode FlatShade	
Graphics Mode HiddenLine	
Graphics Mode PolygonFill	
Graphics Mode Painters	
Graphics Mode SmoothShade	
Graphics Mode WireFrame	
Graphics Perspective	page 63
Graphics Perspective {on off}	
Graphics Perspective Angle	page 63
Graphics Perspective Angle <view_angle_in_degrees>	
Graphics WindowSize	page 59
Graphics WindowSize <width_in_pixels> <height_in_pixels>	
Graphics WindowSize Maximum	page 59
Graphics WindowSize Maximum	
Graphics Zoom	page 63
[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>	
Graphics Zoom Reset	page 64
[Graphics] Zoom Reset	
Hardcopy	page 69
Hardcopy '<filename>' [encapsulated postscript eps] [color monochrome]	
Help	page 74
Help	
Help <keyword>	
Hyperhelp	page 74
Hyperhelp <keyword> [<identifier>]	
Import Acis	page 84
Import Acis '<acis_filename>'

Import Fastq Import Fastq '<fastq_filename>'	page 85
Import Mesh Import Mesh <exodusII_filename> Block <block_id> Volume <volume_id>	page 116
Intersect Intersect <body_id> With <body_id>	page 89
Journal Off Journal Off	page 57
Label Label {on off}	page 68
Label All Label All { on off }	page 68
Label Body Label Body { on off }	page 68
Label Curve Label Curve { on off }	page 68
Label Edge Label Edge { on off }	page 68
Label Face Label Face { on off }	page 68
Label Geometry Label Geometry { on off }	page 68
Label Hex Label Hex { on off }	page 68
Label Mesh Label Mesh { on off }	page 68
Label Node Label Node { on off }	page 68
Label Surface Label Surface { on off }	page 68
Label Vertex Label Vertex { on off }	page 68
Label Volume Label Volume { on off }	page 68
List Body List Body <body_id_range>	page 71

List Curve	page 71
List Curve <curve_id_range>	
List Face	page 71
List Face <mesh_face_id_range>	
List Hex	page 71
List Hex <hex_id_range>	
List Node	page 71
List Node <node_id_range>	
List Surface	page 71
List Surface <surface_id_range>	
List Totals	page 71
List Totals	
List Vertex	page 71
List Vertex <vertex_id_range>	
List View	page 64
List View	
List Volume	page 71
List Volume <volume_id_range>	
Merge All	page 93
Merge All	
Merge Curve	page 94
Merge Curve <curve_id> With <curve_id>	
Merge Only Curves	page 94
Merge Only Curves	
Merge Only Surfaces	page 94
Merge Only Surfaces	
Merge Surface	page 94
Merge Surface <surface_id> With <surface_id>	
Merge Vertex	page 94
Merge Vertex <vertex_id> With <vertex_id>	
Mesh Body	page 112
Mesh Body <body_id_range>	
Mesh Body All	
Mesh Curve	page 100
Mesh Curve <curve_id_range>	
Mesh Curve All	

Mesh Surface	page 107
Mesh Surface <surface_id_range>	
Mesh Surface All	
Mesh Visibility	page 65
Mesh { on off }	
Mesh Visibility { on off }	
Mesh Volume	page 112
Mesh Volume <volume_id_range>	
Mesh Volume All	
Node Visibility	page 65
Node { on off }	
Node Visibility { on off }	
NodeSet Curve	page 120
NodeSet <nodeset_id> Curve <curve_id_range>	
NodeSet Move	page 114
NodeSet <nodeset_id> Move <X> <Y> <Z>	
NodeSet <nodeset_id> Move To <X> <Y> <Z>	
NodeSet Surface	page 120
NodeSet <nodeset_id> Surface <surface_id_range>	
NodeSet Vertex	page 120
NodeSet <nodeset_id> Vertex <vertex_id_range>	
NodeSet Visibility	page 65
NodeSet { on off }	
NodeSet Visibility { on off }	
NodeSet <nodeset_id_range> { on off }	
NodeSet <nodeset_id_range> Visibility { on off }	
NodeSet Volume	page 120
NodeSet <nodeset_id> Volume <volume_id_range>	
Pause	page 58
Pause	
Playback	page 58
Playback <journal_filename>	
Prism	page 80
[Create] Prism Height <height> Sides <sides> Major <radius>	
Minor <radius>	
[Create] Prism Height <height> Sides <sides> Radius <radius>	
Pyramid	page 81
[Create] Pyramid Height <height> Sides <sides> Major <radius>	
Minor <radius> Top <radius>	
[Create] Pyramid Height <height> Sides <sides> Radius <radius>	

Quit	page 56
Quit	
Exit	
Record	page 57
Record '<journal_filename>'	
Record Stop	page 57
Record Stop	
Reset	page 56
Reset	
Rotate	page 63
[Graphics] Rotate <degree> About {x y z}	
[Graphics] Rotate <degree> About Curve <curve_id>	
[Graphics] Rotate <degree> About Vertex <axis_start_vertex_id>	
Vertex <axis_end_vertex_id>	
SideSet Curve	page 120
SideSet <sideset_id> Curve <curve_id_range>	
SideSet Surface	page 120
SideSet <sideset_id> Surface <surface_id_range>	
SideSet Visibility	page 65
SideSet { on off }	
SideSet Visibility { on off }	
SideSet <sideset_id_range> { on off }	
SideSet <sideset_id_range> Visibility { on off }	
Smooth Surface	page 113
Smooth Surface <surface_id_range>	
Smooth Volume	page 113
Smooth Volume <volume_id_range>	
Sphere	page 81
[Create] Sphere Radius <radius>	
Subtract	page 89
Subtract <body_id> From <body_id>	
Surface Geometry Visibility	page 65
Surface <surface_id_range> Geometry Visibility { on off }	
Surface Interval	page 98
Surface <surface_id_range> Interval <intervals>	
Surface Mesh Visibility	page 65
Surface <surface_id_range> Mesh Visibility { on off }	
Surface Scheme Curvature	page 100
Surface <surface_id_range> Scheme Curvature

Surface Scheme Map	page 101
Surface <surface_id_range> Scheme Map	
Surface Scheme Pave	page 101
Surface <surface_id_range> Scheme Pave	
Surface Scheme Triangle	page 101
Surface <surface_id_range> Scheme Triangle	
Surface Size	page 98
Surface <surface_id_range> Size <intervals>	
Surface Smooth Scheme	page 113
Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]	
Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]	
Weight Jacobian	
Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]	
Weight Area	
Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]	
Weight Inverse [Area]	
Surface Visibility	page 65
Surface <surface_id_range> { on off }	
Surface <surface_id_range> Visibility { on off }	
Title	page 121
Title '<title>'	
Torus	page 82
[Create] Torus Major [Radius] <R1> Minor [Radius] <R2>	
Unite	page 89
Unite <body_id> With <body_id>	
View Up	page 63
[View] Up <X_coord> <Y_coord> <Z_coord>	
Version	page 56
Version	
Vertex	Undocumented
Vertex <id> <X_coord> <Y_coord> <Z_coord>	
Vertex Visibility	page 65
Vertex { on off }	
Vertex Visibility { on off }	
Video Initialize	page 69
Video Initialize	
Video Snap	page 69
Video Snap

View At	page 61
[View] At <X_coord> <Y_coord> <Z_coord>	
View From	page 62
[View] From <X_coord> <Y_coord> <Z_coord>	
View List	page 64
View List	
View Up	page 63
[View] Up <X_coord> <Y_coord> <Z_coord>	
Volume Geometry Visibility	page 65
Volume <volume_id_range> Geom { on off }	
Volume <volume_id_range> Geometry Visibility { on off }	
Volume Interval	page 98
Volume <volume_id_range> Interval <intervals>	
Volume Mesh Visibility	page 65
Volume <volume_id_range> Mesh { on off }	
Volume <volume_id_range> Mesh Visibility { on off }	
Volume Scheme Curvature	page 100
Volume <volume_id_range> Scheme Curvature	
Volume Scheme Map	page 108
Volume <volume_id_range> Scheme Map	
Volume Scheme Plaster	page 108
Volume <volume_id_range> Scheme Plaster	
Volume Scheme Project	page 108
Volume <volume_id> Scheme Project Source Surface <surface_id>	
Target Surface <surface_id>	
Volume Scheme Rotate	page 108
Volume <volume_id> Scheme Rotate Source Surface <surface_id>	
Target Surface <surface_id>	
Volume Scheme Translate	page 108
Volume <volume_id> Scheme Translate	
Source Surface <surface_id> Target Surface <surface_id>	
Volume Scheme Weave	page 108
Volume <volume_id_range> Scheme Weave	
Volume Size	page 98
Volume <volume_id_range> Size <intervals>	
Volume Smooth Scheme	page 113
Volume <volume_id_range> Smooth Scheme Laplacian	
Volume <volume_id_range> Smooth Scheme Equipotential [Fixed]

Volume Visibility

page 65

Volume <volume_id_range> { on | off }

Volume <volume_id_range> Visibility { on | off }

WebCut Body

page 92

Webcut Body <body_id> Face <face_id> [Vector <from_vertex> <to_vertex>]

Webcut Body <body_id> Vertex <vertex_1> Vertex <vertex_2>

Vertex <vertex_3> [Vector <from_vertex> <to_vertex>]

Weight Hexes

page 114

Weight Hexes Surface <range> <weight>

Zoom

page 63

[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>

[Graphics] Zoom Reset

Appendix B: Examples

▼ General Comments...137

▼ Simple Internal Geometry Generation...138

▼ Octant of Sphere...139

▼ Airfoil...141

▼ The Box Beam...142

▼ Thunderbird 3D Shell...145

▼ Assembly Components...148

The purpose of this Appendix is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a few examples on the use of CUBIT. Some examples also demonstrate the use of the ACIS test harness as well as other related programs. This Appendix is not intended to be a step-by-step tutorial.

▼ General Comments

CUBIT is based upon the ACIS solid modeling kernel. Solid models can be created within CUBIT or imported in the form of an ACIS geometry file¹. Current means of generating ACIS solid models external to CUBIT include:

- ACIS Test Harness
- FASTQ via the FASTQ to ACIS translator *fsqacs*²
- Aries[®] ConceptStation
- PRO/Engineer via a PRO/Engineer to ACIS translator

These examples show model construction using internal CUBIT geometry creation, the ACIS Test Harness, and the FASTQ translator. Those methods provide the capability of semi-automatically generating a mesh in batch mode in much the same manner as FASTQ [5], GEN3D [10], GREPOS [11], and GJOIN [12].

A CUBIT journal file is included for the examples shown in this appendix. ACIS journal files are also provided for the examples that require geometry generated by ACIS. The user can reproduce the examples interactively by simply entering each of the lines in the journal files as

-
1. ACIS typically adds the filename suffix ".sat" to the output files it writes in text format; therefore, these files are typically referred to as ".sat" files or "ACIS .sat" files. CUBIT cannot read binary ACIS files.
 2. The *fsqacs* users manual is reproduced in Appendix C, "Fsqacs: A FASTQ to ACIS Command Interpreter" on page 153

APPENDIX **B** Examples

commands to CUBIT or ACIS. The examples assume that the command line version of CUBIT will be used. The examples can also be run using the Graphical User Interface version of CUBIT; however, the details for doing this are not given in this appendix. The journal files included in the example are also distributed with CUBIT and they may be executed using the **Playback 'filename'** command.

The examples in this appendix each cover several of CUBIT's mesh generation capabilities. The CUBIT features exercised by each example are shown in Table B-1.

Examples	Geometry Features				Surface Meshing Features						Volume Meshing Features			
	Primitives	Booleans	Geometry Editing	Geometry Consolidation	Curve Bias	Curvature-based	Mapping	Paving	Triangle Tool	Boundary Layer Tool	Project	Translate/Rotate	Mapping	Plastering
Internal Geometry	x	x						x				x		
Sphere Octant	x	x	x	x				x	x		x	x		
Airfoil					x			x		x				
Box Beam				x			x							
Thunderbird								x						
Assembly Components				x				x			x			

Table B-1 CUBIT Features Exercised by Examples.

▼ Simple Internal Geometry Generation

This simple example demonstrates the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. It illustrates the **brick**, **cylinder**, **subtract**, **pave**, and **translate** commands and boolean operations. The geometry to be generated is shown in Figure B-1. This figure also shows the curve and surface labels specified in the CUBIT journal file. The final meshed body is shown in Figure B-2. The CUBIT journal file is:

```
# Internal Geometry Generation Example
Brick Width 10. Depth 10. Height 10.          # Create Cube
Cylinder Height 12. Radius 3.                 # Create cylinder through Cube
View From 3 4 5                               # Update viewing position
Display
Subtract 2 From 1          # Remove cylinder from cube--create hole
```



```

Display
Body 3 Size 1.0 # Default element size for model
Surface 10 Interval 10 # Change intervals on cylinder surface
Curve 15 to 16 Interval 20 # Change intervals around cyl. circ.
Surface 11 Scheme Pave # Front surface paved
Volume 3 Scheme Translate Source 11 Target 12 #Remainder
# of block will be meshed by
# translating front surface to back surface
Mesh Volume 3 # Create the mesh
Graphics Mode Hiddenline
Display # Hiddenline view of cube (Figure B-2)

```

The first two lines create a 10 unit cube centered at the origin and a cylinder with radius 3 units and height of 12 units also centered at the origin. The cylinder height is arbitrary as long as it is greater than the height of the brick. The **subtract** command then performs the boolean by subtracting the cylinder (body 2) from the block (body 1) to create the final geometry (body 3). The remainder of the commands simply assign the desired number of intervals and then generate the mesh. Note that since the cylindrical hole is a "periodic surface," there are no edges joining the two curves so the number of intervals along its axis must be set by the surface interval command. The steps required for generating this geometry and mesh using the Graphical User Interface are given in the Tutorial in Chapter 2.

Figure B-1 Geometry for Cube with Cylindrical Hole

▼ Octant of Sphere

This example also illustrates the internal geometry generation capabilities of CUBIT to generate an octant of a sphere. The procedure used is to generate the octant by intersecting a brick with a sphere. The octant is then split into two pieces—a central "core" and an outer "peel" which are both meshable using the sweeping schemes. This example uses the **sphere**, **brick**, **cylinder**, **intersect**, **copy**, **subtract**, **merge**, **pave**, **project**, and **rotate** commands.

The following annotated CUBIT journal file will generate the mesh shown in Figure B-3.

```

Sphere Radius 10. #Generate Sphere (Body 1)
Brick Width 12 Depth 12 Height 12 #Generate Cube (Body 2)
Body 2 Move 6. 6. 6. #Move Cube to Enclose Octant

```

Figure B-2 Generated Mesh for Cube with Cylindrical Hole

```

Graphics Mode SmoothShade
Display
Intersect 1 With 2
Display
Cylinder Height 22 Radius 3
Body 3 To 5 Copy

#Only way to see a sphere
#Generate Octant (Body 3)
#Generate Cylinder (Body 4)
#Copy Octant (Body 5)
#and Cylinder (Body 6)
#and another octant (Body 7)
#Create Core (Body 8)

Intersect 4 With 5
View From 1 2 3
Intersect 3 With 6
Subtract 8 From 7
Merge All

#
# End of Geometry Generation.
# "Core" is volume/body 9
# "Shell" is volume/body 10
#
volume 9 Size 0.5
Surface 33 Scheme Pave
Mesh Surface 33
volume 9 Scheme Project Source 33 Target 31
Mesh volume 9
Display

#
volume 10 Size 0.5
Surface 37 Scheme Pave
Mesh Surface 37
volume 10 Scheme Rotate Source 37 Target 40
Mesh volume 10
Display
Export Genesis 'Octant.gen'

#Pave end of core
#Generate core mesh
#Make sure it's there
#Make intervals agree for rotate
#Pave face of peel
#Generate Peel Mesh
#Write out the mesh

```

If the generated mesh should consist of one material, the **block** command could be used to merge the peel and core into a single material block. Note that during a boolean operation (unite, intersect, and subtract), the bodies used in that boolean are destroyed so it is sometimes necessary to create extra copies of a body prior to using them in a boolean operation. Also,

Figure B-3 Generated Mesh for Octant of Sphere

during boolean operations, many bodies are created and deleted and it is difficult to remember which bodies exist at certain times¹. It is recommended that comments be added to the journal file to make it easier to determine what is being done in the file.

▼ Airfoil

A simple two-dimensional airfoil is used in this example to demonstrate the use of the boundary layer tool and paving. The commands used to generate the geometry for this problem, using the ACIS Test Harness, are not included here. This example uses the **curve bias**, **boundarylayer** and **pave** commands. The CUBIT commands used to mesh this problem are:

```
# File: foil.jou
#
# Air Foil Example
#
journal off
Import Acis 'foil.sat'
View From 100 0 0 Up 0 0 1          # Set up View
AutoCenter On
Display
Volume 1 Interval 14                  # Set Meshing Parameters
Curve 4 Interval 24
Curve 2 Interval 24
Curve 5 To 6 Interval 18
Curve 6 Bias 1.1
Curve 5 Bias 0.909
BoundaryLayer 1 First Layer 0.5 Growth 1.3
BoundaryLayer 1 Surface 1 Curve 5 to 6
Surface 1 Scheme Pave
Mesh Surface 1                        # Create the Mesh
Display
```

1. The CUBIT Developers are very much aware of the problems this causes during the generation of complicated meshes and are implementing methods to permit user-defined naming of bodies and volumes. This capability relies on the persistent ID concept recently added to ACIS.

```
Graphics zoom .25 .4 .45 .6
geometry visibility off
display
geometry visibility on
```

The mesh generated by these commands is shown in Figure B-4. In this example, curves 5 and 6 (the curves used to define the shape of the airfoil) use biased interval spacing to place more elements towards the front of the airfoil. A boundary layer is designated on either side of the airfoil, which produces elements with high aspect ratios for several layers around the airfoil. The parameters to the **boundarylayer** command specify the depth of the first and second rows of elements, with the boundary layer growth factor inferred from these data. The paving scheme generates the mesh outside the boundary layer.

Figure B-4 Airfoil mesh generated using the boundary layer tool and paving.

▼ The Box Beam

A simple example using ACIS/CUBIT is the box beam buckling problem shown in Figure B-5. A description of an analysis which uses this type of mesh is found in Reference [15]. This example uses the **merge**, **nodeset** and **block** commands and the mapping mesh generation scheme.

The input file for the ACIS Test Harness for the box beam example is¹:

1. This file must be preprocessed by Aprepro prior to being input to the ACIS Test Harness.

Figure B-5 Box Beam example

```
# File: boxBeam.mon
# Side = {Side = 1.75}
# Height = {Height = 12.0}
# Upper = {Upper = 2.0}

block lowerSection width {Side/2.0} depth {Side/2.0} height
{Height - Upper}
block upperSection width {Side/2.0} depth {Side/2.0} height
{Upper}

move lowerSection {Side/4.0} {Side/4.0} {(Height - Upper)/2.0}
move upperSection {Side/4.0} {Side/4.0} {Upper/2.0 + Height -
Upper}

group lowerSection upperSection as boxBeam

save boxBeam to boxBeam.sat
```

In this example, it is assumed that subsequent analyses will take advantage of the problem symmetry and therefore only one-quarter of the box beam will be meshed. It is worth noting that there are a variety of ways to construct a solid model for this problem; however, experience thus far with ACIS and CUBIT indicates that the easiest way to model the box beam is to use ACIS block primitives¹. Even though subsequent meshing will only be performed on the faces of the solid model, the entire 3D body is saved as an ACIS.sat file. The CUBIT journal file for the box beam example is:

1. This geometry can also be generated using the internal CUBIT Brick primitive.

```
# File: boxBeam.jou

# Thickness = {Thickness = 0.06}
# Crease = {Crease = 0.01}
# XYInts = {XYInts = 10}
# ZInts = {ZInts = 90}
# UpperInts = {UpperInts = 15}

Import Acis 'boxBeam.sat'
#Display

Merge All
Label Surface
#Display

Label Curve
#Display

Curve 1 To 8 Interval {XYInts}
Curve 13 To 16 Interval {XYInts}

Curve 9 To 12 Interval {ZInts-UpperInts}
Curve 21 To 24 Interval {UpperInts}

Mesh Surface 3
Mesh Surface 6
Mesh Surface 9
Mesh Surface 12

NodeSet 1 Curve 1
NodeSet 2 Curve 4

NodeSet 1 Move {-Crease} 0 0
NodeSet 2 Move 0 {Crease} 0

Block 2 Surface 3
Block 2 Surface 6

Block 1 Surface 9
Block 1 Surface 12

Block 1 To 2 Attribute {Thickness}

Export Genesis 'boxBeam.exoII'
Quit
```

Commands worth noting in the CUBIT journal file include:

- **Block, Block Attribute** Allows the user to specify that shell elements for the surfaces of the solid model are to be written to the output (EXODUSII) database, and that shell elements be given a thickness attribute. This is necessary since CUBIT defaults to three-dimensional hexahedral meshing of solid model volumes.
 - **NodeSet Move** Allows the user to actually move the specified nodes by a vector (Δx , Δy , Δz). This is advantageous for the buckling problem, since the numerical simulation requires a small "crease" in the beam in order to perform well.
 - **Merge** Allows the user to combine geometric features (e.g. edges and surfaces).
- Other commands in the journal file should be straightforward. Since the problem is sufficiently simple to mesh using a mapping transformation, specification of a meshing "scheme" is unnecessary (mapping is the default in CUBIT).

Finally, note that both the ACIS monitor file (**boxBeam.mon**) and the CUBIT journal file (**boxBeam.jou**) contain macros that are evaluated using Aprepro. The *makefile* used to semi-automatically generate the mesh is given below:

File: Makefile

```
boxBeam.g:boxBeam.exoII
    exo2exol boxBeam.exoII boxBeam.g

boxBeam.exoII:boxBeam.sat boxBeam.jou
    aprepro boxBeam.jou | cubitb
    rm cubit.jou

boxBeam.sat: boxBeam.mon
    aprepro boxBeam.mon | acis
    rm wjbohnhl.*

clean:
    @-rm *.sat *.exoII *.g
```

While this particular example is a trivial use of the software, it does serve to demonstrate a few of the capabilities offered by ACIS and CUBIT.

▼ Thunderbird 3D Shell

This example is the three-dimensional paving of a shell shown in Figure B-6. The 2D wireframe geometry of the thunderbird is given by the following FASTQ file:

Figure B-6 Sandia Thunderbird 3D shell

APPENDIX B

Examples

#File: tbird.fsq

TITLE

MESH OF SANDIA THUNDERBIRD

```
$ block (e = .2) int= (isq = 20)
$ number of elements in block thick (iblkt = 5 ) block thickness
(blkt=.2 )
$ block angle (angle=15)
$ magnification factor = (magnificationFactor=1.0)
$ bird (bthick = .018) (ithick = 3) (idepth = 20)
$ (pi = 3.14159265359) (rad=magnificationFactor/pi) (bdepth=1.)
$ preferred normalized element size = (elementSize=0.06)
$ number of intervals along outside edges =
$ (border_int=5) (corner_int=10) (side_int=20)
$ (outsideIntervals= 2*corner_int+side_int)
$ (boxTop=.2) (topIntervals = 8)
```

```
$ (insideCurveInt=8)
```

```
$ (MAG=magnificationFactor/3.0)
```

```
$ (middleInside=MAG*0.97)
$ (xCurveStartInside=MAG*0.60)
$ (yCurveStartInside=MAG*0.93)
$ (curveMiddleInside=MAG*0.81)
```

```
$ (xCurveStartOutside=MAG*0.75)
$ (yCurveStartOutside=MAG*1.17)
$ (middleOutside=MAG*1.20)
$ (curveMiddleOutside=MAG*1.01)
$ (boundingBox = MAG*1.5)
```

```
$ Thunderbird Coordinates
```

```
POINT 1 {MAG*-.40} {MAG*.78}
POINT 2 {MAG*-.40} {MAG*.59}
POINT 3 {MAG*-.22} {MAG*.59}
POINT 4 {MAG*-.22} {MAG*.40}
POINT 5 {MAG*-.75} {MAG*.40}
POINT 6 {MAG*-.78} {MAG*-.09}
POINT 7 {MAG*-.75} {MAG*-.58}
POINT 8 {MAG*-.53} {MAG*-.60}
POINT 9 {MAG*-.54} {MAG*-.23}
POINT 10 {MAG*-.42} {MAG*-.23}
POINT 11 {MAG*-.42} {MAG*.07}
POINT 12 {MAG*-.24} {MAG*.07}
POINT 13 {MAG*-.27} {MAG*-.80}
POINT 14 {MAG*.27} {MAG*-.80}
POINT 15 {MAG*.24} {MAG*.07}
POINT 16 {MAG*.42} {MAG*.07}
POINT 17 {MAG*.42} {MAG*-.23}
POINT 18 {MAG*.54} {MAG*-.23}
POINT 19 {MAG*.53} {MAG*-.60}
POINT 20 {MAG*.75} {MAG*-.58}
POINT 21 {MAG*.78} {MAG*-.09}
POINT 22 {MAG*.75} {MAG*.40}
POINT 23 {MAG*.22} {MAG*.40}
POINT 24 {MAG*.21} {MAG*.78}
POINT 25 {MAG*0.0} {MAG*.80}
```

```
$ lines for Tbird
```

```
LINE 1 STR 1 2
LINE 2 STR 2 3
LINE 3 STR 3 4
LINE 4 STR 4 5
LINE 5 CIRM 5 7 6
LINE 6 STR 7 8
LINE 7 STR 8 9
LINE 8 STR 9 10
```



```

LINE 9 STR 10 11
LINE 10 STR 11 12
LINE 11 STR 12 13
LINE 12 STR 13 14
LINE 13 STR 14 15
LINE 14 STR 15 16
LINE 15 STR 16 17
LINE 16 STR 17 18
LINE 17 STR 18 19
LINE 18 STR 19 20
LINE 19 CIRM 20 22 21
LINE 20 STR 22 23
LINE 21 STR 23 24
LINE 22 STR 24 1 0 7 1.0

$ REGIONS

SIZE (elementSize*MAG)

REGION 1 1 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 *
      -16 -17 -18 -19 -20 -21 -22

SCHEME 0 X
BODY 1
EXIT

```

A command interpreter program, **fsqacs**¹, has been developed to convert FASTQ geometry commands to equivalent ACIS Test Harness commands (outputs an ACIS monitor file). Note, **fsqacs** ignores any meshing information in the FASTQ file since there is currently no means of passing the mesh parameters through the ACIS solid modeler to the CUBIT session. It should be noted that the 2D wireframe geometry can be directly constructed using wires in the ACIS Test Harness; however, there may be instances when it is more convenient to use the command interpreter.

After executing **fsqacs**, the resulting ACIS monitor file may be included in a subsequent ACIS session by simply using the **include** command as illustrated by the following file:

#File: tbird3d.mon

```

include tbird.acs
roll
view scale 200
#draw
cylinder cyl1 height 1.25 radius 0.5
rotate cyl1 by 90 about x
#draw
sweep wire f1 by 1.0
#draw
intersect f1 with cyl1 as tbird3d
#draw
list
save tbird3d to tbird3d.sat

```

Note that the ACIS.mon file demonstrates how 3D solid models may be constructed starting from an initial FASTQ profile followed by typical solid modeling commands (e.g. sweep, intersect) resulting in the desired geometry.

In this example, only the 3D shell of the thunderbird is desired for the finite element model, and thus, the block command is used to specify that only elements on the surface are to be created. The following CUBIT journal file demonstrates current 3D paving capability:

1. The *fsqacs* users manual is reproduced in Appendix C, "Fsqacs: A FASTQ to ACIS Command Interpreter" on page 153

#File: tbird3d.jou

```
Import Acis 'tbird3d.sat'
#Display
View From 6 3 10
Label Surface
Display
Draw Surface 23
Draw Surface 24

Surface 24 Size 0.03
Surface 24 Scheme Pave
Mesh Surface 24
Draw Surface 24

Block 1 Surface 24
Block 1 Attribute 0.03
```

▼ Assembly Components

Finally, a more practical example of ACIS/CUBIT is demonstrated by meshing an electronics assembly package. Figure B-7 shows a section of the assembly model containing three components: the accelerometer, the timer, and the radar. Also shown is the low density foam encapsulating these components. Note that the foam is of conical shape and the timer and radar units both have draft angles.

Figure B-7 Components in electronics assembly package.

In this case, the ACIS solid model is constructed on a component by component basis, and the final model called **accelLayer.sat** is generated by grouping the separate ACIS volumes together as one ACIS body. The user may prefer to create the entire solid model in a single ACIS session. However, for demonstration purposes, the model constructed here consists of five ACIS.mon files and one FASTQ input file that is converted to ACIS input using **fsqacs**. A

makefile is used to manage the input and output files and efficiently generate the model. The mesh generated for this assembly is shown in Figure B-8.

Figure B-8 Generated mesh for the electronics assembly package.

The complete geometric description is given by the following input files.

#File: timer.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

intersect topBlock with cyll as timer
#draw
save timer to timer.sat
```

#File: radar.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
```

APPENDIX **B** Examples

```
reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#view from 0 0 1 scale 50
#draw

unite topBlock with sliceBlocks
#draw
subtract sliceBlocks from cyl1 as radar
#draw

block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0

subtract bottomBlock from radar
#draw
save radar to radar.sat
```

#File: accel.mon

```
include accel.acs
view from 0 0 1 scale 50
#draw

sweep wire f1 by 2.107 direction 0 0 1
move f1 0 0 -1.0535
copy f1 as accel
save accel to accel.sat
```

#File: foam.mon

```
option props on
cylinder cyl1 height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#draw

subtract sliceBlocks from cyl1 as hole
block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0
subtract bottomBlock from hole
#draw hole

retrieve accel.sat as accel
unite accel with hole
#draw hole

cylinder foam height 2.107 radius 2.124 top 2.486
subtract hole from foam
#draw
save foam to foam.sat
```

#File: accelLayer.mon

```
option props on
view from 0 0 1 scale 50

retrieve timer.sat as timer
retrieve radar.sat as radar
retrieve accel.sat as accel
retrieve foam.sat as foam
```

```
#draw

group timer radar accel foam as accelLayer
#draw
save accelLayer to accelLayer.sat
```

ACIS commands worth noting in this example include:

- **option props on** Inserts an edge or “scribe line” along the outer surface of a cylinder. This changes the *periodic*¹ surface into a surface with only one bounding exterior loop of edges. Some CUBIT meshing algorithms require this type of solid model when constructing geometry using cylinders, spheres, or tori.
- **save** Individual components may be saved as separate ACIS solid models.
- **retrieve** Any valid ACIS.sat file may be retrieved and used to perform booleans and/or transformations in an ACIS session.
- **group** Individual components (ACIS bodies) may be grouped together to create a single ACIS.sat file for an assembly.

The resulting solid model is meshed in CUBIT using the following commands.

```
#File: accelLayer.jou
journal off
Import Acis 'accelLayer.sat'
Merge All
Display
View From 3 4 -5
Display
# front face of foam encapsulant
Surface 28 Size .07
Surface 28 Scheme Pave
Mesh Surface 28
# front face of accelerometer
Surface 3 Size .07
Surface 3 Scheme Pave
Mesh Surface 3
# front face of radar
Surface 7 Size .07
Surface 7 Scheme Pave
Mesh Surface 7
# front face of timer
Surface 16 Size .07
Surface 16 Scheme Pave
Mesh Surface 16
Display
# foam encapsulant
Volume 4 Interval 12
Volume 4 Scheme Project Source 28 Target 29
Mesh Volume 4
# accelerometer
Volume 3 Interval 12
Volume 3 Scheme Project Source 16 Target 17
Mesh Volume 3
# radar
Volume 2 Interval 12
Volume 2 Scheme Project Source 7 Target 10
Mesh Volume 2
Volume 2 Interval 12
# timer
Volume 1 Scheme Project Source 3 Target 4
```

1. A periodic surface is one which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction.

APPENDIX **B** Examples

```
Mesh Volume 1
#Display
Block 1 Volume 1
Block 2 Volume 2
Block 3 Volume 3
Block 4 Volume 4
Export Genesis 'accelLayer.exoII'
```

This example demonstrates that setting the number of intervals for every edge in a 3D solid model can be a very tedious task. When possible, users should use geometry consolidation to reduce the amount of effort involved in performing this step. Additionally, clever use of the body interval command can also significantly reduce time and effort. In this example, all components have the same number of intervals in the z-direction. It is advantageous to set this value for all edges parallel to the z-axis by using the body interval command. Finally, when a mesh is projected from a source surface to a target surface, if one of the surfaces is larger than the other (i.e., if the swept region contains a draft angle), a better quality mesh will usually be generated if the smaller of the two surfaces is used as the source surface.

Appendix C: Fsqacs: A FASTQ to ACIS Command Interpreter

▼ Description...153

▼ Program Execution...153

*A text converter program, **fsqacs**, has been developed to provide a means of generating ACIS solid models from a FASTQ input deck. The program should be considered more as a command interpreter rather than a "true" translator since the output is a set of ACIS Test Harness commands as opposed to an actual solid model.*

▼ Description

The **fsqacs** command interpreter is intended to generate a set of ACIS Test Harness commands which will create the same two-dimensional geometry profile as the geometry described in a FASTQ file¹. The output of **fsqacs** is a file which can be input to the ACIS Test Harness to construct a two- or three-dimensional solid model

No mechanism currently exists that allows mesh attributes (for example, interval settings, nodesets, sidesets, and material specifications) to be attached to the ACIS solid model. Therefore, **fsqacs** ignores all mesh information in the FASTQ file. The only FASTQ commands that are converted are those that represent the geometry definition, i.e., POINT, LINE, SIDE, REGION, HOLE, and BODY commands. The mesh interval field in the LINE command and the Element Block ID field in the REGION command are ignored.

▼ Program Execution

The **fsqacs** program is executed using the following UNIX command:

fsqacs [-aprepro] [-nocover] [-tolerance <val>] in_file.fsq [outputfile]

If no output file is specified, the default is: **in_file.acs**.

The command line options are:

-tolerance <value> Specifies the distance used in ACIS to determine whether two lines intersect. See the following text for more information.

1. FASTQ is the two-dimensional mesh generation program previously used by most analysts at Sandia National Laboratories. The **fsqacs** translator was written to provide a means for analysts to continue work in progress which previously used FASTQ, to try CUBIT on geometries previously meshed using FASTQ and to provide some measure of backward compatibility.

-nocover Specifies that the two-dimensional profile will be subsequently used to generate three-dimensional geometry in ACIS and therefore, the profile should not be covered to generate a surface.

-aprepro Specifies that the FASTQ file should be processed by Aprepro prior to being translated.

Specifying the correct tolerance value is very important, particularly when the model contains circular lines. Due to computational inaccuracies and roundoff, it is possible that connected lines translated from FASTQ to ACIS may not intersect within the default acis tolerance distance of 1.0e-6. If problems are experienced producing a correctly closed "wire" in ACIS, the tolerance value should be increased. A tolerance value of 1.0e-4 has been a good value for many **fsqacs** users.

The **-nocover** option is provided to control whether a closed ACIS wire should be covered or not covered. By default, **fsqacs** assumes that the FASTQ file will be used to create a two-dimensional surface model which must be "covered" in the ACIS Test Harness to generate a surface. If a three-dimensional solid model is desired, the wire should not be "covered" and the **-nocover** option should be specified.

The **-aprepro** option will preprocess the FASTQ input file using Aprepro prior to performing the translation.

The resulting file consisting of ACIS commands must then be processed by executing the ACIS Test Harness to generate the solid model that is imported in CUBIT. See "Importing Geometry" on page 84 and "Examples" on page 137 for more information.

▼ Limitations

Due to the differences in geometry representation between FASTQ and ACIS, **fsqacs** has some limitations:

- Mesh attributes (interval settings, nodesets, sidesets, and material specifications) are not translated.
- Only the STR, CIRC, and CIRM line types are supported. All other line types will be converted to straight lines and a warning message will be printed.
- All points defining a line must be defined prior to encountering the LINE command.
- All lines defining a region must be defined prior to encountering the REGION command.
- A BODY command must be specified.
- The FASTQ file must end with the EXIT command.
- Unless the ACIS Test Harness is reconfigured from the default distributed copy, only 30 edges may be joined to form a region and only 20 regions may be grouped to define a body.

Appendix D: CUBIT Installation

▼ Licensing...155

▼ Distribution Contents...156

▼ Installation...156

This Appendix contains information about the licensing and redistribution restrictions attached to CUBIT, the distribution contents, and installation instructions. All questions pertaining to obtaining a license for CUBIT should be directed to:

*Marilyn K. Smith
Computational Mechanics & Visualization Department
Division 1425, MS-0441
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0441
Fax: (505) 844-9297, Email: mksmith@sandia.gov*

▼ Licensing

CUBIT is distributed in statically linked executable form for each supported platform. Supported platforms include the HP 9000 series running HP-UX¹, Sun SPARCstations running SunOS², and the SGI running IRIX³. Additional platforms will be added as required.

Note: CUBIT installations have use restrictions. **THE CUBIT CODE CANNOT BE COPIED TO ANOTHER COMPUTER AND THE NUMBER OF USER SEATS ON EACH COMPUTER OR LAN IS LIMITED.** If additional user seats or additional copies of CUBIT are required, you **MUST** contact us to acquire them.

CUBIT incorporates code modules developed by outside code vendors and our license agreements with them limit the number of user seats at Sandia National Laboratories and limit the number of users who are doing work in conjunction with Sandia National Laboratories. Hence, CUBIT cannot be copied and redistributed without affecting the licensing agreement with the vendors who have proprietary interests in code modules within CUBIT.

-
1. HP-UX is a registered trademark of Hewlett-Packard Company.
 2. Sun and SunOS are registered trademarks of Sun Microsystems, Inc.
 3. IRIX is a registered trademark of Silicon Graphics, Inc.

CHAPTER

Code distributions within Sandia National Laboratories are managed by an informal memorandum. Code distributions outside Sandia National Laboratories are managed by either a Use Notice memorandum or by a formal license agreement depending upon the code recipient. Use Notice and license agreement formats have been developed by the legal department at Sandia National Laboratories to protect the copyrights of code vendors and to protect the commercialization of Sandia National Laboratories copyrights to the CUBIT and SEACAS codes.

▼ Distribution Contents

In addition to the CUBIT executable, a code distribution can include example inputs and a test suite for CUBIT and, depending upon the nature of the request for CUBIT, a code distribution could include certain codes from the Sandia National Laboratories Engineering Analysis Code Access System [14] (SEACAS). Codes in SEACAS which could be used with CUBIT include finite element analysis codes, graphical postprocessing codes, and non-graphical pre- and postprocessing codes. Note that all codes, whether CUBIT or SEACAS codes, run under UNIX¹ operating systems.

Distributions containing other programs in addition to CUBIT will be supplied in tar format. For users who cannot access the tar file through ftp, the tar file will be written to magnetic or CD-ROM media and mailed. Due to possible exposure of the code and subsequent violation of copyrights and export control regulations, no electronic mailing of CUBIT or other codes is permitted.

▼ Installation

CUBIT and supporting CUBIT examples are installed simply by unpacking the tar file and moving the executables to their final directory. Examples and test problems for CUBIT include a README file which provides information needed to run the test problems and examples.

Any SEACAS code distributed with CUBIT will be in source code only. The compilation, linking, and installation of executables is managed by a very complete and extensive installation script. A complete set of installation procedures is provided with the SEACAS codes.

1. UNIX is a registered trademark of UNIX Systems Laboratories Inc.

Appendix E: Available Colors

Table E-1 in this Appendix lists the colors available in CUBIT at this time. All color commands require the specification of the color name. The table in this appendix lists the color number (#), color name, and the red, green, and blue components corresponding to each color for reference.

Table E-1 Available Colors

#	Color Name	Red	Green	Blue
0	black	0.000	0.000	0.000
1	red	1.000	0.000	0.000
2	green	0.000	1.000	0.000
3	yellow	1.000	1.000	0.000
4	blue	0.000	0.000	1.000
5	magenta	1.000	0.000	1.000
6	cyan	0.000	1.000	1.000
7	white	1.000	1.000	1.000
8	grey	0.500	0.500	0.500
9	orange	1.000	0.647	0.000
10	pink	1.000	0.753	0.796
11	brown	0.647	0.165	0.165
12	gold	1.000	0.843	0.000
13	lightblue	0.678	0.847	0.902
14	lightgreen	0.000	0.800	0.000
15	salmon	0.980	0.502	0.447
16	coral	1.000	0.498	0.314
17	purple	0.627	0.125	0.941
18	paleturquoise	0.686	0.933	0.933
19	lightsalmon	1.000	0.627	0.478

Table E-1 Available Colors

#	Color Name	Red	Green	Blue
20	springgreen	0.000	1.000	0.498
21	slateblue	0.416	0.353	0.804
22	sienna	0.627	0.322	0.176
23	seagreen	0.180	0.543	0.341
24	deepskyblue	0.000	0.749	1.000
25	khaki	0.941	0.902	0.549
26	lightskyblue	0.529	0.808	0.980
27	turquoise	0.251	0.878	0.816
28	greenyellow	0.678	1.000	0.184
29	powderblue	0.690	0.878	0.902
30	mediumturquoise	0.282	0.820	0.800
31	skyblue	0.529	0.808	0.922
32	tomato	1.000	0.388	0.278
33	lightcyan	0.878	1.000	1.000
34	dodgerblue	0.118	0.565	1.000
35	aquamarine	0.498	1.000	0.831
36	lightgoldenrodyellow	0.980	0.980	0.824
37	darkgreen	0.000	0.392	0.000
38	lightcoral	0.941	0.502	0.502
39	mediumslateblue	0.482	0.408	0.933

APPENDIX **E** Available Colors

Table E-1 Available Colors

#	Color Name	Red	Green	Blue
40	lightseagreen	0.125	0.698	0.667
41	goldenrod	0.855	0.647	0.125
42	indianred	0.804	0.361	0.361
43	mediumspringgreen	0.000	0.980	0.604
44	darkturquoise	0.000	0.808	0.820
45	yellowgreen	0.604	0.804	0.196
46	chocolate	0.824	0.412	0.118
47	steelblue	0.275	0.510	0.706
48	burlywood	0.871	0.722	0.529
49	hotpink	1.000	0.412	0.706
50	saddlebrown	0.545	0.271	0.075
51	violet	0.933	0.510	0.933
52	tan	0.824	0.706	0.549
53	mediumseagreen	0.235	0.702	0.443
54	thistle	0.847	0.749	0.847
55	palegoldenrod	0.933	0.910	0.667
56	firebrick	0.698	0.133	0.133
57	palegreen	0.596	0.984	0.596
58	lightyellow	1.000	1.000	0.878
59	darksalmon	0.914	0.588	0.478
60	orangered	1.000	0.271	0.000
61	palevioletred	0.859	0.439	0.576
62	limegreen	0.196	0.804	0.196

Table E-1 Available Colors

#	Color Name	Red	Green	Blue
63	mediumblue	0.000	0.000	0.804
64	blueviolet	0.541	0.169	0.886
65	deeppink	1.000	0.078	0.576
66	beige	0.961	0.961	0.863
67	royalblue	0.255	0.412	0.882
68	darkkhaki	0.741	0.718	0.420
69	lawngreen	0.486	0.988	0.000
70	lightgoldenrod	0.933	0.867	0.510
71	plum	0.867	0.627	0.867
72	sandybrown	0.957	0.643	0.376
73	lightslateblue	0.518	0.439	1.000
74	orchid	0.855	0.439	0.839
75	cadetblue	0.373	0.620	0.627
76	peru	0.804	0.522	0.247
77	olivedrab	0.420	0.557	0.137
78	mediumpurple	0.576	0.439	0.859
79	maroon	0.690	0.188	0.376
80	lightpink	1.000	0.714	0.757
81	darkslateblue	0.282	0.239	0.545
82	rosybrown	0.737	0.561	0.561
83	mediumvioletred	0.780	0.082	0.522
84	lightsteelblue	0.690	0.769	0.871
85	mediumaquamarine	0.400	0.804	0.667

Appendix F: CUBIT Application Defaults File

The CUBIT Application Defaults file (CUBIT.ad) file is reproduced here. The use of this file is discussed in "Graphics Customization" on page 19.

```
cubit.defaultFontList: *-helvetica-bold-r-*--*-120-75-75-*--iso8859-1
```

```
*webCutShell.background: Grey
*webCutShell.x: 264
*webCutShell.y: 113
cubit*XmTextField.background: LightBlue
cubit*webCutWindow*XmText.background: AntiqueWhite
```

```
cubit*XmForm*capture_beginBtn.background: Wheat
cubit*XmForm*capture_endBtn.background: Wheat
cubit*XmForm*applyBtn.background: Wheat
cubit*XmForm*helpBtn.background: Wheat
cubit*XmForm*cancelBtn.background: Wheat
cubit*XmForm*closeBtn.background: Wheat
cubit*XmForm*picker_helpBtn.background: Wheat
cubit*XmForm*picker_applyBtn.background: Wheat
cubit*XmForm*picker_cancelBtn.background: Wheat
cubit*XmForm*pickerBtn.background: Wheat
```

```
cubit*consoleShell.background: Grey
cubit*consoleWindow.x: 16
cubit*consoleWindow.y: 746
cubit*consoleWindow.background: Grey
cubit*XmScrolledWindow.consoleText.background: Grey
cubit*XmForm.btnRC.background: Grey
cubit*XmForm.XmRowColumn.btn1.background: Bisque
cubit*XmForm.XmRowColumn
.btn2.background: Bisque
```

```
cubit*XmForm.XmRowColumn.btn3.background: Bisque
cubit*XmForm.XmRowColumn.btn4.background: Bisque
cubit*XmForm.XmRowColumn.btn5.background: Bisque
cubit*XmForm.XmRowColumn.btn6.background: Bisque
cubit*XmForm.XmRowColumn.btn7.background: Bisque
cubit*XmForm.XmRowColumn.btn8.background: Bisque
cubit*XmForm.XmRowColumn.btn9.background: Bisque
cubit*XmForm.XmRowColumn.btn10.background: Bisque
```

APPENDIX **F** CUBIT Application Defaults File

! Don't recommend changing, but wanted it here for easy changing
cubit*webCutWindow.helpInfo.value: Create a webcut by selecting a body,\n\
supplying a plane for the cut and an optional\n\
vector.\n\
\n\
The plane may be either a face or 3 vertices to\n\
make a face and an optional vector.\n\
\n\
If vector is omitted, default will be 360 degrees.

```
*cubitmMain.x: 96
*cubitmMain.y: 319
*cubitmMain.title: CUBITM
*cubitmMain.iconName: CUBITM
*cubitmBB.x: 96
*cubitmBB.y: 319
*topLevelShell.x: 595
*topLevelShell.y: 309
*bulletinBoard.x: 595
*bulletinBoard.y: 309
*bulletinBoard.cubitmDA.x: 10
*bulletinBoard.cubitmDA.y: 10
cubit*background: Grey
```

References

- 1 T. D. Blacker and M. B. Stephenson, 'Paving: a new approach to automated quadrilateral mesh generation', SAND90-0249, Sandia National Laboratories, (1990).
- 2 M. B. Stephenson, S. A. Canann, and T. D. Blacker, 'Plastering: a new approach to automated, 3D hexahedral mesh generation', SAND89-2192, Sandia National Laboratories, (1992).
- 3 G. D. Sjaardema, et. al., *CUBIT Mesh Generation Environment, Volume 2: Developers Manual*, SAND94-1101, Sandia National Laboratories, (1994).
- 4 Spatial Technology, Inc., *ACIS Test Harness Application Guide Version 1.4*, Spatial Technology, Inc., Applied Geometry, Inc., and Three-Space, Ltd., (1992).
- 5 T. D. Blacker, *FASTQ Users Manual Version 1.2*, SAND88-1326, Sandia National Laboratories, (1988).
- 6 L. A. Schoof, *EXODUS II Application Programming Interface*, internal memo, Sandia National Laboratories, (1992).
- 7 W. A. Cook and W. R. Oakes, 'Mapping methods for generating three-dimensional meshes', *Comp. mech. eng.*, **Volume 1**, 67-72 (1982).
- 8 R. E. Jones, *QMESH: A Self-Organizing Mesh Generation Program*, SLA - 73 - 1088, Sandia National Laboratories, (1974).
- 9 R. E. Tipton, 'Grid Optimization by Equipotential Relaxation', unpublished, Lawrence Livermore National Laboratory, (1990).
- 10 A. P. Gilkey and G. D. Sjaardema, *GEN3D: A GENESIS Database 2D to 3D Transformation Program*, SAND89-0485, Sandia National Laboratories, (1989).
- 11 G. D. Sjaardema, *GREPOS: A GENESIS Database Repositioning Program*, SAND90-0566, Sandia National Laboratories, (1990).
- 12 G. D. Sjaardema, *GJOIN: A Program for Merging Two or More GENESIS Databases*, SAND92-2290, Sandia National Laboratories, (1992).
- 13 G. D. Sjaardema, *APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses*, SAND92-2291, Sandia National Laboratories, (1992).
- 14 G. D. Sjaardema, *Overview of the Sandia National Laboratories Engineering Analysis Code Access System*, SAND92-2292, Sandia National Laboratories, (1993).
- 15 S. C. Lovejoy and R. G. Whirley, *DYNA3D Example Problem Manual*, UCRL-MA--105259, University Of California and Lawrence Livermore National Laboratory, (1990).
- 16 Open Software Foundation, Inc., *OSF/Motif™ User's Guide Revision 1.2*, PTR Prentice Hall, Englewood Cliffs, New Jersey, (1993).
- 17 J. M. Osier, *Keeping Track, Managing Messages with GNATS. The GNU Problem Report Management System*, Users manual for GNATS Version 3.2, Cygnus Support, October 1993.
- 18 L. M. Taylor and D. P. Flanagan, *Pronto 3D—A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912, Sandia National Laboratories, (1989).
- 19 S. W. Attaway, unpublished, (1993).

CHAPTER

Glossary

B

- Body.** A body is simply a collection or set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume. 76
- Brick.** A brick is a hexahedral element defined by six connected faces. A brick is owned by the enclosing volume. 96
- Button.** A button is a rectangular 3-dimensional object that appears to be raised. When you click on a button, a single action is usually performed. 51

C

- Cascade Button.** Cascade Buttons are only present in menus. These buttons always have an arrow following the text description on the button. When the user moves the cursor over the arrow, another series of menus normally "pops up" or "cascades." 52
- Click.** Click means to press and release the mouse button once. Unless otherwise specified, the mouse button to press will be Mouse Button 1 (this is usually the left-most button on a three button mouse). 52
- Curve.** A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. 76

D

- Dialog Box.** A dialog box is a window used to acquire user input or to display a message. 52
- Double-click.** Double-clicking means to quickly press and release the mouse button twice. The period of time between clicks to qualify a double-click is dependent upon user specified preferences established at the X Window system level. 52
- Drag.** Drag means to click and hold a mouse button and move the cursor across the screen 52

E

- Edge.** An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge on a curve is owned by that curve, an edge in a surface is owned by that surface, and an edge in a volume is owned by that volume 96
- Element Blocks.** Element Blocks (also referred to as simply, Blocks) are a logical grouping of elements all having the same basic geometry and number of nodes. 117

F

Face. A face is defined by four connected edges. A face on a surface is owned by that surface, a face in the interior of a volume is owned by that volume. 96

File Selection Dialog Box. File selection dialog boxes allow the user to browse through the directory tree of the users system, filter specific files and file types, and to explicitly input a filename. Click apply to accept the selection or cancel to hide 52

G

Geometry primitives. Classes of general geometric shapes which are differentiated by basic parameters. CUBIT supports the brick, pyramid, prism, cylinder, torus, frustum, and sphere. 77

H

Hard Point. A vertex which is located in the interior of a surface. It is used to force a node location to that specific geometric location. 75

M

Menu Bar. This is an object that is present in the main window. A menu bar contains multiple text descriptions of menus. 52

Menu Item. A menu item is a button under one of the text descriptions of the menu bar or a series of buttons within an option menu. 52

N

Node A node is a single point in space. A node at a vertex is owned by that geometric vertex, a node on a curve is owned by that curve, a node on the interior of a surface is owned by that surface, and a node in a volume is owned by that volume. 95

Nodeset. Nodesets are a logical grouping of nodes also accessed through a single ID known as the Nodeset ID. 117

O

Option Menu. An option menu consists of multiple menu items. To the user, an option menu looks like a button until it is clicked. Once an option menu is clicked, it reveals a series of menu items that may be selected. The current selection is then display 52

P

Periodic Surface. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction. 76

GLOSSARY

R

Radio Button. A radio button is a diamond shaped button that is highlighted in a noticeable color if selected. A radio button is usually one of many buttons grouped together. Only one button may be active at a time. 52

S

Scrolled List. In CUBIT, a scrolled list usually contains selected geometry or mesh entities. The scrolled list will appear as only a depressed window until enough information is contained in the list to activate scroll bars on the sides of the list. 52

Sideset. Sidesets represent a grouping of element sides and are also referenced using an integer Sideset ID. 118

Surface. A surface in CUBIT is a finite bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the volume in a volume. A surface is bounded by a set of curves. 76

T

Text Field. A text field is a rectangular object that has a depressed, 3-dimensional appearance. Click on a text field to activate it and then type the text required. Text in CUBIT is usually an integer or a real value. 52

Toggle Button. A toggle button is a rectangular shaped button that is highlighted in a noticeable color if selected. To activate a toggle button click once. 53

V

Vertex. A vertex occupies a single point in space. A vertex is used to bound a curve and/or to specify a specific location for a node. 75

Volume. Volumes are volumetric regions and are always bounded by one or more surfaces. For practical consideration, volumes will always be bounded by two or more surfaces. 76

GLOSSARY

Index

Symbols

\$HOME/.cubit 18
.cubit 17, 18
.Xdefaults 19
.Xresources 19

Numerics

2-manifold topology 76

A

ACIS 22, 129, 137
 Import 84
 Test Harness 22, 84, 92, 137, 147, 153, 154
adaptivity 23
airfoil 141
Angle
 Perspective 62, 63, 129
Animate
 Geometry 70, 123
 Mesh 70, 123
 Overlay 70, 123
Apply 55
Aprepro 85, 145
-aprepro 154
Arc 84
Aries® ConceptStation 22, 92, 137
aspect ratios 142
Assembly Components 148
At 61, 62, 64, 124, 135
Attribute 119
 Block 119, 124
Autocenter 61, 128
Autoclear 61, 66, 128
Axis 61, 129

B

Background Color 59, 67, 125
-batch 18
Bias 99, 127
 Reverse 126
Block 140, 144
 Attribute 119, 124
 Color 67, 125
 Curve 119
 Draw 127
 Element Type 119, 124
 Geometry Color 67
 Geometry Type 124
 Mesh Color 67
 Surface 119
 Visibility 124
 Volume 119
Block Identifier
 Dialog Box 118
Body 76
 Color 67, 125
 Copy 86, 124
 Decomposition 92, 136
 Draw 66, 127
 Geometry Color 67, 125
 Geometry Visibility 124
 Interval 98, 124, 152
 Interval Size 98, 124
 Label 68, 130
 List 71, 130
 Mesh 131
 Mesh Color 67, 125
 Mesh Visibility 124
 Move 87, 124
 Reflect 88, 124
 Restore 88
 Rotate 88, 124
 Scale 87, 124

- Visibility 65, 124
- Webcut 92, 136
- Booleans 89
 - Intersect 89, 130, 139
 - Subtract 26, 89, 133, 138, 139
 - Unite 89, 134
- Border 61, 129
- Boundary Condition 96, 117
 - Contact Surface 118
 - Menu 53
 - NodeSet 117, 120
 - SideSet 118, 120
- BoundaryLayer 105, 106, 141
 - Curve 125
 - Parameters 125
 - Surface 125
- Box 84
- Box Beam 142
- Brick 26, 77, 78, 125, 126, 138, 139
 - Dialog Box 79
- Button 51
 - Apply 55
 - Cancel 52
 - Cascade 52
 - Clear 54
 - Clear All 55
 - Filter 52
 - Get All 55
 - Help 74
 - Pick 54
 - Radio 52
 - Toggle 53
 - Update Display 54
 - Zoom 54
- By (range) 21

C

- CANCEL 52
- Cascade Button 52
- Cellular Topology 76
- Center 61, 129
- Chamfer 84
- Clear 54, 61, 66, 129
- Clear All 55
- Click 52

- Color 66
 - Background 59, 67, 125
 - Block 67, 125
 - Body 67, 125
 - Geometry 125
 - Mesh 125
 - Dialog Box 67
 - Geometry 125
 - Menu 66
 - Mesh
 - Surface 125
 - Volume 126
 - Node 67, 125
 - NodeSet 67
 - Nodeset 125
 - SideSet 67, 125
 - Surface 67, 125
 - Geometry 125
 - Mesh 125
 - Table 157
 - Volume 67, 126
 - Geometry 126
 - Mesh 126
- Command Line
 - Echo 56, 128
 - Editing 50
 - History 50
 - Interface 49
 - Scroll Window 54
 - Text Field 54
- Command Recall 51
- Console Window 55
- Constraints Menu 53, 120
- Contact Surface 118
- Copy 139
 - Body 86, 124
 - Dialog Box 86
 - Mesh 112, 126
- Create 77
 - Brick 77, 78, 126
 - Cylinder 77, 78, 127
 - Dialog Box 78
 - Frustum 77, 80, 126, 128
 - Prism 77, 79, 126, 132
 - Pyramid 77, 81, 126, 132
 - Sphere 77, 81, 126, 133

- Torus 77, 81, 126, 134
- Cube with Hole 26, 138
- cubit 17, 18
- CUBIT.ad 19, 159
- CUBIT_HELP_DIR 19
- CUBIT_OPT 18, 19
- cubitb 17, 50
- cubitHelpGUI.hlp 19
- CUBITM_GUI 18
- cubitx 17, 18
- Curvature
 - Curve Scheme 100, 127
 - Surface Scheme 103, 133
 - Volume Scheme 135
- Curve 76, 119, 126
 - Bias 127, 141
 - Block 119
 - BoundaryLayer 106
 - Curvature 100, 103, 127
 - Delete Mesh 115, 127
 - Draw 66, 127
 - Equal 127
 - Interval 98, 126
 - Interval Size 98, 127
 - Label 68, 130
 - List 71, 72, 131
 - Merge 94, 131
 - Mesh 100, 131
 - NodeSet 120, 132
 - Reverse Bias 100, 126
 - Scheme
 - Bias 99, 127
 - Curvature 100, 127
 - Equal 99, 127
 - SideSet 120, 133
 - Type 127
- Cylinder 26, 77, 78, 126, 127, 138, 139
 - Dialog Box 79

D

- Decomposition 91, 92, 136
- Delete
 - Mesh 115, 127
- Dialog 52
- Dialog Box

- Block Identifier 118
- Brick 79
- Color 67
- Copy 86
- Cylinder 79
- File Selection 52, 57, 58
- Frustum 81
- Graphics Draw 66
- Graphics View 62
- Hardcopy 69
- Intersect 90
- Journal Record/Play 57
- Label Visibility 68
- List 70
- Merge 94
- Mesh Delete 115
- Mode 59
- Move 87
- NodeSet 121
- Pause 58
- Prism 80
- Pyramid 82
- Reflect 89
- Rotate 88
- Scale 87
- SideSet 121
- Sketch 83
- Sphere 82
- Subtract 90
- Torus 83
- Unite 90
- Visibility 65
- WebCut 91
- DISPLAY 18
- Display 59, 64, 127
 - Button 54
 - Update 54
- Double-click 52
- Drag 52
- Draw 64, 66
 - Block 127
 - Body 66, 127
 - Curve 66, 127
 - Edge 66, 127
 - Face 66, 127
 - Hex 66, 127

- Loop 127
- Loop Circle 128
- Menu 66
- Node 66, 128
- NodeSet 66, 128
- SideSet 66, 128
- Skeleton 128
- Surface 66, 128
- Vertex 66, 128
- Volume 66, 128
 - Mesh 135

E

- Echo 56, 128
- Edge
 - Draw 66, 127
 - Label 68, 130
- Editing
 - Mesh 112
- Element Block 22, 117
- Element Type 97, 124
 - Block 119
- Encapsulated 129
- Environment Variable
 - CUBIT_HELP_DIR 19
 - CUBIT_OPT 18, 19
 - CUBITM_GUI 18
 - DISPLAY 18
 - HOME 18
 - HOOPS_PICTURE 18
 - PATH 18
- EPS 68, 69, 129
- Equal 99, 127
- Equipotential 113
 - Area 113
- Example
 - Assembly Components 148
 - Box Beam 142
 - Cube with Hole 26, 138
 - Octant of Sphere 139
 - Thunderbird 3D Shell 145
- Execution Options
 - batch 18
 - help 17
 - initfile 17, 18

- input 18
- noinitfile 18
- nojournal 18, 19, 57
- solidmodel 18

- Exit 56, 128
- Exodus 117
- ExodusII 116
- Export
 - Genesis 121, 128

F

- Face
 - Draw 66, 127
 - Label 68, 130
 - List 71, 73, 131
- False (toggle) 21
- FASTQ 22, 85, 130, 153
 - Import 85
 - Translator 85
- File 53
 - Dialog Box 58
- File Selection Dialog Box 52, 57
- Filename 21
- Files
 - \$HOME/.cubit 18
 - .Xdefaults 19
 - .Xresources 19
 - CUBIT.ad 19, 159
 - cubitHelpGUI.hlp 19
 - Exodus 117
 - ExodusII 116
 - Genesis 117, 121, 128
 - Menu 56
- Filter 52
- FlatShade 60, 129
- From 62, 64, 128, 135
- Frustum 77, 80, 126, 128
 - Dialog Box 81
- fsqacs 85, 137, 147, 153, 154
 - aprepro 154
 - nocover 154
 - tolerance 153

G

Genesis 117, 121, 128

Export 121

Geometry

Animate 70, 123

Booleans 89

Color 125

Body 67

NodeSet 67

SideSet 67

Surface 67, 125

Volume 67, 126

Creation 77

Decomposition 91

Webcut 91

Definition 75

Label 68, 130

Manipulation 85

Menu 53, 78, 82

Merge 22, 68, 139, 144, 152

Primitives 77

Type 124

Visibility 64, 124, 128

Surface 133

Volume 135

Get All 55

Global 64

Graphics

Autocenter 61, 128

Autoclear 61, 66, 128

Axis 61, 129

Border 61, 129

Center 61, 129

Clear 54, 61, 66, 129

Display 59, 64

Draw 66

Hardcopy 68

Line Width 61, 129

List

View 64

Menu 53, 59, 61, 66, 68

Mode

FlatShade 60, 129

HiddenLine 60, 129

Painters 60, 129

PolygonFill 60, 129

SmoothShade 60, 129

WireFrame 59, 129

Mode Type 59

Perspective 63, 129

Angle 62, 63, 64, 129

Rotate 63

Window 18

WindowSize 59, 129

Zoom 51, 54, 63, 129, 136

Button 54

Reset 54, 64, 129

GUI 49

H

Hard point 75

Hard Set 98

Hardcopy 68, 129

Dialog Box 69

Menu 68

Hardware Platforms 23

Help 57, 74, 129

Button 74

Hyperhelp 74, 129

Menu 53, 74

-help 17

Hex

Draw 66, 127

Label 68, 130

List 71, 73, 131

Weight 114, 136

Weight Surface 114, 136

Weighting Function 113

HiddenLine 60, 129

HOME 18

HOOPS_PICTURE 18

Hyperhelp 74, 129

I

Import

Acis 84, 129

Fastq 85, 130

Mesh 116, 130

-initfile 17, 18

Initialization File 17

Initialize

Video 69, 134

-input 18

Intersect 89, 130, 139

Dialog Box 90

Interval

Adjustment 98

Body 98, 124

Curvature Based 100

Curve 98, 126

Default 96

Hard Set 98

Size

Body 98, 124

Curve 98, 127

Surface 98, 134

Volume 98, 135

Specification 97

Surface 98, 133

Volume 98, 135

J

Journal

Dialog Box 57

Pause 58, 132

Playback 57, 58, 132

Record 57, 133

Journal Off 18, 57, 130

L

Label 67

All 68, 130

Body 68, 130

Curve 68, 130

Dialog Box 68

Edge 68, 130

Face 68, 130

Geometry 68, 130

Hex 68, 130

Menu 68

Mesh 68, 130

Node 68, 130

Surface 68, 130

Vertex 68, 130

Volume 68, 130

Laplacian 113

Length-weighted Laplacian 113

Line 84

Line Width 61, 129

List 70

Body 71, 130

Curve 71, 72, 131

Dialog Box 70

Face 71, 73, 131

Hex 71, 73, 131

Info 70

Node 73, 131

Nodes 71

Scrolled 52

Surface 71, 72, 131

Totals 70, 71, 131

Vertex 71, 73, 131

View 64, 131, 135

Volume 71, 131

Loop 108, 127, 128

Draw 127

Loop Circle

Draw 128

M

Main GUI Window 53

makefile 145

Manifold model 76

Map 101

Scheme

Surface 134

Volume 108, 135

Menu

Bar 52

Color 66

Constraints 53, 120

Draw 66

File 53

Geometry 53, 78, 82

Graphics 53, 59, 61, 66, 68

Graphics Mode Type 59

- Help 53
- Item 52
- Label 68
- Merge 93
- Mesh 53
- Option 52
- Special 53, 57, 70
- View 61
- Visibility 64
- Merge 22, 68, 139, 144, 152
 - All 93, 131
 - Curve 94, 131
 - Dialog Box 94
 - General 93
 - Menu 93
 - Only Curves 94, 131
 - Only Surfaces 94, 131
 - Surface 94, 131
 - Vertex 94, 131
- Mesh
 - Animate 70, 123
 - Body 131
 - Color 125
 - Block 67
 - Body 67
 - NodeSet 67
 - SideSet 67
 - Surface 67, 125
 - Volume 67
 - Copy 112, 126
 - Curve 100, 131
 - Delete 115, 127
 - Dialog Box 115
 - Deletion 115
 - Editing 112
 - GUI 114
 - Import 116, 130
 - Label 68, 130
 - Menu 53
 - Modify Smooth 112
 - Primitives 102
 - Smooth
 - Modify 112
 - Surface 107, 132
 - Visibility 64, 124, 132
 - Surface 133

- Volume 107, 112, 132
 - Visibility 135
- Mode
 - Dialog Box 59
- Model
 - attributes 22
- Motif 51
- Move 124
 - Body 87
 - Dialog Box 87
 - NodeSet 114, 132

N

- No (toggle) 21
- nocover 154
- Node
 - Color 67, 125
 - Draw 66, 128
 - Label 68, 130
 - List 71, 73, 131
 - Repositioning 114
 - Visibility 64, 132
- NodeSet 22, 117, 120
 - Color 67, 125
 - Curve 120, 132
 - Dialog Box 121
 - Draw 66, 128
 - Geometry Color 67
 - Mesh Color 67
 - Move 114, 144
 - Move To 132
 - Surface 120, 132
 - Vertex 120, 132
 - Visibility 65, 132
 - Volume 120, 132
- noinitfile 18
- nojournal 18, 19, 57
- Non-manifold topology 76

O

- Octant of Sphere 139
- Off (toggle) 21
- On (toggle) 21

Option Menu 52
option props on 151

P

Painters 60, 129
Parameter 21
 Optional 21
PATH 18
Pause 58, 132
 Dialog Box 58
Pave 22, 101, 138, 139, 141
 Surface Scheme 134
Perspective 63, 129
 Angle 62, 63, 64, 129
Pick Button 54
Picker Window 54
Picking
 Screen 51
Plaster 23
 Volume Scheme 108, 111, 135
Playback 57, 58, 132
PolygonFill 60, 129
PostScript 69, 129
PostScript Begin 23, 69
PostScript End 68
Primitives 77
 Arc 84
 Box 84
 Brick 77, 78, 125, 126
 Chamfer 84
 Cylinder 77, 78, 126, 127
 Dialog Box 78
 Frustum 77, 80, 126, 128
 Geometry 77
 Line 84
 Mesh 102
 Prism 77, 79, 126, 132
 Pyramid 77, 81, 126, 132
 Round 84
 Sphere 77, 81, 126, 133
 Torus 77, 81, 126, 134
Prism 77, 79, 126, 132
 Dialog Box 80
PRO/Engineer 22, 85, 92, 137
Project 139

Volume Scheme 108, 109, 135
Pyramid 77, 81, 126, 132
 Dialog Box 82

Q

Quit 56, 57, 128, 133

R

Radio Button 52
Range 21
Record 57, 133
 Stop 57, 133
Reflect
 Body 88, 124
 Dialog Box 89
Repositioning
 Node 114
Reset 56, 133
 Zoom 129
Restore
 Body 88
Reverse Bias 100, 126
Rotate 63, 133, 139
 Body 88, 124
 Continuous 63
 Dialog Box 88
 Volume Scheme 108, 111, 135
Round 84

S

Scale
 Body 87, 124
 Dialog Box 87
Scheme 96
 Bias 99
 Curvature 127, 133
 Designation 108
 Equal 99
 Map 101
 Surface 134
 Volume 108
 Pave 101, 134

- Plaster 108, 111
- Project 108, 109
- Rotate 108, 111
- Sweep 108
- Translate 108, 110
- Triangle 101, 102, 134
- Volume 108
 - Curvature 135
 - Map 135
 - Plaster 135
 - Project 135
 - Rotate 135
 - Translate 135
 - Weave 135
- Weave 108, 111
- Screen picking 51
- Scrolled List 52
- Selective 64
- SideSet 22, 118, 120
 - Color 67, 125
 - Curve 120, 133
 - Dialog Box 121
 - Draw 66, 128
 - Geometry Color 67
 - Mesh Color 67
 - Surface 120, 133
 - Visibility 65, 133
- Size
 - Body 124
 - Curve 127
 - Surface 134
 - Volume 135
- Skeleton
 - Draw 128
- Sketch
 - Arc 84
 - Box 84
 - Chamfer 84
 - Dialog Box 82, 83
 - Line 84
 - Round 84
- Smooth
 - Equipotential 113
 - Equipotential Area 113
 - Equipotential Fixed 113
 - Equipotential Generic 113
 - Equipotential Inverse Area 113
 - Equipotential Jacobian 113
 - GUI 114
 - Laplacian 113
 - Length-weighted Laplacian 113
 - Menu 114
 - Method
 - GUI 114
 - Modify Mesh 112
 - Scheme 134, 135
 - Surface 112, 113, 133
 - Volume 113, 133
 - Weight
 - Area 113
 - Inverse Area 113
 - Jacobian 113
- SmoothShade 60, 129
- Snap
 - Video 69, 134
- solidmodel 18
- Source Surface 108, 152
- Special
 - Menu 70
- Special Menu 53, 57
- Sphere 77, 81, 126, 133, 139
 - Dialog Box 82
- Step (range) 21
- String 21
- Subtract 26, 89, 133, 138, 139
 - Dialog Box 90
- Surface 76, 119, 133
 - Block 119
 - BoundaryLayer 106
 - Color 67, 125
 - Copy
 - Mesh 112
 - Curvature 133
 - Delete Mesh 115, 127
 - Draw 66, 128
 - Geometry Color 67, 125
 - Interval 98, 133
 - Interval Size 98, 134
 - Label 68, 130
 - List 71, 72, 131
 - Mapping 101
 - Merge 94, 131

- Mesh 107, 132
 - Visibility 133, 134
- Mesh Color 67, 125
- NodeSet 120, 132
- Scheme
 - Curvature 103, 133
 - Map 101, 134
 - Pave 101, 134
 - Triangle 101, 134
- SideSet 120, 133
- Smooth 112, 133
 - Equipotential 113
 - Equipotential Area 113
 - Equipotential Fixed 113
 - Equipotential Generic 113
 - Equipotential Inverse Area 113
 - Equipotential Jacobian 113
 - Scheme 134
 - Weight
 - Area 113
 - Inverse Area 113
 - Jacobian 113
- Source 108
- Target 108
- Visibility 65, 134
- Weight Hexes 114, 136
- Sweep
 - Volume Scheme 108

T

- Target Surface 108
- Test Harness 22
- Text Field 52
- Through (range) 21
- Thru (range) 21
- Thunderbird 3D Shell 145
- Title 121, 134
- To (range) 21
- Toggle 21
- Toggle Button 53
- tolerance 153
- Topology
 - 2-manifold 76
 - Cellular 76
 - Non-manifold 76

- Torus 77, 81, 126, 134
 - Dialog Box 83
- Totals
 - List 71
- Transform 86
- Translate 138
 - Volume Scheme 108, 110, 135
- Triangle 101, 102
 - Surface Scheme 134
- True (toggle) 21

U

- Unite 89, 134
 - Dialog Box 90
- Up 62, 63, 64, 134, 135
- Update Display 54
- User interface 49

V

- Version 56, 134
- Vertex 75, 134
 - Delete Mesh 115, 127
 - Draw 66, 128
 - Label 68, 130
 - List 71, 73, 131
 - Merge 94, 131
 - NodeSet 120, 132
 - Visibility 64, 134
- Video 69
 - Initialize 69, 134
 - Snap 69, 134
- View
 - At 61, 62, 64, 124, 135
 - Autocenter 61
 - Autoclear 61, 66
 - Border 61
 - Clear 66
 - FlatShade 60
 - From 62, 64, 128, 135
 - HiddenLine 60
 - List 64, 131, 135
 - Menu 61
 - Painters 60

Perspective
 Angle 64
 PolygonFill 60
 SmoothShade 60
 Up 62, 63, 64, 134, 135
 WindowSize 59
 WireFrame 59
 Visibility 64
 Block 124
 Body 65, 124
 Body Mesh 124
 Dialog Box 65
 Geometry 64, 124, 128
 Volume 135
 Global Geometry Type 64
 Menu 64
 Mesh 64, 132
 Surface 133, 134
 Mode 64
 Node 64, 132
 NodeSet 65, 132
 SideSet 65, 133
 Surface 65
 Geometry 133
 Mesh 133, 134
 Vertex 64, 134
 Volume 65, 136
 Mesh 135
 Volume 76, 119
 Block 119
 Color 67, 126
 Copy
 Mesh 112
 Delete Mesh 115, 127
 Draw 66, 128
 Geometry
 Color 126
 Visibility 135
 Geometry Color 67
 Interval 98, 135
 Size 135
 Interval Size 98
 Label 68, 130
 List 71, 131
 Map 108
 Mesh 112, 132

Color 126
 Draw 135
 Visibility 135
 Mesh Color 67
 Meshing 107
 NodeSet 120, 132
 Scheme 108
 Curvature 135
 Map 108, 135
 Plaster 108, 111, 135
 Project 108, 109, 135
 Rotate 108, 111, 135
 Sweep 108
 Translate 108, 110, 135
 Weave 108, 111, 135
 Smooth 113, 133
 Equipotential 113
 Equipotential Fixed 113
 Laplacian 113
 Scheme 135
 Visibility 65, 136

W

Weave
 Volume Scheme 108, 111, 135
 Web Cutting 91
 Webcut
 Body 92, 136
 Dialog Box 91
 Weight 113
 Weight Hexes
 Surface 114, 136
 Weighting Function
 Hex 113
 Window
 Command Line 54
 Console 55
 Graphics 18
 List 70
 Main GUI 53
 Picker 54
 WindowSize 59, 129
 WireFrame 59, 129

Yes (toggle) 21

INDEX

Y

Yes (toggle) 21

Z

Zoom 51, 54, 63, 129, 136

Button 54

Reset 54, 64, 129, 136

DISTRIBUTION

William R. (Rob) Oakes
Los Alamos National Laboratory
Technology Modeling and Analysis
MS F609
Los Alamos, NM 87545

Dr. Steve Benzley & Research Assistants
Associate Dean
College of Engineering & Technology
Brigham Young University
Provo, UT 84602

Don Dewhirst
Ford Research Lab
VP1100
23400 Michigan Ave.
Dearborn, MI 48124

Michael Engelman & Harry Bennett
Fluid Dynamics International
500 Davis Street #600
Evanston, IL 60201
(2 copies)

Paul Kinney
Ford CAD/CAM Department
Building 3, Room 2228
PO Box 2053
Dearborn, MI 48124

Bruce Johnston
MSC - Aries Division
600 Suffolk Street
Lowell, MA 01854

John Longhini
MSC
815 Colorado Blvd.
Los Angeles, CA 90041

Bob Phillips & Scott Canann
PDA Engineering
2975 Redhill Avenue
Costa Mesa, CA 92626

(2 copies)

Eric Trimble
7061 Canosa Ct.
West Minster CO 80030

Bob Benedict & Dave Wismer
D/431A
Goodyear Technical Center
PO Box 3531
Akron, OH 44309-3531
(4 copies)

INTERNAL DISTRIBUTION

MS1184	1239	J. F. Dempsey
MS0321	1400	E. H. Barsis
MS0821	1404	J. A. Ang
MS1110	1422	S. A. Mitchell
MS1110	1423	E. F. Brickell
MS0441	1425	S. W. Attaway
MS0441	1425	J. H. Biffle
MS0441	1425	T. D. Blacker
MS0441	1425	M. L. Blanford
MS0441	1425	W. J. Bohnhoff
MS0441	1425	J. R. Hipp
MS0441	1425	J. F. Mareda
MS0441	1425	C. J. Pavlakos
MS0441	1425	L. A. Schoof
MS0441	1425	M. K. Smith (50)
MS0441	1425	T. J. Tautges
MS0441	1425	T. J. Wilson
MS0819	1431	J. S. Peery
MS0820	1432	F. R. Norwood
MS0439	1434	J. L. Dohner
MS0439	1434	C. R. Dohrmann
MS0439	1434	G. R. Eisler
MS0439	1434	M. S. Eldred
MS0439	1434	J. T. Foley
MS0439	1434	C. W. Fulcher
MS0439	1434	T. Hinnerichs
MS0439	1434	D. W. Lobitz
MS0439	1434	D. B. Longcope
MS0439	1434	E. L. Marek
MS0439	1434	D. R. Martinez
MS0439	1434	G. G. Parker

MS0439	1434	J. R. Red-Horse	MS0437	1562	R. K. Thomas
MS0439	1434	J. M. Redmond	MS1134	1833	F. J. Zanner
MS0439	1434	G. M. Reese	MS0630	2800	J. F. Jones
MS0439	1434	D. J. Segalman	MS0629	2801	J. R. Yoder
MS0439	1434	W. R. Witkowski	MS0660	2861	M. E. Olson
MS0841	1500	D. J. McCloskey	MS0660	2861	A. L. Ames
MS0836	1501	C. W. Peterson	MS0660	2861	T. L. Edwards
MS0827	1502	A. E. Hodapp	MS0660	2861	J. C. Kelly
MS0827	1502	P. J. Hommert	MS0660	2861	R. R. Lober
MS0827	1511	D. K. Gartling	MS0660	2861	R. E. Parks
MS0827	1511	J. S. Rottler	MS0660	2861	S. W. Parratt
MS0827	1511	P. A. Sackinger	MS0660	2861	R. H. Robison
MS0827	1511	P. R. Schunk	MS0660	2861	R. J. Sikorski
MS0827	1511	J. A. Schutt	MS0661	2862	M. R. Ashby
MS0834	1512	M. R. Baer	MS0625	2884	R. L. Burchard
MS0834	1512	R. J. Gross	MS0705	6114	J. R. Waggoner
MS0834	1512	M. L. Hobbs	MS1325	6313	J. F. Holland
MS0835	1513	D. R. Helmich	MS1325	6313	J. Jung
MS0835	1513	R. E. Hogan, Jr	MS1325	6313	R. S. Longenbaugh
MS0835	1513	R. D. Skocypec	MS1345	6331	E. A. Boucheron
MS0443	1561	J. G. Arguello	MS1175	6513	D. S. Oscar
MS0443	1561	S. N. Burchett	MS9043	8743	M. Callabresi
MS0443	1561	R. S. Chambers	MS9043	8743	M. Horstemeyer
MS0443	1561	M. W. Heinsteins	MS9018	8523-2	Central Technical Files
MS0443	1561	E. L. Hoffman	MS0899	7141	Technical Library (5)
MS0443	1561	S. W. Key	MS0619	7151	Technical Publications
MS0443	1561	J. R. Koteras	MS0100	7613-2	Document Processing
MS0443	1561	H. S. Morgan			for DOE/OSTI (10)
MS0443	1561	M. K. Neilsen			
MS0443	1561	V. L. Porter			
MS0443	1561	C. M. Stone			
MS0443	1561	B. J. Thorne			
MS0443	1561	J. R. Weatherby			
MS0443	1561	G. W. Wellman			
MS0437	1562	C. R. Adams			
MS0437	1562	E. P. Chen			
MS0437	1562	J. D. Gruda			
MS0437	1562	K. W. Gwinn			
MS0437	1562	F. J. Mello			
MS0437	1562	K. E. Metzinger			
MS0437	1562	E. D. Reedy			
MS0437	1562	A. M. Sastry			
MS0437	1562	K. W. Schuler			
MS0437	1562	G. D. Sjaardema			
MS0437	1562	A. M. Slavin			
MS0437	1562	J. W. Swegle			

DATE

FILMED

9 / 19 / 94

END