SCIENTIFIC VISUALIZATION OF A 3-D VECTOR FIELD IN 3-D SPACE USING X-WINDOWS

By

RAVI K. GUNDIMEDA

Master of Technology

Indian Institute of Technology

Kanpur, India

1986

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1992

SCIENTIFIC VISUALIZATION OF A 3-D
VECTOR FIELD IN 3-D SPACE
USING X-WINDOWS

Thesis Approved:

William David Miller
Thesis Adviser

J Chandler

D. E. [illegible]

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGEMENTS

I profoundly thank my graduate advisor Dr. David Miller for his unstinted help and guidance. His constructive criticism helped me in gaining confidence during my graduate program. My sincere thanks to Drs. G. E. Hedrick and B. Mayfield for serving on my graduate committee. Their suggestions and support were very helpful throughout the study.

I would like to express my gratitude to the faculty and staff of the Computer Science department for their help and support. In particular, I offer my thanks to Dr. Samadzadeh, Mr. R. Stolfa and Mr. Vassol for their help. My special thanks are extended to my friends Dipti, Shashi, Pamela, Jayathirtha, Amir, Bhupesh and Aouni for helping me throughout my study in Stillwater. I would also like to thank my wife Anu for being so understanding and cooperative, and my parents for their encouragement.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

NOMENCLATURE

Display	A workstation consisting of a keyboard, a pointing device such as a mouce, and one or more screens.
Drawables	Windows and Pixmaps are collectively known as drawables.
Eigenpairs	The pair consisting of an eigenvalue and the corresponding eigenvector.
Eigenvalue	A scalar, λ, which along with a vector X and a square matrix A, satisfy the equation $AX = \lambda X$.
Eigenvector	A vector X, which along with a scalar λ, and a square matrix A, satisfies the equation $AX = \lambda X$.
Jacobian	A square matrix consisting of the partial derivatives of a linear system.
Normalization	The process of transforming a vector, so that it's of unit length
Pixel	A point in a drawable.
Pixmap	An array of pixel values.
Resources	Complex abstraction maintained by the server to reduce network traffic. Examples of resources are windows, widgets, pixmaps, cursor, graphics context, etc..

CHAPTER I

INTRODUCTION

Introduction to Scientific Visualization

Scientific visualization is concerned with techniques that allow the extraction of knowledge from the results of simulations and computations. Massive amounts of data are being produced and collected as a result of increased computational power. There is a need to communicate this data to the scientist so that effective use can be made of human creative and analytic capabilities. Computer generated images coupled with human vision backed by the principles of perceptual psychology are the means used in scientific visualization to achieve this communication [13]. Scientific visualization is not only a tool for discovery and understanding, but is also a tool for teaching and communication [7].

Eigenvalue Problems

Given a square matrix A, the eigenvalue problem consists of computing those non-zero vectors $\mathbf{X}$ and those scalars λ with the property as given by the following equation.

$$A\mathbf{X} = \lambda\mathbf{X}. \qquad (1)$$

The pair (λ,**X**) is called an eigenpair, consisting of the eigenvalue λ, and the eigenvector **X**. The solution to a linear differential equation with constant coefficients can be expressed in terms of the eigenpairs of the coefficient matrix. Differential equations describing a vector field could be reduced to a Jacobian matrix, the eigenpairs of which yield an insight into the qualitative behavior of the vector field.

Many processes in the real world involve calculating sets of numbers that can be considered to define a vector field in an n-dimensional space. Often, to understand the general behavior of such a vector field, one needs to look at the variation of vectors with position. The Jacobian matrix consisting of partial derivatives of the vector components provides a systematic description of this variation. The essential geometry of this variation is represented by the eigenvalues and eigenvectors of the Jacobian matrix. Scientific visualization of these eigenvalues and eigenvectors of the vector field would therefore reveal the structure of the vector field, including the saddle points, attracting and repelling nodes, foci, etc.

The problem, then, is to determine an effective means of communicating the invariant geometric properties of the vector field using eigenvectors and eigenvalues of the vector field as the basis of communication.

This study involves the development of a scientific visualization tool to display the eigenvectors and eigenvalues of a three-dimensional vector field in a three-dimensional space, using X-Windows.

Introduction to X-Windows

X-Windows is a network-based windowing system for bit-mapped graphics displays, developed jointly by MIT's Project Athena and Digital Equipment Corporation, with contributions from many other companies[21]. The X-Window system is a large and complex system. Much of that complexity comes from X attempting to deal with virtually every type of computer graphics display available. It also attempts to provide a complete graphics system - complete enough to create windowing interfaces and complete enough to handle graphics - intensive page design or computer-aided design (CAD) packages.

The popularity of X-Windows has expanded past the academic environment into a windowing system of choice in the workstations and UNIX worlds, and is poised to expand further into the microcomputer and minicomputer worlds.

This popularity stems from the fact that X-Windows offers a solution to a seemingly intractable problem: How to provide a common interface across many different computers, running a number of operating systems, with a number of different displays. This problem has gained significance in this era of open systems in the computer industry, where industry standards are paramount and proprietary architectures are out. X provides the perfect solution to this problem as it provides a graphical interface similar to that provided by Macintosh, MS Windows, etc. that runs on everything and is not tied to a single vendor. It also allows execution of

application programs on remote computers with the results being displayed on a local workstation.

One of the many features of X-Windows is to provide a mechanism for developing a user-interface for an application rather than a policy; the user-interface policy is left to the client. Thus it is also an ideal tool for developing user interfaces.

Present Work

Statement of the Problem

Given a three dimensional vector field in a three dimensional space, how do we display the eigenvalues and eigenvectors of the vector field, so that the dynamic behavior of the vector field is effectively communicated to the viewer.

Design Objectives

The effectiveness of the display of the eigenvectors and eigenvalues for a grid of points in a 3-D vector space should meet the following objectives:

(1) No formal training should be required to read and understand the display. It is assumed that the viewer has prior theoretical knowledge of the concepts involving eigenvectors and eigenvalues of a vector field, besides knowing the principles of geometry and graphics.

(2) The display package should be portable to a variety of networked systems.

(3) The display should communicate the behavior of the vector field.

This thesis involves the development of a scientific visualization package to display the dynamic behavior of a 3-D vector field in 3-D space. The tool would be developed in a networked environment, using X-Windows system and the C programming language.

Outline of the Thesis

Chapter 2 presents a review of the current literature on scientific visualization, techniques to determine eigenvector and eigenvalues of a Jacobian matrix and X-Windows. The design implementation methodology of this software package is presented in Chapter 3. In Chapter 4, the results and conclusions of this work is presented. Finally, directions for future work on this topic are suggested in Chapter 5.

CHAPTER II

REVIEW OF CURRENT LITERATURE

Scientific visualization is the use of visual images to represent and communicate information about relationships among multidimensional quantities[17]. The basic building tools for creating visual images are graphics (the points, the lines, etc.) and geometry (relative positions of points, lines, etc.). The combination of these two well developed fields result in the development of powerful visualization tools that help scientists and engineers gain a better understanding of the physical world. Visual images utilizing color, intensity, transparency, and texture etc., can convey a tremendous amount of information in a short period of time.

For example, a function $y = f(x)$ represented as a two-dimensional graph, conveys the relationship between y and x more readily than the functional formula itself. But representing higher dimensional data as a readily understandable image poses a bigger challenge than just a two-dimensional graph. To this end, scientists and engineers have constantly strived to refine the existing methods of displaying visual images.

But when do we say that one visual image looks better than another one displaying the same scientific phenomenon. For example, numerous techniques in computer graphics including texture, specular light, and ray tracing, etc., help improve the quality

of an image of a car body on a screen by making the image look the way we know it looks. Science and engineering on the other hand, produce objects which do not belong to every day life. These objects do not have to look "real". What does "real" mean for the diagram of, say, a vector field ? In the scientific field, if a picture is designed to help us understand a complicated phenomena, we must understand the picture. We may thus say that a specific picture looks "real" when the relevant picture symbols, which we have learned to read are present.

Though visual images (snapshots) provide a highly leveraged means to convey information, sequences of images can yield even more information. This is called Animation and is an important dimension of scientific visualization [10, 6]. Yet another important dimension of scientific visualization is the impact of interaction. Allowing the researcher to interactively direct the computations increases the chances for new insights and understanding [17]. There are many other aspects of visualization which govern the effective utilization of human perception and vision, such as spatial attributes, feature extraction, depth perception, etc. [11].

Many advanced scientific and engineering applications are using scientific visualization as a tool for discovery and understanding. Chernoff suggested the use of cartoon faces with variable attributes (such as the length of the nose, the curve of the mouth, etc.) to graphically represent higher-dimensional data like fossil shell data and geological core arrays[3]. The author backs his suggestion by providing evidence that these faces can be used for cluster, discriminant, and time-series analysis. Asimov, on the other hand,

suggests the extraction of a sequence of images from the infinite number of orthogonal projections of multidimensional data onto two dimensions [2]. The author enumerates the desired qualities of these sequences besides describing techniques to obtain them. The author admits in his concluding remarks that, to understand these sequences, substantial training is required.

Visualization techniques are also being used in radiological therapy, joint disease diagnosis, CT and MR imaging and molecular modelling [8]. Helman et al. discuss the display, interaction, and implementation of vector field topology in computational fluid dynamics [14]. Rosenblum uses animation techniques to explore temperature and salinity effects on salt fingering convection [20]. Other fields of science and engineering, wherein scientific visualization is currently being used include mathematics, geosciences (meteorology), space exploration, astrophysics, and finite element analysis [5].

Besides being used as a tool for discovery and understanding, scientific visualization is also being used as a tool for communication and teaching. Many abstract and multi-dimensional concepts of modern science and technology would not be communicated effectively without the use of visualization techniques [5]. Cunningham et al., discuss the importance of scientific visualization in education [4]. They also outline the barriers facing the use of visualization as an education tool and suggest ways to overcome these barriers.

Eigenproblems

Many applications in engineering and science involve the calculation of sets of numbers which can be considered to define a vector field in an n-dimensional space. Understanding the variation of the vectors in space with respect to position would yield an insight to the general behavior of the vector field. The eigenvalues and eigenvectors derived from the Jacobian matrix (consisting of the derivatives of the vector components) yield the geometry of the variation of the vector field. The concepts involving vector fields, eigenvectors and eigenvalues, and numerical techniques to solve them are well established. Detailed description about these concepts can be found in [12, 15, 9, 6].

X-Windows

The first major publication on the X-Window system appeared in 1986 [21], and as such, is required reading for anyone working in the field of graphical interfaces and X-Windows. The X-Windows system, developed at MIT, is a window management system that provides virtual terminal interfaces to users. X-Windows was developed to provide the UNIX operating system with a graphical interface similar to that provided by Macintosh, MS-Windows, etc.. Written in C, it has been successfully ported to a great variety of machines and systems. The X-Window system is based on the client-server model. One of the strong points of this system is that it is device-independent. This means multiple window managers, each with a different look and feel, can be built on top of the window

system [18, 21]. Thus, display windows can be maintained transparently across a local area network. Scheifer et al., describe the design of the system in four main areas: system software, programming interface, the input and the output structure [21]. The authors also present the design objectives of the X-Window system. More on window-based man-machine interaction and windows management can be found in [6] and [18].

X-Windows programmers generally base applications on libraries that provide an interface to the base window system. The most widely used low level interface to X is the standard C language library known as XLib. XLib defines an extensive set of functions that provide complete access and control over the display, windows, and input devices[24]. Using XLib to develop applications however prove to be error prone and tedious; many programmers prefer to use one of the higher level toolkits designed to be used with X. The X toolkit is one such high level toolkit which is commonly used. Young presents the programming techniques with the XToolkit and the OSF/MOTIF widget set in an effective manner.[24].

CHAPTER III

METHODOLOGY - DESIGN AND IMPLEMENTATION

Many of the concepts used in developing the tool are widely known; the aim of this software is to combine known techniques and concepts in a way so as to yield a visualization tool which would be simple to use and effective in communicating the desired characteristics of the vector field.

In this chapter, the overall structure of the software along with a more detailed description of the implementation of the important parts of this software is presented. Finally the limitations of this package are presented.

Overall Structure Of The Software

The overall structure of the software, named "Evevwin" hereafter, is presented in Figure 1. Creation of a visual image of a 3-D vector field in Evevwin involves three steps:

(1) Determination of eigenpairs of a given Jacobian for a grid of points.

(2) Graphical composition of the coordinates of the eigenpairs.

(3) The user-interface which presents the user a window containing the visual image and a set of options.

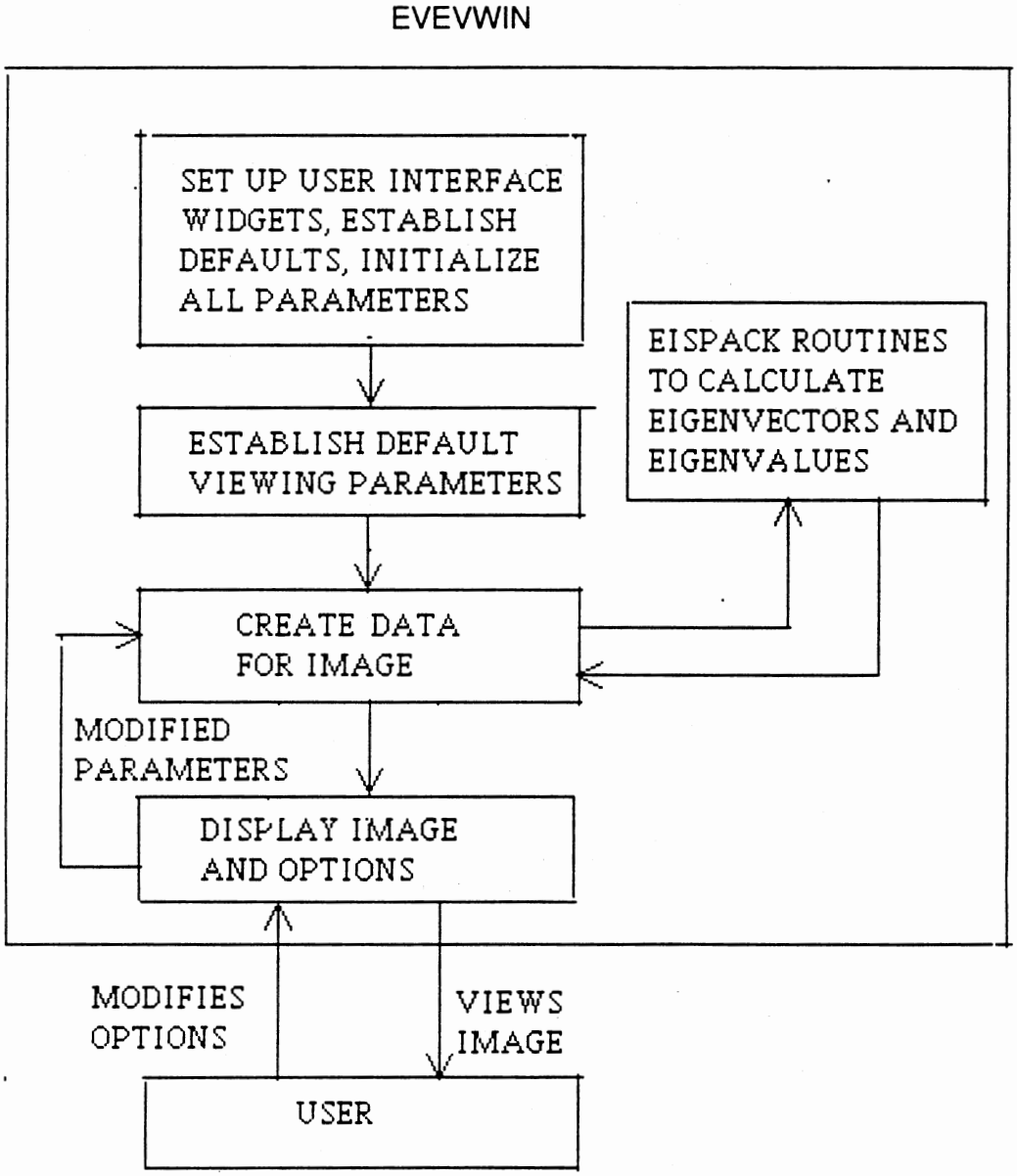

Figure 1. Structure of EVEVWIN - a scientific visualization tool for eigenpairs of a 3-D vector field using X-Windows

The user of Evevwin has a host of modifiable viewing parameters at his disposal. When a viewing parameter is modified ,the image of the vector field is recreated and displayed. The C programming language has been used for this package along with X-windows.

Determination of Eigenpairs

One of the most widely used subroutine packages in the determination of eigenpairs of a matrix is EISPACK[12]. Since this package is widely available it has been used in the present work. These subroutines are in FORTRAN77 and they use FORTRAN library routines. Since the present work is in C, an interface to these FORTRAN subroutines was developed and used. This avoided the necessity of converting these routines to C and thereby the need to check the converted version for bugs and accuracy. These subroutines are available both in single- precision and double-precision. For better accuracy, the double-precision version of these subroutines has been used.

In particular, the following subroutines of EISPACK were used in the following order, given a real Jacobian matrix:

(1) BALANC: which applies a balancing transformation to a general matrix.

(2) ORTHES: which reduces a general matrix to upper Hessenberg form using Householder matrices.

(3) HQR2: which computes all the eigenpairs of an upper Hessenberg matrix using the implicit OR method.

(4) ORTBAK: which computes all the eigenpairs of the original general matrix, given the eigenpairs of the upper Hessenberg matrix.

(5) BALBAK: which inverts the balancing transformations made by BALANC.

The two dominating issues involved in writing the C interface to these subroutines were:

(1) All variables in FORTRAN are passed to procedures by reference, whereas in C variables can be passed to procedures either by reference or by value.

(2) Arrays in C are accessed in a row-major fashion, while arrays in FORTRAN are accessed in a column-major fashion.

Graphical Transformation

The graphical transformation of a 3-D image to 2-D device coordinates involved the following steps:

(1) Translate the image to viewing coordinates from the world coordinates.

(2) Transform the image to fit a canonical view volume based on the type of projection.

(3) Clip in 3-D using Cohen-Sutherland algorithm.

(4) Project the clipped image into 3-D viewport coordinates.

(5) Project the image in 3-D viewport coordinates onto 2-D device coordinates.

For a detailed description of the above process and an explanation of the various graphical terms, the interested can refer to "Computer Graphics: Principles and practice" by Foley et al.[6].

The user can interactively change any of the projection parameters used in the above process to gain a better perspective of the vector field. The ability to change these parameters interactively and view the vector field at different ranges and angles underlines the goal of scientific visualization.

User-Interface

The user-interface for this software package has been developed using the basic functions provided by XLib, and the tools provided by the Xtoolkit and the Motif Window manager. It comprises of a top-level window consisting of two children:

(1) The Command Panel

(2) The Drawing Area

The Command Panel consists of a set of push buttons, which, when activated, allow the user to enter viewing parameters, to save options or to quit the application. Apart from a QUIT button and a SAVE button, the following buttons are provided:

(1) VRP Button: This allows the user to modify the view reference point of the vector field - the origin of the view coordinate system(VRC).

(2) VUP Button: This allows the user to modify the view up direction vector which would define the V-axis of the VRC system.

(3) VPN Button: This allows the user to modify the view-plane normal direction vector which define the W-axis of the VRC system.

(4) PRP Button: This allows the user to modify the projection reference point defined in the VRC system.

(5) WINDOW Button: This allows the user to modify the window coordinates of the viewing plane.

(6) VPORT Button: This allows the user to modify viewport coordinates.

(7) WVIEW Button: This allows the user to modify the vector space he wishes to view. It also allows the user to modify the spacing of the grid of points in evaluating the Jacobian.

(8) PRTYPE Button: This allows the user to select the type of projection he/she wishes to view.

(9) SETNO Button: For demonstration purposes, three vector fields incorporated into the package. This button allows the user to select one of them.

Many of these terms for the user-interface buttons are adapted from Foley et al.[6].

In order to speed up the software, buffering techniques were used in addition to calculating the points representing the image only if necessary.

The Visual Image of a 3-D Vector

As mentioned in Chapter II, a picture designed to understand a complicated scientific phenomena should consist of picture symbols which the user would have learnt to read in the past. This work deals only with real Jacobian matrices which yield only the real

eigenvalues and corresponding real eigenvectors or one real eigenvalue and two complex conjugate eigenvalues. Thus the picture symbols used for depicting the eigenvector at a point in space in this work consist of directed solid lines for the real eigenvectors, a rectangular plate for the eigenplane of pairs of complex conjugate eigenvalues and dashed lines for real vectors obscured by the plates..

Since the X-terminals on which this software was developed are monochromatic, the image was limited to represent only the eigenvectors at a point.

Figure 2 presents the picture symbols used in representing a 3-D eigenvector at a point in space. In Figure 2(a), all the three component vectors are real. Figure 2(b) and 2(c) present the case wherein one component vector is real and the remaining are complex.

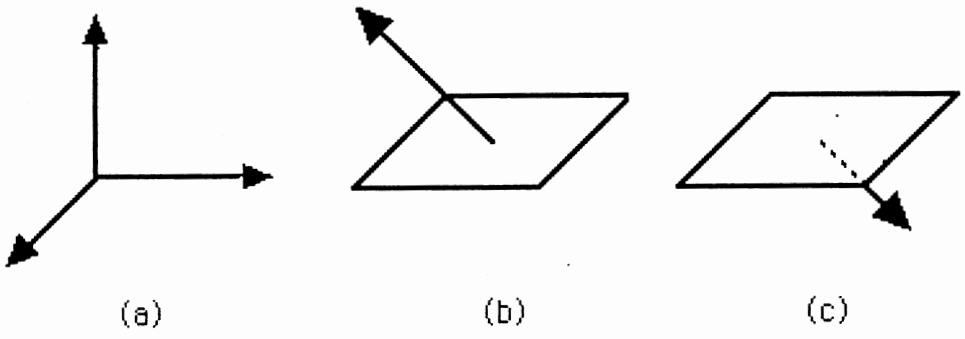

Figure 2. Picture Symbols used to represent a 3-D vector at a point in space
(a) All three real vector components
(b) One real and two complex conjugate vector components
(c) One real and two complex conjugate vector components, with the real component partially obscured

Overall Algorithm

The overall algorithm used in the software is as follows:

(1) Initialize the FORTRAN library.

(2) Initialize the server and open a top-level window.

(3) Set up two windows as children of the top-level window; one window for displaying image and the other a command panel which allows the user to interactively modify the viewing parameters.

(4) Initialize all the necessary data structures used for the user-interface.

(5) Get the defaults for the data.

(6) While the user does not quit the application, Do

  (a) Create the data for the image including the following:

    (i) Determine the eigenpairs for each point in the world volume coordinates, therefore the 3-D coordinates of the vertices of the vectors.

    (ii) If the eigen pair consists of only one real eigenvector, determine the 3-D coordinates of the vertices of a rectangular plate perpendicular to the real vector.

    (iii) Transform these 3-D coordinates to 2-D device coordinates.

  (b) Create the image and display in the appropriate window.

  (c) Wait for user interaction.

  End Do.

(7) Close the FORTRAN library and the server display and quit.

CHAPTER IV

RESULTS AND CONCLUSIONS

The goal of this work was to display a 3-D vector field in 3-D space using easily understood picture symbols so that the qualitative behavior of the vector field is conveyed to the user. In this chapter, the method of using the software, the testing methodology, limitation of this package and conclusions are presented.

User-Manual for the package

The application can be started by typing the command "evevwin" at the command prompt. After a certain delay (spent in initializing all the relevant data structures), the top-level window consisting of the image of simple diverging vector field using Orthogonal Projection is displayed along with a command panel as described in the previous chapter. This window is shown in Figure 3 on the following page.

The user can then proceed to modify the viewing parameters for the same vector field and view the image or choose another vector field. For demonstration purposes, three sets of vector fields have been used. They are as follows:

(1) A Diverging Vector Field: The equation set describing this vector field is given below:

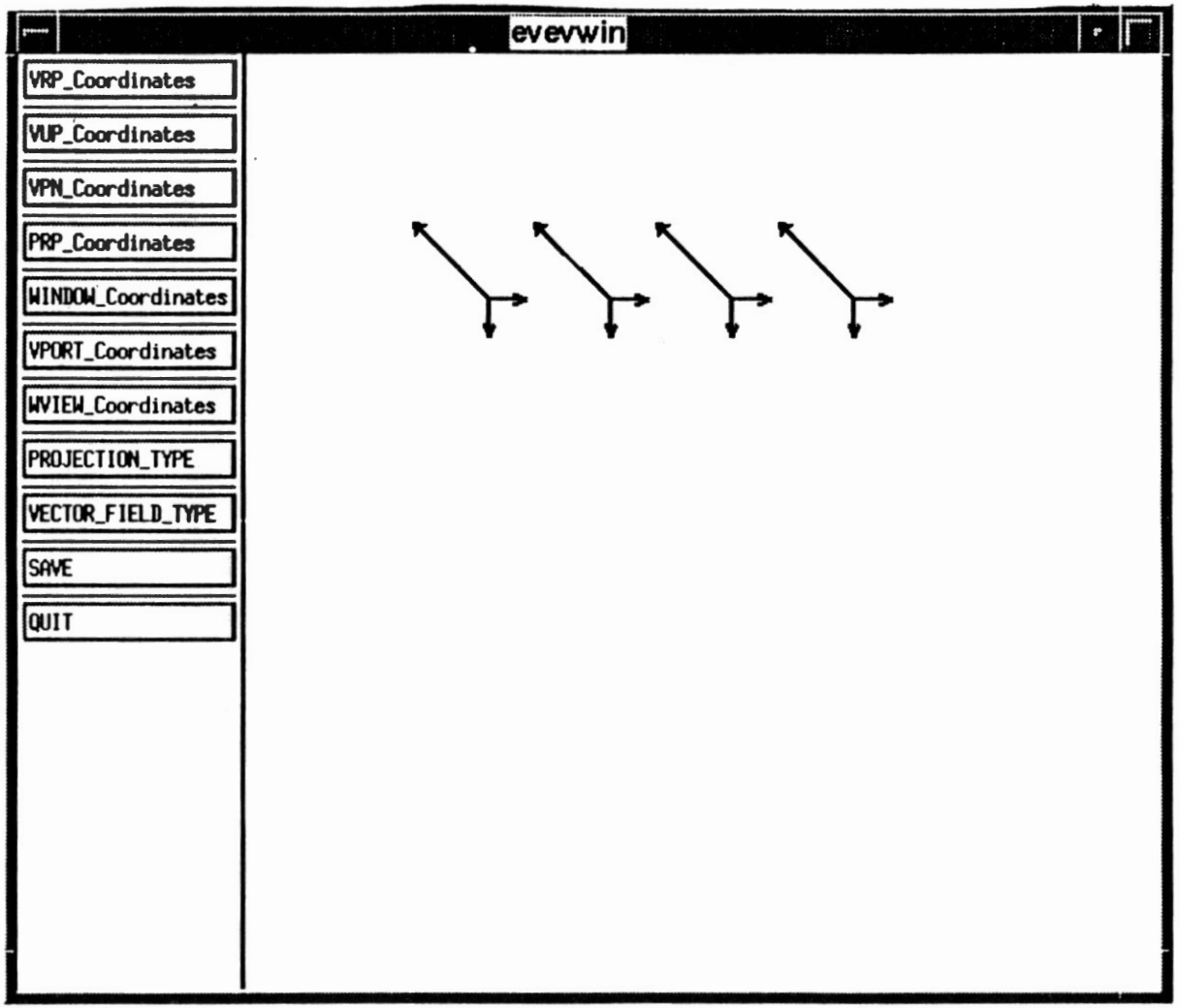

Figure 3. Top-level window of Evevwin

$$f_1(x,y,z) = \alpha * x \quad (2)$$

$$f_2(x,y,z) = -\alpha * y \quad (3)$$

$$f_3(x,y,z) = 0.0 \quad (4)$$

For demonstration α is made equal to 5.0. The Orthogonal projection image of this vector field is presented in Figure 4.

(2) A Circular Vector Field: The equation set describing this vector field is given below:

$$f_1(x,y,z) = -\alpha * y \quad (5)$$

$$f_2(x,y,z) = \alpha * x \quad (6)$$

$$f_3(x,y,z) = 0.0 \quad (7)$$

For demonstration α is made equal to 5.0. A perspective projection image of this vector field is presented in Figure 5.

(3) An Arbitrary Vector Field: The equation set for this vector field is given below:

$$f_1(x,y,z) = 3 * x^2 + 2 * y + z \quad (8)$$

$$f_2(x,y,z) = 4 * x + 5 * y + 6 * z^2 \quad (9)$$

$$f_3(x,y,z) = 8 * x - 9 * y^2 + 10 * z \quad (10)$$

An orthogonal projection image of this vector field is presented in Figure 6.

The user can either modify the viewing parameters, choose a different vector field, save his options, or quit the application. Two

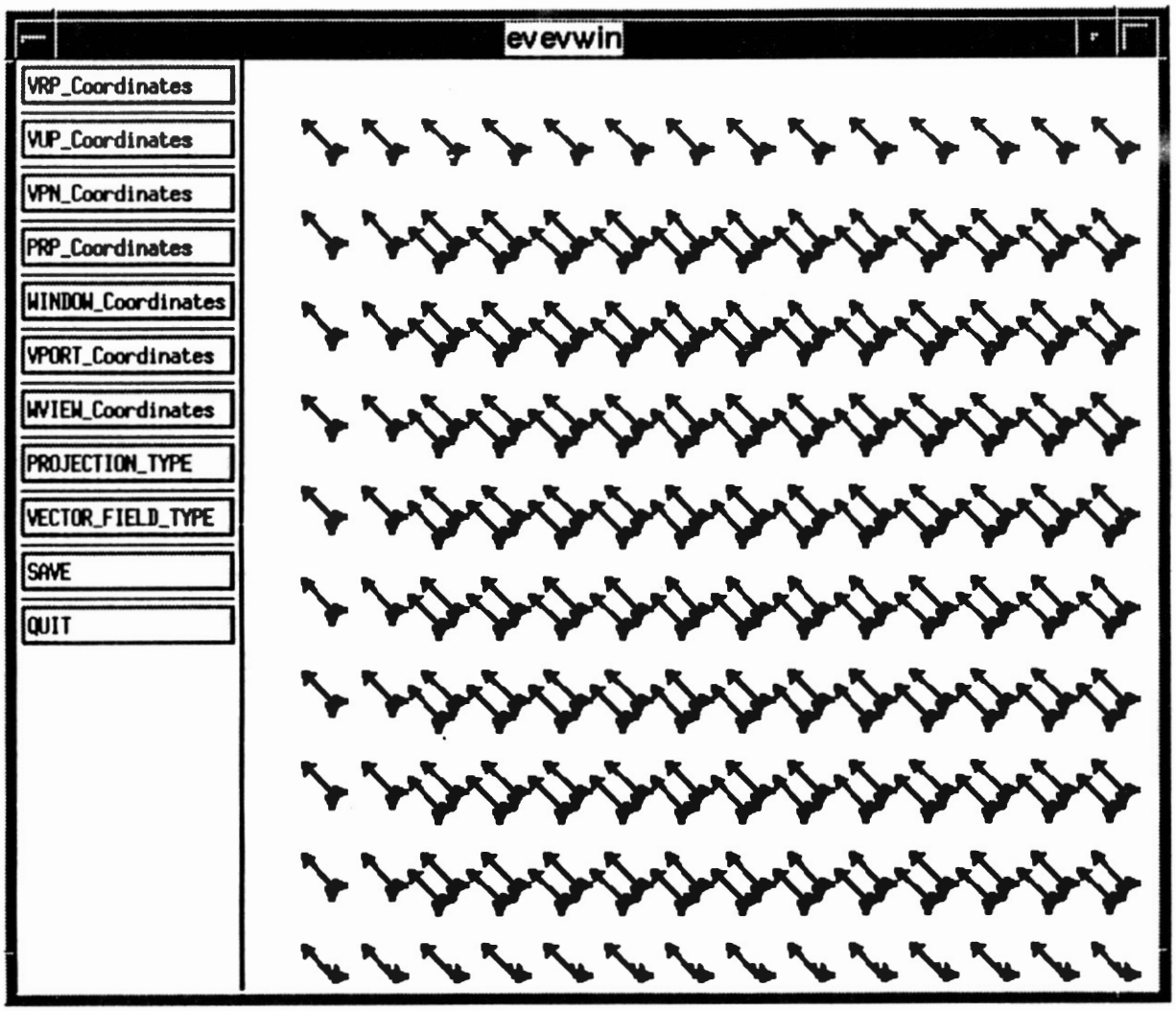

Figure 4. Orthogonal Projection of a Diverging Vector Field

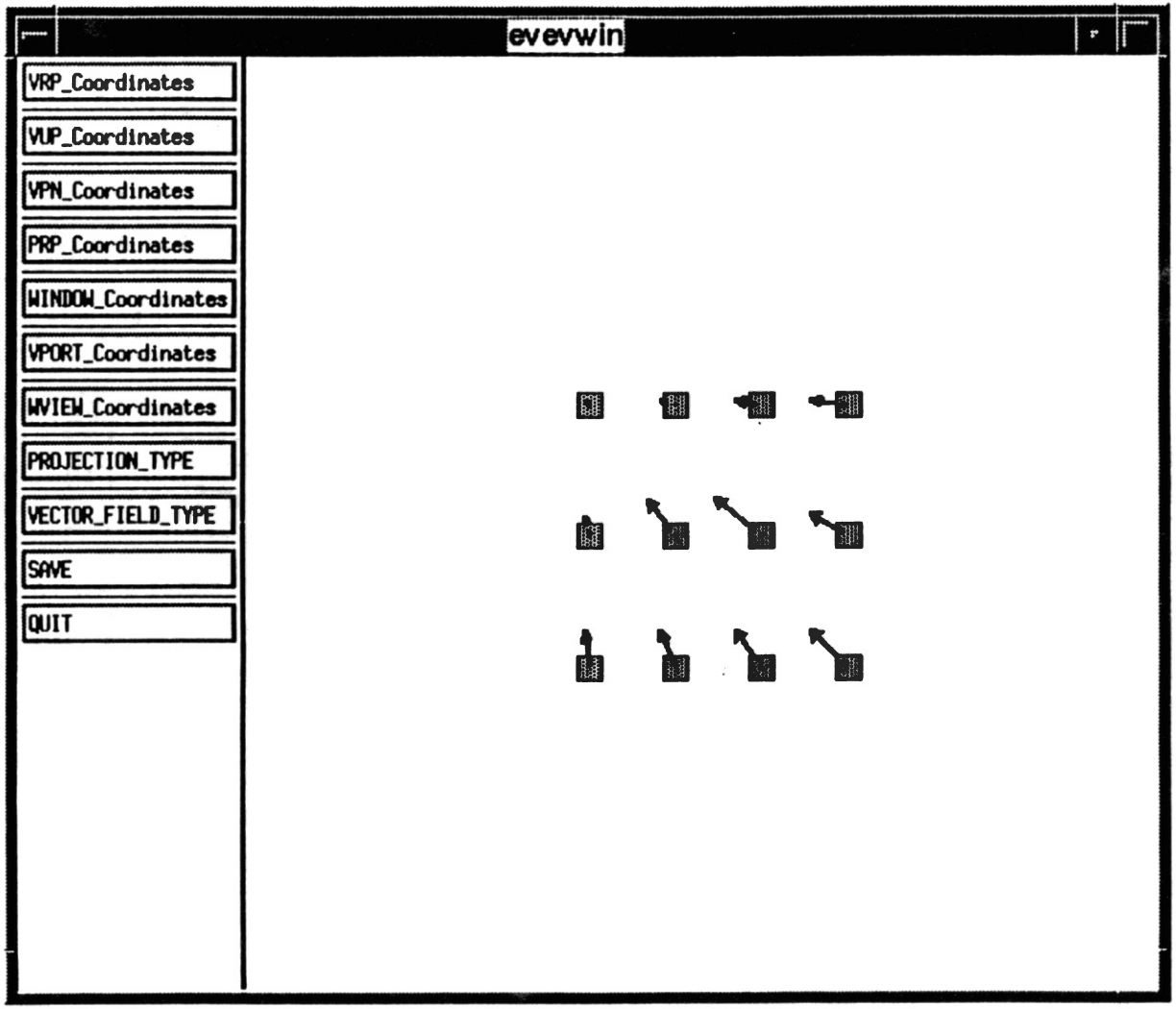

Figure 5. Perspective Projection of a Circular Vector Field

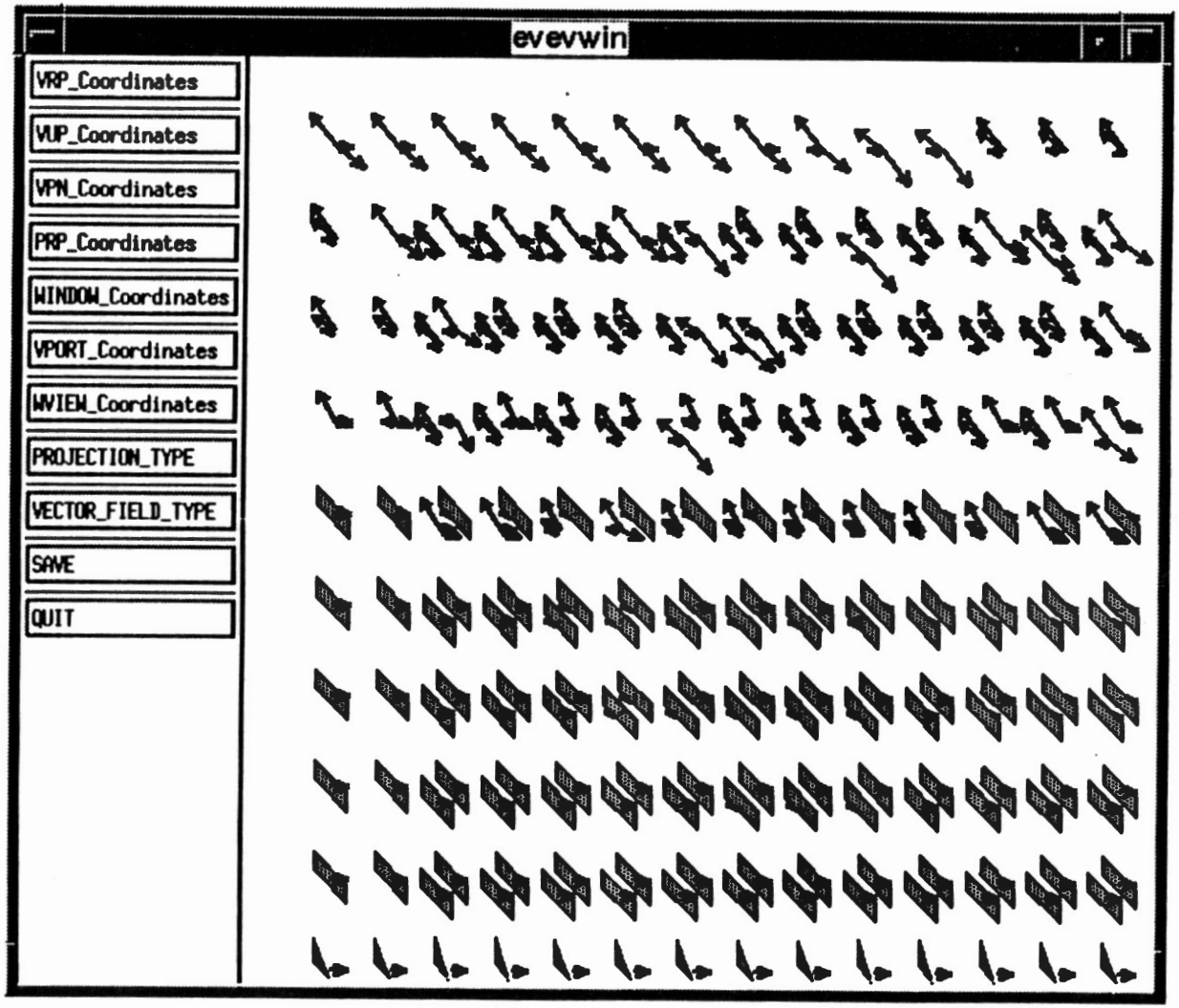

Figure 6. Orthogonal Projection of an Arbitrary Vector Field

typical modification processes are presented in Figures 7 and 8. In Figure 7 the user is presented with a form wherein he can modify each field. In Figure 8, the user is presented with a scrollable list of vector fields to choose from.

Testing Methodology

The following methodology for testing the software was followed:

(1) The C interface to the FORTRAN subroutines of EISPACK was tested by using textbook example matrices as input and comparing the results obtained from the package with that of the textbook. In addition, residual balance for each of the eigenpairs was computed to ensure that they were zero.

(2) The X-Windows and User-Interface part of the package was tested for accuracy.

(3) The graphics part of the package was tested by manual calculation and visual comparison.

The results obtained from these testing proceures proved that the software did not have any obvious bugs.

Limitations of the Software Package

The limitations of this software package are as follows:

(1) Since the terminals used were monochrome, there was obviously a limitation on the perceptual impact of the software package

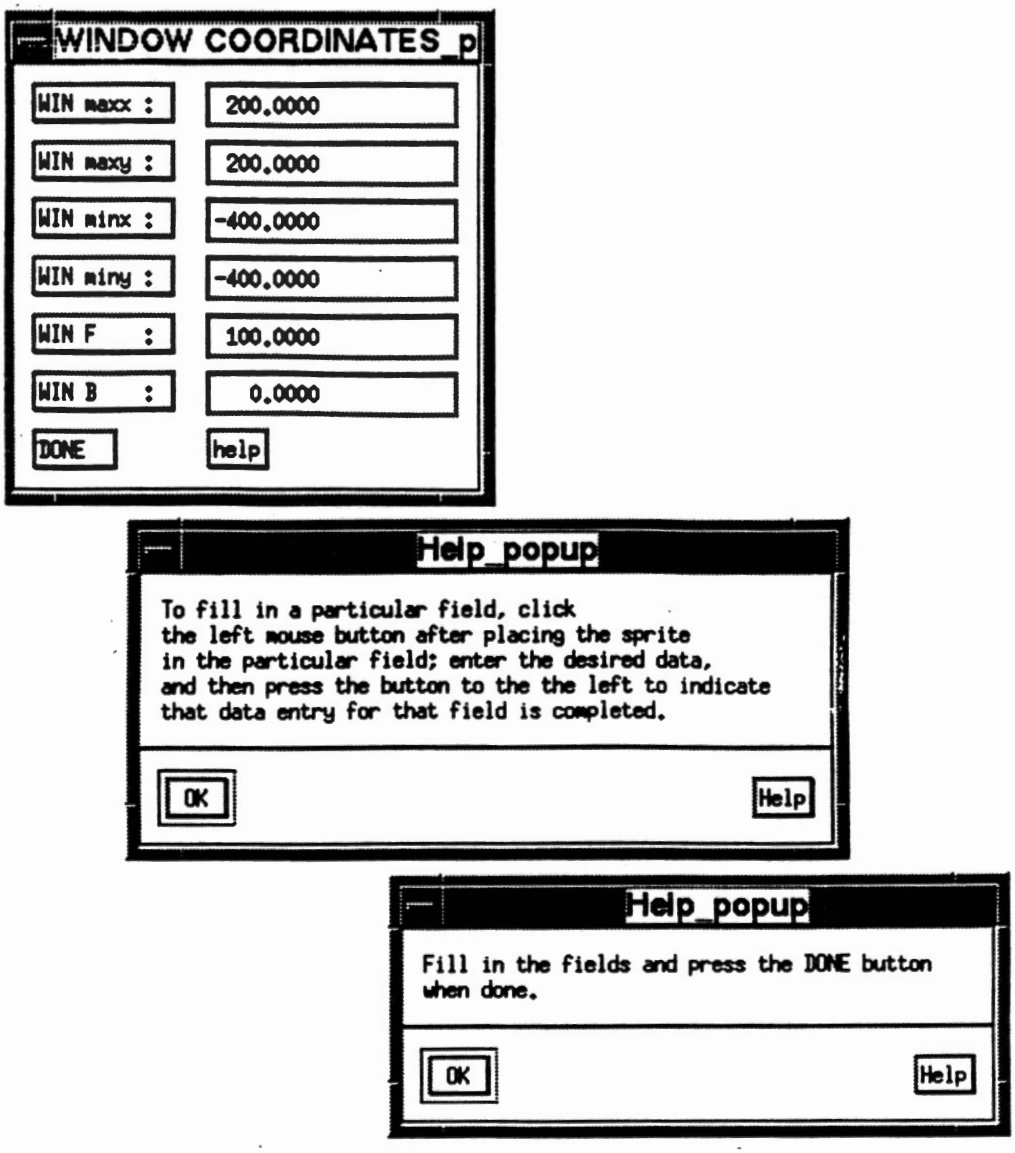

Figure 7. A typical form consisting of modifiable Viewing parameters, along with some Help screens

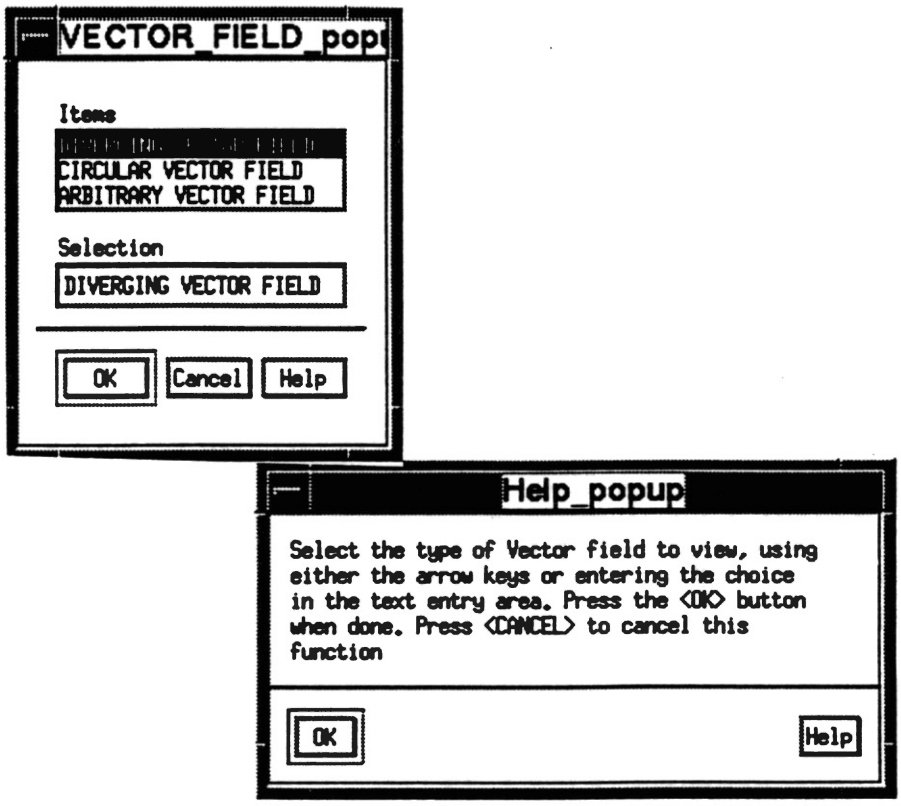

Figure 8. A typical selection list

because of the lack of colour to display the image.

(2) Only real Jacobian matrices are dealt within this software.

(3) Error checking for the user input has not been implemented because of lack of time.

(4) This package deals with three vector fields incorporated for demonstration purposes only. In order to allow the user to input an arbitrary vector field, an interface to a mathematical package like Mathematica would be required.

Summary

The methodology described in Chapter III has been implemented. Preliminary testing shows that the onset of circulation in a vector field is displayed in a very striking manner. Though there are some limitations, this package adequately demonstrates the aim of any scientific visualization tool- to convey information of a scientific phenomenon in an easy but effective manner.

CHAPTER V

FUTURE WORK

Use of animation techniques would greatly enhance the quantity and quality of information transferred to the viewer and hence is suggested to anyone interested in enhancing the software.

This software deals only with 3-D real matrices. It can be extended to deal with any general 3-D matrix, and as such is suggested as future work.

The major limitation of this software is its inability to let a user specify a 3-D vector field using an arbitrary set of equations. Future work might include interfacing this program with a mathematical software package like Mathematica to allow the user to input a set of equations describing a vector field.

Other easy but useful facilities might include the ability to adjust the proportions of the display symbols according to the spacing of the points of evaluation.

BIBLIOGRAPHY

[1] Arnheim, R. , "Visual Thinking", University of California Press, Berkeley, California, 1969.

[2] Asimov, D. , "The Grand Tour: A Tool for Viewing Multidimensional Data", SIAM Journal of Scientific and Statistical Computing, Vol 6, No. 1, 1985, pp. 128-143.

[3] Chernoff, H. , "The Use of Faces to Represent Points in a K-Dimensional Space Graphically", Journal of the American Statistical Association, Vol. 68, No. 342, 1973, pp. 361-368.

[4] Cunningham, S. , Brown, J. R. , and McGrath, M. , "Visualization in Science and Engineering Education", in "Visualization in Scientific Computing", Nielson, G. M. and Shriver, B. , eds. , IEEE Computer Society Press, California, 1990, pp. 48-57.

[5] DeFanti, T. A. , Brown, M. D. , and McCormick, B. H. , "Visualization: Expanding Scientific and Engineering Research Opportunities", Computer, Vol. 22, No 8, Aug. 1989, pp. 48-51.

[6] Foley, J. , Van Dam, A. , Feiner, S. , and Hughes, J. , eds. , "Computer Graphics: Principles and Practice", Addison-Wesley Publishing Company, 1990.

[7] Fox, E. A. , Editor-in-chief, "Resources in Human Computer Interaction", ACM Press, New York, 1990.

[8] Fuchs, H. , Levoy, M. , and Pizer, S. M. , "Interactive Visualization of 3D medical data", Computer, Vol. 22, No. 8, Aug. 1989, pp. 46-51.

[9] Golub, G. H. and Van Loan, C. F. , "Matrix Computation", The Johns Hopkins University Press, 1991.

[10] Haber, R. B. and McNabb, D. A. , "Visualization Idioms: A conceptual Model for Scientific Visualizations Systems", Computer, Vol. 22, No. 8, Aug. 1989, pp. 46-51.

[11] Haber, R. N. and Wilkinson, L. , "Perceptual Components of Computer Displays", IEEE Computer Graphics and Applications, Vol. 2, No. 3, May 982, pp. 23-35.

[12] Hager, W. W. , "Applied Numerical Linear Algebra", Prentice Hall, Englewood Cliffs, N. Jersey, 1988.

[13] Hamming R. W. "Numerical Methods for Scientists and Engineers", McGraw-Hill, Newyork, 1962.

[14] Helman, J. and Hesselink, L. , "Representation and Display of Vector Field Topology in Fluid Data Sets", Computer, Vol. 22, No. 8, Aug. 1989, pp. 27-36.

[15] Kreyszig, E. , "Advanced Engineering Mathematics", Wiley Eastern Ltd. , 1989.

[16] McCormick, B. H. , Defanti, T. A. , and Brown, M. D. , eds. , "Visualization in Scientific Computing", Computer Graphics, Vol. 21, No 6, Nov. 1987.

[17] Nielson, G. M., and Shriver B. , eds. , "Visualization in Scientific Computing", IEEE Computer Society Press, Los Alamitos, California, 1990.

[18] Nye, A. , "Definition Guide to X-Window Systems-XLIB Programming Manual", O'Reilly and Associates, Inc., Nov. 1988.

[19] Phillips, R. L. , "Distributed Visualization at Los Alamos National Laboratory", Computer, Vol 22, No. 8, Aug. 1989, pp. 70-77.

[20] Rosenblum, L. J. , "Visualization Oceanographic Data" IEEE Computer Graphics and Applications, Vol. 9. , No. 3, May 1989, pp. 14-19.

[21] Scheifler, R. W. and Gettys, J. , "The X-Windows System", ACM Transaction Graph, 5, 2, April 1986, pp. 79-109.

[22] Schidlt, H. , "C: The Complete Reference", Osborne McGraw-Hill, Berkeley, California, 1987.

[23] Semmler, K. D., Buser, P., and Radic, B., "Surface Visualization", in "Scientific Visualization and Graphics Simulation", Thalmann, D., ed., John Wiley and Sons, Inc., New York, 1990, pp. 27-42.

[24] Young, A. D., "The X Windows System - Programming and Application with Xt, OSF/MOTIF Edition", Prentice Hall, New Jersey, 1990.

APPENDIX

SOURCE CODE OF EVEVWIN

```
/*
 * evevwin    : A Scientific Visualization tool to visualize
 *              3-D vectors in 3-D space using X-Windows.
 * Written by:  Ravi Kumar Gundimeda (MS THESIS)
 * Dated      :  May 18th, 1992
 */

#include "libXs.h"
#include "trans.h"
#include "evevwin.h"


/*********************************************************/
/*  main : This is the driver routine. It initializes the */
/*         FORTRAN library, the Xt Intrinsics, sets up the*/
/*         top-level, the drawing area, and the command   */
/*         widgets, sets up their configuration, creates  */
/*         the data from the defaults, and enters the MAIN*/
/*         event loop .                                   */
/*********************************************************/

main(argc, argv)
int argc;
char **argv;
{
  Widget top_level, sep,canvas,framework,command,button,dialog;
  Image_data data;
  int i;
  Arg wargs[10];
  int n;
  COMB_DATA user_data;

   /* Initialize FORTRAN library for "-unix" i/o */
   _F_init(argc, argv, 0, 0);

  /* Initialize the Intrinsics */
  top_level = XtInitialize(argv[0],"Evevwin", NULL,0,&argc, argv);

  /* Add the string to float type converter */
  XtAddConverter(XtRString, XtRFloat, xs_cvt_str_to_float, NULL, 0);

  /* Get the application resources */
  XtGetApplicationResources(top_level, &data, resources,
                XtNumber(resources), NULL, 0);

  /* Set the defaults for the image */
  set_defaults(&data);

  /* Create a Frame to hold the commands and the drawing area */
  framework = XtCreateManagedWidget("framework", xmFormWidgetClass,
                                    top_level, NULL, 0);

  /* Create a child of framework to contain all commands */
  command = XtCreateManagedWidget("command", xmRowColumnWidgetClass,
                                  framework, NULL, 0);
```



```
  /* Create a canvas for drawing the image */
  canvas = XtCreateManagedWidget("canvas", xmDrawingAreaWidgetClass,
                                 framework, NULL, 0);

   /* Set up a separator between the commands and the drawing area */
   n = 0;
   XtSetArg(wargs[n], XmNorientation, XmVERTICAL); n++;
   XtSetArg(wargs[n], XmNseparatorType, XmSHADOW_ETCHED_IN); n++;
   XtSetArg(wargs[n], XmNshadowThickness, 6); n++;
   sep = XtCreateManagedWidget("Separator", xmSeparatorWidgetClass,
                               framework, NULL, 0);

   /* Add callbacks */
   XtAddCallback(canvas, XmNexposeCallback, redisplay, &data);
   XtAddCallback(canvas, XmNresizeCallback, resize, &data);

   /* Set up configuration of frame */
   n = 0;
   XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_FORM); n++;
   XtSetValues(command, wargs, n);
   n = 0;
   XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n], XmNleftWidget, command);             n++;
   XtSetValues(sep, wargs, n);
   n = 0;
   XtSetArg(wargs[n], XmNtopAttachment, XmATTACH_FORM);    n++;
   XtSetArg(wargs[n], XmNbottomAttachment, XmATTACH_FORM); n++;
   XtSetArg(wargs[n], XmNleftAttachment, XmATTACH_WIDGET); n++;
   XtSetArg(wargs[n], XmNleftWidget, sep);                 n++;
   XtSetArg(wargs[n], XmNrightAttachment, XmATTACH_FORM); n++;
   XtSetValues(canvas, wargs, n);

   /* Initialize the drawing data */
   init_data(canvas, &data);

   /* Create data for drawing the image using defaults */
   if(pparms.change_flag) {
        create_data(&data);
        pparms.change_flag = FALSE;
        }
    user_data.data = &data;
    user_data.canvas = canvas;

  /* Create the drawing commands */
  create_command_panel(command,&user_data);


  /* Realize the top-level widget */
  XtRealizeWidget(top_level);
  resize(canvas, &data, NULL);
```

```
   /* Enter the main event loop */
   XtMainLoop();

   /* Exit the system */
   EXIT(0);
 }


/*********************************************************/
/** create_command_panel : This function sets up all the */
/**                        required buttons for changing */
/**                        the viewing parameters, saving*/
/**                        the viewing options or to exit*/
/*********************************************************/
void create_command_panel(command, user_data)
Widget command;
COMB_DATA *user_data;
{
   Widget vpn_button,vpn_dialog;
   Widget vrp_button,vrp_dialog;
   Widget vup_button,vup_dialog;
   Widget prp_button,prp_dialog;
   Widget window_button,window_dialog;
   Widget vport_button,vport_dialog;
   Widget wview_button,wview_dialog;
   Widget prtype_button,prtype_dialog;
   Widget setno_button,setno_dialog;
   Widget save_button;
   Arg wargs[2];
   int i;


   vrp_button = XtCreateManagedWidget("VRP_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   vrp_dialog = vrp_create_dialog(vrp_button,user_data);
   XtAddCallback(vrp_button, XmNactivateCallback,
                 show_dialog, vrp_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   vup_button = XtCreateManagedWidget("VUP_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   vup_dialog = vup_create_dialog(vup_button,user_data);
   XtAddCallback(vup_button, XmNactivateCallback,
                 show_dialog, vup_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);


   vpn_button = XtCreateManagedWidget("VPN_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   vpn_dialog = vpn_create_dialog(vpn_button,user_data);
   XtAddCallback(vpn_button, XmNactivateCallback,
                 show_dialog, vpn_dialog);
   XtCreateManagedWidget("separator",
```



```
                xmSeparatorWidgetClass, command, NULL, 0);

   prp_button = XtCreateManagedWidget("PRP_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   prp_dialog = prp_create_dialog(prp_button,user_data);
   XtAddCallback(prp_button, XmNactivateCallback,
                 show_dialog, prp_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   window_button = XtCreateManagedWidget("WINDOW_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   window_dialog = window_create_dialog(window_button,user_data);
   XtAddCallback(window_button, XmNactivateCallback,
                 show_dialog, window_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   vport_button = XtCreateManagedWidget("VPORT_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   vport_dialog = vport_create_dialog(vport_button,user_data);
   XtAddCallback(vport_button, XmNactivateCallback,
                 show_dialog, vport_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   wview_button = XtCreateManagedWidget("WVIEW_Coordinates",
                xmPushButtonWidgetClass, command, NULL, 0);
   wview_dialog = wview_create_dialog(wview_button,user_data);
   XtAddCallback(wview_button, XmNactivateCallback,
                 show_dialog, wview_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   prtype_button = XtCreateManagedWidget("PROJECTION_TYPE",
                xmPushButtonWidgetClass, command, NULL, 0);
   prtype_dialog = prtype_create_dialog(prtype_button,user_data);
   XtAddCallback(prtype_button, XmNactivateCallback,
                 show_dialog, prtype_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   setno_button = XtCreateManagedWidget("VECTOR_FIELD_TYPE",
                xmPushButtonWidgetClass, command, NULL, 0);
   setno_dialog = setno_create_dialog(setno_button,user_data);
   XtAddCallback(setno_button, XmNactivateCallback,
                 show_dialog, setno_dialog);
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);


   save_button = XtCreateManagedWidget("SAVE",
                xmPushButtonWidgetClass, command, NULL, 0);
   XtAddCallback(save_button, XmNactivateCallback,
                 save_parms, NULL);
```

```
   XtCreateManagedWidget("separator",
                xmSeparatorWidgetClass, command, NULL, 0);

   /* Create the quit button */
   xs_create_quit_button(command);
}
/*********************************************************/
/**save_parms:  This function s called when the user     */
/**             presses the save button. All it does is  */
/**             to write to a output file "vparm.out",   */
/**             all the relevant viewing parameters      */
/*********************************************************/

static void save_parms(w, client_data, call_data)
Widget w;
caddr_t *client_data;
XmAnyCallbackStruct *call_data;
{
   FILE *fp;


   fp = fopen("vparm.out","w");
   if(fp == NULL) {
      printf("Error opening output file\n");
      exit(1);
      }

   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.vrp.x,pparms.vrp.y,
         pparms.vrp.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.vup.x,pparms.vup.y,
         pparms.vup.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.vpn.x,pparms.vpn.y,
         pparms.vpn.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.prp.x,pparms.prp.y,
         pparms.prp.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf %9.4lf %9.4lf %9.4lf\n",
         pparms.window.wmaxx,pparms.window.wmaxy,
         pparms.window.wminx,pparms.window.wminy,pparms.window.Bmax,
         pparms.window.Fmin);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf %9.4lf %9.4lf %9.4lf\n",
         pparms.vport_max.x,pparms.vport_max.y,
         pparms.vport_max.z,pparms.vport_min.x,pparms.vport_min.y,
         pparms.vport_min.z );
   fprintf(fp,"%9.4lf %9.4lf %9.4lf %9.4lf %9.4lf %9.4lf\n",
         pparms.vport_max.x,pparms.vport_max.y,
         pparms.vport_max.z,pparms.vport_min.x,pparms.vport_min.y,
         pparms.vport_min.z );
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.wview_max.x,pparms.wview_max.y,
         pparms.wview_max.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.wview_min.x,pparms.wview_min.y,
         pparms.wview_min.z);
   fprintf(fp,"%9.4lf %9.4lf %9.4lf\n",pparms.wview_delta.x,pparms.wview_delta.y,
         pparms.wview_delta.z);
   fprintf(fp,"%2d\n",pparms.type_proj);
   fprintf(fp,"%2d\n",pparms.setno);
   fclose(fp);
```



```
}
/*********************************************************/
/**create_rect_plate: This creates a rectangular plate. **/
/*********************************************************/

void create_rect_plate(xx,yy,zz)
double xx[],yy[],zz[];
{

   xx[0] = -MULTFACT*0.5;
   yy[0] = MULTFACT*0.5;
   zz[0] = 0;
   xx[1] = MULTFACT*0.5;
   yy[1] = MULTFACT*0.5;
   zz[1] = 0;
   xx[2] = MULTFACT*0.5;
   yy[2] = -MULTFACT*0.5;
   zz[2] = 0;
   xx[3] = -MULTFACT*0.5;
   yy[3] = -MULTFACT*0.5;
   zz[3] = 0;
}

/*********************************************************/
/**initalize_arrays: Initializes all arrays used         */
/*********************************************************/
initialize_arrays(x,y,z,xorg,yorg,zorg)
double x[][MAXPTS],y[][MAXPTS],z[][MAXPTS],xorg[],yorg[],zorg[];
{
   int i,j;

   for(i=0; i < NUMPTSONSCREEN; i++){
    for(j=0; j < MAXPTS; j++) {
       x[i][j] = 0.0;
       y[i][j] = 0.0;
       z[i][j] = 0.0;
       }
    xorg[i] = 0.0;
    yorg[i] = 0.0;
    zorg[i] = 0.0;
    }

}
/*********************************************************/
/**fix_origins: This function fixes the origins of all   */
/**             the points in the vector field, where the*/
/**             Jacobian would be evaluated.             */
/*********************************************************/
fix_origins(xorg,yorg,zorg,data3,npts)
PROJECTION *data3;
double xorg[],yorg[],zorg[];
int *npts;
{
   int i,j,k,flag;
   double dx,dy,dz,maxx,maxy,maxz,minx,miny,minz;
   double xcount, ycount,zcount;
```

```
    minx = data3->wview_min.x;
    miny = data3->wview_min.y;
    minz = data3->wview_min.z;
    maxx = data3->wview_max.x;
    maxy = data3->wview_max.y;
    maxz = data3->wview_max.z;
    dx = data3->wview_delta.x;
    dy = data3->wview_delta.y;
    dz = data3->wview_delta.z;


     for(i=0, zcount = maxz; (zcount >= minz) && (i < NUMPTSONSCREEN);
                                                    zcount -= dz) {
       for(ycount = miny; (ycount <= maxy) && (i < NUMPTSONSCREEN);
                                                    ycount += dy) {
          for(xcount = minx; (xcount <= maxx) && (i < NUMPTSONSCREEN);
                                                    xcount += dx,i++) {
                xorg[i] = xcount;
                yorg[i] = ycount;
                zorg[i] = zcount;
                }
          }
       }
    *npts = i;
}
/*********************************************************/
/**create_data: This is the function that does the job of*/
/**             creating the image from the user-choices */
/**             of viewing parameters.                   */
/*********************************************************/
create_data(data)
image_data *data;
{

    int        i,j;
    PROJECTION data3;
    double     point[MROWS],ppoint[MROWS],M[MROWS][MCOLS],M1[MROWS][MCOLS];
    double     M2[MROWS][MCOLS],x[NUMPTSONSCREEN][MAXPTS];
    double     y[NUMPTSONSCREEN][MAXPTS],z[NUMPTSONSCREEN][MAXPTS];
    double     xx[MAXPTS],yy[MAXPTS],zz[MAXPTS],zmin,D1,thx,thy;
    double     xorg[NUMPTSONSCREEN],yorg[NUMPTSONSCREEN],zorg[NUMPTSONSCREEN];
    COMPOSITION data2;
    int        sx,sy,sz,dx,dy,dz,npts = 0,setn = 0;


  initialize_arrays(x,y,z,xorg,yorg,zorg);
  copy_projection(&data3);
  fix_origins(xorg,yorg,zorg,&data3,&npts);
  data->pattern = "2S_foreground";
  data->numptsonscreen = npts;
  setn = data3.setno;
  i=0;
  while (i < npts) {
       data->numpts[i] = MAX_ROWS;
       get_coordinates_of_vectors(x[i],y[i],z[i], &(data->cplx_flag[i]),
```



```
                                        setn, xorg[i],yorg[i],zorg[i]);
       if(data->cplx_flag[i] == TRUE) {
          data->numpts[i] = 1;
          if(z[i][0] > 0.0) data->flag[i] = TRUE;
          thy = PI/2.0 -(atan2 ( z[i][0],x[i][0])) ;
          D1  = calc_hyp(z[i][0], x[i][0]);
          thx = - atan2 ( y[i][0], D1);
          fill_compose_data(&data2,0,0,0,0,0,0,0,thx,thy,0.0);
          create_rect_plate(xx,yy,zz);
          compose_3d_point(xx,yy,zz,4,&data2);
          for(j=0; j < 4; j++) {
            x[i][j+1] = xx[j];
            y[i][j+1] = yy[j];
            z[i][j+1] = zz[j];
            }
          }
        i++;
       }


  create_arrow(&(data->a));
  /* Translate the set of points to the actual origins of the vectors */
  for(j=0; j < data->numptsonscreen; j++) {
     for(i=0; i < data->numpts[j]; i++) {
       x[j][i] += xorg[j];
       y[j][i] += yorg[j];
       z[j][i] += zorg[j];
       }

     if(data->numpts[j] == 1)
       for(i=0; i < 4; i++){
         x[j][i+1] += xorg[j];
         y[j][i+1] += yorg[j];
         z[j][i+1] += zorg[j];
         }
     }

  /* Transform from World coordinates to Viewing Coordinates */
  get_transformation_matrix(M2,&data3);
  get_normalizing_transformation(M1, &data3,&zmin);
  matmul4d(M1,M2,M);
  canonical_transformation(x,y,z,xorg,yorg,zorg,M,data);
  clip_in_3d(&data3,data,x,y,z,xorg,yorg,zorg,zmin);
  construct_mgeneral(M2,&data3);
  map_to_viewport(&data3,M1);
  matmul4d(M1,M2,M);

  for(j=0; j < data->numptsonscreen; j++) {
     for(i=0; i < data->numpts[j]; i++)
       project_to_2d(M,x[j][i],y[j][i], z[j][i],
                        &(data->x[j][i]),&(data->y[j][i]));
     if(data->numpts[j] == 1)
       for(i=0; i < 4; i++)
         project_to_2d(M,x[j][i+1],y[j][i+1], z[j][i+1],
                          &(data->x[j][i+1]),&(data->y[j][i+1]));
     project_to_2d(M,xorg[j],yorg[j],
```

```
                    zorg[j],&(data->x_origin[j]),&(data->y_origin[j]));
      }
}

/*********************************************************/
/**canonical_transformation: Transforms image to fit     */
/**                         canonical view volume.      */
/*********************************************************/
canonical_transformation(x,y,z,xorg,yorg,zorg,M,data)
double x[][MAXPTS],y[][MAXPTS],z[][MAXPTS],xorg[],yorg[],zorg[],
       M[][MCOLS];
image_data *data;
{
  int i,j;

  for(j=0; j < data->numptsonscreen; j++) {
     for(i=0; i < data->numpts[j]; i++)
       apply_transformation(M,x[j][i],y[j][i], z[j][i],
                    &(x[j][i]),&(y[j][i]),&(z[j][i]));
     if(data->numpts[j] == 1)
       for(i=0; i < 4; i++)
         apply_transformation(M,x[j][i+1],y[j][i+1], z[j][i+1],
                     &(x[j][i+1]),&(y[j][i+1]),&(z[j][i+1]));
     apply_transformation(M,xorg[j],yorg[j],
                    zorg[j],&(xorg[j]),&(yorg[j]),&(zorg[j]));
     }
}
/*********************************************************/
void project_to_2d(M,x,y,z,X,Y)
double x,y,z;
double *X,*Y;
double M[][MCOLS];
{
       double point[MROWS],ppoint[MROWS];

       point[0] =  x;
       point[1] =  y;
       point[2] =  z;
       point[3] = 1.0;
       project_point(M,point,ppoint);
       *X = (ppoint[0]/ppoint[3]);
       *Y = (ppoint[1]/ppoint[3]);
}
/*********************************************************/
void apply_transformation(M,x,y,z,X,Y,Z)
double x,y,z,*X,*Y,*Z;
double M[][MCOLS];
{
       double point[MROWS],ppoint[MROWS];

       point[0] = x;
       point[1] = y;
       point[2] = z;
       point[3] = 1.0;
       matmul_4d(point,M,ppoint);
       *X = ppoint[0]/ppoint[3];
```



```
       *Y = ppoint[1]/ppoint[3];
       *Z = ppoint[2]/ppoint[3];
}
/*********************************************************/
void copy_projection(data3)
PROJECTION *data3;
{
  data3->type_proj = pparms.type_proj;
  data3->setno     = pparms.setno;
  data3->alpha     = pparms.alpha;
  data3->d         = pparms.d;
  /* In world coordinates */
  data3->vrp.x     = pparms.vrp.x;
  data3->vrp.y     = pparms.vrp.y;
  data3->vrp.z     = pparms.vrp.z;
  /* In view reference coordinates */
  data3->prp.x     = pparms.prp.x;
  data3->prp.y     = pparms.prp.y;
  data3->prp.z     = pparms.prp.z;
  /* In world coordinates */
  data3->vup.x     = pparms.vup.x;
  data3->vup.y     = pparms.vup.y;
  data3->vup.z     = pparms.vup.z;
  /* In world coordinates */
  data3->vpn.x     = pparms.vpn.x;
  data3->vpn.y     = pparms.vpn.y;
  data3->vpn.z     = pparms.vpn.z;
  /* In view reference coordinates */
  data3->window.wmaxx = pparms.window.wmaxx;
  data3->window.wmaxy = pparms.window.wmaxy;
  data3->window.wminx = pparms.window.wminx;
  data3->window.wminy = pparms.window.wminx;
  data3->window.Bmax = pparms.window.Bmax;
  data3->window.Fmin = pparms.window.Fmin;
  /* In view reference coordinates */
  data3->vport_max.x = pparms.vport_max.x;
  data3->vport_max.y = pparms.vport_max.y;
  data3->vport_max.z = pparms.vport_max.x;
  data3->vport_min.x = pparms.vport_min.x;
  data3->vport_min.y = pparms.vport_min.y;
  data3->vport_min.z = pparms.vport_min.x;
  /* In world coordinates */
  data3->wview_max.x = pparms.wview_max.x;
  data3->wview_max.y = pparms.wview_max.y;
  data3->wview_max.z = pparms.wview_max.z;
  data3->wview_min.x = pparms.wview_min.x;
  data3->wview_min.y = pparms.wview_min.y;
  data3->wview_min.z = pparms.wview_min.z;
  data3->wview_delta.x = pparms.wview_delta.x;
  data3->wview_delta.y = pparms.wview_delta.y;
  data3->wview_delta.z = pparms.wview_delta.z;
}

/*********************************************************/
set_defaults(data)
image_data *data;
```

```
{
  FILE *fp;

  pparms.change_flag = TRUE;
  pparms.type_proj = MPPER;
  pparms.setno =  DIVERGING;
  pparms.alpha     = (double) (PI / 4.00);
  pparms.d         = 1.0;
  fp = fopen("vparm.out","r");
  if(fp != NULL) {

   fscanf(fp,"%lf %lf %lf",&(pparms.vrp.x),&(pparms.vrp.y),
         &(pparms.vrp.z));
   fscanf(fp,"%lf %lf %lf",&(pparms.vup.x),&(pparms.vup.y),
         &(pparms.vup.z));
   fscanf(fp,"%lf %lf %lf",&(pparms.vpn.x),&(pparms.vpn.y),
         &(pparms.vpn.z));
   fscanf(fp,"%lf %lf %lf",&(pparms.prp.x),&(pparms.prp.y),
         &(pparms.prp.z));
   fscanf(fp,"%lf %lf %lf %lf %lf %lf",&(pparms.window.wmaxx),
           &(pparms.window.wmaxy),
         &(pparms.window.wminx),&(pparms.window.wminy),&(pparms.window.Bmax),
         &(pparms.window.Fmin));
   fscanf(fp,"%lf %lf %lf %lf %lf %lf",
         &(pparms.vport_max.x),&(pparms.vport_max.y), &(pparms.vport_max.z),
         &(pparms.vport_min.x),&(pparms.vport_min.y), &(pparms.vport_min.z) );
   fscanf(fp,"%lf %lf %lf %lf %lf %lf",
         &(pparms.vport_max.x),&(pparms.vport_max.y), &(pparms.vport_max.z),
         &(pparms.vport_min.x),&(pparms.vport_min.y), &(pparms.vport_min.z) );
   fscanf(fp,"%lf %lf %lf",&(pparms.wview_max.x),&(pparms.wview_max.y),
         &(pparms.wview_max.z));
   fscanf(fp,"%lf %lf %lf",&(pparms.wview_min.x),&(pparms.wview_min.y),
         &(pparms.wview_min.z));
   fscanf(fp,"%lf %lf %lf",&(pparms.wview_delta.x),&(pparms.wview_delta.y),
         &(pparms.wview_delta.z));
   fscanf(fp,"%d",&(pparms.type_proj));
   fscanf(fp,"%d",&(pparms.setno));
   fclose(fp);
   return(0);
   }

  /* In world coordinates */
  pparms.vrp.x     =  0.0;
  pparms.vrp.y     =  0.0;
  pparms.vrp.z     =  0.0;  /* 1.0 */

  /* In view reference coordinates */
  pparms.prp.x     =  0.5;  /* 0.0;*/
  pparms.prp.y     =  0.5;  /* 0.0;*/
  pparms.prp.z     =  50.0;  /* 150.0;*/

  /* In world coordinates */
  pparms.vup.x     =  0.0;
  pparms.vup.y     =  1.0;
  pparms.vup.z     =  0.0;
```



```
  /* In world coordinates */
  pparms.vpn.x     =  0.0;
  pparms.vpn.y     =  0.0;
  pparms.vpn.z     =  1.0;


  /* In view reference coordinates */
  pparms.window.wmaxx = 200.0;
  pparms.window.wmaxy = 200.0;
  pparms.window.wminx = -400.0;
  pparms.window.wminy = -400.0;
  pparms.window.Bmax = 0.0;       /* B <= F */
  pparms.window.Fmin = 100.0;

  pparms.vport_max.x = 1.0;
  pparms.vport_max.y = 1.0;
  pparms.vport_max.z = 1.0;
  pparms.vport_min.x = 0.0;
  pparms.vport_min.y = 0.0;
  pparms.vport_min.z = 0.0;

  pparms.wview_max.x = 400.0;
  pparms.wview_max.y = 400.0;
  pparms.wview_max.z =  20.0;
  pparms.wview_min.x = -200.0;
  pparms.wview_min.y = -200.0;
  pparms.wview_min.z = 20.0;
  pparms.wview_delta.x = 40.0;
  pparms.wview_delta.y = 60.0;
  pparms.wview_delta.z = 15.0;
}
/*********************************************************/
init_data(w, data)
Widget w;
image_data *data;
{
  int y,i,j;
  Arg wargs[10];
  XGCValues values;
  char   *pattern = "slant_left";

  XtSetArg(wargs[0], XtNwidth, &data->width);
  XtSetArg(wargs[1], XtNheight, &data->height);
  XtGetValues(w, wargs, 2);
  values.line_style = LineSolid;
  values.line_width = 2;
  XtSetArg(wargs[0],XtNforeground, &values.foreground);
  XtSetArg(wargs[1],XtNbackground, &values.background);
  XtGetValues(w, wargs, 2);
  data->gc = XCreateGC(XtDisplay(w), DefaultRootWindow(XtDisplay(w)),
                       GCLineStyle | GCLineWidth | GCBackground |
                       GCForeground, &values);
  data->pix = NULL;
  for(j=0; j < NUMPTSONSCREEN; j++) {
          for(i=0; i < MAXPTS; i++) {
             data->start_x[j][i] = 0;
```

```
            data->start_y[j][i] = 0;
            data->x[j][I] = 0;
            data->y[j][i] = 0;
            }
          data->flag[j] = FALSE;
          data->draw_symbol[j] = TRUE;
          data->cplx_flag[j] = FALSE;
          data->numpts[j] = 0;
          }
  data->numptsonscreen = 0;
  data->last_x = 0;
  data->last_y = 0;
}
/*********************************************************/
/**create_image : This function draws the image in the   */
/**               appropriate window.                     */
/*********************************************************/
void create_image(w, data)
Widget w;
image_data *data;
{
  int         i,j,mx1,my1,prod,num,denom;
  ARROW       comp_a[4];
  COMPOSITION data1[4];
  double      theta,theta1,slope;
  Dimension dim;

  if(data->height < data->width) dim = data->height;
  else dim = data->width;
  for(j=0; j < data->numptsonscreen; j++) {
   if(data->draw_symbol[j]) {
    scale_it(data->x_origin[j],data->y_origin[j], &(data->last_x),
             &(data->last_y), dim);
    for(i=0; i < data->numpts[j]; i++)
      scale_it(data->x[j][i],data->y[j][i], &(data->start_x[j][i]),
               &(data->start_y[j][i]), dim);
    if(data->numpts[j] == 1)
              for(i=1; i < 5; i++)
                  scale_it(data->x[j][i],data->y[j][i], &(data->start_x[j][i]),
                           &(data->start_y[j][i]), dim);
    /* Creating and composing the arrows for each vector */
    for(i=0; i < data->numpts[j]; i++) {
       denom = data->start_x[j][i] - data->last_x;
       num   = data->start_y[j][i] - data->last_y;
       mx1 = data->start_x[j][i];
       my1 = data->start_y[j][i];
       theta = atan2 ((double)num, (double)denom);
       fill_compose_data(&(data1[i]),mx1,my1,0,0,0,0,theta,0.0,0.0);
       compose(&(data->a),&(data1[i]),&(comp_a[i]));
       }
     XSetForeground(XtDisplay(w), data->gc, BlackPixelOfScreen(XtScreen(w)));
     XSetBackground(XtDisplay(w), data->gc, WhitePixelOfScreen(XtScreen(w)));
    if(XtIsRealized(w)) draw_image_in_window(data,w,comp_a,j);
    draw_image_in_pixmap(data,w,comp_a,j);
    }
  }
```



```
}
/*********************************************************/
void draw_image_in_pixmap(data,w,comp_a,j)
Widget w;
image_data *data;
ARROW comp_a[4];
int j;
{
    int i;

    if((!(data->flag[j])) && (data->numpts[j] == 1)) {
         draw_perp_disk(w, data->pix,1, data->gc,
             data->start_x[j], data->start_y[j]);
         data->tile = XmGetPixmap (XtScreen(w), data->pattern,
                           BlackPixelOfScreen(XtScreen(w)),
                           WhitePixelOfScreen(XtScreen(w)));
         XSetFillStyle(XtDisplay(w), data->gc, FillTiled);
         XSetTile(XtDisplay(w), data->gc, data->tile);
         fill_perp_disk(w, data->pix,1, data->gc,
                        data->start_x[j], data->start_y[j]);
         XSetFillStyle(XtDisplay(w), data->gc, FillSolid);
         }
     for(i=0; i < data->numpts[j]; i++) {
        XDrawLine(XtDisplay(w), data->pix, data->gc,
              data->start_x[j][i], data->start_y[j][i],
              data->last_x, data->last_y);
        draw_arrow(w, data->pix,1, data->gc, &(comp_a[i]));
        }
    if((data->flag[j]) && (data->numpts[j] == 1)) {
         draw_perp_disk(w, data->pix,1, data->gc,
             data->start_x[j], data->start_y[j]);
         data->tile = XmGetPixmap (XtScreen(w), data->pattern,
                           BlackPixelOfScreen(XtScreen(w)),
                           WhitePixelOfScreen(XtScreen(w)));
         XSetFillStyle(XtDisplay(w), data->gc, FillTiled);
         XSetTile(XtDisplay(w), data->gc, data->tile);
         fill_perp_disk(w, data->pix,1, data->gc,
                        data->start_x[j], data->start_y[j]);
         XSetFillStyle(XtDisplay(w), data->gc, FillSolid);
         XSetLineAttributes(XtDisplay(w), data->gc, 2,
                LineOnOffDash, CapButt, JoinMiter);
        XDrawLine(XtDisplay(w), data->pix, data->gc,
              data->start_x[j][0], data->start_y[j][0],
              data->last_x, data->last_y);
         XSetLineAttributes(XtDisplay(w), data->gc, 2,
                LineSolid, CapButt, JoinMiter);
         }
}

/*********************************************************/
void draw_image_in_window(data,w,comp_a,j)
Widget w;
image_data *data;
int j;
ARROW comp_a[4];
```

```
{
      int i;

      if(((!(data->flag[j])) && (data->numpts[j] == 1)) {
          draw_perp_disk(w, data->pix,0, data->gc,
             data->start_x[j], data->start_y[j]);
          data->tile = XmGetPixmap (XtScreen(w), data->pattern,
                    BlackPixelOfScreen(XtScreen(w)),
                    WhitePixelOfScreen(XtScreen(w)));
          XSetFillStyle(XtDisplay(w), data->gc, FillTiled);
          XSetTile(XtDisplay(w), data->gc, data->tile);

          fill_perp_disk(w, data->pix,0, data->gc,
             data->start_x[j], data->start_y[j]);

          XSetFillStyle(XtDisplay(w), data->gc, FillSolid);
          }
      for(i=0; i < data->numpts[j]; i++) {
         XDrawLine(XtDisplay(w), XtWindow(w), data->gc,
              data->start_x[j][i], data->start_y[j][i],
              data->last_x, data->last_y);
        draw_arrow(w, data->pix,0, data->gc, &(comp_a[i]));
         }
      if((data->flag[j]) && (data->numpts[j] == 1)) {
          draw_perp_disk(w, data->pix,0, data->gc,
             data->start_x[j], data->start_y[j]);
          data->tile = XmGetPixmap (XtScreen(w), data->pattern,
                    BlackPixelOfScreen(XtScreen(w)),
                    WhitePixelOfScreen(XtScreen(w)));
          XSetFillStyle(XtDisplay(w), data->gc, FillTiled);
          XSetTile(XtDisplay(w), data->gc, data->tile);

          fill_perp_disk(w, data->pix,0, data->gc,
             data->start_x[j], data->start_y[j]);

          XSetFillStyle(XtDisplay(w), data->gc, FillSolid);
        XSetLineAttributes(XtDisplay(w), data->gc, 2,
              LineOnOffDash, CapButt, JoinMiter);
       XDrawLine(XtDisplay(w), XtWindow(w), data->gc,
              data->start_x[j][0], data->start_y[j][0],
              data->last_x, data->last_y);
        XSetLineAttributes(XtDisplay(w), data->gc, 2,
             LineSolid, CapButt, JoinMiter);
          }
}
/*********************************************************/
get_jacobian(Jacobian,setno,xorg,yorg,zorg)
double xorg,yorg,zorg;
int setno;
double      Jacobian[][MAX_COLS];
{
  init_matrix(Jacobian);
  switch(setno) {
   case DIVERGING : get_diverging_jacobian(Jacobian,xorg,yorg,zorg);
                    break;
   case CIRCULAR  : get_circular_jacobian(Jacobian,xorg,yorg,zorg);
```



```
                    break;
   case OTHER     : get_other_jacobian(Jacobian,xorg,yorg,zorg);
                    break;
   default        : printf("Error in choosing set no\n");
                    exit(1);
   }
}
/*********************************************************/
get_diverging_jacobian(Jacobian,xorg,yorg,zorg)
double xorg,yorg,zorg;
double      Jacobian[][MAX_COLS];
{
    /* The equation set describing the vector is:
       X1 = Alpha * x + 0  * y + 0 * z;
       X2 = 0 * x - Alpha  * y + 0 * z;
       X3 = 0 * x + 0 * y + 0 * z;
       In this case, Alpha has been taken as 5.0
     */

     Jacobian[0][0] = 5.0;
     Jacobian[1][1] = -5.0;
}
/*********************************************************/
get_circular_jacobian(Jacobian,xorg,yorg,zorg)
double xorg,yorg,zorg;
double      Jacobian[][MAX_COLS];
{
    /* The equation set describing the vector is:
       X1 = 0 * x - Alpha  * y + 0 * z;
       X2 = Alpha * x + 0 * y + 0 * z;
       X3 = 0 * x + 0 * y + 0 * z;
       In this case, Alpha has been taken as 5.0
     */

     Jacobian[0][1] = -5.0;
     Jacobian[1][0] = 5.0;
}

/*********************************************************/

get_other_jacobian(Jacobian,xorg,yorg,zorg)
double xorg,yorg,zorg;
double      Jacobian[][MAX_COLS];
{
    /* The equation set describing the vector is:
       f1 = 3 * x^2 + 2 * y + z;
       f2 = 4 * x + 5 * y + 6 * z^2;
       f3 = 8 * x - 9 * Y^2 + 10 * z;
     */

     Jacobian[0][0] =  6.0 * xorg;
     Jacobian[0][1] =  2.0;
     Jacobian[0][2] =  1.0;
     Jacobian[1][0] =  4.0;
     Jacobian[1][1] =  5.0;
     Jacobian[1][2] = 12.0 * zorg;
```

```
     Jacobian[2][0] = 8.0;
     Jacobian[2][1] = -18.0 * yorg;
     Jacobian[2][2] = 10.0 ;
}
/*******************************************************/
get_coordinates_of_vectors(x,y,z,flag,setno,xorg,yorg,zorg)
double xorg,yorg,zorg;
int setno;
double x[],y[],z[];
int *flag;
{
   double Z[MAX_ROWS][MAX_COLS],WR[MAX_ROWS],WI[MAX_ROWS];
   double     Jacobian[MAX_ROWS][MAX_COLS];
   int j,i;

   *flag = FALSE;
   get_jacobian(Jacobian,setno,xorg,yorg,zorg);

   get_evev(WR,WI,Z,Jacobian);
   for(j=0; (j < MAX_COLS) && (*flag == FALSE); j++)
      if(WI[j] != 0.0)
         *flag = TRUE;
    j--;
    if(*flag == TRUE) {
        if(j == 0) {
             normalize_vector(Z,2);
             x[0] =  (MULTFACT* Z[0][2]);
             y[0] =  (MULTFACT* Z[1][2]);
             z[0] =  (MULTFACT* Z[2][2]);
             }
        else if(j == 1){
             normalize_vector(Z,0);
             x[0] =  (MULTFACT* Z[0][0]);
             y[0] =  (MULTFACT* Z[1][0]);
             z[0] =  (MULTFACT* Z[2][0]);
             }
        else {
             printf("Error in get_coordinates_of_vectors\n");
             EXIT(1);
             }
        return(TRUE);
        }


   for(i=0; i < 3; i++)
       normalize_vector(Z,i);

   for(j=0; j < 3; j++) {
       x[j] =  (MULTFACT* Z[0][j]);
       y[j] =  (MULTFACT* Z[1][j]);
       z[j] =  (MULTFACT* Z[2][j]);
       }

   return(TRUE);

}
```



```
/*******************************************************/
normalize_vector(Z,i)
int i;
double Z[][MAX_COLS];
{

   double denom;

   denom = sqre(Z[0][i]) + sqre(Z[1][i]) + sqre(Z[2][i]);
   denom = sqrt(denom);
   Z[0][i] = Z[0][i] / denom;
   Z[1][i] = Z[1][i] / denom;
   Z[2][i] = Z[2][i] / denom;
}
/*******************************************************/
double sqre(a)
double a;
{
  double b = 2.0;
  return(pow(a,b));
}

/*******************************************************/
void draw_arrow(w,pix, flag, gc, comp_a)
Widget w;
Pixmap pix;
GC gc;
ARROW *comp_a;
{
  Display *dpy = XtDisplay(w);
  Window win = XtWindow(w);

  if(!flag) {
     XDrawLine(XtDisplay(w), win, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_base_x,
           comp_a->arrow_base_y);
     XDrawLine(XtDisplay(w), win, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_lwing_x,
           comp_a->arrow_lwing_y);
     XDrawLine(XtDisplay(w), win, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_rwing_x,
           comp_a->arrow_rwing_y);
     }
  else  {
     XDrawLine(XtDisplay(w), pix, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_base_x,
           comp_a->arrow_base_y);
     XDrawLine(XtDisplay(w), pix, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_lwing_x,
           comp_a->arrow_lwing_y);
     XDrawLine(XtDisplay(w), pix, gc, comp_a->arrow_tip_x,
           comp_a->arrow_tip_y, comp_a->arrow_rwing_x,
           comp_a->arrow_rwing_y);
     }
}
/*******************************************************/
```

```
void draw_perp_disk(w,pix, flag, gc, x, y)
Widget w;
Pixmap pix;
GC gc;
int x[],y[];
{
  Display *dpy = XtDisplay(w);
  Window win = XtWindow(w);
  int i,prev_x,prev_y;
  XPoint points[5];

  for(i=1; i < 5; i++) {
      points[i-1].x = x[i];
      points[i-1].y = y[i];
      }
  points[4].x = x[1];
  points[4].y = y[1];

   if(!flag) XDrawLines(dpy,win,gc,points,5,CoordModeOrigin);
   else XDrawLines(dpy,pix,gc,points,5,CoordModeOrigin);
}
/*********************************************************/
void fill_perp_disk(w,pix, flag, gc, x, y)
Widget w;
Pixmap pix;
GC gc;
int x[],y[];
{
  Display *dpy = XtDisplay(w);
  Window win = XtWindow(w);
  int i,prev_x,prev_y;
  XPoint points[5];

  for(i=1; i < 5; i++) {
      points[i-1].x = x[i];
      points[i-1].y = y[i];
      }
  points[4].x = x[1];
  points[4].y = y[1];
  if(!flag) XFillPolygon(dpy,win,gc,points,4,Complex,CoordModeOrigin);
  else XFillPolygon(dpy,pix,gc,points,4,Complex,CoordModeOrigin);
}
/*********************************************************/
void scale_it(x1,y1,scaled_x, scaled_y, win_dim)
int *scaled_x,*scaled_y;
double x1,y1;
Dimension win_dim;
{
   *scaled_x = x1 * ((int) win_dim);
   *scaled_y = y1 * ((int) win_dim);
}
/*********************************************************/
void redisplay(w, data, call_data)
Widget w;
image_data *data;
XmDrawingAreaCallbackStruct *call_data;
```



```
{

   XExposeEvent *event = (XExposeEvent *) call_data->event;
   XCopyArea(XtDisplay(w), data->pix, XtWindow(w), data->gc,
             event->x, event->y, event->width, event->height,
            event->x, event->y);
}
/*********************************************************/
void resize(w, data, call_data)
Widget w;
image_data *data;
caddr_t call_data;
{
   Arg wargs[10];

   XtSetArg(wargs[0], XtNwidth, &data->width);
   XtSetArg(wargs[1], XtNheight, &data->height);
   XtGetValues(w, wargs, 2);
   if(XtIsRealized(w))
      XClearArea(XtDisplay(w), XtWindow(w), 0, 0, 0, 0,
                 TRUE);
   if(data->pix)
      XFreePixmap(XtDisplay(w), data->pix);
   data->pix = XCreatePixmap(XtDisplay(w),
                  DefaultRootWindow(XtDisplay(w)),
                  data->width, data->height,
                  DefaultDepthOfScreen(XtScreen(w)));
   XSetForeground(XtDisplay(w), data->gc,
                  WhitePixelOfScreen(XtScreen(w)));
   XFillRectangle(XtDisplay(w), data->pix, data->gc, 0, 0,
                  data->width, data->height);
   create_image(w, data);
}
/*********************************************************/
int matherr(x)
struct exception *x;
{
  if(x->type != DOMAIN) return(0);
  else  return(1);
}
/*********************************************************/
static void show_dialog(w, dialog, call_data)
Widget w;
Widget dialog;
XmAnyCallbackStruct *call_data;
{
  XtManageChild(dialog);
}
/*********************************************************/
Widget setno_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, selection, button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
```

```
  char buff[80];
  XmString xmstr[4];

  xmstr[0] = XmStringCreate(setno[0], XmSTRING_DEFAULT_CHARSET);
  xmstr[1] = XmStringCreate(setno[1], XmSTRING_DEFAULT_CHARSET);
  xmstr[2] = XmStringCreate(setno[2], XmSTRING_DEFAULT_CHARSET);
  xmstr[3] = NULL;
  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"VECTOR_FIELD",wargs,n);
  user_data->wsetno = bb;
  n = 0;
  XtSetArg(wargs[n], XmNlistItems, xmstr); n++;
  XtSetArg(wargs[n], XmNlistItemCount, 3); n++;
  XtSetArg(wargs[n], XmNlistVisibleItemCount, 3); n++;
  selection = XtCreateWidget("VECTOR_FIELD",xmSelectionBoxWidgetClass,
            bb,wargs,n);
  XtUnmanageChild(XmSelectionBoxGetChild(selection,XmDIALOG_APPLY_BUTTON));
  XtAddCallback(selection, XmNokCallback, get_setno, user_data);
  XtAddCallback(selection, XmNcancelCallback, remove_setno, user_data);
  XtAddCallback(selection,XmNhelpCallback,
                xs_help_callback, help_str2);
  XtManageChild(selection);
  return(bb);
}
/*********************************************************/
void remove_setno(w, user_data, call_data)
Widget w;
COMB_DATA *user_data;
XmSelectionBoxCallbackStruct *call_data;
{
  XtUnmanageChild(user_data->wsetno);
}
/*********************************************************/
Widget prtype_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, selection, button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];
  XmString xmstr[3];

  xmstr[0] = XmStringCreate(prtype[0], XmSTRING_DEFAULT_CHARSET);
  xmstr[1] = XmStringCreate(prtype[1], XmSTRING_DEFAULT_CHARSET);
  xmstr[2] = NULL;
  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"PROJECTION",wargs,n);
  user_data->wprtype = bb;
  n = 0;
  XtSetArg(wargs[n], XmNlistItems, xmstr); n++;
  XtSetArg(wargs[n], XmNlistItemCount, 2); n++;
  XtSetArg(wargs[n], XmNlistVisibleItemCount, 2); n++;
  selection = XtCreateWidget("PROJECTION",xmSelectionBoxWidgetClass,
            bb,wargs,n);
  XtUnmanageChild(XmSelectionBoxGetChild(selection,XmDIALOG_APPLY_BUTTON));
```



```
  XtAddCallback(selection, XmNokCallback, get_prtype, user_data);
  XtAddCallback(selection, XmNcancelCallback, remove_prtype, user_data);
  XtAddCallback(selection,XmNhelpCallback, xs_help_callback, help_str1);
  XtManageChild(selection);
  return(bb);
}
/*********************************************************/
void remove_prtype(w, user_data, call_data)
Widget w;
COMB_DATA *user_data;
XmSelectionBoxCallbackStruct *call_data;
{
  XtUnmanageChild(user_data->wprtype);
}
/*********************************************************/
Widget vpn_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[3], button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"VPN COORDINATES",wargs,n);
  user_data->wvpn = bb;
  for(i=0; i < XtNumber(editors); i++) {
    edit[i] = XtCreateWidget(editors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.vpn.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.vpn.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.vpn.z); break;
       default: printf("Error somewhere \n"); exit(1);
       }
    XmTextSetString(edit[i],buff);
    }
  for(i=0; i < XtNumber(buttons); i++){
    button[i]=XtCreateWidget(buttons[i],xmPushButtonWidgetClass,bb,NULL,0);
     switch(i) {
       case 0: XtAddCallback(button[i],XmNactivateCallback,get_vpnx,edit[i]);
               break;
       case 1: XtAddCallback(button[i],XmNactivateCallback,get_vpny,edit[i]);
               break;
       case 2: XtAddCallback(button[i],XmNactivateCallback,get_vpnz,edit[i]);
               break;
       default: printf("Error somewhere \n");
                exit(1);
       }
    }
  done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(done_button,XmNactivateCallback, done_callback, user_data);
  /* Add a help button */
  help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(help_button,XmNactivateCallback,xs_help_callback,help_str);
  XtManageChildren(edit, XtNumber(editors));
```

```
    XtManageChildren(button, XtNumber(buttons));
    return(bb);
}
/*********************************************************/
Widget vup_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[3], button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"VUP COORDINATES",wargs,n);
  user_data->wvup = bb;
  for(i=0; i < XtNumber(editors); i++) {
    edit[i] = XtCreateWidget(editors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.vup.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.vup.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.vup.z); break;
       default: printf("Error somewhere \n"); exit(1);
       }
    XmTextSetString(edit[i],buff);
    }
  for(i=0; i < XtNumber(buttons); i++){
     button[i]=XtCreateWidget(buttons[i],xmPushButtonWidgetClass,bb,NULL,0);
     switch(i) {
       case 0: XtAddCallback(button[i],XmNactivateCallback,get_vupx, edit[i]);
               break;
       case 1: XtAddCallback(button[i], XmNactivateCallback,get_vupy,edit[i]);
               break;
       case 2: XtAddCallback(button[i], XmNactivateCallback,get_vupz,edit[i]);
               break;
       default: printf("Error somewhere \n"); exit(1);
       }
    }
  done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(done_button,XmNactivateCallback,done_callback,user_data);

  /* Add a help button */
  help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(help_button,XmNactivateCallback,xs_help_callback,help_str);
  XtManageChildren(edit, XtNumber(editors));
  XtManageChildren(button, XtNumber(buttons));
  return(bb);
}
/*********************************************************/
Widget vrp_create_dialog(parent,user_data)
COMB_DATA  *user_data;
Widget parent;
{
  Widget bb, edit[3], button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
```



```
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"VRP COORDINATES",wargs,n);
  user_data->wvrp = bb;
  for(i=0; i < XtNumber(editors); i++) {
    edit[i] = XtCreateWidget(editors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.vrp.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.vrp.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.vrp.z); break;
       default: printf("Error somewhere \n"); exit(1);
       }
    XmTextSetString(edit[i],buff);
    }
  for(i=0; i < XtNumber(buttons); i++){
     button[i]=XtCreateWidget(buttons[i],xmPushButtonWidgetClass,bb,NULL,0);
     switch(i) {
       case 0: XtAddCallback(button[i],XmNactivateCallback,get_vrpx,edit[i]);
               break;
       case 1: XtAddCallback(button[i],XmNactivateCallback,get_vrpy,edit[i]);
               break;
       case 2: XtAddCallback(button[i],XmNactivateCallback,get_vrpz,edit[i]);
               break;
       default: printf("Error somewhere \n"); exit(1);
       }
    }
  done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(done_button,XmNactivateCallback,done_callback,user_data);
  /* Add a help button */
  help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(help_button,XmNactivateCallback,xs_help_callback,help_str);
  XtManageChildren(edit, XtNumber(editors));
  XtManageChildren(button, XtNumber(buttons));
  return(bb);
}
/*********************************************************/
Widget prp_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[3], button[3], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"PRP COORDINATES",wargs,n);
  user_data->wprp = bb;
  for(i=0; i < XtNumber(editors); i++) {
    edit[i] = XtCreateWidget(editors[i], xmTextWidgetClass,
                             bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.prp.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.prp.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.prp.z); break;
```

```
        default: printf("Error somewhere \n"); exit(1);
        }
      XmTextSetString(edit[i],buff);
      }
   for(i=0; i < XtNumber(buttons); i++){
      button[i] = XtCreateWidget(buttons[i],xmPushButtonWidgetClass,
                                 bb, NULL, 0);
      switch(i) {
        case 0: XtAddCallback(button[i], XmNactivateCallback,
                  get_prpx, edit[i]); break;
        case 1: XtAddCallback(button[i], XmNactivateCallback,
                  get_prpy, edit[i]); break;
        case 2: XtAddCallback(button[i], XmNactivateCallback,
                  get_prpz, edit[i]); break;
        default: printf("Error somewhere \n"); exit(1);
        }
      }
    done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
    XtAddCallback(done_button,XmNactivateCallback, done_callback, user_data);
    /* Add a help button */
    help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
    XtAddCallback(help_button,XmNactivateCallback, xs_help_callback, help_str);
    XtManageChildren(edit, XtNumber(editors));
    XtManageChildren(button, XtNumber(buttons));
    return(bb);
}
/*********************************************************/
Widget window_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[6], button[6], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"WINDOW COORDINATES",wargs,n);
  user_data->wwindow = bb;
  for(i=0; i < XtNumber(weditors); i++) {
    edit[i] = XtCreateWidget(weditors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.window.wmaxx); break;
       case 1: sprintf(buff,"%9.4f",pparms.window.wmaxy); break;
       case 2: sprintf(buff,"%9.4f",pparms.window.wminx); break;
       case 3: sprintf(buff,"%9.4f",pparms.window.wminy); break;
       case 4: sprintf(buff,"%9.4f",pparms.window.Fmin); break;
       case 5: sprintf(buff,"%9.4f",pparms.window.Bmax); break;
       default: printf("Error somewhere \n"); exit(1);
       }
     XmTextSetString(edit[i],buff);
     }
  for(i=0; i < XtNumber(wbuttons); i++){
     button[i]=XtCreateWidget(wbuttons[i],xmPushButtonWidgetClass,bb,NULL,0);
     switch(i) {
       case 0: XtAddCallback(button[i], XmNactivateCallback,
```



```
                 get_window_maxx, edit[i]); break;
       case 1: XtAddCallback(button[i], XmNactivateCallback,
                 get_window_maxy, edit[i]); break;
       case 2: XtAddCallback(button[i], XmNactivateCallback,
                 get_window_minx, edit[i]); break;
       case 3: XtAddCallback(button[i], XmNactivateCallback,
                 get_window_miny, edit[i]); break;
       case 4: XtAddCallback(button[i], XmNactivateCallback,
                 get_window_F, edit[i]); break;
       case 5: XtAddCallback(button[i], XmNactivateCallback,
                 get_window_B, edit[i]); break;
       default: printf("Error somewhere \n"); exit(1);
       }
     }
   done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
   XtAddCallback(done_button,XmNactivateCallback, done_callback, user_data);
   /* Add a help button */
   help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
   XtAddCallback(help_button,XmNactivateCallback, xs_help_callback, help_str);
   XtManageChildren(edit, XtNumber(weditors));
   XtManageChildren(button, XtNumber(wbuttons));
   return(bb);
}
/*********************************************************/
Widget vport_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[6], button[6], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"VIEW PORT COORDINATES",wargs,n);
  user_data->wvport = bb;
  for(i=0; i < XtNumber(weditors); i++) {
    edit[i] = XtCreateWidget(weditors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.vport_max.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.vport_max.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.vport_max.z); break;
       case 3: sprintf(buff,"%9.4f",pparms.vport_min.x); break;
       case 4: sprintf(buff,"%9.4f",pparms.vport_min.y); break;
       case 5: sprintf(buff,"%9.4f",pparms.vport_min.z); break;
       default: printf("Error somewhere \n"); exit(1);
       }
     XmTextSetString(edit[i],buff);
     }
  for(i=0; i < XtNumber(wbuttons); i++){
     button[i] = XtCreateWidget(wbuttons[i],xmPushButtonWidgetClass,
                                bb, NULL, 0);
     switch(i) {
       case 0: XtAddCallback(button[i], XmNactivateCallback,
                               get_vport_maxx, edit[i]); break;
       case 1: XtAddCallback(button[i], XmNactivateCallback,
```

```
                        get_vport_maxy, edit[i]); break;
      case 2: XtAddCallback(button[i], XmNactivateCallback,
                        get_vport_maxz, edit[i]); break;
      case 3: XtAddCallback(button[i], XmNactivateCallback,
                        get_vport_minx, edit[i]); break;
      case 4: XtAddCallback(button[i], XmNactivateCallback,
                        get_vport_miny, edit[i]); break;
      case 5: XtAddCallback(button[i],XmNactivateCallback,
                        get_vport_minz,edit[i]); break;
      default: printf("Error somewhere \n"); exit(1);
      }
    }
  done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(done_button,XmNactivateCallback,done_callback, user_data);
  /* Add a help button */
  help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(help_button,XmNactivateCallback,xs_help_callback,help_str);
  XtManageChildren(edit, XtNumber(weditors));
  XtManageChildren(button, XtNumber(wbuttons));
  return(bb);
}
/*********************************************************/
Widget wview_create_dialog(parent,user_data)
COMB_DATA *user_data;
Widget parent;
{
  Widget bb, edit[9], button[9], done_button,help_button;
  Arg wargs[10];
  int i, n = 0;
  char buff[80];

  XtSetArg(wargs[n], XmNautoUnmanage, FALSE); n++;
  bb = XmCreateBulletinBoardDialog(parent,"WORLD VIEW COORDINATES",wargs,n);
  user_data->wwview = bb;
  for(i=0; i < XtNumber(wveditors); i++) {
    edit[i] = XtCreateWidget(wveditors[i], xmTextWidgetClass, bb, NULL, 0);
     switch(i) {
       case 0: sprintf(buff,"%9.4f",pparms.wview_max.x); break;
       case 1: sprintf(buff,"%9.4f",pparms.wview_max.y); break;
       case 2: sprintf(buff,"%9.4f",pparms.wview_max.z); break;
       case 3: sprintf(buff,"%9.4f",pparms.wview_min.x); break;
       case 4: sprintf(buff,"%9.4f",pparms.wview_min.y); break;
       case 5: sprintf(buff,"%9.4f",pparms.wview_min.z); break;
       case 6: sprintf(buff,"%9.4f",pparms.wview_delta.x); break;
       case 7: sprintf(buff,"%9.4f",pparms.wview_delta.y); break;
       case 8: sprintf(buff,"%9.4f",pparms.wview_delta.z); break;
       default: printf("Error somewhere \n"); exIt(1);
       }
    XmTextSetString(edit[i],buff);
    }

  for(i=0; i < XtNumber(wvbuttons); i++){
     button[i]=XtCreateWidget(wvbuttons[i],xmPushButtonWidgetClass,bb,NULL,0);

     switch(i) {
       case 0: XtAddCallback(button[i], XmNactivateCallback,
```



```
                        get_wview_maxx, edit[i]);
                break;
      case 1: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_maxy, edit[i]);
                break;
      case 2: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_maxz, edit[i]);
                break;
      case 3: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_minx, edit[i]);
                break;
      case 4: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_miny, edit[i]);
                break;
      case 5: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_minz, edit[i]);
                break;
      case 6: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_dx, edit[i]);
                break;
      case 7: XtAddCallback(button[i], XmNactivateCallback,
                        get_wview_dy, edit[i]);
                break;
      case 8:XtAddCallback(button[i],XmNactivateCallback,
                        get_wview_dz,edit[i]);
                break;
      default: printf("Error somewhere \n"); exit(1);
      }
    }
  done_button=XtCreateManagedWidget("done",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(done_button,XmNactivateCallback, done_callback, user_data);
  /* Add a help button */
  help_button=XtCreateManagedWidget("help",xmPushButtonWidgetClass,bb,NULL,0);
  XtAddCallback(help_button,XmNactivateCallback,xs_help_callback,help_str);
  XtManageChildren(edit, XtNumber(wveditors));
  XtManageChildren(button, XtNumber(wvbuttons));
  return(bb);
}
/*********************************************************/
void get_setno(w, user_data, call_data)
Widget w;
COMB_DATA *user_data;
XmSelectionBoxCallbackStruct *call_data;
{
  double result;
  char *text;

  XmStringGetLtoR(call_data->value,XmSTRING_DEFAULT_CHARSET,&text);
  if(strcmp(text,"DIVERGING VECTOR FIELD") == 0) {
      if(pparms.setno != DIVERGING) {
       pparms.change_flag = TRUE;
       pparms.setno = DIVERGING;
       }
    }
  else if(strcmp(text,"CIRCULAR VECTOR FIELD") == 0) {
      if(pparms.setno != CIRCULAR) {
```

```
        pparms.change_flag = TRUE;
        pparms.setno = CIRCULAR;
        }
      }
  else {
      if(pparms.setno != OTHER) {
        pparms.change_flag = TRUE;
        pparms.setno = OTHER;
        }
      }
  XtUnmanageChild(user_data->wsetno);
  if(pparms.change_flag) {
     pparms.change_flag = FALSE;
     create_data(user_data->data);
     resize(user_data->canvas, user_data->data, NULL);
     }
  XtFree(text);
}
/*********************************************************/
void get_prtype(w, user_data, call_data)
Widget w;
COMB_DATA *user_data;
XmSelectionBoxCallbackStruct *call_data;
{
  double result;
  char *text;

  XmStringGetLtoR(call_data->value,XmSTRING_DEFAULT_CHARSET,&text);
  if(strcmp(text,"ORTHOGONAL PROJECTION") == 0) {
      if(pparms.type_proj != MORT) {
        pparms.change_flag = TRUE;
        pparms.type_proj = MORT;
        }
      }
  else {
      if(pparms.type_proj != MPPER) {
        pparms.change_flag = TRUE;
        pparms.type_proj = MPPER;
        }
      }
  XtUnmanageChild(user_data->wprtype);
  if(pparms.change_flag) {
     pparms.change_flag = FALSE;
     create_data(user_data->data);
     resize(user_data->canvas, user_data->data, NULL);
     }
  XtFree(text);
}
/*********************************************************/
void get_vpnx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];
```



```
  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1){
      if(pparms.vpn.x != result) {
        pparms.change_flag = TRUE;
        pparms.vpn.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vpn.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vpny(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vpn.y != result) {
        pparms.change_flag = TRUE;
        pparms.vpn.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vpn.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vpnz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vpn.z != result) {
        pparms.change_flag = TRUE;
        pparms.vpn.z = result;
        }
      }
  else {
```

```
      sprintf(buff,"%9.4f",pparms.vpn.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_vupx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1){
      if(pparms.vup.x != result) {
        pparms.change_flag = TRUE;
        pparms.vup.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vup.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_vupy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vup.y != result) {
        pparms.change_flag = TRUE;
        pparms.vup.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vup.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_vupz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
```



```
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vup.z != result) {
        pparms.change_flag = TRUE;
        pparms.vup.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vup.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_prpx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1){
      if(pparms.prp.x != result) {
        pparms.change_flag = TRUE;
        pparms.prp.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.prp.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_prpy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.prp.y != result) {
        pparms.change_flag = TRUE;
        pparms.prp.y = result;
```

```
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.prp.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_prpz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.prp.z != result) {
        pparms.change_flag = TRUE;
        pparms.prp.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.prp.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_window_maxx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.wmaxx != result) {
        pparms.change_flag = TRUE;
        pparms.window.wmaxx = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.wmaxx);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_window_maxy(w, textwidget, call_value)
```



```
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.wmaxy != result) {
        pparms.change_flag = TRUE;
        pparms.window.wmaxy = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.wmaxy);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/

void get_window_minx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0", buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.wminx != result) {
        pparms.change_flag = TRUE;
        pparms.window.wminx = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.wminx);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_window_miny(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
```

```
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.wminy != result) {
        pparms.change_flag = TRUE;
        pparms.window.wminy = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.wminy);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_window_F(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.Fmin != result) {
        pparms.change_flag = TRUE;
        pparms.window.Fmin = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.Fmin);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_window_B(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str,*Reset = "\0",buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.window.Bmax != result) {
        pparms.change_flag = TRUE;
        pparms.window.Bmax = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.window.Bmax);
      XmTextSetString(textwidget,buff);
      }
```



```
  XtFree(str);
}
/*********************************************************/
void get_wview_maxx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_max.x != result) {
        pparms.change_flag = TRUE;
        pparms.wview_max.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_max.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_wview_maxy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_max.y != result) {
        pparms.change_flag = TRUE;
        pparms.wview_max.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_max.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_wview_maxz(w, textwidget, call_value)
Widget w;
Widget textwidget;
```

```
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_max.z != result) {
        pparms.change_flag = TRUE;
        pparms.wview_max.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_max.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_wview_minx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_min.x != result) {
        pparms.change_flag = TRUE;
        pparms.wview_min.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_min.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_wview_miny(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];
```



```
  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_min.y != result) {
        pparms.change_flag = TRUE;
        pparms.wview_min.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_min.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_wview_minz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_min.z != result) {
        pparms.change_flag = TRUE;
        pparms.wview_min.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_min.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*******************************************************/
void get_wview_dx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_delta.x != result) {
        pparms.change_flag = TRUE;
```

```
        pparms.wview_delta.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_delta.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_wview_dy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_delta.y != result) {
        pparms.change_flag = TRUE;
        pparms.wview_delta.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_delta.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_wview_dz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.wview_delta.z != result) {
        pparms.change_flag = TRUE;
        pparms.wview_delta.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.wview_delta.z);
      XmTextSetString(textwidget,buff);
```



```
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_maxx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_max.x != result) {
        pparms.change_flag = TRUE;
        pparms.vport_max.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_max.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_maxy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_max.y != result) {
        pparms.change_flag = TRUE;
        pparms.vport_max.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_max.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_maxz(w, textwidget, call_value)
Widget w;
```

```
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_max.x != result) {
        pparms.change_flag = TRUE;
        pparms.vport_max.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_max.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_minx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_min.x != result) {
        pparms.change_flag = TRUE;
        pparms.vport_min.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_min.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_miny(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
```



```
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_min.y != result) {
        pparms.change_flag = TRUE;
        pparms.vport_min.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_min.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vport_minz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;

  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vport_min.z != result) {
        pparms.change_flag = TRUE;
        pparms.vport_min.z = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vport_min.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vrpx(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1){
```

```
      if(pparms.vrp.x != result) {
        pparms.change_flag = TRUE;
        pparms.vrp.x = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vrp.x);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vrpy(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vrp.y != result) {
        pparms.change_flag = TRUE;
        pparms.vrp.y = result;
        }
      }
  else {
      sprintf(buff,"%9.4f",pparms.vrp.y);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
void get_vrpz(w, textwidget, call_value)
Widget w;
Widget textwidget;
XmAnyCallbackStruct *call_value;
{
  double result;
  char *str;
  char *Reset = "\0";
  char buff[80];

  str = Reset;
  str  = XmTextGetString(textwidget);
  if(sscanf(str,"%lf",&result) == 1) {
      if(pparms.vrp.z != result) {
        pparms.change_flag = TRUE;
        pparms.vrp.z = result;
        }
      }
  else {
```



```
      sprintf(buff,"%9.4f",pparms.vrp.z);
      XmTextSetString(textwidget,buff);
      }
  XtFree(str);
}
/*********************************************************/
static void done_callback(w, user_data, call_data)
Widget w;
COMB_DATA *user_data;
XmAnyCallbackStruct *call_data;
{
  XtUnmanageChild(user_data->wvrp);
  XtUnmanageChild(user_data->wvup);
  XtUnmanageChild(user_data->wvpn);
  XtUnmanageChild(user_data->wprp);
  XtUnmanageChild(user_data->wwindow);
  XtUnmanageChild(user_data->wvport);
  XtUnmanageChild(user_data->wvview);
  if(pparms.change_flag) {
     pparms.change_flag = FALSE;
     create_data(user_data->data);
     resize(user_data->canvas, user_data->data, NULL);
     }
}
/*********************************************************/
clip_in_3d(data3,data,x,y,z,xorg,yorg,zorg,zmin)
PROJECTION *data3;
image_data *data;
double x[][MAXPTS],y[][MAXPTS],z[][MAXPTS],
       xorg[],yorg[],zorg[],
       zmin;
{
  int i,j;
  double x0,y0,z0,x1,y1,z1;
  int flag;

  for(j=0; j < data->numptsonscreen; j++) {
     x0 = xorg[j];
     y0 = yorg[j];
     z0 = zorg[j];
     data->draw_symbol[j] = TRUE;
     for(i=0; (i < data->numpts[j]) && (data->draw_symbol[j]); i++) {
       x1 = x[j][i];
       y1 = y[j][i];
       z1 = z[j][i];
       if((data3->type_proj != MPER) && (data3->type_proj != MPPER))
           CohenSutherlandLineClipPLL(&x0,&y0,&x1,&y1,&z0,&z1,&flag);
       else
           CohenSutherlandLineClipPER(&x0,&y0,&x1,&y1,&z0,&z1,zmin,&flag);
       if((!flag) || (x0 != xorg[j]) ||(y0 != yorg[j]) || (z0 != zorg[j]))
          data->draw_symbol[j] = FALSE;
       else {
          x[j][i] = x1;
          y[j][i] = y1;
          z[j][i] = z1;
          }
```

```
      }
    if(data->numpts[j] == 1) {
      for(i=1; (i < 5) && (data->draw_symbol[j]); i++) {
        x0 = x[j][i];
        y0 = y[j][i];
        z0 = z[j][i];
        if(i != 4) {
           x1 = x[j][i+1];
           y1 = y[j][i+1];
           z1 = z[j][i+1];
           }
        else {
           x1 = x[j][1];
           y1 = y[j][1];
           z1 = z[j][1];
           }
        if((data3->type_proj != MPPER) && (data3->type_proj != MPER))
           CohenSutherlandLineClipPLL(&x0,&y0,&x1,&y1,&z0,&z1,&flag);
        else
           CohenSutherlandLineClipPER(&x0,&y0,&x1,&y1,&z0,&z1,zmin,&flag);
        if(!flag) data->draw_symbol[j] = FALSE;
        else {
           x[j][i] = x0;
           y[j][i] = y0;
           z[j][i] = z0;
           if(i != 4) {
             x[j][i+1] = x1;
             y[j][i+1] = y1;
             z[j][i+1] = z1;
             }
           else {
             x[j][1] = x1;
             y[j][1] = y1;
             z[j][1] = z1;
             }
           }
        }
      }
    }/* for j < NUMPTSONSCREEN */
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAXPTS 10
#define NUMPTSONSCREEN 5000
#define MULTFACT 10
#define DIVERGING 1
#define CIRCULAR  2
#define OTHER     3


void resize();
void redisplay();
void create_command_panel();
void create_image();
void scale_it();
void draw_perp_disk();
void fill_perp_disk();
void draw_arrow();
void draw_image_in_window();
void draw_image_in_pixmap();
void apply_transformation();
void project_to_2d();
void get_projection_specs();
void done_callback();
void copy_projection();

Widget prtype_create_dialog();
void get_prtype();
void remove_prtype();

void save_parms();

Widget setno_create_dialog();
void get_setno();
void remove_setno();

void show_dialog();
Widget vpn_create_dialog();
void get_vpnx();
void get_vpny();
void get_vpnz();

Widget vup_create_dialog();
void get_vupx();
void get_vupy();
void get_vupz();

Widget vrp_create_dialog();
void get_vrpx();
void get_vrpy();
void get_vrpz();

Widget prp_create_dialog();
void get_prpx();
void get_prpy();
```

```
void get_prpz();

Widget window_create_dialog();
void get_window_maxx();
void get_window_maxy();
void get_window_minx();
void get_window_miny();
void get_window_F();
void get_window_B();

Widget vport_create_dialog();
void get_vport_maxx();
void get_vport_maxy();
void get_vport_maxz();
void get_vport_minx();
void get_vport_miny();
void get_vport_minz();

Widget wview_create_dialog();
void get_wview_maxx();
void get_wview_maxy();
void get_wview_maxz();
void get_wview_minx();
void get_wview_miny();
void get_wview_minz();
void get_wview_dx();
void get_wview_dy();
void get_wview_dz();

double sqre();
extern int  create_arrow();
extern int  fill_compose_data();
extern int  compose();
extern int construct_mgeneral();
extern int get_transformation_matrix();
extern int matmul4D();
extern int project_point();
extern int compose_point();
extern int compose_3d_point();
extern int EXIT();
extern int get_evev();
extern double calc_hyp();

typedef struct {
  float real;
  float imag;
  } complex;

typedef struct {
  int (*current_func)();
  float       range;
  double      x_origin[NUMPTSONSCREEN];
  double      y_origin[NUMPTSONSCREEN];
  int         draw_symbol[NUMPTSONSCREEN];
  int         start_x[NUMPTSONSCREEN][MAXPTS];
  int         start_y[NUMPTSONSCREEN][MAXPTS];
```



```
  int         last_x,last_y;
  double      x[NUMPTSONSCREEN][MAXPTS];
  double      y[NUMPTSONSCREEN][MAXPTS];
  int         numpts[NUMPTSONSCREEN];
  int         numptsonscreen;
  int         cplx_flag[NUMPTSONSCREEN];
  int         maxx,maxy,minx,miny;
  ARROW       a;
  GC          gc;
  Pixmap      pix;
  Pixmap      tile;
  Dimension   width,
              height;
  char        *pattern;
  int         flag[NUMPTSONSCREEN];
  }image_data, *image_data_ptr;


static XtResource resources[] = {
  {"range", "Range", XtRFloat, sizeof(float),
    XtOffset(image_data_ptr, range), XtRString, "1.0"},
  };


char *buttons[] = { "button1", "button2", "button3");
char *editors[] = {"field1", "field2", "field3"};
char *wbuttons[] = { "button1", "button2", "button3",
                     "button4", "button5", "button6"};
char *weditors[] = {"field1", "field2", "field3",
                    "field4", "field5", "field6"};
char *wvbuttons[] = { "button1", "button2", "button3",
                      "button4", "button5", "button6",
                      "button7", "button8", "button9"};
char *wveditors[] = {"field1", "field2", "field3",
                     "field4", "field5", "field6",
                     "field7", "field8", "field9"};
char *prtype[] = {"ORTHOGONAL PROJECTION",
                  "PERSPECTIVE PROJECTION"};
char *setno[] = {"DIVERGING VECTOR FIELD",
                 "CIRCULAR VECTOR FIELD",
                 "ARBITRARY VECTOR FIELD"};

char *help_str[] = {
  "Fill in the fields and press the DONE button ",
  "when done. ",
  "",
  "To fill in a particular field, click ",
  "the left mouse button after placing the sprite ",
  "in the particular field; enter the desired data. ",
  "and then press the button to the the left to indicate ",
  "that data entry for that field is completed. ",
  "",
  "Sorry, more detailed help is unavailable. ",
  "",
  ""};
```

```
typedef struct {
   int change_flag;
   int type_proj;
   int setno;
   double d;
   double alpha;
   RPOINT vrp;       /* In World coordinates */
   RPOINT vpn;       /* In World coordinates */
   RPOINT vup;       /* In World coordinates */
   RPOINT prp;       /* In View reference coordinates */
   WINDOW window;      /* In View reference coordinates */
   RPOINT vport_max;
   RPOINT vport_min;
   RPOINT wview_max;
   RPOINT wview_min;
   RPOINT wview_delta;
   } PPARM;

typedef struct {
  Widget canvas;
  image_data *data;
  Widget wvrp;
  Widget wvup;
  Widget wvpn;
  Widget wprp;
  Widget wwindow;
  Widget wvport;
  Widget wwview;
  Widget wprtype;
  Widget wsetno;
  } COMB_DATA;

PPARM pparms;

char *help_str1[] = {
  "Select the type of Projection desired, using ",
  "either the arrow keys or entering the choice ",
  "in the text entry area. Press the <OK> button",
  "when done. Press <CANCEL> to cancel this ",
  "function",
  "",
  "The Projection Types are ORTHOGONAL  ",
  "or PERSPECTIVE. ",
  "",
  "Sorry, more detailed help is unavailable. ",
  "",
  ""};


char *help_str2[] = {
  "Select the type of Vector field to view, using ",
  "either the arrow keys or entering the choice ",
  "in the text entry area. Press the <OK> button",
  "when done. Press <CANCEL> to cancel this ",
```



```
  "function",
  "",
  "The currently available vector fields",
  "are, DIVERGING, CIRCULAR and an ",
  "ARBITRARY vector fields. ",
  "",
  "Sorry, more detailed help is unavailable. ",
  "",
  ""};
```

```
#include "trans.h"
/* trans.c : This file contains most of the functions required
 *           for graphical transformations. The remaining functions
 *           are in evevwin.c
 * Written by: Ravi Kumar Gundimeda
 * Dated     : May 18th, 1992.
*/
/*****************************************************************/
compose_point(a,b,num_points,data)
int a[],b[];
COMPOSITION *data;
{
   double T[MAX_ROWS][MAX_COLS];
   double R[MAX_ROWS][MAX_COLS];
   double point[MAX_ROWS];
   double cpoint[MAX_ROWS];
   int i;

   create_translation_matrix(T,data);
   create_rotation_matrix(R,data);

  for(i=0; i < num_points; i++) {
   point[0] = (double) a[i];
   point[1] = (double) b[i];
   point[2] = 1;
   matmul(point,R,cpoint);
   matmul(cpoint,T,point);
   a[i] = point[0];
   b[i] = point[1];
   }
}
/*****************************************************************/
compose(a,data,comp_a)
ARROW *a,*comp_a;
COMPOSITION *data;
{
   double T[MAX_ROWS][MAX_COLS],R[MAX_ROWS][MAX_COLS];
   double point[MAX_ROWS],cpoint[MAX_ROWS];

   create_translation_matrix(T,data);
   create_rotation_matrix(R,data);
   point[0] = a->arrow_base_x;
   point[1] = a->arrow_base_y;
   point[2] = 1;
   matmul(point,R,cpoint);
   matmul(cpoint,T,point);
   comp_a->arrow_base_x = point[0];
   comp_a->arrow_base_y = point[1];
   point[0] = a->arrow_tip_x;
   point[1] = a->arrow_tip_y;
   point[2] = 1;
   matmul(point,R,cpoint);
   matmul(cpoint,T,point);
   comp_a->arrow_tip_x = point[0];
   comp_a->arrow_tip_y = point[1];
   point[0] = a->arrow_lwing_x;
```



```
   point[1] = a->arrow_lwing_y;
   point[2] = 1;
   matmul(point,R,cpoint);
   matmul(cpoint,T,point);
   comp_a->arrow_lwing_x = point[0];
   comp_a->arrow_lwing_y = point[1];
   point[0] = a->arrow_rwing_x;
   point[1] = a->arrow_rwing_y;
   point[2] = 1;
   matmul(point,R,cpoint);
   matmul(cpoint,T,point);
   comp_a->arrow_rwing_x = point[0];
   comp_a->arrow_rwing_y = point[1];
}
/*****************************************************************/
matmul(a,b,c)
double a[],c[],b[][MAX_COLS];
{
  int i,j,k;

  for(i=0; i < MAX_ROWS; i++) {
      c[i] = 0.0;
      for(j=0; j < MAX_COLS; j++)
         c[i] += b[i][j] * a[j];
      }
}
/*****************************************************************/
create_rotation_matrix(R,data)
COMPOSITION *data;
double R[][MAX_COLS];
{
   double sth,cth;

   sth = sin ( data->rotation_angle);
   cth = cos (data->rotation_angle);
   init_matrix(R);
   R[0][0] = cth;
   R[0][1] = 0.0 - sth;
   R[1][0] = sth;
   R[1][1] = cth;
   R[2][2] = 1.0;
}
/*****************************************************************/
create_translation_matrix(T,data)
COMPOSITION *data;
double T[][MAX_COLS];
{
    init_matrix(T);
    T[0][0] = 1.0;
    T[1][1] = 1.0;
    T[2][2] = 1.0;
    T[0][2] = data->trans_x;
    T[1][2] = data->trans_y;
}
/*****************************************************************/
create_arrow(a)
```

```
ARROW *a;
{
    a->arrow_base_x = 0;
    a->arrow_base_y = 0;
    a->arrow_tip_x = 0;
    a->arrow_tip_y = 0;
    a->arrow_lwing_x = -7;
    a->arrow_lwing_y = 3;
    a->arrow_rwing_x = -7;
    a->arrow_rwing_y = -3;
}
/*****************************************************************/
fill_compose_data(data,xl,yl,oox,ooy,nox,noy,theta,thetay, thetaz)
double theta,thetay,thetaz;
int xl,yl,oox,ooy,nox,noy;
COMPOSITION *data;
{
    data->scale_x = 1;
    data->scale_y = 1;
    data->trans_x = xl;
    data->trans_y = yl;
    data->rotation_angle = theta;
    data->rotation_angle_y = thetay;
    data->rotation_angle_z = thetaz;
    data->old_origin_x = oox;
    data->old_origin_y = ooy;
    data->new_origin_x = nox;
    data->new_origin_y = noy;
}
/*****************************************************************/
init_matrix(T)
double T[][MAX_COLS];
{
  int i,j;

  for(i=0; i < MAX_ROWS; i++)
    for(j=0; j < MAX_COLS; j++)
        T[i][j] = 0.0;
}
/*****************************************************************/
construct_mgeneral(M,proj_parms)
PROJECTION *proj_parms;
double M[][4];
{
    int i,j;
    double zp,Q,dx,dy,dz,a;

    /* Default values */
    dx = 0.0;
    dy = 0.0;
    dz = - 1.0;
    Q  = HUGE_VAL;
    zp = 0.0;
    switch(proj_parms->type_proj) {
       case MORT     : break;
       case MPER     : zp = proj_parms->d; Q = zp; break;
```



```
       case MPPER    : Q = proj_parms->d; break;
       case MORT_CAV : dx = cos (proj_parms->alpha);
                       dy = sin (proj_parms->alpha);
                       break;
       case MORT_CAB : dx = cos (proj_parms->alpha) / 2.0;
                       dy = sin (proj_parms->alpha) / 2.0;
                       break;
       default       : printf("Error in construct_mgeneral\n"); exit(1);
       }

    init_4d_matrix(M);
    a  = (zp / Q) * dz;
    /* Now construct the matrix */
    M[0][0] = 1.0;
    M[1][1] = 1.0;
    M[0][2] = 0.0 - (dx / dz);
    M[0][3] = - (M[0][2] * zp);
    M[1][2] = - (dy / dz);
    M[1][3] = -(M[1][2] * zp);
    M[2][2] = - a;
    M[2][3] = zp * (a + 1.0);
    M[3][2] = -( (1.0 / Q) / dz);
    M[3][3] = a + 1.0;
}
/*****************************************************************/
project_point(M,point,ppoint)
double M[][4],point[],ppoint[];
{
   int i,j;

   for(i=0; i <= MAX_ROWS; i++) {
     ppoint[i] = 0.0;
     for(j=0; j < MAX_COLS+1; j++)
         ppoint[i] += M[i][j] * point[j];
     }
}
/*****************************************************************/
init_4d_matrix(M)
double M[][MAX_COLS+1];
{
  int i,j;

  for(i=0; i <= MAX_ROWS; i++)
    for(j=0; j <= MAX_COLS; j++)
       M[i][j] = 0.0;
}
/*****************************************************************/
create_3dx_rotation_matrix(R,data)
COMPOSITION *data;
double R[][MAX_COLS+1];
{
   double sth,cth;

   sth = sin ( data->rotation_angle);
   cth = cos (data->rotation_angle);
   init_4d_matrix(R);
```

```
   R[0][0] = 1.0;
   R[1][1] = cth;
   R[1][2] = 0.0 - sth;
   R[2][1] = sth;
   R[2][2] = cth;
   R[3][3] = 1.0;
}
/*****************************************************************/
create_3dy_rotation_matrix(R,data)
COMPOSITION *data;
double R[][MAX_COLS+1];
{
   double sth,cth;

   sth = sin ( data->rotation_angle_y);
   cth = cos (data->rotation_angle_y);
   init_4d_matrix(R);
   R[0][0] = cth;
   R[0][2] = sth;
   R[1][1] = 1.0;
   R[2][0] = -sth;
   R[2][2] = cth;
   R[3][3] = 1.0;
}
/*****************************************************************/
create_3dz_rotation_matrix(R,data)
COMPOSITION *data;
double R[][MAX_COLS+1];
{
   double sth,cth;

   sth = sin ( data->rotation_angle_z);
   cth = cos (data->rotation_angle_z);
   init_4d_matrix(R);
   R[0][0] = cth;
   R[0][1] = -sth;
   R[1][0] = sth;
   R[1][1] = cth;
   R[2][2] = 1.0;
   R[3][3] = 1.0;
}
/*****************************************************************/
compose_3d_point(a,b,c,num_points,data)
double a[],b[],c[];
COMPOSITION *data;
{
   double RX[MAX_ROWS+1][MAX_COLS+1],RY[MAX_ROWS+1][MAX_COLS+1];
   double point[MAX_ROWS+1],cpoint[MAX_ROWS+1];
   int i;

  create_3dy_rotation_matrix(RY,data);
  create_3dx_rotation_matrix(RX,data);
  for(i=0; i < num_points; i++) {
   point[0] =  a[i];
   point[1] =  b[i];
   point[2] =  c[i];
```



```
   point[3] = 1;
   matmul_4d(point,RX,cpoint);
   matmul_4d(cpoint,RY,point);
   a[i] = point[0];
   b[i] = point[1];
   c[i] = point[2];
   }
}
/*****************************************************************/
matmul_4d(a,b,c)
double a[],c[],b[][MAX_COLS+1];
{
  int i,j,k;

  for(i=0; i < MAX_ROWS+1; i++) {
      c[i] = 0.0;
      for(j=0; j < MAX_COLS+1; j++)
         c[i] += b[i][j] * a[j];
      }
}
/*****************************************************************/
double calc_hyp(op,ad)
double op,ad;
{
  double num;

  num = op*op + ad*ad;
  num = sqrt(num);
  return(num);
}
/*****************************************************************/
create_3d_translation_matrix(T,data)
COMPOSITION *data;
double T[][MAX_COLS+1];
{
    init_4d_matrix(T);
    T[0][0] = 1.0;
    T[1][1] = 1.0;
    T[2][2] = 1.0;
    T[3][3] = 1.0;
    T[0][3] = data->trans_x;
    T[1][3] = data->trans_y;
    T[2][3] = data->trans_z;
}
/*****************************************************************/
create_3d_inverse_rotation(R,RINV)
double R[][MAX_COLS+1];
double RINV[][MAX_COLS+1];
{
   int i,j;

   init_4d_matrix(RINV);
   for(i=0; i < MAX_ROWS+1; i++)
       for(j=0; j < MAX_ROWS+1; j++)
           RINV[i][j] = R[j][i];
}
```

```
/*****************************************************************/
create_3d_inverse_translation(R,RINV)
double R[][MAX_COLS+1];
double RINV[][MAX_COLS+1];
{
   int i,j;
   init_4d_matrix(RINV);
   for(i=0; i < MAX_ROWS+1; i++)
       for(j=0; j < MAX_ROWS+1; j++)
           RINV[i][j] = R[i][j];
   RINV[0][3] = -RINV[0][3];
   RINV[1][3] = -RINV[1][3];
   RINV[2][3] = -RINV[2][3];
}
/*****************************************************************/
get_ncoord_sys(origin,vpn,vup,u,v,w)
RPOINT *origin,*vpn,*vup,   /* Known at this point */
       *u,*v,*w;            /* Normalized unit vectors of new
                               coordinate system */
{
    double r3,r4,r5,s3,s4,s5,t3,t4,t5,MODV;
    RPOINT V3, V4, V5;

    r5 = vpn->x;
    s5 = vpn->y;
    t5 = vpn->z;
    calculate_length(r5,s5,t5,&MODV);
    V5.x = r5 / MODV;
    V5.y = s5 / MODV;
    V5.z = t5 / MODV;
    r3 = vup->x;
    s3 = vup->y;
    t3 = vup->z;
    calculate_length(r3,s3,t3,&MODV);
    V3.x = r3 / MODV;
    V3.y = s3 / MODV;
    V3.z = t3 / MODV;
    calculate_cross_product(&V5,&V3,&V4);
    calculate_cross_product(&V4,&V5,&V3);
    /* At this point V5 is the unit vector in u direction */
    /* At this point V3 is the unit vector in v direction */
    /* At this point V4 is the unit vector in w direction */
    copy_vector(&V5,w);
    copy_vector(&V3,v);
    copy_vector(&V4,u);
}
/*****************************************************************/
copy_vector(m,n)
RPOINT *m,*n;
{
   n->x = m->x;
   n->y = m->y;
   n->z = m->z;
}
/*****************************************************************/
calculate_cross_product(V1,V2,V3)
```



```
RPOINT *V1,*V2,*V3;
{
     V3->x = V1->y * V2->z - V1->z * V2->y;
     V3->y = V1->z * V2->x - V1->x * V2->z;
     V3->z = V1->x * V2->y - V1->y * V2->x;
}
/*****************************************************************/
calculate_length(x,y,z,l)
double x,y,z,*l;
{
   double L;

   L = x * x + y * y + z * z;
   L = sqrt(L);
   *l = L;
}
/*****************************************************************/
get_transformation_matrix(M,data3)
double M[][MCOLS];
PROJECTION *data3;
{
  RPOINT u,v,w;
  RPOINT *vrp,*vup,*vpn;
  double a,b,c,d,zero=0.0;
  double mrx[MROWS][MCOLS];
  double mry[MROWS][MCOLS];
  double mrz[MROWS][MCOLS];
  double invt[MROWS][MCOLS];
  double temp[MROWS][MCOLS];

  vrp = &(data3->vrp);
  vup = &(data3->vup);
  vpn = &(data3->vpn);
  initialize_matrix(M,NOIDENTITY);
  get_ncoord_sys(vrp,vpn,vup,&u,&v,&w);
  /* Rotate vector w about x-axis to coincide with the z-axis (0,0,1) */
  a = w.x;
  b = w.y;
  c = w.z;
  calculate_length(zero,b,c,&d);
  get_trans_matrix(invt,vrp);
  get_3dx_rotmatrix(mrx,b,c,d);
  get_3dy_rotmatrix(mry,a,d);
  matmul4d(mry,mrx,temp);
  matmul4d(temp,invt,M);
}
/*****************************************************************/
matmul4d(a,b,c)
double a[][MCOLS],c[][MCOLS],b[][MAX_COLS+1];
{
  int i,j,k;

  for(i=0; i < MAX_ROWS+1; i++)
      for(j=0; j < MAX_COLS+1; j++) {
          c[i][j] = 0.0;
          for(k=0; k < MAX_ROWS+1; k++)
```

```
            c[i][j] += a[i][k] * b[k][j];
        }
}
/*****************************************************************/
get_trans_matrix(invt,vrp)
double invt[][MCOLS];
RPOINT *vrp;
{
     initialize_matrix(invt,IDENTITY);
     invt[0][3] = -(vrp->x);
     invt[1][3] = -(vrp->y);
     invt[2][3] = -(vrp->z);
}
/*****************************************************************/
get_3dy_rotmatrix(mry,a,d)
double a,d;
double mry[][MCOLS];
{
     initialize_matrix(mry,IDENTITY);
     mry[0][0] = d;
     mry[0][2] = -a;
     mry[2][0] = a;
     mry[2][2] = d;
}
/*****************************************************************/
get_3dx_rotmatrix(mrx,b,c,d)
double b,c,d;
double mrx[][MCOLS];
{
     initialize_matrix(mrx,IDENTITY);
     mrx[1][1] = c/d;
     mrx[2][2] = c/d;
     mrx[1][2] = - (b/d);
     mrx[2][1] = b / d;
}
/*****************************************************************/
initialize_matrix(mrx,flag)
int flag;
double mrx[][MCOLS];
{
   int i,j;

   for(i=0; i < MROWS; i++)
    for(j=0; j < MCOLS; j++)
        mrx[i][j] = 0.0;
   if(flag) {
     mrx[0][0] = 1.0;
     mrx[1][1] = 1.0;
     mrx[2][2] = 1.0;
     mrx[3][3] = 1.0;
     }
}
/*****************************************************************/
get_transformed_coordinates(tp2,ttp2,M)
RPOINT *tp2,*ttp2;
double M[][MCOLS];
```



```
{
   double point[MROWS],ppoint[MROWS];

   point[0] = tp2->x;
   point[1] = tp2->y;
   point[2] = tp2->z;
   point[3] = 1.0;
   matmul_4d(point,M,ppoint);
   ttp2->x = ppoint[0];
   ttp2->y = ppoint[1];
   ttp2->z = ppoint[2];
}
/*****************************************************************/
get_normalizing_trans_pll(NPAR,data3)
PROJECTION *data3;
double NPAR[][MCOLS];
{
    double cw[MROWS],prp[MROWS],dop[MROWS],SHPAR[MROWS][MCOLS],shxpar,shypar;
    double TPAR[MROWS][MCOLS],tx,ty,tz,sx,sy,sz,SPAR[MROWS][MCOLS];
    double temp[MROWS][MCOLS];

     prp[0] = data3->prp.x;
     prp[1] = data3->prp.y;
     prp[2] = data3->prp.z;
     prp[3] = 1;
     tx    = -(data3->window.wmaxx + data3->window.wminx) / 2.0;
     ty    =  -(data3->window.wmaxy + data3->window.wminy) / 2.0;
     tz    = -data3->window.Fmin;
     sx    = 2.0 / (data3->window.wmaxx - data3->window.wminx);
     sy    = 2.0 / (data3->window.wmaxy - data3->window.wminy);
     sz    = 1.0 / (data3->window.Fmin - data3->window.Bmax);
     cw[0] = -tx;
     cw[1] = -ty;
     cw[2] = 0.0;
     cw[3] = 1;
     subtract_matrices(cw,prp,dop);
     shxpar = -(dop[0] / dop[2]);  /* -dopx / dopz */
     shypar = -(dop[1] / dop[2]);  /* -dopy / dopz */
     create_shear_matrix(SHPAR,shxpar,shypar);
     create_trans_matrix(TPAR,tx,ty,tz);
     create_scale_matrix(SPAR,sx,sy,sz);
     matmul4d(SPAR,TPAR,temp);
     matmul4d(temp,SHPAR,NPAR);
}
/*****************************************************************/
create_trans_matrix(TPAR,tx,ty,tz)
double TPAR[][MCOLS],tx,ty,tz;
{
     initialize_matrix(TPAR,IDENTITY);
     TPAR[0][3] = tx;
     TPAR[1][3] = ty;
     TPAR[2][3] = tz;
}
/*****************************************************************/
create_scale_matrix(SPAR,sx,sy,sz)
double SPAR[][MCOLS],sx,sy,sz;
```

```
{
     initialize_matrix(SPAR,IDENTITY);
     SPAR[0][0] = sx;
     SPAR[1][1] = sy;
     SPAR[2][2] = sz;
}
/*****************************************************************/
create_shear_matrix(SHPAR,shxpar,shypar)
double SHPAR[][MCOLS],shxpar,shypar;
{
     initialize_matrix(SHPAR,IDENTITY);
     SHPAR[0][2] = shxpar;
     SHPAR[1][2] = shypar;
}
/*****************************************************************/
subtract_matrices(cw,prp,dop)
double cw[],prp[],dop[];
{
    int i;

    for(i=0; i < 4; i++) dop[i] = cw[i] - prp[i];
}
/*****************************************************************/
get_normalizing_transformation(M,data3,zmin)
double M[][MCOLS];
PROJECTION *data3;
double *zmin;
{
   *zmin = 0;
   if((data3->type_proj != MPPER) && (data3->type_proj != MPER))
            get_normalizing_trans_pll(M,data3);
   else
            get_normalizing_trans_per(M,data3,zmin);
}
/*****************************************************************/
get_normalizing_trans_per(NPER,data3,zmin)
PROJECTION *data3;
double NPER[][MCOLS],*zmin;
{
    double tx,ty,tz,TPAR[MROWS][MCOLS],cw[MROWS],sx,sy,sz,dum1,dum2,dum3;
    double prp[MROWS],prpp[MROWS],vrp[MROWS],vrpp[MROWS],dop[MROWS];
    double SHPAR[MROWS][MCOLS],shxpar,shypar,SPER[MROWS][MCOLS];
    double temp[MROWS][MCOLS];

    tx = -data3->prp.x;
    ty = -data3->prp.y;
    tz = -data3->prp.z;
    create_trans_matrix(TPAR,tx,ty,tz);
     prp[0] = data3->prp.x;
     prp[1] = data3->prp.y;
     prp[2] = data3->prp.z;
     prp[3] = 1;
     vrp[0] = 0.0;
     vrp[1] = 0.0;
     vrp[2] = 0.0;
     vrp[3] = 1.0;
```



```
     tx    = -(data3->window.wmaxx + data3->window.wminx) / 2.0;
     ty    =  -(data3->window.wmaxy + data3->window.wminy) / 2.0;
     tz    = -data3->window.Fmin;
     cw[0] = -tx;
     cw[1] = -ty;
     cw[2] = 0.0;
     cw[3] = 1;
     subtract_matrices(cw,prp,dop);
     shxpar = -(dop[0] / dop[2]);  /* -dopx / dopz */
     shypar = -(dop[1] / dop[2]);  /* -dopy / dopz */
     create_shear_matrix(SHPAR,shxpar,shypar);
     matmul4d(SHPAR,TPAR,temp);
     matmul_4d(vrp,temp,vrpp);
     /* At this point check if vrpp[2] is == -prp[2] */
     dum2 = vrpp[2] + data3->window.Bmax;
     dum1 = (data3->window.wmaxx - data3->window.wminx);
     dum3 = (data3->window.wmaxy - data3->window.wminy);
     sx = 2.0 * vrpp[2] / (dum1 * dum2);
     sy = 2.0 * vrpp[2] / (dum3 * dum2);
     sz = -1.0 / dum2;
     create_scale_matrix(SPER,sx,sy,sz);
     matmul4d(SPER,temp,NPER);
     *zmin = - ((vrpp[2] + data3->window.Fmin)/ dum2 );
}
/*****************************************************************/
compute_outcode(x,y,z,xmax,ymax,zmax,xmin,ymin,zmin,code)
OUTCODE *code;
double x,y,z,xmax,ymax,xmin,ymin,zmax,zmin;
{
  unsigned int inside_mask = 0, behind_mask = 2, front_mask = 1,
               top_mask = 32, bottom_mask = 16, right_mask = 8,
               left_mask = 4;

  code->num = 0;
  if(y > ymax) code->num = code->num | top_mask;
  else if  (y < ymin) code->num = code->num | bottom_mask;
  if(x > xmax) code->num = code->num | right_mask;
  else if (x < xmin) code->num = code->num | left_mask;
  if(z > zmax) code->num = code->num | front_mask;
  else if (z < zmin) code->num = code->num | behind_mask;
}
/*****************************************************************/
CohenSutherlandLineClipPLL(X0,Y0,X1,Y1,Z0,Z1,flag)
double *X0,*Y0,*X1,*Y1,*Z0,*Z1;
int *flag; /* Whether points are accepted or not */
{
   int accept,done;
   OUTCODE outcode0,outcode1,outcodeOut;
   double x,y,x0,y0,x1,y1,z,z0,z1,xmin,xmax,ymin,ymax,zmax,zmin;

   xmin = -1.0; xmax = 1.0; ymin = -1.0; ymax = 1.0; zmax = 0.0; zmin = -1.0;
   x0 = *X0; y0 = *Y0; z0 = *Z0; x1 = *X1; y1 = *Y1; z1 = *Z1;
   accept = FALSE; done = FALSE;
   compute_outcode(x0,y0,z0,xmax,ymax,zmax,xmin,ymin,zmin,&outcode0);
   compute_outcode(x1,y1,z1,xmax,ymax,zmax,xmin,ymin,zmin,&outcode1);
```

```
   do {
      if((outcode0.num == 0) && (outcode1.num == 0)){
        /* Trivially accepted and return */
       accept = TRUE;
       done = TRUE;
       }
      else if((outcode0.num & outcode1.num) != 0)
       /* Trivially reject and return */
       done = TRUE;
      else {
         /* Failed both tests, so calculate the line segment to clip;
         from an outside point to an intersection with a clip edge */
         /* Atleast one endpoint is outside the clip rectangle;
            pick it . */
         if(outcode0.num != 0) outcodeOut.num = outcode0.num;
         else outcodeOut.num = outcode1.num;
         /* Now, find the intersection point; */
         if(outcodeOut.code.top_bit) {
            /* Divivde line at top of clip rectangle */
            x = x0 + (x1-x0) * (ymax-y0)/(y1-y0);
            y = ymax;
            z = z0 + (ymax-y0) *(z1-z0) / (y1-y0);
            }
         else if(outcodeOut.code.bottom_bit) {
            /* Divivde line at bottom of clip rectangle */
            x = x0 + (x1-x0) * (ymin-y0)/(y1-y0);
            y = ymin;
            z = z0 + (ymin-y0) *(z1-z0) / (y1-y0);
            }
         else if(outcodeOut.code.right_bit) {
            /* Divivde line at right of clip rectangle */
            y = y0 + (y1-y0) * (xmax-x0)/(x1-x0);
            x = xmax;
            z = z0 + (xmax-x0) *(z1-z0) / (x1-x0);
            }
         else if(outcodeOut.code.left_bit) {
            /* Divivde line at left of clip rectangle */
            y = y0 + (y1-y0) * (xmin-x0)/(x1-x0);
            x = xmin;
            z = z0 + (xmin-x0) *(z1-z0) / (x1-x0);
            }
         else if(outcodeOut.code.front_bit) {
            /* Divivde line at front of clip rectangle */
            y = y0 + (y1-y0) * (zmax-z0)/(z1-z0);
            x = x0 + (x1-x0) * (zmax-z0)/(z1-z0);
            z = zmax;
            }
         else if(outcodeOut.code.behind_bit) {
            /* Divivde line at back of clip rectangle */
            y = y0 + (y1-y0) * (zmin-z0)/(z1-z0);
            x = x0 + (x1-x0) * (zmin-z0)/(z1-z0);
            z = zmin;
            }

         /* Now, we move outside point to intersection point
            to clip, and get ready for the next pass */
```



```
            if(outcodeOut.num == outcode0.num) {
               x0 = x;
               y0 = y;
               z0 = z;
               compute_outcode(x0,y0,z0,xmax,ymax,zmax,
                               xmin,ymin,zmin,&outcode0);
               }
            else {
               x1 = x;
               y1 = y;
               z1 = z;
               compute_outcode(x1,y1,z1,xmax,ymax,zmax,
                               xmin,ymin,zmin,&outcode1);
               }
            }
         } while ( !done);

   *X0 = x0; *Y0 = y0; *Z0 = z0; *X1 = x1; *Y1 = y1; *Z1 = z1;
   *flag = accept;
}


/*****************************************************************/
computeCodePer(x,y,z,zmax,zmin,code)
OUTCODE *code;
double x,y,z,zmax,zmin;
{
  unsigned int inside_mask = 0, behind_mask = 2, front_mask = 1,
               top_mask = 32, bottom_mask = 16, right_mask = 8, left_mask = 4;

  code->num = 0;
  if(y > -z) code->num = code->num | top_mask;
  else if  (y < z) code->num = code->num | bottom_mask;
  if(x > -z) code->num = code->num | right_mask;
  else if (x < z) code->num = code->num | left_mask;
  if(z > zmax) code->num = code->num | front_mask;
  else if (z < zmin) code->num = code->num | behind_mask;
}
/*****************************************************************/
CohenSutherlandLineClipPER(X0,Y0,X1,Y1,Z0,Z1,ZMIN,flag)
double *X0,*Y0,*X1,*Y1,*Z0,*Z1,ZMIN;
int *flag; /* Whether points are accepted or not */
{
   int accept,done;
   OUTCODE outcode0,outcode1,outcodeOut;
   double x,y,x0,y0,x1,y1,z,z0,z1,xmin,xmax,ymin,ymax,zmax,zmin;

   zmax = ZMIN; zmin = -1.0;
   x0 = *X0; y0 = *Y0; z0 = *Z0; x1 = *X1; y1 = *Y1; z1 = *Z1;
   accept = FALSE; done = FALSE;
   computeCodePer(x0,y0,z0,zmax,zmin,&outcode0);
   computeCodePer(x1,y1,z1,zmax,zmin,&outcode1);
   do {
      if((outcode0.num == 0) && (outcode1.num == 0)){
         /* Trivially accepted and return */
       accept = TRUE;
```

```
        done = TRUE;
        }
      else if((outcode0.num & outcode1.num) != 0)
       /* Trivially reject and return */
       done = TRUE;
      else {
         /* Failed both tests, so calculate the line segment to clip;
         from an outside point to an intersection with a clip edge */
         /* Atleast one endpoint is outside the clip rectangle;
            pick it . */
         if(outcode0.num != 0) outcodeOut.num = outcode0.num;
         else outcodeOut.num = outcode1.num;
         /* Now, find the intersection point; */
          if(outcodeOut.code.top_bit) {
            /* Divivde line at top of clip rectangle */
            x = x0 - ((x1-x0) * (z0+y0)/(y1-y0+z1-z0));
            y = y0 - ((y1-y0)*(z0+y0) / (y1-y0+z1-z0));
            z = -y;
            }
          else if(outcodeOut.code.bottom_bit) {
            /* Divivde line at bottom of clip rectangle */
            x = x0 + (x1-x0) * (z0-y0)/(y1-y0-z1+z0);
            y = y0 + (y1-y0)*(z0-y0) / (y1-y0-z1+z0);
            z = y;
            }
          else if(outcodeOut.code.right_bit) {
            /* Divivde line at right of clip rectangle */
            x = x0 - (x1-x0) * (z0+x0)/(x1-x0+z1-z0);
            y = y0 - (y1-y0)*(z0+x0) / (x1-x0+z1-z0);
            z = -x;
            }
          else if(outcodeOut.code.left_bit) {
            /* Divivde line at left of clip rectangle */
            x = x0 + (x1-x0) * (z0-x0)/(x1-x0-z1+z0);
            y = y0 + (y1-y0)*(z0-x0) / (x1-x0-z1+z0);
            z = x;
            }
          else if(outcodeOut.code.front_bit) {
            /* Divivde line at front of clip rectangle */
            y = y0 + (y1-y0) * (zmax-z0)/(z1-z0);
            x = x0 + (x1-x0) * (zmax-z0)/(z1-z0);
            z = zmax;
            }
          else if(outcodeOut.code.behind_bit) {
            /* Divivde line at back of clip rectangle */
            y = y0 + (y1-y0) * (zmin-z0)/(z1-z0);
            x = x0 + (x1-x0) * (zmin-z0)/(z1-z0);
            z = zmin;
            }

          /* Now, we move outside point to intersection point
             to clip, and get ready for the next pass */
          if(outcodeOut.num == outcode0.num) {
              x0 = x;
              y0 = y;
              z0 = z;
```



```
              computeCodePer(x0,y0,z0,zmax, zmin,&outcode0);
              }
            else {
              x1 = x;
              y1 = y;
              z1 = z;
              computeCodePer(x1,y1,z1,zmax, zmin,&outcode1);
              }
            }
        } while ( !done);

   *X0 = x0; *Y0 = y0; *Z0 = z0; *X1 = x1; *Y1 = y1; *Z1 = z1;
   *flag = accept;
}
/*****************************************************************/
map_to_viewport(data3,M)
double M[][MCOLS];
PROJECTION *data3;
{
   double TPAR[MROWS][MCOLS],SPAR[MROWS][MCOLS],tx,ty,tz,sx,sy,sz;
   double temp[MROWS][MCOLS],TPER[MROWS][MCOLS];

   initialize_matrix(M,NOIDENTITY);
   tx = 1.0;
   ty = 1.0;
   tz = 0.0;
   create_trans_matrix(TPAR,tx,ty,tz);
   sx = (data3->vport_max.x - data3->vport_min.x) / 2.0;
   sy = (data3->vport_max.y - data3->vport_min.y) / 2.0;
   sz = (data3->vport_max.z - data3->vport_min.z) ;
   create_scale_matrix(SPAR,sx,sy,sz);
   tx = data3->vport_min.x;
   ty = data3->vport_min.y;
   tz = data3->vport_min.z;
   create_trans_matrix(TPER,tx,ty,tz);
   matmul4d(TPER,SPAR,temp);
   matmul4d(temp,TPAR,M);
}
/*****************************************************************/
```

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

#define MAX_ROWS 3
#define MAX_COLS 3
#define MORT 0
#define MPER 1
#define MPPER 2
#define MORT_CAV 3
#define MORT_CAB 4
#define MAXPOINTS 500
#define MAXCOLOR 256
#define PI 3.1415926
#define MROWS 4
#define MCOLS 4
#define IDENTITY 1
#define NOIDENTITY 0
#define TRUE 1
#define FALSE 0

typedef struct {
   double x;
   double y;
   double z;
   }RPOINT;


typedef struct {
    int arrow_base_x;
    int arrow_base_y;
    int arrow_tip_x;
    int arrow_tip_y;
    int arrow_lwing_x;
    int arrow_lwing_y;
    int arrow_rwing_x;
    int arrow_rwing_y;
    }ARROW;


typedef struct {
    int scale_x;
    int scale_y;
    int scale_z;
    int trans_x;
    int trans_y;
    int trans_z;
    double rotation_angle;
    double rotation_angle_y;
    double rotation_angle_z;
    int old_origin_x;
    int old_origin_y;
    int old_origin_z;
    int new_origin_x;
    int new_origin_y;
    int new_origin_z;
```



```
    }COMPOSITION;

typedef struct {
    int x;
    int y;
    int z;
    } VPOINT;

typedef struct {
    double wmaxx;
    double wmaxy;
    double wminx;
    double wminy;
    double Zmax, Zmin;
    } WINDOW ;


typedef struct {
   int type_proj;
   int setno;
   double d;
   double alpha;
   RPOINT vrp;       /* In World coordinates */
   RPOINT vpn;       /* In World coordinates */
   RPOINT vup;       /* In World coordinates */
   RPOINT prp;       /* In View reference coordinates */
   WINDOW window;    /* In View reference coordinates */
   RPOINT vport_max;
   RPOINT vport_min;
   RPOINT wview_max;
   RPOINT wview_min;
   RPOINT wview_delta;
   } PROJECTION;


typedef struct {
     unsigned int front_bit   : 1;
     unsigned int behind_bit  : 1;
     unsigned int left_bit    : 1;
     unsigned int right_bit   : 1;
     unsigned int bottom_bit  : 1;
     unsigned int top_bit     : 1;
     unsigned int unused_bits:26;
     }OCODE ;

typedef union {
    OCODE code;
    unsigned int num;
    }OUTCODE;
```

```
#include "evev.h"

/* evev.c : This file is the interface between evevwin and the
 *          FORTRAN library EISPACK which is a collection of
 *          subroutines for eigenvector calculations.
 * Written by: Ravi Kumar Gundimeda
 * Dated  : May 18th, 1992.
/*************************************************************/
get_evev(WR,WI,Z_CVTD,INIT_GEN_MTX)
double      Z_CVTD[MAX_ROWS][MAX_ROWS],
            WR[MAX_ROWS],
            WI[MAX_ROWS];
double INIT_GEN_MTX[MAX_ROWS][MAX_ROWS];
{
    int n,r,c,low,igh,i,j,ierr,m,mb;
    double INIT_GEN_CNVTD_MTX[MAX_ROWS][MAX_ROWS], ORT_MTX[MAX_ROWS][MAX_ROWS],
           scale[MAX_COLS], ORT[MAX_ROWS], Z[MAX_ROWS][MAX_ROWS],
           PZ[MAX_ROWS][MAX_ROWS];

   r = MAX_ROWS; c = MAX_COLS; ierr = 0; low = 1; igh = MAX_COLS;
   init_1dmatrix(WR);
   init_1dmatrix(WI);
   init_2dmatrix(PZ);
   create_identity_matrix(PZ);
   cvt_rmv_to_cmv(PZ, Z);
   cvt_rmv_to_cmv(INIT_GEN_MTX, INIT_GEN_CNVTD_MTX);
   balanc_(&r,&c,INIT_GEN_CNVTD_MTX,&low,&igh,scale);
   orthes_(&r,&c,&low,&igh,INIT_GEN_CNVTD_MTX,ORT);
   copy_matrices(INIT_GEN_CNVTD_MTX,ORT_MTX);
   hqr2_(&r, &c, &low, &igh,INIT_GEN_CNVTD_MTX,WR, WI, Z, &ierr);
   if(ierr != 0) {
      printf("Error computing eigen vectors/eigen values\n");
      printf("Error value: %d\n",ierr);
      EXIT(1);
      }
   m = 3;
   mb =m;
   ortbak_(&r, &low, &igh, ORT_MTX, ORT, &m, Z);
   balbak_(&r,&c,&low,&igh,scale, &mb, Z);
   cvt_rmv_to_cmv(Z, Z_CVTD);
}
/*************************************************************/
EXIT(i)
int i;
{
   /* Shutdown   */
   _F_exit();
   exit(i);
}
/*************************************************************/
print_array(B)
double B[][MAX_ROWS];
{
   int i,j;

   printf("\n\n");
```



```
   for(i=0; i < MAX_ROWS; i++){
     for(j=0; j < MAX_COLS; j++)
        printf("%8.4f ",B[i][j]);
     printf("\n");
     }
   printf("\n\n");
}
/*************************************************************/
cvt_rmv_to_cmv(rm_array, cm_array)
double rm_array[][MAX_COLS];
double cm_array[][MAX_ROWS];
{
   int i, j;

    for(i=0; i < MAX_ROWS; i++)
       for(j=0; j < MAX_ROWS; j++)
            cm_array[j][i] = rm_array[i][j];
}
/*************************************************************/
init_2dmatrix(MATX)
double MATX[][MAX_COLS];
{
   int i,j;

   for(i=0; i < MAX_COLS; i++)
     for(j=0; j < MAX_COLS; j++)
        MATX[i][j] = 0.0;
}
/*************************************************************/
init_1dmatrix(MATX)
double MATX[];
{
   int i;
   for(i=0; i < MAX_COLS; i++)
        MATX[i] = 0.0;
}
/*************************************************************/
create_identity_matrix(Z)
double Z[][MAX_COLS];
{
   int i,j;

   for(i=0; i < MAX_COLS; i++)
     for(j=0; j < MAX_COLS; j++)
        if(i == j) Z[i][j] = 1.0;
}
/*************************************************************/
copy_matrices(INIT_GEN_CNVTD_MTX,ORT_MTX)
double INIT_GEN_CNVTD_MTX[][MAX_COLS], ORT_MTX[][MAX_COLS];
{
   int i,j;

   for(i=0; i < MAX_COLS; i++)
     for(j=0; j < MAX_COLS; j++)
          ORT_MTX[i][j] = INIT_GEN_CNVTD_MTX[i][j];
}
```

```
/*************************************************************/
complex_mult(a,b,c,d,e,f)
double a,b,c,d,*e,*f;
{
   *e = (a*c) - (b*d);
   *f = (a*d) + (b*c);
}
/*************************************************************/
```



```
#include <stdio.h>

#define DEBUG 0

#define MAX_ROWS 3
#define MAX_COLS 3

typedef struct {
   double real_part;
   double imag_part;
   }COMPLEX;
```

VITA

RAVI K. GUNDIMEDA

Candidate for the Degree of

Master of Science

Thesis: SCIENTIFIC VISUALIZATION OF A 3-D VECTOR FIELD IN 3-D SPACE USING X-WINDOWS

Major Field: Computer Science

Biographical:

Personal Data: Born in Guntur, India, March 20th, 1963, son of G.S.Ramakrishna Rao and Radha Devi.

Education : Received Bachelor of Science Degree in Mechanical Engineering from Birla Institute of Technology, India in May 1984; Received Master of Technology degree from Indian Institute of Technology, India in August 1986; completed requirements for the Master of Science at Oklahoma State University in July, 1992.

Professional Experience:

Teaching Assistant, Department of Computer Science, Oklahoma State University, August 1991, to May 1992.
Research Assistant, Department of Agriculture, Oklahoma State University, September 1989 to May 1991.
Software Engineer, Tata Consulting Engineers(R&D), Bangalore, India, September 1986 to July 1989.