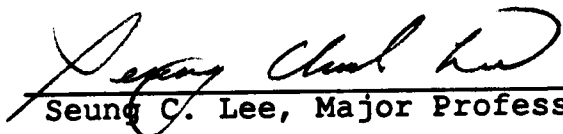


To the Graduate Council:

I am submitting a thesis written by Yukkin Daniel Chan entitled "Design of an X Window Interface for an Engineering Graphics Application." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Seung C. Lee, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:



Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Request for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Daniel Chan

Date 7/23/90

DESIGN OF AN X WINDOW INTERFACE FOR AN
ENGINEERING GRAPHICS APPLICATION

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Yukkin Daniel Chan

August 1990

Copyright © Yukkin Daniel Chan, 1990
All rights reserved

DEDICATION

I would like to dedicate this thesis to my parents, Mr. Tak Hon Chan and Mrs. Lily Chan, and my sister, Mrs. Salina Kwong for their love, encouragement, and unfailing support.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Seung C. Lee for the patience and understanding he continuously displayed during the preparation of this thesis. I would like to also thank Dr. Bruce Whitehead for serving on my thesis committee.

I would like to give special thanks to my third committee member, Dr. Kenneth Kimble, for allowing me to work on the research project, on which the thesis was based.

Lastly, I would like to thank Mr. Patrick Sherley for many hours he spent aiding me in the preparation of this thesis and all the staff in UTSI computer center for helping and supporting me.

ABSTRACT

Many application engineers spend a significant amount of time to develop user interfaces for their applications, but most of these interfaces are machine dependent.

A friendly graphics programming interface tool is developed by using the X Window system, which was developed at MIT, to reduce the development time and to provide user interface independence for engineering graphics applications. X Windows is a distributed system based on a client/server architecture. The programming interface takes advantage of X Window features such as networking connectivity, menu-driven user interface, machine independence, windowing capability, and portability.

PLOT3D, which provides convenient display of computational fluid dynamics, is used to demonstrate the programming interface. This thesis describes how to operate the user interface of PLOT3D in X Window and to utilize the programming interface for other applications. Recommendations for further development of the programming interface and a discussion of the weaknesses and the strengths of the interface are provided.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION	1
II. X WINDOW OVERVIEW	6
2.1 Development of X Window	6
2.2 Network Transparency and Portability	7
2.3 X Protocol, Xlib, and X Toolkit	10
2.4 The X Resource Manager	14
2.5 Window Management	19
III. INTERFACING TECHNIQUES	21
3.1 Interface Independence	21
3.2 Menu Driven User Interface	23
3.3 Fortran and C Interface	24
IV. IMPLEMENTATION OF THE INTERFACE	26
4.1 Overall Structure of the interface	26
4.2 Engineering Application Example	28
4.3 Interfacing between Control and PLOT3D	
Parser Modules	29
4.4 Implementation of the Interface	30
4.4.1 Implementation of the Screen	
Layout	30
4.4.2 Help Facility	39
4.4.3 Additional Window Handling	41
4.4.4 Description of Graphics Module .	43

CHAPTER	PAGE
V. CONCLUSIONS AND RECOMMENDATIONS	49
5.1 Conclusions	49
5.2 Future Research Recommendations	54
LIST OF REFERENCES	57
APPENDICES	60
APPENDIX A. X WINDOW CONTROL MODULE	61
APPENDIX B. GRAPHICS SUBROUTINES FOR USER INTERFACE	100
VITA	110

LIST OF FIGURES

FIGURE		PAGE
1.	The components of X Window	3
2.	X Window's Client/Server architecture	8
3.	Programmer's view of X Window	13
4.	An example application	16
5.	Widget tree for Figure 4	17
6.	The resource file for Figure 4	18
7.	Interaction between end-user, interface, program and machine	21
8.	The control flow of an application using X Window for user interface	27
9.	Screen layout	31
10.	Widget tree for the PLOT3D interface	33
11.	Submenu layout	35
12.	The code for creating a command button in main menu	34
13.	The structure of "MenuSubset"	36
14.	An example of an input window	38
15.	An example of a help module	40
16.	The resource file for PLOT3D	42
17.	Temperature Function	51
18.	Enthalpy Function	52
19.	Density Function	53

Chapter I

INTRODUCTION

In recent years, the use of high performance personal computers and graphic workstations has been increasing in the engineering field. All of these computers consist of a wide variety of hardware and software configurations, especially regarding the computer graphics. They also come with different kinds of graphics packages such as the Graphics Kernel System (GKS)¹ or the Core system². Such disparity of systems has meant great difficulty in porting engineering graphics applications from one machine to another because these applications are simply not developed for some computers. Therefore, application engineers normally spend significant time and energy on designing the user interfaces.

Most of the existing interfaces simply provide primitive command languages. The user controls the application using the rather low-level primitive instruction sets in each specific command language. The development of a more user-friendly and machine-independent user interface for engineering applications is necessary. In order to help solve the user interface problem for the application engineers, there is a need to provide a convenient programming interface.

Developed at Massachusetts Institute of Technology (MIT), "the X Window System is a device independent, multitasking

windowing, and graphics system designed to operate across heterogeneous networks"³. Supported by several major companies such as IBM and Digital Equipment Corporation, X Window System (often referred to as X) is becoming an industry-standard software system that allows programmers to develop portable graphical user interfaces. X is a distributed system based on a client/server architecture⁴; this architecture consists of five main components (see Figure 1):

- (1) **X Library** (or simply Xlib): provides functions for connecting to a particular display server, creating windows, drawing graphics, responding to events, and etc.
- (2) **X Server**: is a program that runs on each system that supports a graphics display, keyboard, and mouse. The server interfaces with the outside world, such as Xlib, using the X network protocol.
- (3) **X Application** (or X client): is a program to perform a variety of tasks, including terminal emulation and window management. Applications need not run on the same machine as the X server program.
- (4) **X Protocol**: serves as a link between X server and X application.
- (5) **X Toolkit**: divides into two libraries that house components called intrinsics and widgets. Intrinsics are low-level utility routines that can implement user-interface codes. Because they are low-level, they must be activated by higher-level routines, the widgets, in order

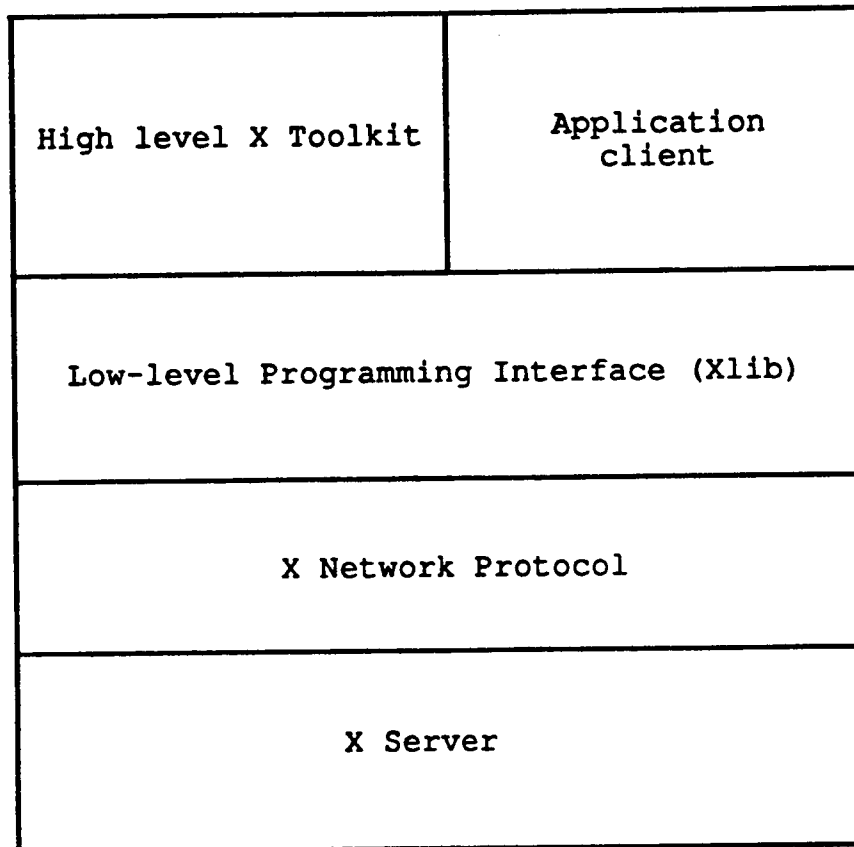


Figure 1. The components of X Window.

to produce screen-based, "look and feel" components that make up a graphical user interface (GUI)⁵.

For now, X Window is the only GUI that really works in a multiuser, multicomputer, networked environment.

X Window has already provided a set of GUI routines written in the C language. X is an extremely large and complex software package, and since most application engineers are mainly familiar with the Fortran, which is quite different from C, programmers have to spend a significant amount of time learning how X Window works and familiar with C in order to program in X Window.

The objective of this research was to design a convenient programming interface to bypass the C language for the engineering graphics application using X Window to provide machine independence, networking connectivity, menu-driven user interface, and windowing capability.

Chapter II presents an overview of X Window. It discusses the history, internal structure, and the resource and the window management of X in detail.

Chapter III discusses the technical approaches for implementing the interface. The approaches include user interface independence, menu-driven user interface, and C and Fortran interface.

Chapter IV presents the actual implementation of the interface. It also discusses how an engineering programmer can

modify codes of the interface in order to adapt to personal applications. PLOT3D is taken as an experimental application using the interface because it is a well developed engineering graphics package. This chapter describes how the user interface works for PLOT3D.

Chapter V presents the conclusion and future development. It discusses the strengths and the weaknesses of the interface and gives suggestions for further improvement of the interface.

Chapter II

X WINDOW OVERVIEW

2.1 Development of X Window

In 1984, MIT had a campus-wide computer network made up of computers donated by various vendors; X was developed by Jim Gettys and Robert Scheifler to achieve portability and compatibility to deal with this heterogeneity. The name, X, as well as some initial design ideas, were derived from a window system called 'W' developed by Brian Reid and Paul Asente at Stanford University.

When MIT first released the version X10.4 in 1986, the response was overwhelming. Many manufacturers expressed interest in X as a commercial product. Finally, MIT formed a consortium with most of the leading workstation manufacturers, such as DEC and Hewlett-Packard, to develop X Window further and had it adopted as an ANSI standard in the summer 1988⁶.

The book entitled "X Window Systems Programming and Application with Xt" by Douglas A. Young stated that "One of the most important features of X is its unique device-independent architecture. X allows programs to display windows containing text and graphics on any hardware that supports the X protocol without modifying, recompiling, or relinking application. This device independence, along with X's position

as an industry standard, allows X-based applications to function in a heterogeneous environment consisting of mainframes, workstations, and personal computers."⁷

2.2 Network Transparency and Portability

X is based on a client-server model (see Figure 2) in order to provide two important features, network transparency and device independence. A server is a single process completely controlling all input and output devices, and a client is an application that uses the facilities provided by the X server. For each physical display, there is a controlling server. A client and a server communicate over a reliable duplex byte stream. The X protocol is layered on top of the byte stream; it can be used over TCP, DECnet, or Chaos. If the client and server are on the same machine, the stream is typically based on a local interprocess communication (IPC) mechanism; otherwise a networking connection is established between them³.

Multiple clients can have connections open to a single server simultaneously, and a client can have connections open to multiple servers simultaneously. The server communicates with the client by sending an event to the client application and by placing the event in a first-in first-out (FIFO) queue that can be read by the client. Each event consists of a data-structure containing the type of event, the window in which

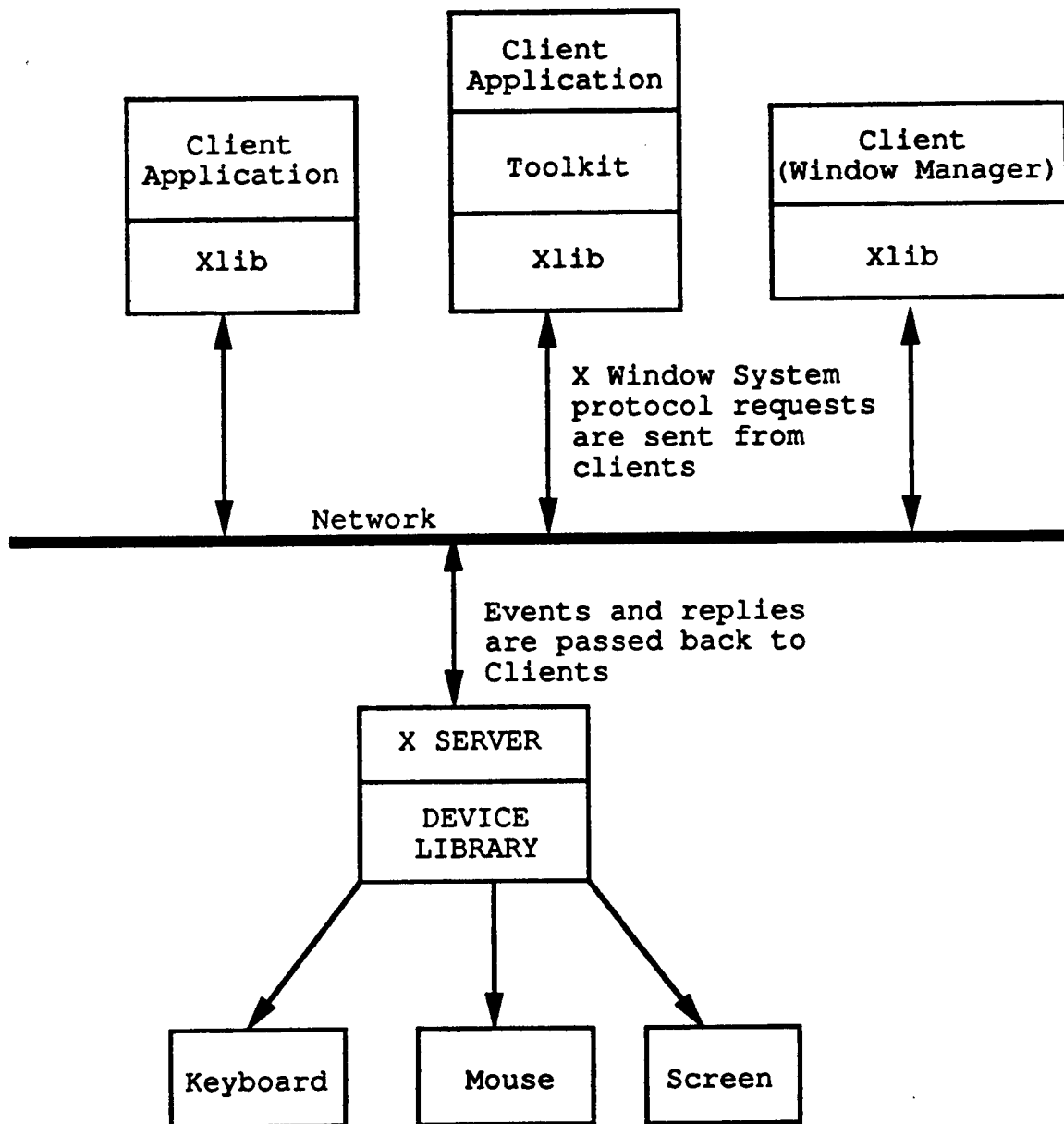


Figure 2. X Window's Client/Server architecture

the event occurred, and other data specific to the particular type of event. X supports 33 types of events, and provides a mechanism that allows clients to define additional event types. The server generates events as a direct or indirect result of a user action (such as pressing a key on the keyboard, moving the mouse, or pressing a mouse button)³. Most X applications are completely event-driven and are designed to wait until an event occurs, respond to the event, and then wait for the next event. The interface design, which is discussed in Chapter IV, also uses the event-driven approach.

An X application can be accessed by any bit-mapped graphics workstation in a network; however, the program actually executes on the available computer best suited to process the program. For instance, if a personal computer lacks the performance and memory to execute a large engineering application, a mainframe can run the program while the users interact with the application through a window on the personal computer. The X Window system's network transparency makes remote applications appear as if they were running on their own computer⁸.

In addition to providing an architecture for network transparency, X also simplifies the porting of software. Because the X server encapsulates the base window system, it provides the fundamental resources and mechanisms required to implement various user interfaces. All device dependencies are implemented on the X server; therefore, the communication

protocol, X protocol, between clients and servers is device independent. By placing all the device dependencies on one end of a network connection, applications are really device independent, so that the users do not need to rewrite, recompile or relink an application for each new hardware display.

2.3 X Protocol, Xlib, and X Toolkit

In addition to portability, X provides a flexible platform for developing and supporting window-based applications and user-interface toolkits.

The X Protocol defines each packet of information that is transferred between the server and Xlib in both directions. Even when the server and the client are running on the same machine, the protocol is used for communication. There are four different kinds of packets that are transferred via the protocol^{5,9}:

- (1) **Request** : A one-way protocol request message is generated by Xlib and instructs the server to perform specific actions.
- (2) **Reply** : A round-trip protocol request message is sent from the server to Xlib in response to certain requests.
- (3) **Event** : An event message is sent from the server to Xlib and contains information about either a device action or a side-effect of a previous device action.

(4) **Error** : An error message is a special type of event message; it reports the server-detected errors in requests.

Xlib is a low-level library of C language routines to interact with the X server; it generates and sends the X protocol request when it is called. Furthermore, Xlib contains approximately 300 graphics routines, many of which resemble those found in any graphics package. For instance the drawing primitives, including XDrawPoints, XDrawLines, XDrawRectangle and XDrawArc, are easy-to-use routines capable of drawing points, lines, rectangles, and circles, ellipses, or arcs. Xlib also provides other functions such as creating and configuring windows, controlling color, manipulating font, and handling events sent from the X server⁵.

Because Xlib defines an extensive set of procedures that provide complete access and control over the display, windows, and input devices, application programmers can define a personal user interface for each of their applications. However, this library can be time-consuming and difficult to use correctly. Just handling the window manager conventions can require hundreds of lines of code. Many programmers prefer to use one of the higher-level toolkits to build their applications⁷; therefore, the interface design, discussed in Chapter IV, uses X toolkits instead of Xlib.

X Toolkit is layered on top of Xlib and provides an easier programming interface than Xlib. It consists of two parts: Intrinsics and a set of user interface components called Widgets. The Intrinsics are a set of user interface primitives that are themselves free of visual and interaction style in order to provide the base mechanisms to build a wide variety of widget sets and application environments. These primitives and the widget set are smoothly integrated with Xlib, so applications that use the additional facilities of the intrinsics and widgets can also use the functions provided by Xlib when needed¹⁰.

There are already several widget sets on the market such as InterView (built at Stanford), Andrew (developed at Carnegie Mellon), and CLUE (developed at Texas Instruments). Widget sets are usually very similar; a programmer who is familiar with one widget set should be able to quickly learn to use any other widget set^{11,12}. For practical reason at UTSI, my interface design uses the X widget developed at Hewlett Packard.

Figure 3 shows that the X Widgets system is layered on top of the X Toolkit Intrinsics, which in turn is layered on top of X Window system, thus extending the basic abstractions provided by X. The primary data type in the X Widget system is the widget. A widget is dynamically allocated and contains state information, and it may be thought of as a rectangle with associated input and output semantics. The X Widget set

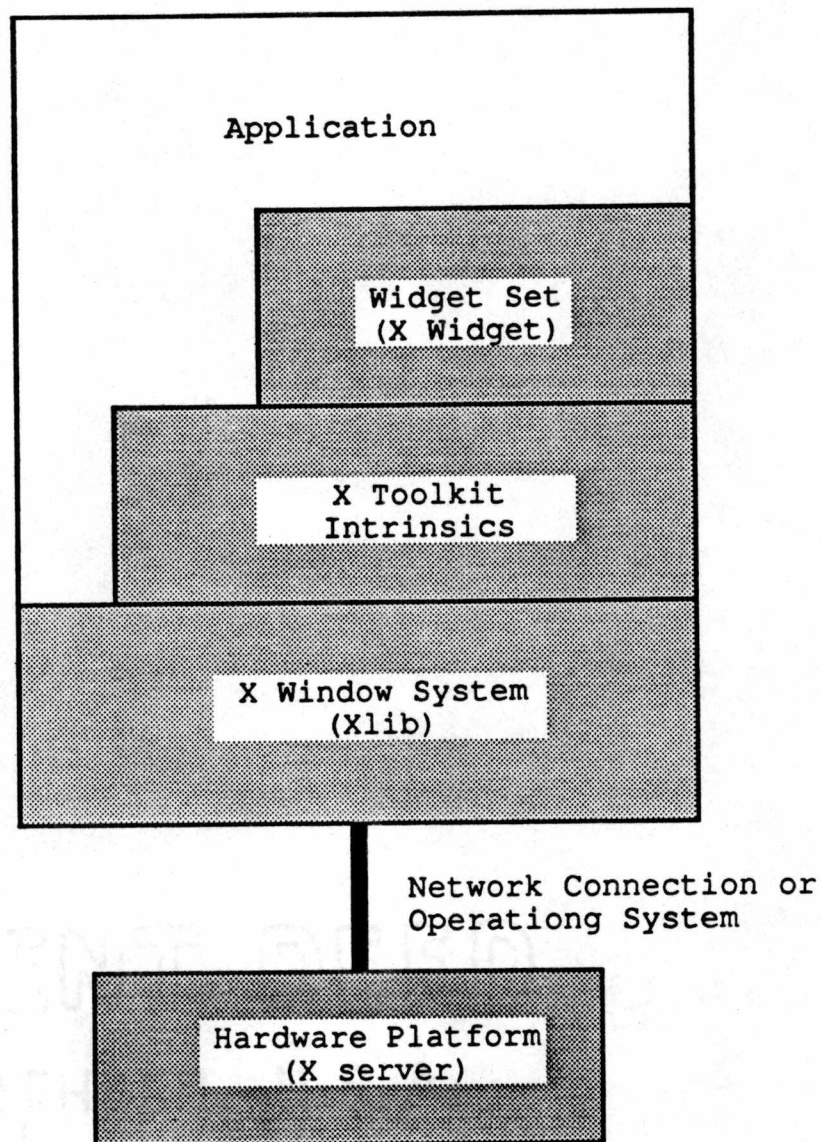


Figure 3. Programmer's view of X Window.

provides programmers with some simple user interface components, including scroll bars, menus, and buttons, to build their own interfaces¹³. A salient disadvantage of the X toolkit, however, is that programmers have to be familiar with C language in order to develop an application in X since both the widgets and the intrinsics are written in C.

2.4 The X Resource Manager

In X, the term "resources" refers to windows, graphics contexts, fonts, color maps, pixmaps, and cursors. The resources are maintained by the server and accessed by clients through a resource identifier, a 32-bit integer containing unique identifying values. Furthermore, they are shareable among all clients which connect to the same X server; any client that knows the identifier of a particular resource can use, alter or destroy that resource. All resource information is stored in the resource database managed by the X resource manager^{4,7}.

The resource manager provides an easy-to-use mechanism for determining user customization and specifying defaults; this allows programmers to write customizable applications. The fundamental resource-manager facilities supported by Xlib permit applications to store and retrieve information from a resource database. Traditional databases contain information that is completely and precisely specified and use imprecise

queries to search for information, but the resource manager uses a slightly different model: the database in X contains general information, usually specified by the user, about the resource used by applications; an application can then query the resource database using a complete specification of the desired resource⁵.

Two specific strings uniquely identify a resource for a particular window or widget. The first string consists of (1) name of the application, (2) the names of each widget in the application's widget tree between the top widget and the widget using the resource, and (3) the name of the resource. The second string is similar but uses class names instead of resource names for the application and each widget. Together, these strings specify a unique path through the application's widget tree⁷. Figure 4 and Figure 5 show the widget layout and hierarchy of an example graphics editor, "draw". Figure 5 shows the name of each widget in Figure 4 with its class name in parentheses.

Most applications allow users to specify look and feel preferences using a simple mechanism. A preference string consists of (1) the name of an application, (2) one or more predefined variables that indicate the preference to set, (3) a colon and one or more spaces, and (4) the actual value of the preference. The user can specify preferences for each resource in a resource file. The example resource file shown in Figure 6, specifies the application the default background

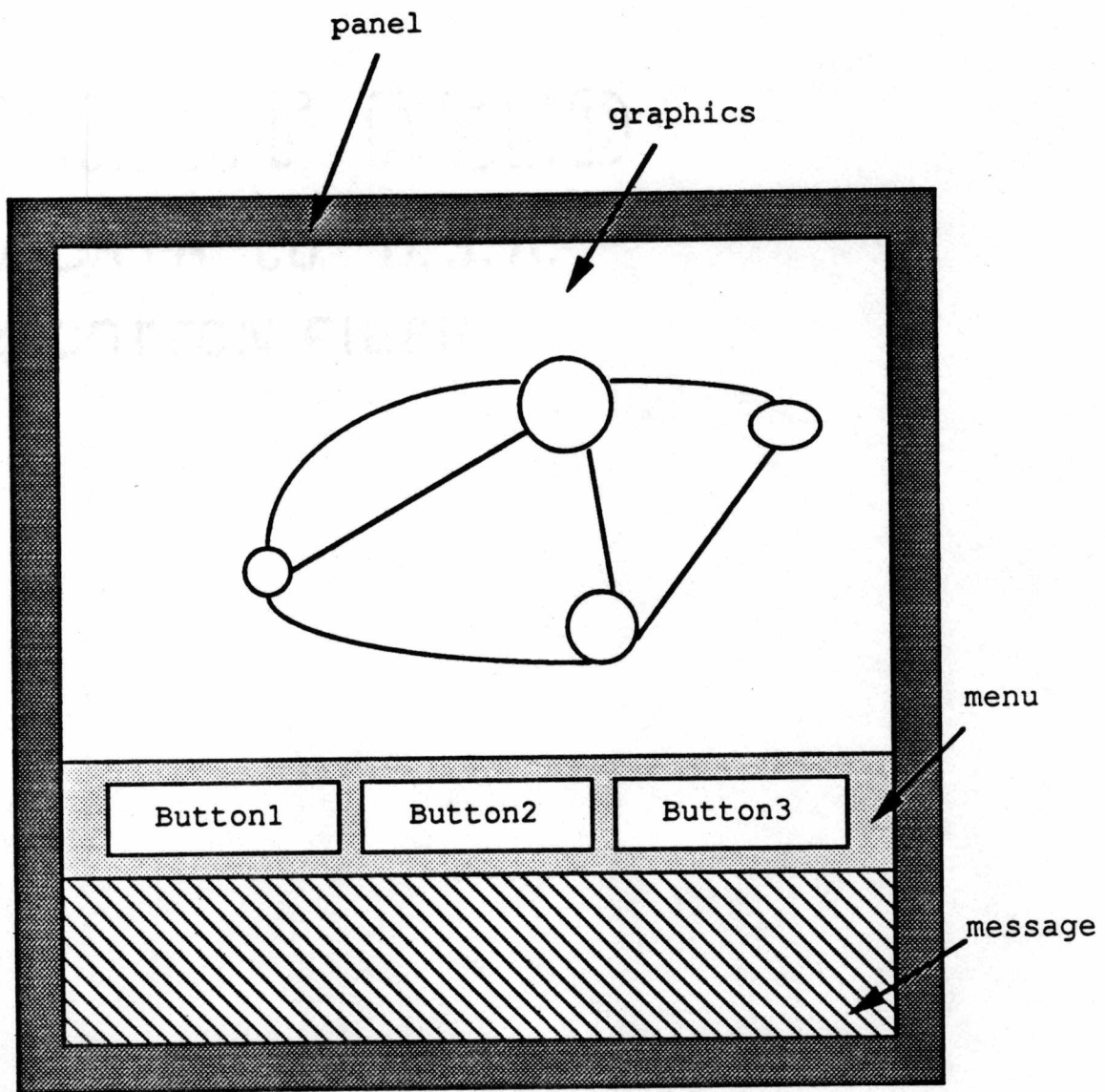


Figure 4. An example application.

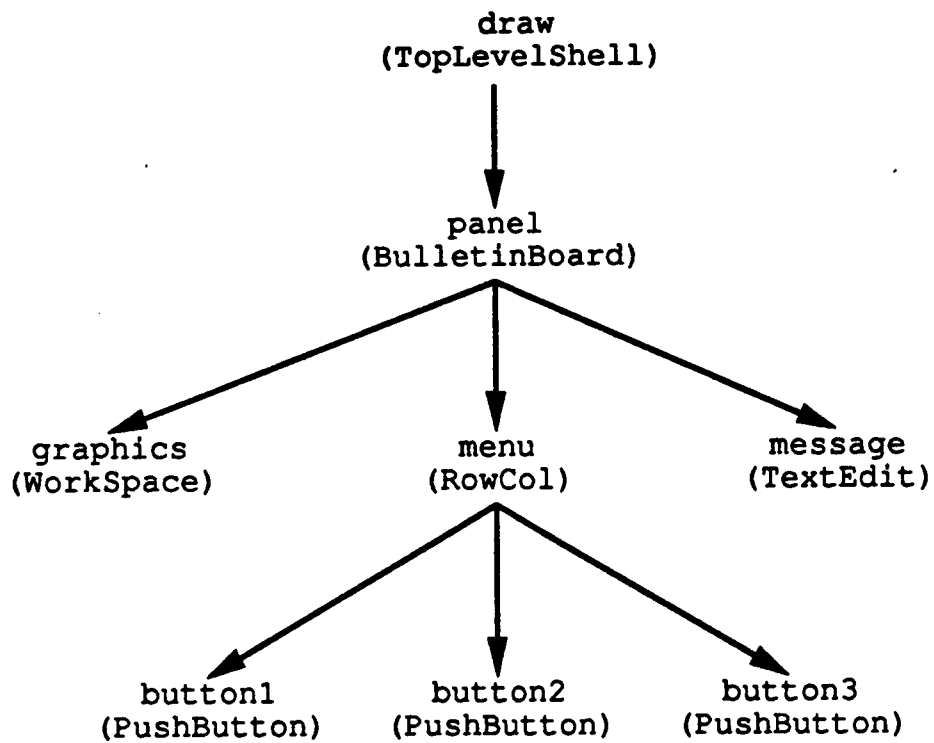


Figure 5. Widget tree for Figure 4.

Draw*button1*background:	white
Draw*button1*Foreground:	red
*button2*background:	green
*button2*Foreground:	yellow
Draw*foreground:	black
Draw*background:	white
*PushButton*background:	blue
Draw*forceSize:	True
*font:	fixed
panel*borderWidth:	4
Draw*menu.yRefName:	graphics
Draw*message.yRefName:	menu
panel*menu*xResizable:	True
panel*menu*yAddWidth:	True
Draw.panel.graphics.yResizable:	True
*message.yAttachBottom:	True
Draw*message*y:	600
*graphics*width:	500
*graphics*height:	500
*message*width:	500
*message*height:	200
*menu*height:	100

Figure 6. The resource file for Figure 4.

color as white and foreground as red of the "button1" in the graphics editor shown in Figure 4. When an application starts running, it automatically loads the user's resource files in the resource database.

2.5 Window Management

In X, windows make it easy for applications to organize the workstation screen into areas for specific purposes, such as menus, texts display, and graphical display. (The book entitled "Introduction to the X Window System" by Oliver Jones has an in-depth discussion of how to create, manage, and manipulate windows in X⁹).

The client doesn't actually control where the window appears or what size it is. Instead, it gives hints, such as where or how it would like to display, to a window manager. A window manager is a program written with the X library and given special power to control the layout of windows on the screen. The window manager allows the user to control the location and size of windows, start new applications, and control the stacking of windows on the screen, but only according to the window manager's window layout policy⁵.

Different window managers have different layout policies, which are sets of rules that specify allowable sizes and position of windows, icons, etc. There are several window managers, developed by different companies and universities,

available for X, such as Ultrix Window Manager (UWM), Tom's Window Manager (TWM), and Generic Window Manager (GWM)⁹. The interface design discussed in the next chapter only works well with the GWM, because GWM allows the application to modify the color map in the system. (UWM and TWM do not permit any applications to modify the existing color map.)

Chapter III

INTERFACING TECHNIQUES

3.1 Interface Independence

A user interface is a program that is the link between the end-user, the application and the machine (see Figure 7).



Figure 7. Interaction between end-user, interface, program and machine.

Typical approaches to interactive engineering applications require the interface module and the computational module to fit tightly together. Consequently, the interfaces become less flexible and cannot be easily modified as the design of the application changes. Moreover, the development of a quality user interface involves an iterative cycle of design and evaluation, and one of the important criteria for development of a good user interface is fast and convenient modification.

Engineering application programmers have a similar problem, the need for convenient data modification without changing the corresponding program. The solution is interface

independence, which allows design decisions affecting only the user interface to be isolated from those affecting only the application system structure and the computational program. This means that the appearance of the interface to the end-user and choices of interaction styles (such as menus, forms, and command languages, used to extract inputs from the end-user) are not known to the computational program¹⁴.

Interface independence separates the design of the interface from the design of the computational component of an application so that modifications in either do not cause changes in the other. For instance, the PLOT3D application, which is discussed in the next chapter, uses the pop-up menu for user interface. But if the end-users prefer to use a pull-down menu instead, they only need to change the user interface portion of the application, because the interface is completely independent.

In addition to easy modification, interface independence also significantly reduces the difficulty of porting the developed application software from one machine to another. Most of the computational portions of an application are highly compatible with different machine architecture and software configurations, but the interface portions are not. The interfaces are usually built on top of the specific graphics packages or require some operating system calls¹⁵. Therefore, in order to adapt an application that was developed

with the interface independence to other machines, one only needs to modify the interface part of the application.

3.2 Menu Driven User Interface

Menu-type and command-language-type are two basic types of software systems that interface with users. In a menu-type user interface (MUI), the user controls the software system solely through the selection of options from a number of choices presented. (It is assumed that only valid choices are presented to the user through the menu; therefore, it is not possible to select options that are not permitted.) In a command-language-type user interface (CLUI), the user controls the software system by using a predefined command language. The user has to know this language before he or she can use the system¹⁶.

The advantage of MUI is that users can become productive with a new system after a very short time because they do not have to learn command language. A naive or intermediate user may explore the operations provided by browsing through the menus. Menu systems may, however, be less satisfactory for expert users who have to work through several menus to get their work done. Proficient users who do not need help tend to prefer a concise CLUI that allows a few keystrokes to achieve the same effect as a number of menu selections¹⁷.

To satisfy both naive and expert users, user interfaces should be implemented in a software system so users can choose the interface they prefer. This increases the software's development time, of course; the user interface described in this paper is designed for the naive and intermediate user only, but the future developers of this interface should think of accommodating expert users.

3.3 Fortran and C Interface

As discussed in Chapter II, Xlib and X toolkit are both written in C. Generally speaking, if application programmers are going to develop a program using Xlib and X toolkit, they have to write their program in C. However, engineering application programmers usually write their programs in Fortran, which is dissimilar to C. For instance, the components of a widget in X is a structure of C, but there is no structure capability in Fortran. For a programmer trying to implement a Fortran program with X, the program has to have a set of Fortran routines (which are discussed in the next chapter) built on top of Xlib and X toolkit in order to communicate with the X Window system.

There are some difficulties developing routines to interface with a Fortran program. The following are some of the difficulties I met during this project and some ideas for solving them.

- (1) All Fortran arguments are passed by references, while C arguments are passed by value. For solving this problem, the arguments defined in a C subroutine have to be the corresponding type of pointer.
- (2) The Fortran compiler appends a trailing underscore to all external names in Fortran programs. Programmers need to add an underscore to the name of the C function called.
- (3) For every Fortran argument that is of type CHARACTER, an argument is passed giving the length of the value. The programmer needs to define an extra INTEGER argument for C function following the CHARACTER argument to receive the value.

The difficulties and solutions mentioned above are only examples of the problem interfacing C and Fortran codes comprising the routines discussed in the next chapter. There are a lot of other differences between C and Fortran that created problems for the interface. (SUN Fortran programmer's guide discusses in more detail C and Fortran interfacing¹⁸.)

Chapter IV

IMPLEMENTATION OF THE INTERFACE

4.1 Overall Structure of the Interface

The design of an X Window user interface for engineering graphics application is divided into four modules. Figure 8 shows the relationship among these four modules. They are:

- (1) The X Window control module is the main program of the whole application; it is written in C using Xlib and X toolkit. The main function of the control module is to construct the commands and to pass them to the parser module after it receives an input from an end-user. The control module is discussed in more detail later.
- (2) The Parser module is written in Fortran; its function is to recognize the passing commands from the control module and to perform the corresponding action.
- (3) The Computational module is also written in Fortran. It computes the data based on what the end-user specifies the application to perform.
- (4) The Graphics module provides a set of routines, which are discussed below, for the application to draw the result after the computational module finishes the computation.

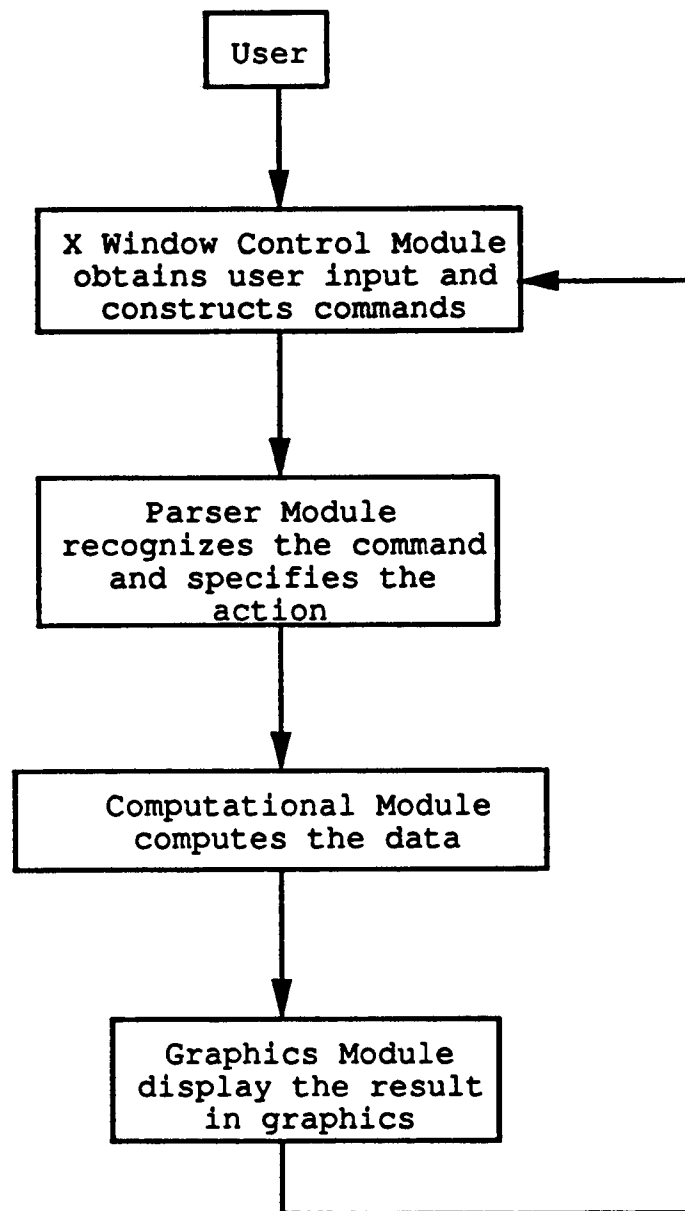


Figure 8. The control flow of an application using X Window for user interface.

4.2 Engineering Application Example

To test my X window interface, an original working engineering application that provided completely developed parser and computational modules was needed. PLOT3D was a good candidate because it had been used since 1980 and is very popular and comprehensive program for visualizing the solution files containing density, momentum, and stagnation energy values for each grid point.

Pieter Buning developed PLOT3D for computational fluid dynamics (CFD) research at NASA Ames Research Center. PLOT3D is a graphics program, written to provide convenient display of all or portions of two- and three-dimensional CFD grids and solutions. Using graphics workstations and other powerful computers, scientists use PLOT3D to see their work and to view the computational models from different angles. It reads grid and solution files and plots functions to produce on-screen graphics in order to help scientists understand the grids and their solutions¹⁹. The most important function of PLOT3D is letting scientists view and compare computed or experimental results. For instance, CFD output can be compared with wind tunnel results by putting wind tunnel data into grid form and running it in PLOT3D. (For more information about PLOT3D, refer to PLOT3D User's Manual, which is distributed by NASA Ames Research Center.)

Currently the user interface with the PLOT3D is running on the UNIX operating system of the SUN386i machine. The software is X Window system version 11.4 released by MIT in 1990.

4.3 Interfacing between Control and PLOT3D Parser Modules

The users interact with the control module via a mouse or a keyboard to determine what action the control module will perform. Then, the module constructs a command string (containing command, qualifier, and data), which is passed to parser module as an argument by calling a main function in the parser module.

After the parser module receives the command string, it extracts the PLOT3D command from the command string. The qualifier and the data are extracted from the rest of the command string, and then they are put in the FIFO queue for later use. After the module recognizes the specific PLOT3D command, it calls the appropriate function in the computational module.

However, this process takes time to be performed. In the future, if part of the parser module's function can put into the control module, it will improve the performance of the application and reduce the development load of the application programmer. For instance, the task of recognizing the command function can be put into the control module. The control

module would only send the qualifier and data to the parser, and the parser module would extract those items and put them into the FIFO queue. Consequently, the control module would directly call the appropriate function of the computational module.

4.4 Implementation of the Interface

This section discusses the development of the X Window control module and graphics module, and how the programmers can modify part of the X control module to use it with their applications.

4.4.1 *Implementation of the Screen Layout*

The control module is completely event-driven; it used Xlib and X toolkit to build the user interface consisting of menu, graphics, and message windows on the screen. The user interface is based on the screen layout in Figure 9.

On the right side of the graphics window is the main menu. This menu permits the user to operate the PLOT3D program. At the bottom of the graphics window is a message window which aids the user in the operation of PLOT3D.

The menu items are organized in alphabetical order except for the "Quit" command. The "Quit" command is located on the top of the main menu, so that the users can easily locate the

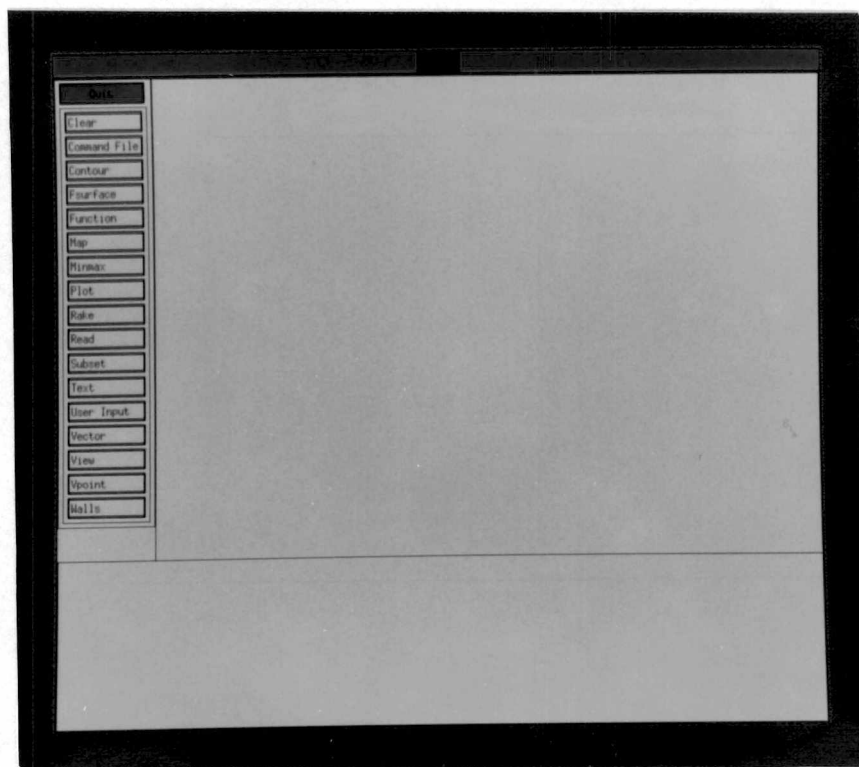


Figure 9. Screen layout.

command if they would like to terminate the program. The quit command is also written in a different color, green, to contrast with other commands, which are written in black.

Besides the "Quit" command, the rest of the main menu consists of the following commands in order from top to bottom: 1) Clear; 2) Command File; 3) Contour; 4) Fsurface; 5) Function; 6) Map; 7) Minmax; 8) Plot; 9) Rake; 10) Read; 11) Subset; 12) Text; 13) User Input; 14) View; 15) Vpoint; and 16) Walls. The user can find the desired command through the alphabetical order. There are many ways to organize the commands in the menu. For instance, the commands could be organized by frequency of use. The only constraint against using this in the interface is that the frequency of command usage depends on the particular user; therefore, there is no one absolute frequency use order to satisfy all the users.

Each command in the menu is created by using X toolkit. Figure 10 shows the widget tree of the PLOT3D user interface. All the commands are push-button widgets, which consist of a text label surrounded by a button border, and are activated by placing the cursor over the desired button and pressing a button on the mouse. Then the command needed to complete the operation will be activated by the particular subroutine associated with that button. When the user selects a button, the interior of the button will be inverted (the background will be filled with the foreground color and the text will be written in background color).

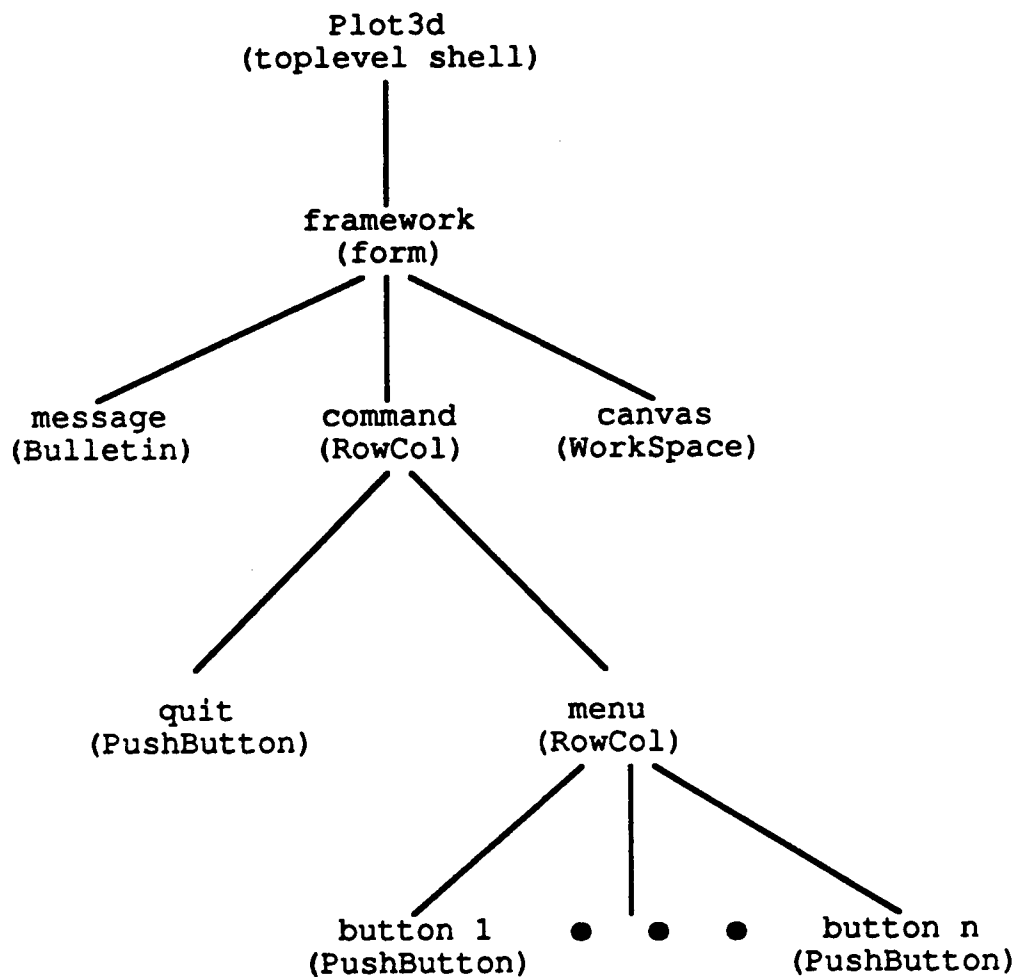


Figure 10. Widget tree for the PLOT3D interface.

```

1) w = XtcreateManagedWidget ("Subset", XwpushButtonWidgetClass,
                               commands, NULL, 0);
2) ws = create_menu_manager(w, "menu_mgr");
3) create_p(ws, "menu_mgr", "MAIN", MenuSubset, XtNumber(MenuSubset));

```

Figure 12. The code for creating a command button in main menu.

Except for the "Quit" button, all of the command buttons are associated with a pop-up submenu. Some of the entries in these submenus will generate other submenus if selected. A selection is made by pressing a mouse button and dragging the cursor through the list of menu items. The item highlighted is selected when the mouse button is released. Figure 11 shows one of the submenu layouts. The application programmer can create a main menu by modifying the code in the X Window control module. The three statements from the control module in Figure 12 show how to create a command button in the main menu. For these three statements, the label of this command button is "Subset", which is managed by the RowCol Widget, called "commands", created previously in the module. All of the commands in the main menu are managed by this RowCol widget. The submenu items of the "Subset" command are stored in the "MenuSubset" structure, which is discussed later. The first statement creates a push button widget with the name "Subset". If application programmers want different names for their command buttons, they change the word "Subset" to anything else. The second statement creates a pop-up manager to be associated with the command button, and the third statement generates a submenu using the "MenuSubset"

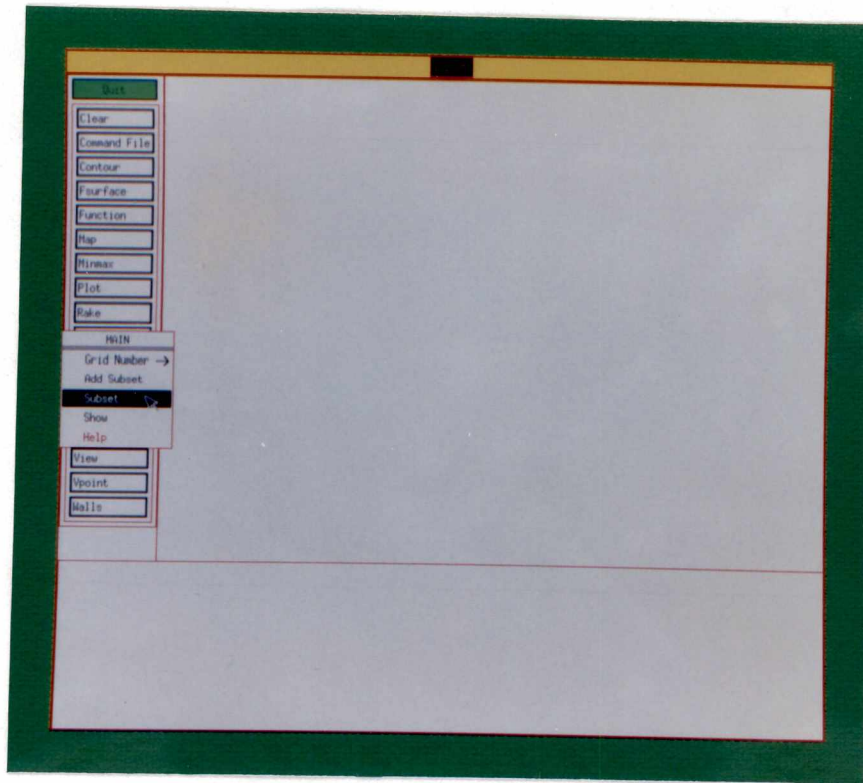


Figure 11. Submenu layout.

```

static menu_struct MenuSubset[] = {
    {"Grid Number",      dummy,      " "},
    {"Add Subset",       walls_sub,   "subs/add/att"},
    {"Subset",           walls_sub,   "subs/att"},
    {"Show",             pass_cmd,    "show subs"},
    {"Help",             help_cmd,    "subs"},
};

```

Figure 13. The structure of "MenuSubset".

structure. The submenu has a title called "MAIN". Programmers may change the submenu title by replacing the character string with another character string. This submenu is associated with the pop-up manager, which was created by the second statement. When the users move the cursor to this command button and press a button on the mouse, it will activate the pop-up manager to bring up the submenu.

Every command has the same submenu structure; the differences are the name and related function of sub command and the character string passed to the function. Figure 13 shows the structure of "MenuSubset". Figure 11 shows the layout of the submenu of "Subset" command. The commands of the submenu from top to bottom are: 1) Grid Number; 2) Add Subset; 3) Subset; 4) Show; 5) Help.

As shown in the structure of "MenuSubset", the first field is the sub-command name, the second field is the calling function, and the third field is the character string passed to the calling function. It is very easy for the programmers to customize their own submenu by changing these three fields. However, they need to generate their own calling function associated with that particular button. Appendix A shows the

control module; programmers may modify some of the calling functions for their own use.

Some of the items in the main menu require users to input data before the control module constructs a command which is then passed to the parser module. If this is a case, an input window will appear on the screen for users to input data. The input window consists of three different kinds of boxes, which are message, input, and control. The window is divided into two columns, on the right-hand side is the input box and on the left-hand side is the message box, with any number of rows. The message box displays a character string instructing users what they should enter in the input box adjacent to the corresponding message box. All the rows are made up of message box on the left and input box on the right except for the last row. The last row contains the control box which is a push button with a label written in pink for attracting attention. The function of this control button is to tell the system that the user has finished inputting data. Figure 14 shows one of the input windows.

For entering data, users need to place the cursor inside the input box. If there is more than one input box in the input window, users need to move the cursor to other input boxes in order to enter data. Furthermore, if users do not enter data for the input box, the system will use the default value (except for some input data for which the system does not have a default value). For example, the 'read' command

The image shows a screenshot of a software input window. The window has a green background. In the center is a white rectangular form with a thin black border. At the top of the form is a title bar with a small red square icon on the left and the text "INPUT" in the center. Below the title bar are eight rows of input fields. Each row consists of a text label on the left and a text input box on the right. The labels are: "I start[,end[,inc]] use ':' to separate another set of I", "J start[,end[,inc]] use ':' to separate another set of J", "K start[,end[,inc]] use ':' to separate another set of K", "LINES, HIDDEN_LINE, or SHADED_SURFACE", "Color", "Line type", "Line thickness factor", "Symbol type", and "Symbol size factor". Each input box has a small upward-pointing arrow on its left side. At the bottom of the form are two buttons: "DONE" on the left and "NEXT INPUT" on the right.

INPUT	
I start[,end[,inc]] use ':' to separate another set of I	
J start[,end[,inc]] use ':' to separate another set of J	
K start[,end[,inc]] use ':' to separate another set of K	
LINES, HIDDEN_LINE, or SHADED_SURFACE	
Color	
Line type	
Line thickness factor	
Symbol type	
Symbol size factor	
DONE NEXT INPUT	

Figure 14. An example of an input window.

must have an input file name. After users finish entering data, they need to move the cursor to one of the control boxes and press the mouse button to confirm to the system that they have finished entering data. The input window will disappear from the screen, but if the user enters wrong data, enters no data in an input box that doesn't have a default value, or selects the "next" control box, the same input window will appear again for the user to enter another set of data. For wrong input data and no input data for input boxes with no default value, an error message will display on the message window.

4.4.2 *Help Facility*

All of the submenus have a "Help" command. The "Help" command is written in red and other items in the submenu are written in black. Therefore, users could easily spot the command if they need help. When users select the "Help" command, the help message (about the command related to this "Help" command) will display on the message window.

The help message is stored separately in a help file in the system. Whenever users choose the "Help" command of one of the commands in the main menu, the application opens the help file to find the appropriate help message to display on the message window. Application programmers can create help

```

1Minmax
  0   Set the plot ranges.
  2Qualifiers
  -2/X
  -2/Y
  -2/Z
  0   Controls which set of axis limits are to be changed.

```

Figure 15. An example of a help module.

messages for their commands by modifying or creating a help file.

A help file may contain one or more help modules. A help module is the line of help text related to the same topic or key. Each module within a help file contains a group of related keys or topics, numbered key 1 through key 20. Each key represents a level within the hierarchy of the module. The key-1 name identifies the main topic of help information, for example, the name of a command in your program that requires explanation. The key-2 through key-20 names identify subtopics that are related to the key-1 name, for example, the command's parameters or qualifiers or both. However, if the subtopic is started with a "/" character, programmers need to enter a negative sign followed by the subkey. Figure 15 shows an example of a help module.

Each line in the module consists of the key number in the first to third column, followed by the name of the key. The subtopic lines following key-2 through key-20 consist of the subkey number followed by the name of the subkey. The help line has a key-0 in the third column to identify the following text as a help text for the topic or the subtopic right above

it. The system recognizes a group of key-1 and subkey lines, and their associated text, as a module. A module is terminated either by another key-1 line or by an end-of-file record.

Programmers can specify their help file by replacing the default help file name in the resource file for the application. As shown in Figure 16, the PLOT3D resource file, "plot3d.hlb", is the default help file name. This resource file is also used to generate the screen layout in Figure 9. If programmers want a different kind of screen layout, they only need to change the resource file. For instance, the width and height of the graphics window is 500 pixels. In addition, programmers can specify a default value in the resource file for some of the variables in the X control module; therefore, they do not need to recompile the control module when they just want to change the default value for the variables.

4.4.3 *Additional Window Handling*

When the submenus or other windows appear on the screen, they may cover all or part of the graphics window. After they disappear or move away from the graphics window, the region of the original picture that had been covered will sometimes be erased.

There are two approaches to deal with this erased region problem. One is to draw the whole picture again. The application can save the picture in a pixmap, which is stored

*form*columns:	2
Plot3d*quit*background:	grey
Plot3d*Help*Foreground:	black
Plot3d*foreground:	white
Plot3d*background:	yellow
*form*foreground:	white
*form*PushButton*background:	green
*form*background:	yellow
*forceSize:	True
Plot3d*file:	plot3d.hlb
Plot3d*canvas.xRefName:	command
Plot3d*canvas.yRefName:	command
Plot3d*message*y:	501
*canvas*width:	500
*canvas*height:	500
*message*width:	500
*message*height:	500
Plot3d*canvas*xResizable:	True
Plot3d*canvas*xAddWidth:	True
Plot3d*canvas.yResizable:	True
Plot3d*canvas*xAttachRight:	True
Plot3d*canvas*xAttachLeft:	True
Plot3d*canvas*yAttachTop:	True
Plot3d*canvas*yAttachBottom:	True
*command.yAttachBottom:	True

Figure 16. The resource file for PLOT3D.

in off-screen memory in the X server, and then copy the pixmap to the window when a region of the window exposes again. However, generating a pixmap takes a lot of time and memory space because the application needs to draw into the pixmap and then copy the pixmap to the window each time the window changes instead of only drawing to the window. By another method, the application can issue all the Xlib commands to the X server to redraw the picture. It takes time for the application to send commands to the X server and for the server to process these commands.

The second approach is to ignore the erased region of the picture. This means that the application will not do anything to restore the picture. The second approach has been taken since it will not waste time in re-generating the old picture. Users' selection of commands in the submenus also indicates that they want to view a new picture.

When users move the cursor inside any one of the application windows (message, command, graphics), the cursor changes from a crossbar to an arrow in order to indicate that users are operating the application (PLOT3D).

4.4.4 Description of Graphics Module

The Graphics module is a set of primitive graphics routines allowing application programmers to display their results on the graphics window. Table 1 shows the routine's

Table 1. The routine of the graphics module.

Subroutine	Argument	Description
charst(str, length)	CHARACTER str() INTEGER length	draw a character string
clear	NONE	clear the graphics window
cmov(x,y,z)	REAL x, y, z	move the 3D current character position
cmov2(x,y)	REAL x,y	move the 2D current character position
color(c)	INTEGER c	specify the current color†
draw(x, y, z)	REAL x, y, z	draw a 3D line
draw2(x, y)	REAL x, y	draw a 2D line
ginit	NONE	start the graphics package
move(x, y, z)	REAL x, y, z	move the 3D current position
move2(x, y)	REAL x, y	move the 2D current position
ortho2(left, right,lower, upper)	REAL left, right, lower, upper	use parallel projection
perspe(fovy, aspect,near, far)	REAL fovy, aspect, near, far	use perspective projection
sclsys(vector, xsc,ysc,zsc)	REAL vector(4), xsc, ysc, zsc Input: vector(4) Output: xsc,ysc,zsc	compute screen coordinates from viewport and setdepth
xfmsys(untvec, wvec)	REAL untvec(4), wvec(4) Input: untvec(4) Output: wvec(4)	perform the projection transformation of the coordinate

†see Table 2.

Table 2. Color Table.

Argument	Color
0	black
1	red
2	green
3	yellow
4	blue
5	magenta
6	cyan
7	white

name, its arguments, and its description; the source code of the routines are shown in Appendix B.

All of the routines are built on top of an engineering package, available on the market, called Vogle (A very ordinary graphics learning environment). One of the features of Vogle must be changed before it can be used, because although Vogle is written in Xlib, it opens its own separate window to display new drawings each time one is created and does not cooperate with other windows (message and command) in the design. Therefore portion of Vogle's coding is re-written in X toolkit in order to create a graphics window for drawing pictures that would cooperate with other windows in the screen layout.

The description of how each routine works is given in the following:

- (1) **charst**: allows programmers to display a string starting at current character position using current font and color.
- (2) **clear**: sets the graphics window to the current color and current graphics position and sets the current character position to undefined.
- (3) **cmov**: sets current character position in 3D, which after transformation into screen coordinate, becomes the new character position.

- (4) **cmov2:** projects the position from model space to screen coordinate, determining current character position for subsequent string display.
- (5) **color:** sets the current color map index value for subsequent pixels written to bit plane memory (see the color table for the color index).
- (6) **draw:** draws from current graphics position to the argument point.
- (7) **draw2:** draws a 2D line from the current graphics position to the argument point.
- (8) **ginit:** initializes all the graphics variables and clears the window.
- (9) **move:** sets the given point as current graphics position without any visible change on the screen.
- (10) **move2:** sets the given 2D point as current graphics point and sets the z coordinate to zero.
- (11) **ortho2:** defines the viewing box into the world coordinate system for mapping from the eye-fixed coordinate system to the screen coordinate system, using the rules of 2D orthographic projection.
- (12) **perspe:** defines the viewing pyramid into the world coordinate system for mapping from the eye-fixed coordinate system to the screen coordinate system using the rules of perspective.

- (13) **scsys:** computes the screen coordinate from the viewport.
- (14) **xfmsys:** performs the coordinate transformation of the projection.

Chapter V

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

The major objectives of this research in developing the programming interface were 1) to provide application programmers with a better and easy-to-use user interface tool, 2) to separate the computational module and the user interface module, and 3) to create the capability of porting from one machine architecture to others.

With the X Window interface developed through this work, programmers can modify part of the coding in the X control module (see Appendix A) to incorporate with their own applications, and create their own style (look and feel) in user interface. For instance, they can create their own command buttons and submenus to put into the main menu.

The graphics module of the interface provides a set of graphics routines that make it easy to display the results of the application after computation. Programmers do not need to learn how to use the Xlib and the X toolkit in order to use X; they also do not need to know C language because the routines provide the C and Fortran interface.

Now programmers can create a help message relating to any command by placing it in the help file, and by placing a

"Help" item in the submenu for the user to select to display the message. There are some rules for creating the message.

The resource file of X Window provides the flexibility for the user interface. The programmer may change the screen layout or default value for variables in the X control module without recompiling the program by modifying the resource file.

With the X Window interface, porting application from one machine to another can be easily performed. The interface written in X also provides the networking capability. If the application is running on one machine in the network, it can display the result or image on another machine in the same network. Therefore, a slow-speed and small-memory machine can still access the big application programs through the network.

The X Window interface control module provides a user interface independence that allows programmers to modify the computational module without affecting the user interface, and vice versa. In addition, the control module enables programmers to easily cooperate the user interface with their applications. Thus they may concentrate more of their time on developing applications.

For PLOT3D, the user interface is simple to learn. Users can use the program as a tool to help them picture the CFD grids and solutions. Figures 17 through 19 show some of the CFD grids and solutions using different functions produced by PLOT3D.

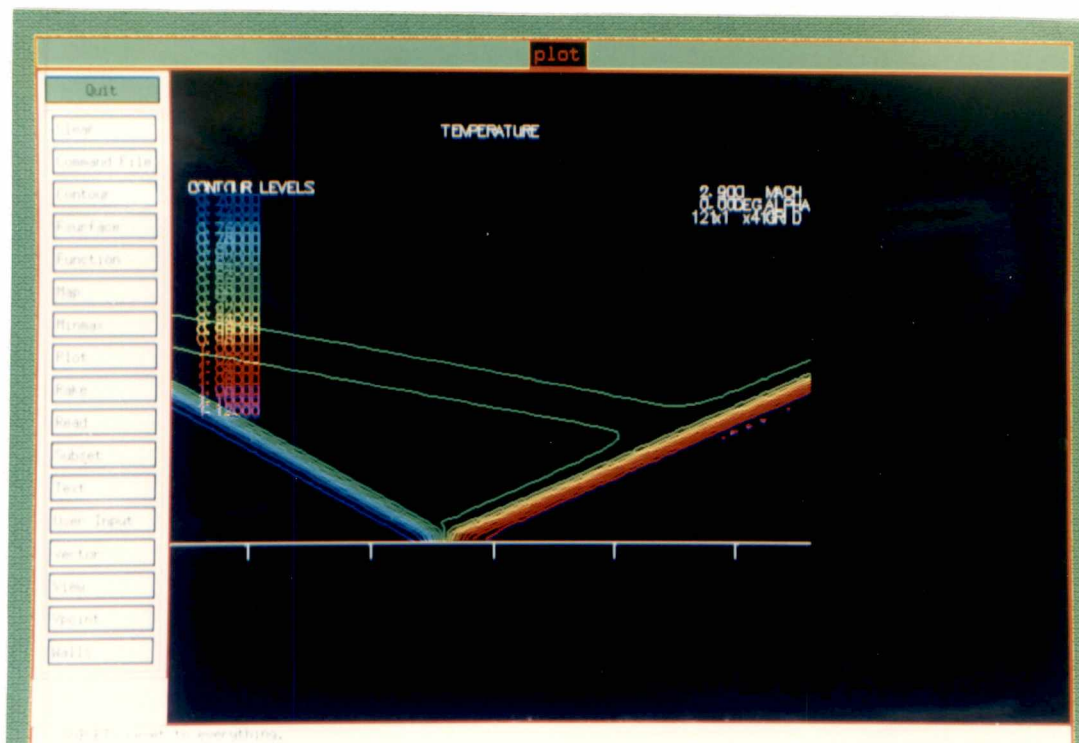


Figure 17. Temperature Function.

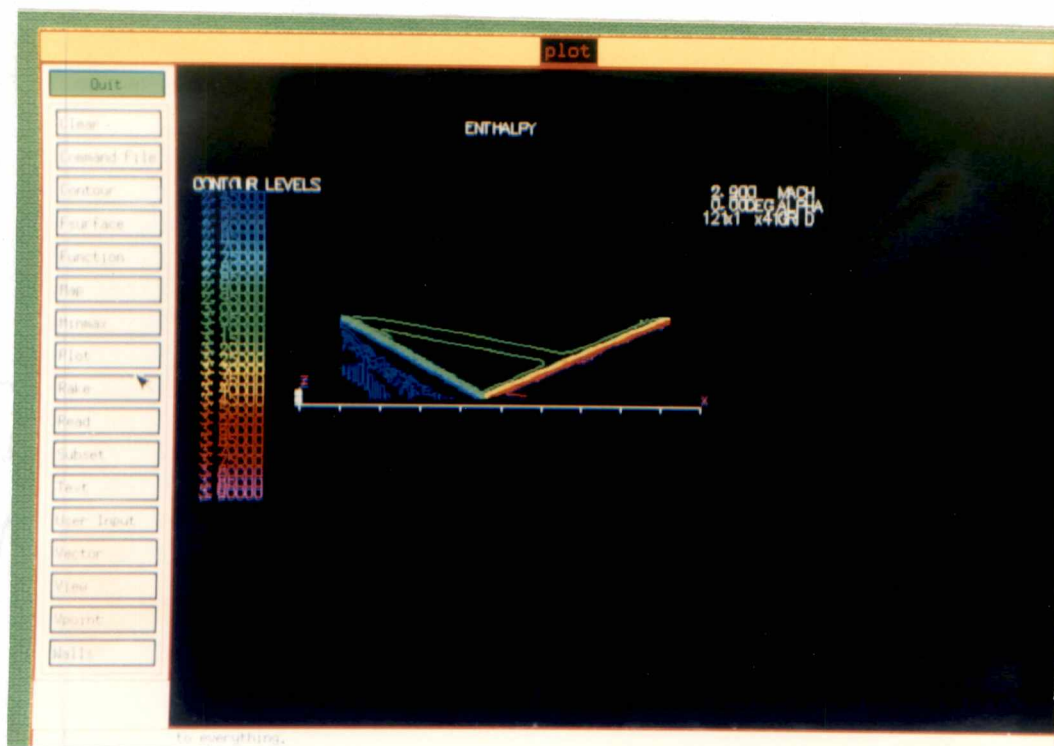


Figure 18. Enthalpy Function.

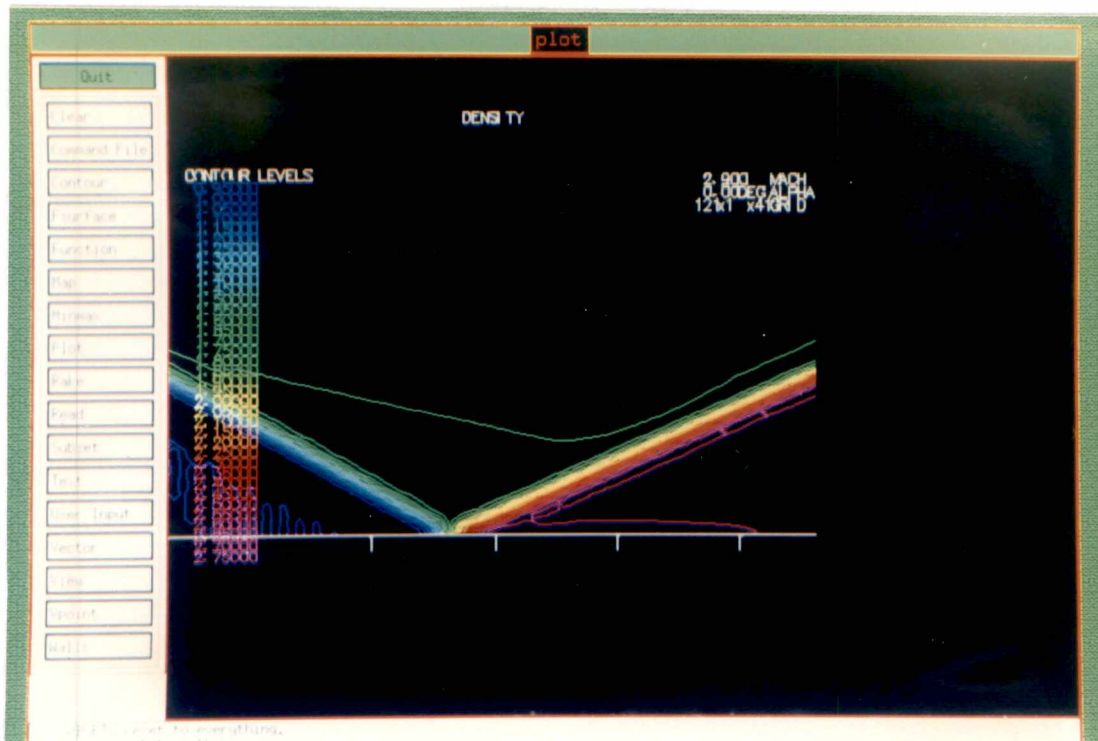


Figure 19. Density Function.

The menu setup of the X Window interface has an advantage over command-language interface in that no special vocabulary needs to be learned by the users; specifications depend on recognizing the command in the menu. Other advantages include each submenu contain a "help" item to aid the users for understanding what the command is about.

The menu selections are made by the cursor controlled by the mouse. The menu is set up in alphabetical order, and menu items are all in text form, static, always in the same place in the command window, and always visible to the user.

There are several window managers available in UTSI for the user interface; however, only one of them, the Generic Window Manager, will allow the application to modify the default color map. The others restrict the modification of the system color map.

5.2 Future Research Recommendations

The user interface developed through this research is an example of how to help the engineering programmer to produce a better user interface within short period of time. The interface has some room for improvement, however, and is being left to others for further development. A few suggestions are given below.

A developer could include the parser module in the interface such that engineering programmers do not have to

generate their own parser for their applications. This would significantly reduce the development time for an application. Although the commands of the parser would be limited, developers could provide a set of rules for programmers to follow for cooperating programmers' applications with the interface.

Vogle is a primitive three-dimensional graphics package built on top of X. Developers could replace the Vogle with an advanced 3D graphics package, such as PEX (PHIGS/PHIGS+ Extension to X), for building the graphics module. PEX is an extension of the X Window System that has been designed to provide the capabilities of PHIGS and other 3D graphics standards in the X environment^{20,21}. It would provide a better performance for the applications.

The interface uses the message window, which shows only eighteen text lines, to display messages for the end-user. If there are more than eighteen lines to display, the first line at the top of the window disappears and each line moves up one to make room for the new line at the bottom. If developers could make the message window into a scrolled window, users would be able to see the message off the window by scrolling the window backward.

The response for remote machines trying to access an application (PLOT3D) through the network is slow. If the application could be made to port to a faster machine, it may have a better performance.

As shown in Figures 17 through 19, PLOT3D shows the contour level on the right-hand side of the picture. Sometimes the contour level covers part of the picture, but if it could be put in the pop-up menu, users could press the mouse button inside the graphics window to display it.

LIST OF REFERENCES

LIST OF REFERENCES

1. Computer Graphics, Special GKS Issue. New York: Associate for Computer Machinery, February 1984.
2. Computer Graphics, Special Report on the Graphics Planning Committee. New York: Association for Computer Machinery, 1979.
3. Scheifler, Robert W. and Gettys, Jim. "The X Window System", ACM Transactions on Graphics, 5(No. 2): 78-109, April 1986.
4. Nye, Adrian. Volume One Xlib Programming Manual for Version 11 of the X Window System. Sebastopol: O'Reilly & Associates, Inc., 1988.
5. Pountain, Dick. "The X Window System", Byte, Pp. 353-360, January 1989.
6. Gettys, Jim. "Networking Windowing Using X Window System", Dr. Dobb's Journal, Pp. 42-53, March 1989.
7. Young, Douglas A. X Window Systems Programmer and Applications with Xt, New Jersey: Prentice-Hall, Inc., 1989.
8. McGregor, Scott. "Designing User Interface Tools for the X Window System", Digest of Papers. COMPCON Spring '89. Thirty-four IEEE Computer Society International Conference: Intellectual Leverage, San Francisco, CA, USA, 27 Feb.-3 March 1989, Pp.243-6.
9. Jones, Oliver. Introduction to the X Window System, New Jersey: Prentice-Hall, Inc., 1989.
10. McCormack, Joel, Paul Asente, Ralph R. Swick. X Toolkit Intrinsics - C Language Interface, Cambridge: MIT, 1988.
11. Young, Douglas A. "Programming with the X Toolkit", UnixWorld, Pp. 123-135, January 1990.
12. Young, Douglas A. "Handling Input with the X Toolkit", UnixWorld, Pp. 103-115, February 1990.
13. Programming with the HP X Widgets, Cambridge: MIT, November 1988.
14. Hartson, H. Rex and Hix, Debarah. "Human-Computer Interface Development Concepts and System for its Management", ACM Computing Surveys, 21(No. 1): 5-92, March 1989.

15. Stelovsky, Jan and Conti, Paolo. "Towards Portability of Dialog Systems", Pp. 83-8, edited by G. Salvendy, Cognitive Engineering in the Design of Human-Computer Interaction and Expert System, Amsterdam: Elsevier Science Publishers B.V., 1987.
16. Kantorowitz, Eliezer and Sudarsky, Oded. "The Adaptable User Interface", Communication of the ACM, 32(No. 11): 1352-8, November 1987.
17. Steitz Norbert A. "Cognitive Compatibility as a Central Issue in Human-Computer Interaction: Theoretical Framework And Empirical Finding", Pp. 75-82, edited by G. Salvendy, Cognitive Engineering in the Design of Human-Computer Interaction and Expert System, Amsterdam: Elsevier Science Publishers B.V., 1987.
18. Sun FORTRAN Programmer's Guide, Sun Microsystems, Inc., 1988.
19. PLOT3D User's Manual, Moffett Field: NASA Ames Research Center, 1990.
20. Rost, Randi J. "Adding a Dimension to X" Unix Review, 6(No. 11): 51-9, October 1988.
21. Nishimoto, Peter, Randi Rost, and Jeffery Friedberg. "PEX: A Network-Transparent 3D Graphics System", IEEE Computer Graphics and Applications, Pp. 14-26, July 1989.

APPENDICES

APPENDIX A
X WINDOW CONTROL MODULE

```

/*****
*****
*****      Include File for Control Module (control.h)      *****
*****
*****/

/*
 * Include all the necessary X Window System header files .
 */

#include <stdio.h>
#include <X11/StringDefs.h>
#include <X11/cursorfont.h>
#include <X11/Intrinsic.h>
#include <X11/Xutil.h>
#include <X11/Shell.h>
#include <Xw/MenuBtn.h>
#include <Xw/Cascade.h>
#include <Xw/PopupMgr.h>
#include <Xw/Xw.h>
#include <Xw/WorkSpace.h>
#include <Xw/Form.h>
#include <Xw/RManager.h>
#include <Xw/PButton.h>
#include <Xw/SRaster.h>
#include <Xw/SText.h>
#include <Xw/BBoard.h>
#include <Xw/TextEdit.h>

/*
 * Global variable structure.
 */

typedef struct {
    Widget      msg;
    char        *commsg;
    char        *flag;
} internal_data;

/*
 * Input/Output variable structure.
 */

typedef struct {
    Widget      incoming_msg[12];
    char        dpyststring[12][80];
    char        comstring[80];
} input_struct;

/*
 * Define all the variables for the main program.
 */

Arg            wargs[10];
internal_data  data;
input_struct   input;
int            first;
FILE           *fopen(), *fclose(), *fp;
Display        *msg_display;
Window         msg_winder;
Drawable       msg_drawable;
GC             msg_gc;

```

```

XGCValues      gcv;
#define         col 81
#define         row 16
static char    list[row][col];

/*
 * Define the type of all the external functions.
 */

extern Widget   create_quit_button();
extern Widget   create_one_line_text_widget();
extern Widget   create_menu_manager();
extern void     create_p();
extern void     ptloop_();
extern void     plot3d_();

/*
 * Menu variable structure.
 */

typedef struct {
    char        *name;
    void        (*func)();
    char        *com;
} menu_struct;

/*
 * declare the type of the function.
 */

void dummy();
void vpoint();
void minmax();
void command_file();
void read_option();
void wrlin_();
void done();
void get_text();
void user_input();
void loop_handler();
void help_cmd();
void pass_cmd();
void read_cmd();
void title_cmd();
void len_scale();
void scale_func();
void scale_fact();
void wall_origin();
void contour();
void noattrib();
void manual();
void con_filnam();
void wr_confile();
void write_to_file();
void comfile();
void contour_msg();
void wall_msg();
void walls_sub();
void wall_sub_grid();
void rake();
void rake_read();
void rake_write();

```

```

void xyz_msg();
void ijk_msg();
void attribute_msg();

/*
 * Global data structure to store variable retrieve from the
 * resource file.
 */

typedef struct {
    String    file;
    String    tempfile;
} AppData, *AppDataPtr;
AppData defvalue;

/*
 * Specify how the resource manager should retrieve these
 * resources from the database.
 */

static XtResource resources[] = {
    { "file", "File", XtRString, sizeof(String),
      XtOffset(AppDataPtr, file), XtRString, "plot3d.hlb"},
    { "tempfile", "TempFile", XtRString, sizeof(String),
      XtOffset(AppDataPtr, tempfile), XtRString, "plot3dtemp.com"},
};

/*
 * The submenu definition of Plot command.
 */

static menu_struct MenuPlot[] = {
    {"Plot3d", pass_cmd, "plot/3d"},
    {"Plot2d", pass_cmd, "plot/2d"},
    {"Help",    help_cmd, "plot"},
};

/*
 * The submenu definition of Clear command.
 */

static menu_struct MenuClear[] = {
    {"Internal Array", pass_cmd, "clear"},
    {"Help", help_cmd, "clear"},
};

/*
 * The submenu definition of View Command.
 */

static menu_struct MenuView[] = {
    {"Show", pass_cmd, "show view"},
    {"XY",    pass_cmd, "view xy"},
    {"XZ",    pass_cmd, "view xz"},
    {"YZ",    pass_cmd, "view yz"},
    {"YX",    pass_cmd, "view yx"},
    {"ZX",    pass_cmd, "view zx"},
    {"ZY",    pass_cmd, "view zy"},
    {"Help",  help_cmd, "view"},
};

```

```

* The submenu definition of Minmax Command.
*/

static menu_struct MenuMinmax[] = {
    {"Plot Range", minmax, "minmax "},
    {"Show",      pass_cmd, "show minmax"},
    {"Help",      help_cmd, "minmax"},
};

/*
* The submenu definition of Map command.
*/

static menu_struct MenuMap[] = {
    {"Usage of WORK", pass_cmd, "map"},
    {"Help",          help_cmd, "map"},
};

/*
* The submenu definition of Vpoint command.
*/

static menu_struct MenuVpoint[] = {
    {"Viewpoint", vpoint, "vpoint "},
    {"Show",      pass_cmd, "show vpoint"},
    {"Help",      help_cmd, "vpoint"},
};

/*
* The submenu definition of Function command.
*/

static menu_struct MenuFunc[] = {
    {"Grid",      dummy, " "},
    {"Scalar",    dummy, " "},
    {"Vector",    dummy, " "},
    {"Particle",  dummy, " "},
    {"Show",      pass_cmd, "show func"},
    {"Help",      help_cmd, "func"},
};

/*
* Sub-submenu definition of Function command associated with
* sub-command Grid.
*/

static menu_struct MenuGrid[] = {
    {"Walls Alone", pass_cmd, "fun 0"},
    {"Grid",        pass_cmd, "fun 1"},
    {"Outline IBLANK", pass_cmd, "fun 2"},
};

/*
* Sub-submenu definition of Function command associated with
* sub-command Scalar.
*/

static menu_struct MenuScalar[] = {
    {"Density",      pass_cmd, "fun 100"},
    {"Pressure",     pass_cmd, "fun 110"},
    {"Temperature",  pass_cmd, "fun 120"},
    {"Enthalpy",     pass_cmd, "fun 130"},
};

```

```

        {"Energy",      pass_cmd, "fun 140"},
        {"U Velocity",  pass_cmd, "fun 150"},
        {"X-Momentum",  pass_cmd, "fun 160"},
        {"Entropy",     pass_cmd, "fun 170"},
        {"X Vorticity",  pass_cmd, "fun 180"},
        {"Shock",       pass_cmd, "fun 190"},
};

/*
 * Sub-submenu definition of Function command associated with
 * sub-command Vector.
 */

static menu_struct MenuVector[] = {
    {"Velocity",      pass_cmd, "fun 200"},
    {"Vorticity",     pass_cmd, "fun 201"},
    {"Momentum",      pass_cmd, "fun 202"},
    {"Perturb Vel",   pass_cmd, "fun 203"},
    {"Pressure",      pass_cmd, "fun 204"},
};

/*
 * Sub-submenu definition of Function command associated with
 * sub-command Particle.
 */

static menu_struct MenuParticle[] = {
    {"Part Traces",  pass_cmd, "fun 300"},
    {"Vortex Line",  pass_cmd, "fun 301"},
};

/*
 * The submenu definition of Read command.
 */

static menu_struct MenuRead[] = {
    {"Read Data File", read_cmd, "read/mg/form"},
    {"Help",          help_cmd, "read"},
};

/*
 * The submenu definition of Comfile command.
 */

static menu_struct MenuComfile[] = {
    {"command File", command_file, "@"},
    {"Help",        help_cmd, " @"},
};

/*
 * The submenu definition of Text command.
 */

static menu_struct MenuText[] = {
    {"Title",  title_cmd, "text "},
    {"Help",   help_cmd, "text"},
};

/*
 * The submenu definition of Vector command.
 */

```

```

static menu_struct MenuVec[] = {
    {"Arrow", pass_cmd, "vector/arrow"},
    {"Noarrow", pass_cmd, "vector/noarrow"},
    {"Length Scale", len_scale, "vector/length="},
    {"Scalar Func", scale_func, "vector/scalar="},
    {"Noscal Func", pass_cmd, "vector/noscalar"},
    {"Show", pass_cmd, "show vector"},
    {"Help", help_cmd, "vector"},
};

/*
 * The submenu definition of Fsurface command.
 */

static menu_struct MenuFsurface[] = {
    {"Contour", pass_cmd, "fsurface/contour"},
    {"Nocontour", pass_cmd, "fsurface/nocont"},
    {"Scale", scale_fact, "fsurface/scale="},
    {"Auto Scale", pass_cmd, "fsurface/scale=auto"},
    {"Walls", wall_origin, "fsurface/wall="},
    {"Auto Walls", pass_cmd, "fsurface/walls=auto"},
    {"Show", pass_cmd, "show fsurface"},
    {"Help", help_cmd, "fsurface"},
};

/*
 * The submenu definition of Contour command.
 */

static menu_struct MenuContour[] = {
    {"Attributes", contour, "cont/attrib "},
    {"Noattrib", noattrib, "cont/noatt "},
    {"Automatic", contour, "cont/auto "},
    {"Increment", contour, "cont/increment "},
    {"Manual", manual, "cont/manual "},
    {"Range", contour, "cont/range "},
    {"Show", pass_cmd, "show contour"},
    {"Help", help_cmd, "contour"},
};

/*
 * The submenu definition of Wall command.
 */

static menu_struct MenuWall[] = {
    {"Grid Number", dummy, " "},
    {"Add Wall", walls_sub, "wall/add/att"},
    {"Wall", walls_sub, "wall/att"},
    {"Show", pass_cmd, "show wall"},
    {"Help", help_cmd, "wall"},
};

/*
 * Sub-submenu definition of Wall command associated with
 * sub-command Grid Number.
 */

static menu_struct SubMenuWall[] = {
    {"Add Wall", wall_sub_grid, "wall/grid/add/att"},
    {"Grid", wall_sub_grid, "wall/grid/att"},
};

```

```

/*
 * The submenu definition of Subset command.
 */

static menu_struct MenuSubset[] = {
    {"Grid Number", dummy, " "},
    {"Add Subset", walls_sub, "subs/add/att"},
    {"Subset", walls_sub, "subs/att"},
    {"Show", pass_cmd, "show subs"},
    {"Help", help_cmd, "subs"},
};

/*
 * Sub-submenu definition of Subset command associated with
 * sub-command Grid Number.
 */

static menu_struct SubMenuSubset[] = {
    {"Add Subset", wall_sub_grid, "subs/grid/add/att"},
    {"Grid", wall_sub_grid, "subs/grid/att"},
};

/*
 * The submenu definition of Rake command.
 */

static menu_struct MenuRake[] = {
    {"Read File", dummy, " "},
    {"Write File", dummy, " "},
    {"Add Rake", dummy, " "},
    {"Rake IJK", dummy, " "},
    {"Rake XYZ", dummy, " "},
    {"Show", pass_cmd, "show rake"},
    {"Help", help_cmd, "rake"},
};

/*
 * Sub-submenu definition of Rake command associated with
 * sub-command Read File.
 */

static menu_struct SubMenuRakeRead[] = {
    {"Read IJK", dummy, " "},
    {"Read XYZ", dummy, " "},
};

/*
 * Sub-sub-submenu definition of Rake command associated with
 * sub-command Read File and sub-sub-command Read IJK.
 */

static menu_struct SSMenURReadIJK[] = {
    {"-Time", rake_read, "rake/read/ijk/-time"},
    {"+Time", rake_read, "rake/read/ijk/+time"},
    {"+-Time", rake_read, "rake/read/ijk/+-time"},
    {"Add -Time", rake_read, "rake/read/add/ijk/-time"},
    {"Add +Time", rake_read, "rake/read/add/ijk/+time"},
    {"Add +-Time", rake_read, "rake/read/add/ijk/+-time"},
};

/*
 * Sub-sub-submenu definition of Rake command associated with

```

```

* sub-command Read File and sub-sub-command Read XYZ.
*/

static menu_struct SSMenuRReadXYZ[] = {
    {"-Time",      rake_read, "rake/read/xyz/-time"},
    {"+Time",      rake_read, "rake/read/xyz/+time"},
    {"+-Time",     rake_read, "rake/read/xyz/+-time"},
    {"Add -Time",  rake_read, "rake/read/add/xyz/-time"},
    {"Add +Time",  rake_read, "rake/read/add/xyz/+time"},
    {"Add +-Time", rake_read, "rake/read/add/xyz/+-time"},
};

/*
* Sub-submenu definition of Rake command associated with
* sub-command Add Rake.
*/

static menu_struct SubMenuRakeAdd[] = {
    {"IJK -Time",  rake, "rake/ijk/add/-time"},
    {"IJK +Time",  rake, "rake/ijk/add/+time"},
    {"IJK +-Time", rake, "rake/ijk/add/+-time"},
    {"XYZ -Time",  rake, "rake/xyz/add/-time"},
    {"XYZ +Time",  rake, "rake/xyz/add/+time"},
    {"XYZ +-Time", rake, "rake/xyz/add/+-time"},
};

/*
* Sub-submenu definition of Rake command associated with
* sub-command Rake IJK.
*/

static menu_struct SubMenuRakeIJK[] = {
    {"-Time", rake, "rake/ijk/-time"},
    {"+Time",  rake, "rake/ijk/+time"},
    {"+-Time",  rake, "rake/ijk/+-time"},
};

/*
* Sub-submenu definition of Rake command associated with
* sub-command Rake XYZ.
*/

static menu_struct SubMenuRakeXYZ[] = {
    {"-Time", rake, "rake/xyz/-time"},
    {"+Time",  rake, "rake/xyz/+time"},
    {"+-Time",  rake, "rake/xyz/+-time"},
};

/*
* Sub-submenu definition of Rake command associated with
* sub-command Write File.
*/

static menu_struct SubMenuRakeWrite[] = {
    {"Write IJK", dummy, " "},
    {"Write XYZ", dummy, " "},
};

/*
* Sub-sub-submenu definition of Rake command associated with
* sub-command Write File and sub-sub-command Write IJK.
*/

```

```

static menu_struct SSMenuRWriteIJK[] = {
    {"-Time", rake_write, "rake/write/ijk/-time"},
    {"+Time", rake_write, "rake/write/ijk/+time"},
    {"+-Time", rake_write, "rake/write/ijk/+-time"},
    {"Add -Time", rake_write, "rake/write/add/ijk/-time"},
    {"Add +Time", rake_write, "rake/write/add/ijk/+time"},
    {"Add +-Time", rake_write, "rake/write/add/ijk/+-time"},
};

/*
 * Sub-sub-submenu definition of Rake command associated with
 * sub-command Write File and sub-sub-command Write XYZ.
 */

static menu_struct SSMenuRWriteXYZ[] = {
    {"-Time", rake_write, "rake/write/xyz/-time"},
    {"+Time", rake_write, "rake/write/xyz/+time"},
    {"+-Time", rake_write, "rake/write/xyz/+-time"},
    {"Add -Time", rake_write, "rake/write/add/xyz/-time"},
    {"Add +Time", rake_write, "rake/write/add/xyz/+time"},
    {"Add +-Time", rake_write, "rake/write/add/xyz/+-time"},
};

/*
 * A dummy function.
 */

void dummy(w, cl_data, call_data)
    Widget w;
    caddr_t cl_data, call_data;
{
    return;
}

/*****
*****
*****          X Window Control Module          *****
*****
*****/

#include "control.h"

main(argc, argv)
    int argc;
    char *argv[];
{
    Widget toplevel, canvas, framework, command;
    Display *dpy;

    /*
     * Initialize the top level widget of the widget tree, and get the
     * default value from the resource file.
     */

    toplevel = XtInitialize(argv[0], "Plot3d", NULL, 0, &argc, argv);
    XtGetApplicationResources(toplevel, &defvalue, resources,
                              XtNumber(resources), NULL, 0);

```

```

/*
 * Create a form widget to manage the message, command, and canvas
 * windows.
 */

framework = XtCreateManagedWidget ("framework", XwformWidgetClass,
                                     toplevel, NULL, 0);

/*
 * Create a message window with certain height and width.
 */

XtSetArg (wargs[0], XtNwidth, 800);
XtSetArg (wargs[1], XtNheight, 170);
data.msg = XtCreateManagedWidget ("message", XwbulletinWidgetClass,
                                   framework, wargs, 2);

/*
 * Create the column to hold the commands in the menu.
 */

command = XtCreateManagedWidget ("command", XwrowColWidgetClass,
                                   framework, NULL, 0);

/*
 * Add a quit button in the menu.
 */

create_quit_button("Quit", command);

/*
 * Create the rest of the commands in the menu.
 */

create_commands_menu(command);

/*
 * Create the drawing surface and define a cursor type.
 */

canvas = XtCreateManagedWidget ("canvas", XworkSpaceWidgetClass,
                                 framework, NULL, 0);

dpy = XtDisplay(toplevel);
XtRealizeWidget(toplevel);
XDefineCursor(dpy, XtWindow(toplevel), XCreateFontCursor(dpy,
                                                           XC_sailboat));

/*
 * Initialize the graphics data and get into infinite loop.
 */

init_data(canvas);
XtMainLoop();
}

```

```

/*-----*
*          C R E A T E _ C O M M A N D S _ M E N U          *
*          *          *          *          *          *          *
*      Effect:   Create command in the user menu.          *
*-----*/

create_commands_menu(parent)
    Widget parent;
{
Widget w, ws, commands;

/*
* Group all commands in a column.
*/

commands = XtCreateManagedWidget("commands", XwrowColWidgetClass,
                                parent, NULL, 0);

/*
* Create a button for each Plot3d command.
*/

/* CLEAR COMMAND */

w = XtCreateManagedWidget("Clear", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuClear, XtNumber(MenuClear));

/* COMMAND FILE COMMAND */

w = XtCreateManagedWidget("Command File", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuComfile, XtNumber(MenuComfile));

/* CONTOUR COMMAND */

w = XtCreateManagedWidget("Contour", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuContour, XtNumber(MenuContour));

/* FSURFACE COMMAND */

w = XtCreateManagedWidget("Fsurface", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuFsurface, XtNumber(MenuFsurface));

/* FUNCTION COMMAND */

w = XtCreateManagedWidget("Function", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuFunc, XtNumber(MenuFunc));
create_p(ws, "Grid", "Sub Menu", MenuGrid, XtNumber(MenuGrid));
create_p(ws, "Scalar", "Sub Menu", MenuScalar, XtNumber(MenuScalar));
create_p(ws, "Vector", "Sub Menu", MenuVector, XtNumber(MenuVector));
create_p(ws, "Particle", "Sub Menu", MenuParticle,
        XtNumber(MenuParticle));

```

```

/* MAP COMMAND */

w = XtCreateManagedWidget("Map", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuMap, XtNumber(MenuMap));

/* MINMAX COMMAND */

w = XtCreateManagedWidget("Minmax", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuMinmax, XtNumber(MenuMinmax));

/* PLOT COMMAND */

w = XtCreateManagedWidget("Plot", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuPlot, XtNumber(MenuPlot));

/* RAKE COMMAND */

w = XtCreateManagedWidget("Rake", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuRake, XtNumber(MenuRake));
create_p(ws, "Read File", "Sub Menu", SubMenuRakeRead,
        XtNumber(SubMenuRakeRead));
create_p(ws, "Write File", "Sub Menu", SubMenuRakeWrite,
        XtNumber(SubMenuRakeWrite));
create_p(ws, "Add Rake", "Sub Menu", SubMenuRakeAdd,
        XtNumber(SubMenuRakeAdd));
create_p(ws, "Rake IJK", "Sub Menu", SubMenuRakeIJK,
        XtNumber(SubMenuRakeIJK));
create_p(ws, "Rake XYZ", "Sub Menu", SubMenuRakeXYZ,
        XtNumber(SubMenuRakeXYZ));
create_p(ws, "Read IJK", "Read Submenu", SSMenuRReadIJK,
        XtNumber(SSMenuRReadIJK));
create_p(ws, "Read XYZ", "Read Submenu", SSMenuRReadXYZ,
        XtNumber(SSMenuRReadXYZ));
create_p(ws, "Write IJK", "Write Submenu", SSMenuRWriteIJK,
        XtNumber(SSMenuRWriteIJK));
create_p(ws, "Write XYZ", "Write Submenu", SSMenuRWriteXYZ,
        XtNumber(SSMenuRWriteXYZ));

/* READ COMMAND */

w = XtCreateManagedWidget("Read", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuRead, XtNumber(MenuRead));

/* SUBSET COMMAND */

w = XtCreateManagedWidget("Subset", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuSubset, XtNumber(MenuSubset));
create_p(ws, "Grid Number", "Sub Menu",
        SubMenuSubset, XtNumber(SubMenuSubset));

```

```

/* TEXT COMMAND */

w = XtCreateManagedWidget("Text", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuText, XtNumber(MenuText));

/* USER INPUT COMMAND */

w = XtCreateManagedWidget("User Input", XwpushButtonWidgetClass,
                           commands, NULL, 0);
XtAddCallback(w, XtNselect, user_input, NULL);

/* VECTOR COMMAND */

w = XtCreateManagedWidget("Vector", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuVec, XtNumber(MenuVec));

/* VIEW COMMAND */

w = XtCreateManagedWidget("View", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuView, XtNumber(MenuView));

/* VPOINT COMMAND */

w = XtCreateManagedWidget("Vpoint", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuVpoint, XtNumber(MenuVpoint));

/* WALL COMMAND */

w = XtCreateManagedWidget("Walls", XwpushButtonWidgetClass,
                           commands, NULL, 0);
ws = create_menu_manager(w, "menu_mgr");
create_p(ws, "menu_mgr", "MAIN", MenuWall, XtNumber(MenuWall));
create_p(ws, "Grid Number", "Sub Menu", SubMenuWall,
        XtNumber(SubMenuWall));
}

/*-----*
*              I N I T _ D A T A              *
*-----*
* Effect:  Initialize all the global variables and *
*          starting up the vogle graphics package. *
*-----*/

init_data(w)
    Widget w;
{
    int i;
    vinit(w);
    plot3d();
    msg_display = XtDisplay(data.msg);
    msg_winder = XtWindow(data.msg);
    msg_drawable = (Drawable)msg_winder;
    gcv.foreground = 4;
}

```

```

        msg_gc = XCreateGC(msg_display, msg_drawable, GCForeground, &gcval);
    }

/*-----*
 *          L O O P _ H A N D L E R          *
 *-----*
 * Effect:   Passing the constructed command to PLOT3D *
 *           package to perform the appropriate action. *
 *-----*/

void loop_handler ()
{
    ptloop_(data.flag, data.commsg, strlen(data.flag),
            strlen(data.commsg));
}

/*-----*
 *          H E L P _ C M D          *
 *-----*
 * Effect:   Passing the HELP command to the help *
 *           function. *
 *-----*/

void help_cmd(w, keywrld, call_data)
    Widget w;
    char *keywrld;
    caddr_t call_data;
{
    int nkey = 1;
    help_(defvalue.file, &nkey, keywrld, strlen(defvalue.file),
          sizeof(nkey), strlen(keywrld));
}

/*-----*
 *          P A S S _ C M D          *
 *-----*
 * Effect:   Construct the passing command to PLOT3D *
 *           and the flag (if the flag is "no", the *
 *           parser will not repeat in the parser loop. *
 *           Otherwise, it will repeat in the parser *
 *           loop until it complete the command. *
 *-----*

void pass_cmd (w, comword, call_data)
    Widget w;
    char *comword;
    caddr_t call_data;
{
    data.commsg = comword;
    data.flag = "no";
    loop_handler();
}

/*-----*
 *          M I N M A X          *
 *-----*
 * Effect:   Construct the minmax command to pass to *
 *           parser module. *
 *-----*/

/*

```

```

* Setup the instruction message to ask for the user to enter,
* and copy the command string to a global variable.
*/

void minmax(w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;

{
    static char dpy[]="Plot limits Xmin,Xmax, Ymin,Ymax, [,Zmin,Zmax]";
    strcpy(input.dpystring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          S C A L E _ F A C T          *
*          *                             *
* Effect:   Construct the fsurface command with scale *
*           qualifier to pass to parser module.      *
*-----*/

void scale_fact(w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;

{
    static char dpy[]="Scale factor for function axis: ";
    strcpy(input.dpystring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          W A L L _ O R I G I N          *
*          *                             *
* Effect:   Construct the wall command to pass to *
*           parser module.                      *
*-----*/

void wall_origin(w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;

{
    static char dpy[]="Body and Walls origin offset";
    strcpy(input.dpystring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          V P O I N T                    *
*          *                             *
* Effect:   Construct the vpoint command to pass to *
*           parser module.                      *
*-----*/

void vpoint(w, qual, call_data)

```

```

Widget w;
char *qual;
caddr_t call_data;

{
    static char dpy[]="X,Y,Z viewpoint for 3D plots";
    strcpy(input.dpyststring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          L E N _ S C A L E          *
*                                     *
* Effect:   Construct the vector command with length *
*           qualifier to pass to parser module.      *
*-----*/

void len_scale(w, qual, call_data)
Widget w;
char *qual;
caddr_t call_data;

{
    static char dpy[]="Scale Factor for Vector Lenghts";
    strcpy(input.dpyststring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          S C A L E _ F U N C          *
*                                     *
* Effect:   Construct the vector command with scalar *
*           qualifier to pass to parser module.      *
*-----*/

void scale_func(w, qual, call_data)
Widget w;
char *qual;
caddr_t call_data;

{
    static char dpy[]="Scalar Function number";
    strcpy(input.dpyststring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*          C O M M A N D _ F I L E          *
*                                     *
* Effect:   Construct the COMMAND FILE command to pass *
*           to parser module.                        *
*-----*/

void command_file(w, qual, call_data)
Widget w;
char *qual;
caddr_t call_data;

{

```

```

        static char dpy[]="Command File";
        strcpy(input.dpyststring[0] , dpy);
        strcpy(input.comstring , qual);
        read_option(1, 0);
    }

/*-----*
*                U S E R _ I N P U T                *
*                *                                     *
* Effect:   Construct the ANY command for PLOT3D to   *
*           pass to parser module.                   *
*-----*/

void user_input(w, cl_data, call_data)
    Widget w;
    caddr_t cl_data, call_data;

{
    static char dpy[]="User Input";
    static char com[] = "";
    strcpy(input.dpyststring[0] , dpy);
    strcpy(input.comstring , com);
    read_option(1, 0);
}

/*-----*
*                R E A D _ C M D                    *
*                *                                     *
* Effect:   Construct the READ command to pass to   *
*           parser module.                           *
*-----*/

void read_cmd(w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;

{
    static char dpy[]="XYZ File";
    static char dpy1[]="Q File";
    strcpy(input.dpyststring[0] , dpy);
    strcpy(input.dpyststring[1] , dpy1);
    strcpy(input.comstring , qual);
    read_option(2, 0);
}

/*-----*
*                T I T L E _ C M D                  *
*                *                                     *
* Effect:   Construct the TEXT command to pass to   *
*           parser module.                           *
*-----*/

void title_cmd(w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;

{
    static char dpy[]="Title 1";
    static char dpy1[]="Title 2";
    strcpy(input.dpyststring[0] , dpy);

```

```

        strcpy(input.dpystring[1] , dpy1);
        strcpy(input.comstring , qual);
        read_option(2, 0);
    }

/*-----*
 *                D O N E                *
 *                *                        *
 * Effect:   Destroy the input window.    *
 *-----*/

void done (w, shell, call_data)
    Widget w, shell;
    caddr_t call_data;
{
    XtUnrealizeWidget(shell);
}

/*-----*
 *                G E T _ T E X T        *
 *                *                        *
 * Effect:   Read the data in the input window into the *
 *           system.                                *
 *-----*/

void get_text(w, linenum, call_data)
    Widget w;
    int linenum;
    caddr_t call_data;
{
    char *p, g[80];
    int i;

/*
 * Initialize all the character string to detect the command
 * for PLOT3D.
 */

static char xfile[] = "/x=";
static char qfile[] = "/q=";
static char space[] = " ";
static char quote[] = {'\42', '\0'};

/*
 * Copy the command string and input string to the temporary
 * variables.
 */

    for (i = 0; i < strlen(input.comstring)+1; i++)
        g[i] = input.comstring[i];
    p = (char*)XwTextCopyBuffer(input.incoming_msg[0]);

/*
 * If the input string is not empty, find out what kind of command
 * string it is.
 */

    if (strlen(p))
    {
        switch (linenum)
        {

```

```

/*
 * Contour Command: check for the input string. If is a legal
 * value, append it to the command string.
 */

    case 1:
        if ((p[0] == '0') && (g[0] == 'c') && (g[1] == 'o'))
        {
            wrln_("Wrong Input for Contour Level", 1);
            return;
        }
        else
            strcat (g, p);
        break;
    case 2:

/*
 * Read Command: append the X and Q file name with the command string.
 */

        if (g[0] == 'r')    /* check for 'r' */
        {
            strcat (g, xfile);
            strcat (g, p);
            strcat (g, qfile);
            p = (char*)XwTextCopyBuffer(input.incoming_msg[1]);
            strcat (g, p);
        }

/*
 * Text Command: put the text string between two quote mark and append
 * with the command string.
 */

        else
        {
            strcat (g, quote);
            strcat (g, p);
            strcat (g, quote);
            strcat (g, space);
            strcat (g, quote);
            p = (char*)XwTextCopyBuffer(input.incoming_msg[1]);
            if (strlen(p))
                strcat (g, p);
            else
                strcat (g, space);
            strcat (g, quote);
        }
        break;

/*
 * Other commands: it will put all the input data in the command file.
 */

    case 8:
        if ((p[0] == '0') && (g[0] == 'c') && (g[1] == 'o'))
        {
            wrln_("Wrong Input for Contour Level", 1);
            return;
        }
        else
        {

```

```

        first = 1;
        write_to_file(linenum);
        fclose(fp);
        comfile();
        return;
    }
    break;
}

/*
 * Set the temporary variable value to the global variable and
 * then call loop_handler to pass the command to the parser
 * module.
 */

    data.commsg = g;
    if (data.commsg[0] == '@')
        data.flag = "yes";
    else
        data.flag = "no";
    loop_handler();
}
else
    return;
}

/*-----*
 *           W R L I N _           *
 *                                     *
 * Effect:   Write text to the message window. *
 *-----*/

void wrlin_ (str, length)
    char *str;
    int  length;
{
    int i;

/*
 * Move all the line one line up so the last one will be lost.
 */

    for (i = row-1; i > 0; i--)
        strcpy(list[i] , list[i-1]);

/*
 * Copy the given string to the first element of the array.
 */

    for (i = 0; i < col ; i++)
        if (i == col-1)
            list[0][i] = '\0';
        else
            list[0][i] = str[i];

/*
 * Clear the message window and write the whole array to the window.
 */

    XClearWindow(msg_display, msg_winder);
    for (i = row-1; i >= 0; i--)

```

```

        XDrawString(msg_display,msg_drawable,msg_gc,10,(row-i)*10,
                    list[i],strlen(list[i]));
    }

/*-----*
*           N O A T T R I B           *
*                                     *
*   Effect:   Construct the CONTOUR command with no      *
*             attribute qualifier to pass to parser      *
*             module.                                     *
*-----*/

void noattrib (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    static char dpy[]="Number of contour levels (Don't enter 0)";
    strcpy(input.dpystring[0] , dpy);
    strcpy(input.comstring , qual);
    read_option(1, 0);
}

/*-----*
*           C O N T O U R           *
*                                     *
*   Effect:   Construct the CONTOUR command to pass to   *
*             parser module.                             *
*-----*/

void contour (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    static char dpy[] = "Number of contour levels (Don't enter 0)";
    strcpy(input.dpystring[0], dpy);
    contour_msg(1);
    strcpy(input.comstring, qual);
    read_option(8, 0);
}

/*-----*
*           C O N T O U R _ M S G           *
*                                     *
*   Effect:   Copy the instructing message to the array  *
*             to aid the user to enter data.             *
*-----*/

void contour_msg (start)
    int start;
{
    static char dpy[7][50] =
        {"LINES, SURFACES, GRID, COLOR_CON, or DOTS",
         "Color sequence for color map",
         "Line type",
         "Line thickness factor",
         "Symbol type",
         "Symbol size factor",
         "Plane Orientation"};
    int i;

```

```

        for (i = start; i < (start + 7); i++)
            strcpy(input.dpystring[i], dpy[i-start]);
    }

/*-----*
*               M A N U A L               *
*-----*
* Effect:   Construct CONTOUR command with manual *
*           qualifier to pass to parser module.   *
*-----*/

void manual (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    static char dpy[50] = "Contour Start [End Inc]";

    strcpy(input.dpystring[0], dpy);
    contour_msg(1);
    strcpy(input.comstring, qual);
    first = 1;
    read_option(8, 1);
}

/*-----*
*           R E A D _ O P T I O N           *
*-----*
* Effect:   Create an input window for user to enter *
*           data to the appropriate input box having a *
*           instructing message beside it.           *
*-----*/

void read_option(linenum, rflag)
    int linenum;
{
    Widget shell2, quit, form, cont;
    int i;
    static char *message[12] = { "message1", "message2", "message3",
                                "message4", "message5", "message6",
                                "message7", "message8", "message9",
                                "message10", "message11", "message12"};
    static char *field[12] = { "field1", "field2", "field3",
                                "field4",
                                "field5", "field6", "field7", "field8",
                                "field9", "field10", "field11", "field12"};

/*
* Create another toplevel widget to handle the input window using
* the column manager to manage the components inside the window.
*/

    XtSetArg(wargs[0], XtNallowShellResize, (XtArgVal)TRUE);
    XtSetArg(wargs[1], XtNinput, (XtArgVal)TRUE);
    shell2 = XtCreateApplicationShell ("shell2",
                                      topLevelShellWidgetClass, wargs, 2);
    form = XtCreateManagedWidget ("form", XwrowColWidgetClass, shell2,
                                   NULL, 0);

/*
* Create the input message corresponding to the input box.
*/

```

```

    for (i = 0; i < linenum ; i ++){
        XtSetArg(wargs[0], XtNstring, input.dpystring[i]);
        XtCreateManagedWidget (message[i],XwstatictextWidgetClass,
                                form, wargs, 1);
        XtSetArg(wargs[0], XtNeditType, XwtextEdit);
        input.incoming_msg[i]= create_one_line_text_widget
                                (field[i], form, wargs, 1);
    }

/*
 * The "DONE" button is used to signal the system the users finished
 * enter data if they select it.
 */

    quit = XtCreateManagedWidget("DONE", XwpushButtonWidgetClass,
                                form, NULL, 0);

/*
 * If the rflag is 1, it means the system will keep asking the users
 * to enter data as long as they selecting the "NEXT INPUT" button.
 * Otherwise It will only ask the user once.
 */

    if (rflag)
    {
        cont = XtCreateManagedWidget("NEXT INPUT",
                                XwpushButtonWidgetClass, form, NULL, 0);
        XtAddCallback(quit, XtNselect, con_filnam, linenum);
        XtAddCallback(cont, XtNselect, wr_confile, linenum);
        XtAddCallback(cont, XtNselect, done, shell2);
    }
    else
        XtAddCallback(quit, XtNselect, get_text, linenum);
    XtAddCallback(quit, XtNselect, done, shell2);
    XtRealizeWidget(shell2);
}

/*-----*
 *                W R _ C O N F I L E                *
 *                *                                     *
 * Effect:   Detect the command of the PLOT3D to decide *
 *           what action should perform.               *
 *-----*/

void wr_confile(w, linenum, call_data)
Widget w;
int linenum;
caddr_t call_data;
{
    char *p;
    int i, count;

/*
 * Copy the input data to the temporary variable. If there
 * is no input data, it will skip the whole function and return.
 */

    p = (char *)XwTextCopyBuffer(input.incoming_msg[0]);
    if (strlen(p))

```

```

/*
 * Check to see what type of PLOT3D command and perform different
 * type of action to construct the command file to pass to the parser
 * module.
 */

    switch(input.comstring[0])
    {

/*
 * Contour Command: Check the first three input data line. If they are
 * empty, it will not write the data to the command file. Otherwise,
 * it will write to the command file. Then it will ask the for the user
 * input again.

        case 'c':
            if (wrong_numarg())
            {
                read_option(linenum, 1);
                return;
            }
            else
            {
                write_to_file(linenum);
                read_option(linenum, 1);
            }
            break;

/*
 * Subset and Wall command: check for the first 4 input lines
 * to see it is empty or not. If it is not, write the input
 * data to the command file passing to parser module.
 */

        case 's':
        case 'w':
            if (linenum == 10)
            {
                if (write_rsw(4, linenum))
                    read_option((linenum - 1), 1);
                else
                    read_option(linenum, 1);
            }
            else
            {
                write_rsw(3, linenum);
                read_option(linenum, 1);
            }
            break;

/*
 * Rake Command: Check the input data. If it is valid, copy the data
 * to the command file.
 */

        case 'r':
            switch (linenum)
            {
                case 6:
                    write_rsw(1, linenum);
                    read_option(linenum, 1);
                    break;

```

```

        case 7:
        if (write_rsw(2, linenum))
            read_option((linenum-1), 1);
        else
            read_option(linenum, 1);
        break;
        case 11:
        write_rsw(4, linenum);
        read_option(linenum, 1);
        break;
        case 12:
        if (write_rsw(5, linenum))
            read_option((linenum-1), 1);
        else
            read_option(linenum, 1);
        break;
    }
    break;
}
else
{
    if (first)
    {
        wrmlin("No data from the first input field", 1);
        XBell(XtDisplay(data.msg), 100);
        return;
    }
    else
    {
        wrmlin("No data from the first input field", 1);
        wrmlin("Try again ...", 1);
        XBell(XtDisplay(data.msg), 100);
        read_option(linenum, 1);
    }
}

}

/*-----*
*               N O I N P U T               *
*-----*
* Effect:   It is a function use to check there is *
*           Input data or not.                   *
*-----*/

int noinput(ptr)
    int ptr;
{
    char *p;
    p = (char*) XwTextCopyBuffer(input.incoming_msg[ptr]);
    if (strlen(p))
        return(0);
    else
        return(1);
}

/*-----*
*               W R I T E _ R S W             *
*-----*/

```

```

*
* Effect:      Error checking for input data for Rake,
*              Subset, and Wall command. If there is no
*              error, it will write the input data to the
*              command file passing to the parser module.
*-----*

int write_rsw(checknum, linenum)
    int checknum, linenum;
{
    FILE *fp1;
    char *p;
    char temp[80];
    int i, j, k, finish;

/*
* Check the empty input data line for the variable won't accept
* it. If this is a case, it will display the error message to
* let the users realize they input the wrong data.
*/

    for (i = 0; i < checknum; i++)
        if (noinput(i))
        {
            strcpy(temp, "No input data in field number ");
            j = strlen(temp);
            temp[j] = (char)(i + 49);
            temp[j+1] = '\0';
            wrlin_(temp, 1);
            wrlin_("Try again ...", 1);
            XBell(XtDisplay(data.msg), 100);
            return(0);
        }

/*
* Check to see the file is existed or not. If it is not existed,
* It will display the error message and beep the user.
*/

    if (checknum == 2)
    {
        p = (char*) XwTextCopyBuffer(input.incoming_msg[0]);
        if ((fp1 = fopen(p, "r")) == NULL)
        {
            strcpy(temp, "No Data File ---> ");
            strcat(temp, p);
            wrlin_(temp, 1);
            wrlin_("Try Again ...", 1);
            XBell(XtDisplay(data.msg), 100);
            return(0);
        }
        else
            fclose(fp1);
    }

/*
* If it is the first time entering this function, it will open
* a command file for writing the input data. Otherwise, it will
* only write into the command file which is already opened.
*/

    if (first)

```

```

        {
            fp = fopen(defvalue.tempfile, "w");
            first = 0;
            fputs(input.comstring, fp);
            fputs("\n", fp);
        }
    if ((checknum == 4) || (checknum == 5))
    {
        p = (char*) XwTextCopyBuffer(input.incoming_msg[0]);
        fputs(p, fp);
        fputs("\n", fp);
        if (checknum == 5)
        {
            p = (char*) XwTextCopyBuffer(input.incoming_msg[1]);
            fputs(p, fp);
            fputs("\n", fp);
        }
    }

/*
 * Write the value separate by the colon in a new line for the first
 * three input line of command Rake and Wall.
 */

    if (!(checknum == 2) || (checknum == 1))
    {
        for (i = checknum - 3; i < checknum; i++)
        {
            p = (char*) XwTextCopyBuffer(input.incoming_msg[i]);
            k = 0;
            finish = 0;
            for (j = 0; finish == 0; j++)
            {
                if (p[j] == '\0') finish = 1;
                if ((p[j] != ':') && (p[j] != '\0'))
                {
                    temp[k] = p[j];
                    k++;
                }
                else
                {
                    if (k != 0)
                    {
                        temp[k] = '\0';
                        fputs(temp, fp);
                        fputs("\n", fp);
                        k = 0;
                    }
                }
            }
            fputs("\n", fp);
        }
    }
    else
        checknum = 0;

/*
 * Write the rest of the input line to command file starting from
 * checking line.
 */

    for (i = checknum; i < linenum; i++)
    {
        p = (char*) XwTextCopyBuffer(input.incoming_msg[i]);

```

```

        fputs(p, fp);
        fputs("\n", fp);
    }

/*
 * If it doesn't need the first input data line after the first
 * entry, the system will skip the first input data line for the
 * next entry.
 */

    if (((checknum == 4) && (input.comstring[0] != 'r')) ||
        (linenum == 7) || (checknum == 5))
        for (i = 0; i < linenum; i++)
            strcpy(input.dpyststring[i], input.dpyststring[i+1]);
    return(1);
}

/*-----*
 *          W R I T E _ T O _ F I L E          *
 *-----*
 * Effect:  If the First flag is 1, it will open a
 *           command file for writing input data.
 *           Otherwise, it just write to the command file.
 *-----*/

void write_to_file(linenum)
    int linenum;
{
    char *p;
    int i;

    if (first)
    {
        fp = fopen(defvalue.tempfile, "w");
        first = 0;
        fputs (input.comstring, fp);
        fputs ("\n", fp);
    }
    for (i = 0; i < linenum; i++)
    {
        p = (char*) XwTextCopyBuffer(input.incoming_msg[i]);
        fputs(p, fp);
        fputs("\n", fp);
    }
}

/*-----*
 *          W R O N G _ N U M A R G          *
 *-----*
 * Effect:  check to make sure there is only 1 or 3
 *           value in the input line.
 *-----*/

int wrong_numarg ()
{
    int i, j, count = 0;
    char *p = (char *)XwTextCopyBuffer(input.incoming_msg[0]);

    for (i = 0; p[i] != '\0'; i++)
        if (p[i] != ' ')
        {
            count++;
        }
}

```

```

        for (j = i+1; ((p[j] != 0) &&
            (p[j] != 32)); j++)
            i = j - 1;
    }
    if ((count != 3) && (count != 1))
    {
        wrlin_
        ("An increment is required if you give both a start and a end",
        1);
        XBell(XtDisplay(data.msg), 100);
        return(1);
    }
    return(0);
}

```

```

/*-----*
*          C O N _ F I L N A M          *
*          *                             *
* Effect:  Check the command type and perform appropriate *
*          action. It will do some error checking too.    *
*-----*

```

```

void con_filnam(w, linenum, call_data)
Widget w;
int linenum;
caddr_t call_data;
{
    char *p;
    int i, flag;

/*
 * Initialize the local variable.
 */

    flag = 0;
    p = (char*)XwTextCopyBuffer(input.incoming_msg[0]);

/*
 * Checking the to see the input line is empty or not.
 */

    if (strlen(p))
    {

/*
 * Check the command type.
 */

        switch(input.comstring[0])
        {

/*
 * Check for the contour command.
 */

            case 'c':
                if (wrong_numarg())
                    flag = 1;
                else
                    write_to_file(linenum);
                break;

```

```

/*
 * Check for Subset and Wall command.
 */

        case 's':
        case 'w':
        if (linenum == 10)
        {
            if (i=write_rsw(4,linenum) == 0)
                flag = 1;
        }
        else
            if (write_rsw(3, linenum) == 0)
                flag = 1;
        break;

/*
 * Check for rake command.
 */

        case 'r':
        switch (linenum)
        {
            case 6:
            if (!(write_rsw(1, linenum)))
                flag = 1;
            break;
            case 7:
            if (!(write_rsw(2, linenum)))
                flag = 1;
            break;
            case 11:
            if (!(write_rsw(4, linenum)))
                flag = 1;
            break;
            case 12:
            if (!(write_rsw(5, linenum)))
                flag = 1;
            break;
        }
    }

/*
 * If the flag is 1, it will ask the user for another input.
 * Otherwise, it closes the close the command file.
 */

        if (flag)
        {
            read_option(linenum, 1);
            return;
        }
        else
        {
            fputs("\n", fp);
            fclose(fp);
            comfile();
        }
    }
    else
    {

```

```

/*
 * If the flag is 1, it will ask the user for another input.
 * Otherwise, it closes the close the command file.
 */

        if (first)
            return;
        else
        {
            fputs("\n", fp);
            fclose(fp);
            comfile();
        }
    }
}

/*-----*
 *                C O M F I L E                *
 *-----*
 * Effect:  construct the command file command ( put the @ *
 *          sign in front of the command file name) and *
 *          sends it to parser module.                  *
 *-----*/

void comfile()
{
    static char atsign[] = "@";
    char temp[80];

    strcpy(temp, atsign);
    strcat(temp, defvalue.tempfile);
    data.commsg = temp;
    data.flag = "yes";
    loop_handler();
}

/*-----*
 *                W A L L S _ S U B                *
 *-----*
 * Effect:  Construct the wall and subset command to *
 *          pass to parser module.                  *
 *-----*/

void walls_sub (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    wall_msg(0);
    first = 1;
    strcpy(input.comstring, qual);
    read_option(9, 1);
}

/*-----*
 *                W A L L _ S U B _ G R I D                *
 *-----*
 * Effect:  Construct the wall and subset command with *
 *          grid qualifier pass to parser module.      *
 *-----*/

```

```

void wall_sub_grid (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    static char dpy[] = "Grid Number";
    first = 1;
    strcpy(input.dpystring[0], dpy);
    wall_msg(1);
    strcpy(input.comstring, qual);
    read_option(10, 1);
}

/*-----*
*           W A L L _ M S G           *
*                                     *
* Effect:   Part of the instruction message for Wall *
*           command.                     *
*-----*/

void wall_msg (start)
    int start;
{
    static char dpy[] = "LINES, HIDDEN_LINE, or SHADED_SURFACE";
    int i;

    ijk_msg(start);
    strcpy(input.dpystring[start+3], dpy);
    attribute_msg(start+4);
}

/*-----*
*           A T T R I B U T E _ M S G           *
*                                     *
* Effect:   Instruction message for attribute qualifier *
*           of any command.                     *
*-----*/

void attribute_msg (start)
    int start;
{
    static char dpy[5][80] =
        {"Color",
         "Line type",
         "Line thickness factor",
         "Symbol type",
         "Symbol size factor"};
    int i;

    for (i = start; i < (start + 5); i++)
        strcpy(input.dpystring[i], dpy[i-start]);
}

/*-----*
*           I J K _ M S G           *
*                                     *
* Effect:   Instruction message for get IJK value from *
*           the users.                     *
*-----*/

void ijk_msg (start)
    int start;

```

```

{
    static char dpy[3][80] =
        {"I start[,end[,inc]] use ':' to separate another set of I",
         "J start[,end[,inc]] use ':' to separate another set of J",
         "K start[,end[,inc]] use ':' to separate another set of K"};
    int i;

    for (i = start; i < (start + 3); i++)
        strcpy(input.dpystring[i], dpy[i-start]);
}

/*-----*
*           X Y Z _ M S G           *
*                                     *
* Effect:   Instruction message for get XYZ value from *
*           the users.                 *
*-----*/

void xyz_msg (start)
    int start;
{
    static char dpy[3][80] =
        {"X start[,end[,inc]] use ':' to separate another set of X",
         "Y start[,end[,inc]] use ':' to separate another set of Y",
         "Z start[,end[,inc]] use ':' to separate another set of Z"};
    int i;

    for (i = start; i < (start + 3); i++)
        strcpy(input.dpystring[i], dpy[i-start]);
}

/*-----*
*           R A K E _ R E A D         *
*                                     *
* Effect:   Construct the rake command with read *
*           qualifier to pass to the parser module. *
*-----*/

void rake_read (w, qual, call_data)
    Widget w;
    char *qual;
    caddr_t call_data;
{
    static char dpy[2][50] = {"File to read particle trace from",
                              "LINES or DOTS"};
    first = 1;
    strcpy(input.dpystring[0], dpy[0]);
    strcpy(input.dpystring[1], dpy[1]);
    attribute_msg(2);
    strcpy(input.comstring, qual);
    read_option(7, 1);
}

/*-----*
*           R A K E _ W R I T E       *
*                                     *
* Effect:   Construct the rake command with write *
*           qualifier to pass to the parser module. *
*-----*/

void rake_write (w, qual, call_data)
    Widget w;

```

```

char *qual;
caddr_t call_data;
{
    static char dpy[4][80] =
        {"File to write particle trace to",
         "Grid number for rake points",
         "IJ,IK,JK,XY,XZ, or YZ to restrict particles traces",
         "LINES or DOTS"};
    first = 1;
    strcpy(input.dpystring[0], dpy[0]);
    strcpy(input.dpystring[1], dpy[1]);
    if (qual[5] == 'x')
        xyz_msg(2);
    else
        ijk_msg(2);
    strcpy(input.dpystring[5], dpy[2]);
    strcpy(input.dpystring[6], dpy[3]);
    attribute_msg(7);
    strcpy(input.comstring, qual);
    read_option(12, 1);
}

```

```

/*-----*
*               R A K E               *
*-----*
* Effect: Construct the rake command to pass to the *
*         parser module.                       *
*-----*/

```

```

void rake (w, qual, call_data)
Widget w;
char *qual;
caddr_t call_data;
{
    static char dpy[3][80] =
        {"Grid number for rake points",
         "IJ,IK,JK,XY,XZ, or YZ to restrict particles traces",
         "LINES or DOTS"};
    first = 1;
    strcpy(input.dpystring[0], dpy[0]);
    if (qual[5] == 'x')
        xyz_msg(1);
    else
        ijk_msg(1);
    strcpy(input.dpystring[4], dpy[1]);
    strcpy(input.dpystring[5], dpy[2]);
    attribute_msg(6);
    strcpy(input.comstring, qual);
    read_option(11, 1);
}

```

```

/*-----*
*       C R E A T E _ M E N U _ M A N A G E R       *
*-----*
* Effect: Create a pop-up manager associate to the *
*         the given push button.                   *
*-----*/

```

```

Widget create_menu_manager(parent, mgrname)
Widget parent;
char *mgrname;
{

```

```

Widget shell = XtCreatePopupShell(mgrname, shellWidgetClass,
                                parent, NULL, 0);

Widget menu_mgr = XtCreateManagedWidget(mgrname,
                                         XwpopupmgrWidgetClass,
                                         shell, NULL, 0);

return (menu_mgr);
}

/*-----*
*               C R E A T E _ P               *
*-----*
* Effect:  Create the button and submenu of the for the *
*          main menu.                                   *
*-----*/

void create_p(mgr, mgrname , name, menulist, nitems)
    Widget      mgr;
    char        *name, *mgrname;
    menu_struct *menulist;
    int         nitems;
{
    Arg         wargs[1];
    Widget      menupane, pane_shell;
    int         i;
    WidgetList  buttons;

/*
 * Allocate a widget list to hold all button widgets.
 */

    buttons = (WidgetList) XtMalloc(nitems * sizeof(Widget));

/*
 * Create a popup shell to hold this pane.
 */
    pane_shell = XtCreatePopupShell("pane_shell",
                                    shellWidgetClass,
                                    mgr, NULL, 0);

/*
 * Create a Cascade menu pane, and attach it to the given menu manager.
 */

    XtSetArg(wargs[0], XtNattachTo, (XtArgVal) mgrname);
    menupane = XtCreateManagedWidget(name, XwcascadeWidgetClass,
                                      pane_shell, wargs, 1);

/*
 * Create a menu button for each item in the menu.
 */

    for(i=0; i<nitems; i++){
        buttons[i] = XtCreateManagedWidget(menulist[i].name,
                                            XwmenubuttonWidgetClass,
                                            menupane, NULL, 0);
        XtAddCallback(buttons[i], XtNselect, menulist[i].func,
                    menulist[i].com);
    }

/*
 * Manage all button widgets.
 */

```

```

*/

    XtManageChildren(buttons, nitems);
}

/*-----*
*           Q U I T _ C A L L B A C K           *
*-----*
* Effect:   Define a callback to exit(). Make it static *
*           - no need to make it known outside this file. *
*-----*/

static void quit_callback(w, client_data, call_data)
    Widget w;
    caddr_t client_data;
    caddr_t call_data;
{
    exit(0);
}

/*-----*
*           C R E A T E _ Q U I T _ B U T T O N   *
*-----*
* Effect:   Function to add a quit button. *
*-----*/

Widget create_quit_button(label, parent)
    char *label;
    Widget parent;
{
    Arg wargs[1];
    Widget w;

    XtSetArg(wargs[0], XtNlabel, label);
    w = XtCreateManagedWidget("quit", XwpushButtonWidgetClass, parent,
                               wargs, 1);

/*
* Quit on the release (button up), so the application can do something
* on the select (button down) if needed.
*/

    XtAddCallback(w, XtNrelease, quit_callback, NULL);
    return(w);
}

#define FONTHEIGHT(f) ((f)->max_bounds.ascent + (f)->max_bounds.descent)

/*-----*
*           B E E P           *
*-----*
* Effect:   Just ring the terminal bell. *
*-----*/

static void beep(w, event, params, num_params)
    Widget w;
    XEvent *event;
    String *params;
    int num_params;

```

```

{
    XBell(XtDisplay(w), 100);
}

/*
 * Associate the action "beep" with the function.
 */

static XtActionsRec actionsTable [] = {
    {"beep", beep},
};

/*
 * Override all translations that enter a newline.
 */

static char defaultTranslations[] =
    "Ctrl<Key>J:      beep()  \n\
    Ctrl<Key>O:      beep()  \n\
    Ctrl<Key>M:      beep()  \n\
    <Key> Return:    beep()";

/*-----*
 *   C R E A T E _ O N E _ T E X T _ W I D G E T   *
 *-----*
 * Effect: Create one line for user to type in the input *
 *         data.                                           *
 *-----*/

Widget create_one_line_text_widget(name, parent, args, nargs)
    char *name;
    Widget parent;
    Arg args[];
    int nargs;
{
    XFontStruct *font;
    Widget w;
    Arg wargs[2];
    XtTranslations trans_table;

/*
 * Add the actions and compile the translations.
 */

    XtAddActions(actionsTable, XtNumber(actionsTable));
    trans_table = XtParseTranslationTable(defaultTranslations);

/*
 * Create a Textedit widget
 */

    w = XtCreateManagedWidget(name, XwtexteditWidgetClass, parent,
    args, nargs);

/*
 * Install our translations.
 */

    XtOverrideTranslations(w, trans_table);

/*

```

```

* Get the font used by the widget.
*/

    XtSetArg(wargs[0], XtNfont, &font);
    XtGetValues(w, wargs, 1);

/*
* Set Widget height according to the font height.
*/

    XtSetArg(wargs[0], XtNheight, FONTHEIGHT(font) + 6 );
    XtSetArg(wargs[1], XtNwidth, 200);
    XtSetValues(w, wargs, 2);
    return (w);
}

```

APPENDIX B
GRAPHICS SUBROUTINES FOR USER INTERFACE

```

subroutine charst(str,length)

integer*4 length,tctx,tcty
character*(*) str

C
C str      character string to display
C length   number of characters in string
C
C
C Effects:  Display string starting at current character position
C           using current font and color. Leave current character
C           position at pixel to right of last character in the
C           string. Suppress operation if origin (current character
C           position) outside of current viewport. Clip string
C           according to current viewport.
C
C
C           include      '../plot3d.h'

integer*4 i,j,m
character*2 let

C
C Suppress all output if character origin outside of current viewport.
C
C   tctx = ifix(svtctx)
C   tcty = ifix(svtcty)

C   if ((tctx.ge.svvp1t).and.(tctx.le.svvp1t) .and.
C   &    (tcty.ge.svvpbm).and.(tcty.le.svvp1t)) then

C Assume current character position updates regardless of clipping.
C
C   x = float(svvp1t-svvp1t)*(12.0*0.0110677/11.5)*1.4

C   do 100 i=1,length
C     let = str(i:i)
C     j=ichar(let)
C     if (0.le.j.and.j.le.m) then
C       call vmove2(svchxp, svchyp)
C       call drawchar(let)
C       svtctx=svtctx+x
C       svchxp = svtctx
C     endif

100 continue
endif
end

subroutine clear

C
C Effect:   Set screen area within the current viewport to the
C           current color. Current graphics position and current
C           character position undefined.
C
C
C   include      '../plot3d.h'
C

```

c Mess up current graphics position and current character position.

c

```
svgrxp=-1.0
svgryp=-2.0
svgrzp=-3.0
svchxp=-4.0
svchyp=-5.0
svchzp=-6.0
```

c

c Print a black screen.

c

```
call vcolor(BLACK)
call vclear
```

end

subroutine cmov(x,y,z)

real x,y,z

c

c

c

c

c

Effects: Set current character position in 3D, which after transformation into screen coords becomes the new character position.

```
include '../plot3d.h'
real*4 w,untvec(4),wvec(4)
```

```
w=1.0
untvec(1)=x
untvec(2)=y
untvec(3)=z
untvec(4)=w
call xfmsys(untvec,wvec)
```

c

c

c

Ignore clipping.

```
call sclsys(wvec,svchxp,svchyp,svchzp)
svtxtx=svchxp
svtxty=svchyp
end
```

subroutine cmov2(x,y)

real x,y

c

c

c

c

c

c

c

c

c

c

Effects: After projection of position from model space to screen coords, determines current character position for subsequent string display.

```
include '../plot3d.h'
```

```

n      real*4      z,w,unvec(4),wvec(4)

      w=1.0
c
c      Assume to be at z-axis origin (ignored anyways).
c
      z=0.0
      unvec(1)=x
      unvec(2)=y
      unvec(3)=z
      unvec(4)=w
      call xfmsys(unvec,wvec)
c
c      Ignore clipping.
c
      call sclsys(wvec,svchxp,svchyp,svchzp)
      svtxtx=svchxp
      svtxty=svchyp
      end

      subroutine color(c)

      integer*4 c

c
c      Effects:      Sets the current color map index value for subsequent
c                    pixels written to bitplane memory. The current index
c                    value is called the current color attribute.
c
c
      include      '../plot3d.h'
c
c      Check index value within range and then set the color index.
c
      if ((0.le.c).and.c.le.8))
         svclcr=c
         call vcolor(c)
      endif
      end

      subroutine draw(x,y,z)

      real      x,y,z

c
c      x      x coord (rect. model space)
c      y      y coord (rect. model space)
c      z      z coord (rect. model space)
c
c
c
c      Effect:      Draw visible line from prior graphics position
c                    to (x,y,z) which becomes current graphics
c                    position. Appearance of line determined by
c                    current color.
c

```

```

C
C      include      '../plot3d.h'
      real*4      w,untvec(4),wvec(4)
C
C      Initialize all the variables.
C
      svgrxp=x
      svgryp=y
      svgrzp=z
      w=1.0
      untvec(1)=x
      untvec(2)=y
      untvec(3)=z
      untvec(4)=w
      call xfmsys(untvec,wvec)
C
C      Ignore clipping.
C
      call sclsys(wvec,svscxp,svscyp,svsczp)
      call vdraw (svscxp, svscyp, svsczp)

      end

      subroutine draw2(x,y)

      real      x,y

C      x      x coord (rect. model space)
C      y      y coord (rect. model space)
C
C      Effect: Draw visible line from prior graphics position
C              to (x,y) which becomes current graphics
C              position. Appearance of line determined by
C              current color.
C
      include      '../plot3d.h'
      real*4      z,w,untvec(4),wvec(4)
C
C      Ignore current z-axis value.
C
      svgrxp=x
      svgryp=y
      z=0.0
      w=1.0
      untvec(1)=x
      untvec(2)=y
      untvec(3)=z
      untvec(4)=w
      call xfmsys(untvec,wvec)
C
C      Ignore clipping.
C
      call sclsys(wvec,svscxp,svscyp,z)
C
      call vdraw2 (svscxp,svscyp)
      end

```

```

      subroutine ginit

C
C   Effect:   Initialize the graphics package (Vogle) and some of the
C             global variables for the graphics package.
C
      include      '../plot3d.h'
C
C   Clear the graphics window using black color.
C
      call vcolor(0)
      call vclear

C
C   Setup the viewport coordinate.
C

      svzbnr = -(2**15)
      svzbfr = (2**15) - 1
      svvplt = -1.0
      svvpbm = -1.0
      svvprr = 1.0
      svvptp = 1.0

C
C   Turn the clipping mode on
C
      call clipping(1)

C
C   Mess up current graphics position and current character position.
C

      svgrxp=1.0
      svgryp=2.0
      svgrzp=3.0
      svchxp=4.0
      svchyp=5.0
      svchzp=6.0
      svtxtx=-1.0
      svtxty=-2.0
      end

      subroutine move(x,y,z)

      real      x,y,z

C
C   x          x coord (rect. model space)
C   y          y coord (rect. model space)
C   z          z coord (rect. model space)
C
C
C   Effect:    Moves to (x,y,z) as current graphics position
C             without any visible change on the screen.
C
      include      '../plot3d.h'
      real*4      w,untvec(4),wvec(4)
C

```

```

C
C Initialize the global variables.
C
    svgrxp=x
    svgryp=y
    svgrzp=z
    w=1.0
C
C Initialize the transformation matrix.
C
    untvec(1)=x
    untvec(2)=y
    untvec(3)=z
    untvec(4)=w
    call xfmsys(untvec,wvec)
C
C Ignore clipping.
C
    call sclsys(wvec,svscxp,svscyp,svsczp)
    call vmove(svscxp,svscyp,svsczp)
end

subroutine move2(x,y)

    real    x,y

C
C x      x coord (rect. model space)
C y      y coord (rect. model space)
C
C Effect: Moves to (x,y) as current graphics position
C          without any visible change on the screen.
C
    include    '../plot3d.h'
    real*4    z,w,untvec(4),wvec(4)
C
C Initialize the global variables and leave Z-axis alone.
C
    svgrxp=x
    svgryp=y
    z=0.0
    w=1.0
C
C Initialize the transformation matrix.
C
    untvec(1)=x
    untvec(2)=y
    untvec(3)=z
    untvec(4)=w
    call xfmsys(untvec,wvec)
C
C Ignore clipping.
C
    call sclsys(wvec,svscxp,svscyp,z)
    call vmove2(svscxp,svscyp)
end

```

```

subroutine ortho2(gauche, right, touche, top)
  real    gauche, right, touche, top
c
c  gauche      gauchemost x coordinate of the orthographic
c              clipping plane
c  right       rightmost x coordinate of the orthographic
c              clipping plane
c  touche      lowermost y coordinate of the orthographic
c              clipping plane
c  top         uppermost y coordinate of the orthographic
c              clipping plane
c
c  Effects:    Define the viewing box into the world coordinate system
c              for mapping from the eye-fixed coordinate system to the
c              screen coordinate system, using the rules of 2D
c              orthographic projection (z axis values simply collapse
c              on to the screen).
c
c  include     '../plot3d.h'
c
c  Setup the transformation matrix.
c
  svtmcr(1,1)=2.0/(right-gauche)
  svtmcr(1,2)=0.0
  svtmcr(1,3)=0.0
  svtmcr(1,4)=0.0
  svtmcr(2,1)=0.0
  svtmcr(2,2)=2.0/(top-touche)
  svtmcr(2,3)=0.0
  svtmcr(2,4)=0.0
  svtmcr(3,1)=0.0
  svtmcr(3,2)=0.0
  svtmcr(3,3)=-1.0
  svtmcr(3,4)=0.0
  svtmcr(4,1)=-((right+gauche)/(right-gauche))
  svtmcr(4,2)=-((top+touche)/(top-touche))
  svtmcr(4,3)=0.0
  svtmcr(4,4)=1.0
  end

subroutine perspe(fovy, aspect, near, far)
  integer*4  fovy
  real       aspect, near, far
c
c  fovy        field-of-view angle fovy in the y
c              direction of the eye coordinate system
c  aspect      the aspect ratio which determines
c              the field of view in the x direction
c  near        near clipping plane in z direction
c  far         far clipping plane in z direction.
c
c  N.B. near and far are always positive and near <= far, but since
c  the eye coordinate system is right handed, near and far are
c  actually inverted (-value) for true z coordinate values.
c

```

```

C
C      Effects:      Define the viewing pyramid into the world coordinate
C                    system for mapping from the eye-fixed coordinate
C                    system to the screen coordinate system (origin (0,0) at
C                    left lower corner of the screen), using the rules of
C                    (pinhole) perspective.
C
C      include      './plot3d.h'
C      real*4      pi,r,cot
C      PARAMETER (PI=3.1415926)
C
C      Setup the transformation matrix.
C
C      r=((float(fovy)/10.0)/180.0)*pi
C      cot=1.0/tan(r/2.0)
C      svtmcr(1,1)=cot/aspect
C      svtmcr(1,2)=0.0
C      svtmcr(1,3)=0.0
C      svtmcr(1,4)=0.0
C      svtmcr(2,1)=0.0
C      svtmcr(2,2)=cot
C      svtmcr(2,3)=0.0
C      svtmcr(2,4)=0.0
C      svtmcr(3,1)=0.0
C      svtmcr(3,2)=0.0
C      svtmcr(3,3)=-((far+near)/(far-near))
C      svtmcr(3,4)=-1.0
C      svtmcr(4,1)=0.0
C      svtmcr(4,2)=0.0
C      svtmcr(4,3)=-((2.0*far*near)/(far-near))
C      svtmcr(4,4)=0.0
C      end
C
C      subroutine  sclsys(vector,xsc,ysc,zsc)
C
C      real*4      vector(4),xsc,ysc,zsc
C
C      vector      is transformed 4-space vector [x,y,z,w] from xfmsys.
C                  Presumably [x,y,z,w] is within clipping bounds.
C
C      Effects:      Compute screen coordinates from viewport and setdepth
C                    parameters: xsc=(x/w)*Vsx+Vcx
C                               ysc=(y/w)*Vsy+Vcy
C                               zsc=(y/w)*Vsz+Vcz
C
C                    where
C                               Vsx=(right-left)/2.0  Vcx=(left+right)/2
C                               Vsy=(top-bottom)/2.0  Vcy=(bottom+top)/2
C                               Vsz=(far-near)/2.0    Vcz=(near+far)/2
C
C      include      './plot3d.h'
C      integer*2    near,far
C      real*4       w,nr,fr
C
C      Using the global variable to initialize the local variable for the x,
C      y, z coordinate.

```

```

C      near=svzbmr
C      far=svzbfr
C      nr=float(near)
C      fr=float(far)
C      w=vector(4)

C
C      Calculate the screen coordinate.
C
C      if (w.eq.0.0) w=1.0
C      xsc=((vector(1)/w)*((svvprr-svvprr)/2.0))+((svvprr+svvprr)/2.0)+0.5
C      ysc=((vector(2)/w)*((svvprr-svvprr)/2.0))+((svvprr+svvprr)/2.0)+0.5
C      if (zsc .ne. 0.0) then
C          zsc=((vector(3)/w)*((fr-nr)/2.0))+((nr+fr)/2.0))+0.5
C      endif
C      end

C
C      subroutine xfmsys(untvec,wvec)
C      real*4      untvec(4),wvec(4)

C      untvec      is 4-space vector [x',y',z',w']; w' is usually input
C                  as 1.0, i.e. in the unit hyperplane.
C
C      wvec        is 4-space vector [x,y,z,w] output in the w
C                  hyperplane convenient for clipping and scaling (see
C                  sclsys).
C
C      Effects:    Compute [x,y,z,w]=[x',y',z',w']*MVP
C                  where MVP (Model, View, Project) is the current
C                  4x4 transformation matrix

C      include     '../plot3d.h'

C      Compute the transformation matrix
C
C      wvec(1)=untvec(1)*svtmcr(1,1)+
C      &      untvec(2)*svtmcr(2,1)+
C      &      untvec(3)*svtmcr(3,1)+
C      &      untvec(4)*svtmcr(4,1)
C
C      wvec(2)=untvec(1)*svtmcr(1,2)+
C      &      untvec(2)*svtmcr(2,2)+
C      &      untvec(3)*svtmcr(3,2)+
C      &      untvec(4)*svtmcr(4,2)
C
C      wvec(3)=untvec(1)*svtmcr(1,3)+
C      &      untvec(2)*svtmcr(2,3)+
C      &      untvec(3)*svtmcr(3,3)+
C      &      untvec(4)*svtmcr(4,3)
C
C      wvec(4)=untvec(1)*svtmcr(1,4)+
C      &      untvec(2)*svtmcr(2,4)+
C      &      untvec(3)*svtmcr(3,4)+
C      &      untvec(4)*svtmcr(4,4)
C      end

```

VITA

Yukkin Daniel Chan was born in Hong Kong on December 28, 1963. After he graduated from the city high school in the spring 1984, he came to the United States to study at Middle Tennessee State University (MTSU). He received the Bachelor of Science degree with honors in Computer Science from MTSU in December 1987, and then joined The University of Tennessee Space Institute as a graduate research assistant to pursue graduate studies in Computer Science. The Master degree was awarded in August 1990. The author is a member of the Association of Computing Machinery.

