



THEMS

This is to certify that the

dissertation entitled

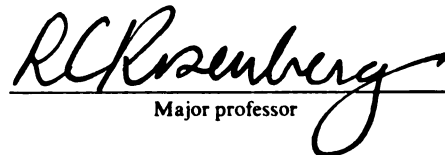
IMPROVING ENGINEERING SYSTEM DESIGN THROUGH
SCIENTIFIC VISUALIZATION METHODS

presented by

GAYLE ELLEN ERMER

has been accepted towards fulfillment
of the requirements for

PH.D. degree in MECHANICAL ENGINEERING


Major professor

Date 15 Nov 94

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

**IMPROVING ENGINEERING SYSTEM DESIGN THROUGH
SCIENTIFIC VISUALIZATION METHODS**

By

Gayle Ellen Ermer

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1994

ABSTRACT

IMPROVING ENGINEERING SYSTEM DESIGN THROUGH SCIENTIFIC VISUALIZATION METHODS

By

Gayle Ellen Ermer

One of the greatest challenges for designers of complex engineering systems is to interpret simulation data. Numerical results need to be presented in a way which is meaningful and encourages understanding of the simulated system behavior. Scientific visualization (SV) is an emerging discipline in which methods are developed for clear presentation of data to scientists and engineers.

This thesis presents a significantly improved design environment for engineering systems which has been developed and implemented using the concepts of SV . The design environment is based on the Bond Graph (BG) as a modeling representation. The BG captures the interconnectedness of the system components and shows power pathways as energy is transferred within the system. Animation of simulation results directly on the BG and the correspondence between the bond graph model and the physical system are both exploited to aid the designer in comprehending the system's behavior.

The research was conducted in two phases for each of which pilot environments were designed. In phase one a two-dimensional animation tool for simulation of variable values on the BG was developed (BGVis-2D). A complex railcar airbrake switching system was modeled and simulated to illustrate the benefits of the visualization. A

second example was developed to show the application of the new visualization tool in a somewhat abstract model space, namely modal analysis . In phase two a tool to represent BGs as 3-D objects was developed using the PHIGS graphics standard (BGVis-3D). A bounce-pitch-roll model of a car body and suspensions was implemented in the 3-D environment and its simulated behavior when encountering a pothole was investigated.

Dedication

To my husband, Eric.

ACKNOWLEDGEMENTS

I would like to sincerely thank my major professor, Dr. Ronald Rosenberg, for his support, guidance, and encouragement throughout my Ph.D. program. Not only was his knowledge invaluable for my research, but I learned many things about teaching from him as well.

I also express my appreciation to the members of my thesis committee: Drs. Clark Radcliffe, Erik Goodman, and Richard Hallgren. Their suggestions and comments were very valuable.

My gratitude also extends to ASME for providing me a graduate teaching fellowship and to the Zonta organization for awarding me the Amelia Earhart Fellowship. These scholarships were very precious to me, not only financially, but in the encouragement they gave me to continue pursuing this degree.

Finally, special thanks go to my parents, Robert and Carol Essenburg, and the rest of my family for always encouraging me to use my talents and for instilling in me the values that made this accomplishment possible.

TABLE OF CONTENTS

List of Tables	viii
List of Figures	ix
Nomenclature	xii
Chapter 1 - Introduction	1
1.1 Problem Definition	1
1.1.1 Importance of Improving the Efficiency of System Design	1
1.1.2 The Role of Insight and Interpretation of Results	2
1.1.3 The Role of Visualization in the Modeling Process	5
1.2 Research Objectives	11
1.3 Description of Thesis Organization	14
Chapter 2 - Bond Graphs and Scientific Visualization	16
2.1 Using Bond Graphs as a Tool for System Analysis and Design	16
2.1.1 Modeling of Engineering Systems	16
2.1.2 Bond Graph Modeling Methodology	19
2.1.3 Computer Support for Simulation	23
2.2 Scientific Visualization	24
2.2.1 Description of the Field	24
2.2.2 Current Tools and Research Areas	27
Chapter 3 - A Two-Dimensional Bond Graph Visualization Environment	30
3.1 Environment Specifications	30
3.2 Design of the Two-Dimensional Visualization Environment	31
3.2.1 Features Selected for Implementation	31

3.2.2 Platform Selected for Implementation	38
3.2.3 User's Guide	40
Chapter 4 - A Three-Dimensional Bond Graph Visualization Environment	45
4.1 Environment Specifications	45
4.2 Design of the Three-Dimensional Visualization Environment	46
4.2.1 Features Selected for Implementation	46
4.2.2 Platform Selected for Implementation	48
4.2.3 Description of the PHIGS Graphics Standard	49
4.2.4 User's Guide	53
Chapter 5 - Visualization Examples Using BGVis-2D	57
5.1 Automobile Modeling Example	57
5.2 Railcar Airbrake Switching System Example	79
5.3 Modal Analysis Example	110
Chapter 6 - Visualization Example Using BGVis-3D	116
Chapter 7 - Conclusions	122
7.1 Summary of Contributions	122
7.1.1 Two-Dimensional Visualization	122
7.1.2 Three-Dimensional Visualization	123
7.2 Areas for Further Research	124
Appendix A - A Brief Introduction to Bond Graph Modeling	128
Appendix B - Program Documentation and Code	139
B.1 Two-Dimensional Environment	139
B.2 Three-Dimensional Environment	203
Appendix C - Example Model Files	254
References	271

LIST OF TABLES

Table 1. Standard Bond Graph Elements	22
Table 2. Car Body and Suspension Model Equations and Parameters	65
Table 3. Response Data from Pothole Simulation	67
Table 4. Eigenvalues/vectors of Mechanical Three DOF System	112
Table A-1. Variables in Various Energy Domains	129
Table A-2. Standard Bond Graph Elements	130
Table A-3. Block Diagram Elements	130
Table A-4 Causality Possibilities for Bond Graph Elements	135

LIST OF FIGURES

Figure 1. A Design Process Description	4
Figure 2. An Engineering Simulation Activity Diagram	5
Figure 3. Interpretation of Results Diagram	11
Figure 4. Scientific Visualization and the Design Process	27
Figure 5. Dynamic Display Key to Widths and Colors	36
Figure 6. ENPORT Postprocessor Module Control Window	40
Figure 7. Dynamic Display Module Control Window	41
Figure 8. PHIGS Modeling Transformation Pipeline	52
Figure 9. 3-D Postprocessor Control Window	53
Figure 10. The 3-D Postprocessor Display Window	56
Figure 11. Vehicle System Illustration	58
Figure 12. Schematic of Car Chassis Unit and Suspensions with Seat	59
Figure 13. Macro Bond Graph of BPR Model	60
Figure 14. 2-D Bond Graph Representation of Seat Unit	61
Figure 15. 2-D Bond Graph Representation of Rigid Chassis Unit	62
Figure 16. Detail of Single Suspension Assembly	63
Figure 17. Two-Dimensional Bond Graph of Car Model	64
Figure 18. Mathematical Model of Pothole Source of Flow	66
Figure 19. Response Data from Pothole Simulation	68
Figure 20. Power Variable Response for BPR Model (Time = 0.050)	70
Figure 21. Power Variable Response for BPR Model (Time = 0.110)	71
Figure 22. Power Variable Response for BPR Model (Time = 0.115)	72

Figure 23. Power Variable Response for BPR Model (Time = 0.135)	73
Figure 24. Power Variable Response for BPR Model (Time = 0.300)	74
Figure 25. Power Variable Response for BPR Model (Time = 0.305)	75
Figure 26. Power Variable Response for BPR Model (Time = 0.325)	76
Figure 27. Glyph Display of Energy Response for BPR Model (Time = 0.135)	77
Figure 28. Time Plot of Displacement Obtained by Selecting Q88	78
Figure 29. Schematic of the Airbrake Control Valve Switch	80
Figure 30. Top Level Bond Graph Decomposition	81
Figure 31. Bond Graph Decomposition of DB-10 Module	82
Figure 32. Schematic of the Service Accelerated Release Valve (2)	83
Figure 33. Bond Graph of the Service Accelerated Release Valve	84
Figure 34. Power Variable Display for Check Valve (Time = 0.5)	87
Figure 35. Power and Effort Variable Display for Check Valve (Time = 0.76)	88
Figure 36. Flow Variable Display for Check Valve (Time = 0.21)	89
Figure 37. Flow Variable Display for Check Valve (Time = 0.6)	90
Figure 38. Bond Graph of the Main Piston System and Reservoirs	92
Figure 39. Power Response for Air Brake Model Charging Mode (Time = 18.9)	94
Figure 40. Power Response for Air Brake Model Charging Mode (Time = 31.8)	95
Figure 41. Power Response for Air Brake Model Charging Mode (Time = 49.8)	96
Figure 42. Power Response for Air Brake Model Charging Mode (Time = 58.2)	97
Figure 43. Power Response for Air Brake Model Braking Mode (Time = 12.9)	99
Figure 44. Power Response for Air Brake Model Braking Mode (Time = 14.1)	100
Figure 45. Power Response for Air Brake Model Braking Mode (Time = 19.95)	101
Figure 46. Flow Response for Air Brake Model Recharging Mode (Time = 2.85)	103
Figure 47. Flow Response for Air Brake Model Recharging Mode (Time = 5.4)	104
Figure 48. Flow Response for Air Brake Model Recharging Mode (Time = 9.45)	105
Figure 49. Flow Response for Air Brake Model Recharging Mode (Time = 14.4)	106

Figure 50. Power Response for Air Brake Model Release Mode (Time = 2.25)	108
Figure 51. Power Response for Air Brake Model Release Mode (Time = 4.2)	109
Figure 52. Schematic of Mechanical Three DOF System	110
Figure 53. Bond Graph of Mechanical Three DOF System	111
Figure 54. Flow Variable Display Showing First Mode Shape	113
Figure 55. Displacement Variable Display Showing Second Mode Shape	114
Figure 56. Displacement Variable Display Showing Third Mode Shape	115
Figure 57. 3-D Bond Graph Representation of Car and Suspensions Model . .	117
Figure 58. 3-D Dynamic Display of Response to Pothole Input (Time = 0.112)	118
Figure 59. 3-D Dynamic Display of Response to Pothole Input (Time = 0.115)	119
Figure 60. 3-D Dynamic Display of Response to Pothole Input (Time = 0.300)	120
Figure 61. 3-D Dynamic Display of Response to Pothole Input (Time = 0.305)	121
Figure A-1. Mechanical Translation Example	132
Figure A-2. Electrical Example	133
Figure A-3. Hydraulic Circuit Example	134
Figure A-4. Schematic of Radar Pedestal Control System	137
Figure A-5. Section (Macro) Bond Graph as Illustration of Decomposition . . .	137
Figure A-6. Bond Graph of the Radar Pedestal System	138
Figure A-7. Response to a Step Input	138

NOMENCLATURE

cg - Center of Gravity
e - Voltage
E - Effort
F - Flow
F - Force
i - Current
P - Momentum
P - Power
Q - Displacement
t - time
V - Velocity
w - Rotational velocity

ABBREVIATIONS

AR - Auxiliary Reservoir
BC - Brake Cylinder
BG - Bond Graph
BP - Brake Pipe
CV - Check Valve
ER - Emergency Reservoir
SV - Scientific Visualization

Chapter 1

Introduction

1.1 Problem Definition

1.1.1 Importance of Improving the Efficiency of System Design

Today's competitive global markets put a great deal of pressure on engineers to efficiently design, manufacture, and maintain engineering systems. It is no longer sufficient to produce systems which are adequate. In the future, engineers must be concerned with achieving the "best" design to meet specifications by improving device performance and eliminating undesirable behaviors. They must produce new products in the shortest possible amount of time in order to stay competitive. They also need to know how much variation in the designed system's behavior can be tolerated to determine manufacturing constraints.

The design process is intended to produce an optimum solution to a problem through iteration (Arora, 1989). This is becoming an increasingly difficult task as engineering systems become larger in scale and span many different engineering disciplines. Automobiles and aircraft are important examples of complex systems which require detailed prediction and assessment of system function to generate improvements in performance and reductions in undesired behaviors such as noise. Studying system behavior by modeling the system with experimental physical prototypes is a time consuming and expensive method for design enhancement. Some systems, such as the

proposed National Aerospace Plane designed to fly at speeds up to Mach 25, may be impossible to study to any great extent in this way.

Therefore, it is necessary to improve the practice of design by making it easier to predict and comprehend the behavior of engineering systems. Mathematical modeling and simulation of physical systems is one of the most cost effective methods for analysis and prediction of system behavior, especially for systems of very large scale. The trends of the future require mathematical modeling and simulation which promote learning and understanding about system behavior to allow engineers and designers to make changes which improve a system. This dissertation describes research to improve engineering system design through the development of tools that improve the efficiency and effectiveness of the design process.

1.1.2 The Role of Insight and Interpretation of Results

Considerable thought has been given to the subject of what components should be included in a design methodology and the order in which they should be executed. Some current views of the process of designing will be briefly described in order to set the stage for a system design progression which shows clearly some areas where improvements would be welcome.

In his book *The Principles of Design*, Suh specifies a design method which begins with the identification of "function requirements" from a given societal need. A product is created and its behavior is compared to the functional requirements. The product is then modified iteratively until the functional requirements are satisfied (Suh, 1990). Medland proposes a computer-based design process which consists of three main phases: concept, analysis, and scheming. These processes are interconnected and information can flow

between any pair of them (Medland, 1986). Shoup defines a more precise six-step process of design which includes: recognition of need, creating a design, preparing a model, testing and evaluating the model, improving the design, and communicating the design. These activities form a control loop of sorts where the design is improved until it passes the testing and evaluation stage (Shoup, et al, 1987). Manufacturability is also a key element in the design of systems, and recently much attention has been given to how this concept should be integrated into the design process (Gadh and Prinz, 1991). These models of design synthesis, along with others in the literature, can be helpful in identifying where improvements to this process are needed. In order to provide a common framework within which the improvements to the design process described in this thesis can be detailed, a description of design practice has been developed which is specific to lumped parameter system design.

Figure 1 presents a general design process description which involves several information objects. This particular diagram is only one view of the process of system design selected from the many which have been proposed, but it covers the main features of this activity as described by others, and it gives a basis for the figures and discussion which follow. The aim of an engineering design process is to fill a given societal need either by inventing a physical system to solve some problem, or by improving the performance of a given system. In this context, the designer's model is the mental picture the designer has of the structure and behavior of the physical object which constitutes the engineering system. In the case where the design is only conceptual, the internal model is the designer's perception of the components and behavior of the resulting design. The designer's model is then inherently hard to identify or control, since it exists only in the mind of the designer. The activity of design involves relating several concrete information objects to the designer's model in order to provide knowledge about potential system improvement and optimization. One information object is the specified behavior for the system to be

designed, which may be numerical or qualitative. Although the relationship between specified behavior and modeling is very important, the examination of this aspect of design is not the focus of this research. Another information object is the measured behavior. This would consist of any experimental lab testing which reveals information about the behavior of the system. Obviously if the design is only in the conceptual stage, there may not be data of this type available. The remaining information state is the simulated behavior of the system. This is information which can be obtained from mathematical modeling and analysis. This research is concerned with the shaded area of the diagram, which has been labeled analysis activity. The focus is on the interaction between the reference model in the engineer's mind and the knowledge which can be gained through the activities of modeling and simulation. It is the improvement of the relation of these two information objects that is valuable in leading to optimization of the design.

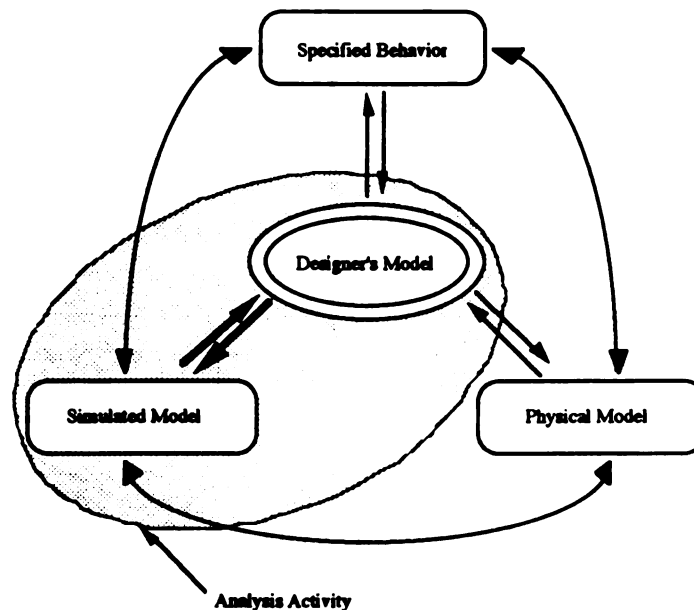


Figure 1. A Design Process Description.

1.1.3 The Role of Visualization in the Modeling Process

Figure 2 shows a more detailed simulation activity diagram which focuses on the analysis portion of Figure 1. This particular process outline is based on the use of system or bond graphs as a structured modeling technique. The nodes in the diagram are model information states, while the connectors represent transformations of information from one state to another. Paths between the nodes other than the ones actually drawn in the diagram exist, but were omitted for clarity. A brief description of each of the steps in Figure 2 follows. Some of the current computer tools available to support these stages are also mentioned.

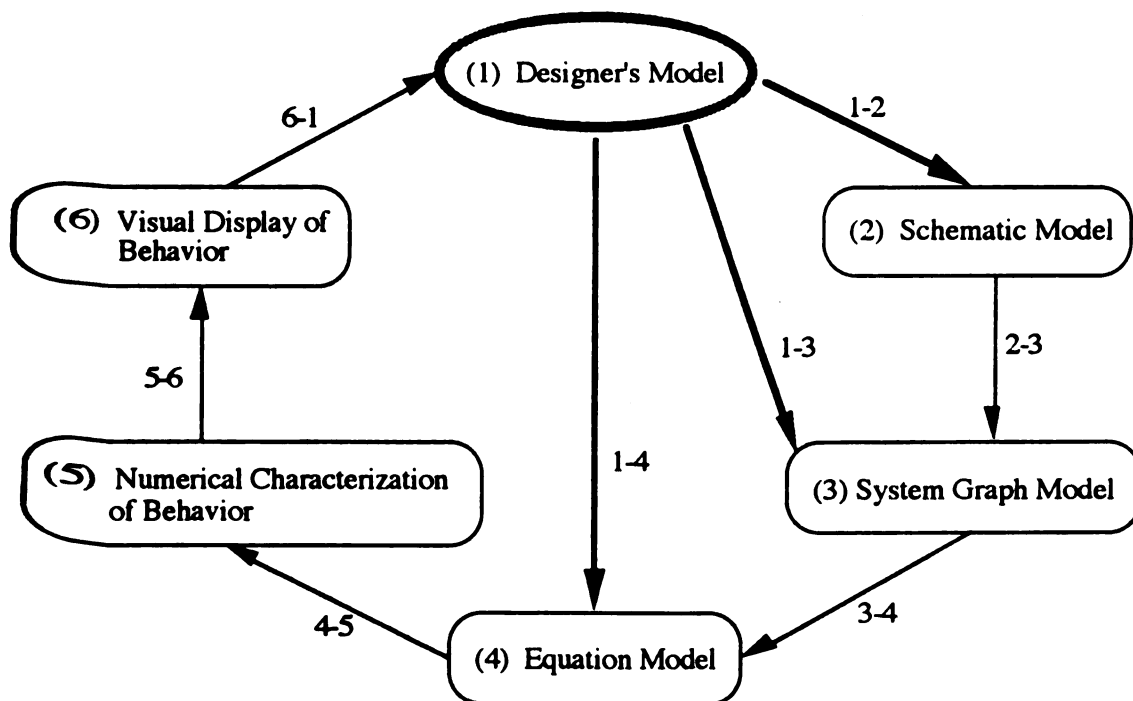


Figure 2. An Engineering Simulation Activity Diagram

Designer's Model (node 1). The first step in physical system analysis is to mentally identify the particular system under consideration. This mental model is primarily visual and has been described as "the mind's eye, the locus of our images of remembered reality and imagined contrivance" (Ferguson, 1992). Building this model includes identifying the boundaries of the system and important system variables, as well as comprehending and visualizing the basic relationships among system components. All characteristics of the system significantly affecting the behavior must be included in the designer's reference model; otherwise the results of the model simulation will not be accurate. In physical systems of realistic complexity, some relatively unimportant characteristics may have to be neglected in order to make the problem manageable.

Schematic model (node 2). The next step typically involves making further assumptions about system characteristics and sketching the operational ones on paper or on a computer screen in a form which is commonly understood. This process is associated with path 1-2 in Figure 2. The schematic model is an idealization of the physical system which includes all important aspects of system behavior to be considered, and it is usually domain specific. For example, mechanics has its own symbols for commonly used items such as springs, masses, dampers, and the connections between them, while electrical engineering has developed a standard set of circuit diagramming techniques. At this point the schematic model is an iconic one (Suckling, 1978). That is, it represents the physical system components and interconnectedness in a spatial way without holding to actual dimensions, but clearly the schematic corresponds to the topology of the actual system. For some simple design problems it may be possible to form an equation model directly from knowledge of the physical system, following a path from 1 to 4, but this is not usually practical for complex systems. The problem with obtaining an equation model from schematic information, following a path from 2 to 4, arises because different parts of a large multidisciplinary system are expressed in domain-specific modes, which leaves no easy

method for a consistent translation to the form of a mathematical model which will allow simulation. Hence the use of bond graph methods becomes desirable for multidomain engineering systems.

Software applications do exist which facilitate the transition from domain specific schematic models to mathematical models directly. Circuit modelers such as PSPICE (PSPICE User Manual, 1989), mechanical multibody systems analyzers such as ADAMS (ADAMS User's Manual, 1989), finite element solvers such as ANSYS (ANSYS User's Manual, 1993), and hydraulic software packages like PipeFlo (Shearing, 1992) all fall into this category. For example, using ADAMS it is possible to create 3-D solid models of rigid bodies and their connectors with a graphical editor, and then to animate the mathematically simulated motions of the system on a computer monitor. Such a display is very helpful in giving the designer a feel for how the design will move under different conditions. These types of software are of great benefit in designing their respective single-discipline systems, but they are not sufficient for use when integrating several disciplines into one system design.

System Graph Model (node 3). In order to put all the parts of the system model in a unified form, a system graph model can be constructed, as in path 2-3. System or bond graph modeling is a standardized modeling approach which deals with energy transfer between model components (Karnopp, Margolis, and Rosenberg, 1990). It sets up a framework into which systems from most domains of engineering can be translated. Information in the form of control signal and block diagram elements can also be included, in which case the bond graph becomes a system graph. A system graph is a topological model, presenting the components of a system and their connections with no regard as to their actual geometry or spatial placement. The advantage of developing a system bond graph is that a complete operational mathematical model of the system behavior can easily

be obtained, and computer support is available for this task. However, the disadvantage of formulating the model in bond graph form is that the apparent correspondence between the model and the physical system is weaker than it is with a schematic. A two-dimensional system graph representation requires visualization effort on the part of the designer when he or she is considering the behavior of a three-dimensional physical system.

Computer software exists which can aid the engineer in creating and manipulating bond graph models as well as in deriving equation models and in calculating numerical solutions. One particular program is ENPORT, which has been designed to implement bond graph modeling techniques (Rosenberg, 1993). The bond graph elements and structure can be input by using a two-dimensional graphical editor. Another bond graph modeling environment is CAMAS (Broenink, 1990). CAMAS is an interactive computer language developed to implement bond graphs. The graphical editing capabilities also permit bond graph development represented in two dimensions.

Equation Model (node 4). A mathematical equation model contains all relevant quantitative information about a system in question. The equation model allows prediction of the responses of an engineering system to various conditions without implementing them physically. For lumped parameter systems, the mathematical models generally consist of sets of ordinary differential equations. However, for many designers, there is very little conceptual correspondence between a three-dimensional physical system and a set of equations. The abstractness of this type of model is what generates the need for intermediate modeling techniques like schematics and bond graphs. It is also very difficult to compose the large sets of equations required to describe complex systems without a structured graphical model. Equations can be generated automatically from system graph models (path 3-4) using ENPORT or from discipline-specific schematics using programs like PSPICE.

Numerical Characterization of Behavior (node 5). At this point the model building task is complete. Given the equation model, computers can be used to perform the simulation (path 4-5), usually by solving the mathematical equations iteratively, since most practical problems do not have closed-form solutions. Various algorithms exist which are capable of performing this task. As real-world systems design is becoming increasingly complex, continuing increases in computational speed and memory size allow for more and more data to be generated related to a system under analysis.

Many computer programs exist for general numerical solution of equations. These facilitate the creation and solution of equation models obtained from physical systems or schematics and can be identified as direct equation or block-diagram type programs. Examples of these are the MATRIX-x software environment (MATRIX-x User's Guide, 1990), and MATHEMATICA, a powerful mathematical symbolic equation solver (Wolfram, 1991). These programs provide two and three dimensional output, but variables are not necessarily directly related to the physical system. Difficulty in using these programs is also encountered when the systems to be analyzed are very large involving many mathematical equations. Programming in this case can be extremely complicated. Many single-discipline modeling programs such as PSPICE have their own solvers based on commonly used algorithms. ENPORT and CAMAS also have their own built-in equation solvers.

Visual Display of Behavior (node 6). A critical part of the simulation process is to take the resulting response data from a numerical simulation and interpret it in a meaningful way so that the physical system itself can be better understood (path 6-1). This involves both quantitative analysis of simulation results and qualitative analysis leading to understanding of global system function. The important features of the system response

need to be highlighted. A designer who understands how a system behaves will be equipped to make the right changes to improve the design. Rosenblum and Brown call this process "realization" (Rosenblum and Brown, 1992). Appropriate displays of system response can greatly improve the interpretation of results process.

Figure 3 shows some of the possible ways to relate visualization to model information states in order to apply simulation results to the designer's conceptual model. The designer visualizes system response in terms of (2), (3), or (4) depending upon which model building strategy was used. If the system is single-domain and has an accepted iconic representation (e.g. rigid bodies or electronic circuits), then the most useful display is the link between (6) and (2); this is denoted (A) in Figure 2. This type of display has been implemented in many domain-specific computer programs. For example, ADAMS shows motions of rigid bodies; PSPICE displays currents related to electric circuit diagrams. Linking the results directly to the equations (C) would allow plotting of variables versus time or cross-plotting of variables, one versus another. Often, however, the degree of abstractness of an equation model makes visualization difficult.

Path (B) can be followed to link simulation results data with the system graph in a display which could show system variables (e.g. powers, energies, forces, currents) directly related to the system graph. By linking results data directly to the system graph in a computer environment, the engineer can take advantage of the structure provided by the bond graph model to understand the overall meaning of the results data. It is this visualization process that is enhanced by the work described here. The usefulness of a coherent design package based on the bond graph modeling technique which aids in all steps of the design process, especially the interpretation step, is becoming clear.

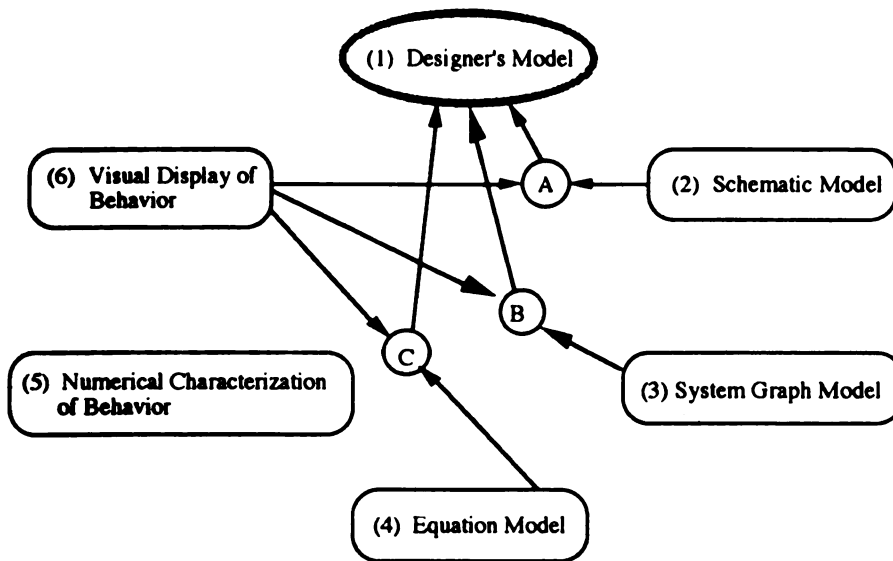


Figure 3. Interpretation of Results Diagram

1.2 Research Objectives

The principal objective of this research was to create and implement a significantly improved design environment for engineering systems based on the use of bond graph methods as a structured modeling technique. A common practice in the context of dynamic system analysis is to build a system graph model, to choose a set of parameters for defining the constitutive equations of each of the elements of the model, to run a numerical simulation of the model for given initial conditions, and finally to compare the output results to the expected or required behavior. In this way an engineer can determine if the model is behaving properly (model verification). Once the model is verified, the engineer can experiment with changing parameters to improve the system performance compared to the design requirements for the output response. The types of questions asked by the designer after a simulation might be: Which subcomponents of the model are most important in determining overall system behavior? Which components use very little power and might

be eliminated to reduce the computational order of the model? Which parts of a mechanical model move the largest distances over the course of a dynamic response? Which voltages in the electrical part of a model fluctuate the most and do they remain within required levels? Do the power levels of all the elements stay within reasonable limits for the chosen parameters and initial conditions? A major goal of this research was to design an environment which makes the interpretation of dynamic system response easier for designers and makes answering these types of questions as straightforward as possible.

This objective was met in two separate stages, for each of which prototype softwares were implemented. The first stage concentrates on improving the interpretation of simulation results by providing a power-related animation directly tied to the 2-D system graph. Current 2-D bond graph computer models were enhanced to provide time-varying power information on the system graph. Computer animation is a natural way of visualizing time-varying results obtained from a simulation. The animation shows global patterns of behavior as well as focusing on individual quantitative details upon demand. Energy transfer and/or other variable value information being indicated by color and line-width variation or glyph-type symbology as an animation in time was explored. The line-width variation technique has previously been implemented in ENPORT to display time-averaged power variables on the system graph (Rosenberg and Zhou, 1988). This approach was extended to dynamic display of time-varying quantities on the system graph, thus providing an energy and information-based animation of the system behavior. Various scaling options are made available to the user to provide for selection of a scaling which most clearly shows the dynamics of the energy propagation in the system. A method was developed for representing the changing direction of each energy flow. Other display codes were also used which involve changes in glyph drawings to reflect the energy change in a node with time. Once these concepts were studied in a 2-D environment, they were applied to the next phase.

The second stage involved building a pilot environment for using three-dimensional system graphs. Three-dimensional visualization is necessary because better correspondence between topological system graph models and actual physical systems or schematics will help the designer apply his or her physical intuition to a model in a more meaningful way. A topological model in general is helpful because it conveys the relationships between variables in the system, but it lacks a direct relationship to the spatial characteristics of the system. An engineer can conceptualize about a physical system because it can be pictured visually in space, but at the point where a two-dimensional bond graph model is constructed, direct correspondence to the physical system is lost to a large degree. In this work, a system graph was allowed to be more than a topological model alone. A graphical environment was developed in which system graph display can correspond directly to the geometric imagery of the physical problem. The concept can be thought of as 'true geometry' in the sense that a three-dimensional system graph can be a scaled representation of a physical system with its elements fixed to the spatial positions of the matching elements of the physical system. Capability for dynamic rotation of the display was provided in order to achieve an actual three-dimensional effect (Hays, 1992). The user is also allowed to manipulate the model as a geometric object in the computer environment. The result is better visualization of the power flow pathways captured in the three-dimensional structure of the bond graph.

The principles of the emerging field of scientific visualization as expressed by some of its proponents (Tufte, 1983 and 1990, Cox, 1990) were used to make the visualization of each stage clear and easily understandable. In the past, traditional methods have been successful in providing knowledge of system behavior to designers, but the production of ever larger, more complex, and interdependent systems require substantially improved design methods like the ones described here, in order to promote comprehension and

learning. This work makes a significant contribution to the area of complex system design by providing a new tool for engineers which complements existing tools. Since the nature of the project is closely tied to computers and their ability to produce color graphics and animation, color printouts and photographs are included in this thesis wherever possible, and a videotape of the example systems as implemented in the pilot 2-D and 3-D computer environments is available upon request from the author directly or through the Mechanical Engineering Department of Michigan State University.

1.3 Description of Thesis Organization

This thesis consists of the sections described in this paragraph. The current chapter contains an introduction to the problem and motivation for the research as well as the order of presentation of further contents. Chapter 2 contains background information relevant to the problem, in particular a short description of bond graphs and a discussion of scientific visualization principles and practice. Chapter 3 includes a complete description of the two-dimensional visualization environment created in answer to the problem statement. Chapter 4 covers the extension of bond graphs to a three-dimensional representation and visualization of results in the three-dimensional environment. Chapter 5 illustrates the use of the designed environment for solving some problems of practical scale, namely an air-brake switching system and a car and suspension model. It also contains a discussion of how the designed environments can be used in educational situations. Finally, Chapter 6 presents a summary of what was accomplished and some ideas for future development. Appendices are included which contain more details on certain topics related to this thesis. Appendix A includes a more comprehensive explanation of bond graph modeling for readers who are not familiar with the techniques. More details on the structure and coding of the visualization software implementations are shown in Appendix B. The complete description of the Air-brake Switch bond graph model and its equations, as well as the

ENPORT model files for both the car and suspensions example and the air-brake switching system, are included in Appendix C. This document concludes with a list of references of particular use in carrying out this research.

Chapter 2

Bond Graphs and Scientific Visualization

2.1 Using Bond Graphs as a Tool for System Analysis and Design

2.1.1 Modeling of Engineering Systems

In the introductory chapter of this thesis the importance of modeling in the process of designing a physical system was emphasized. Models exist in many different forms, but in all cases their purpose is to convey to an engineer as much information about the system's behavior as possible, and to provide the information in such a way that the underlying concepts behind the behavior are clear and insight into the problem is gained. An engineer cannot design a system without a reasonable conception of the way the system operates and the influence that parameters have on the behavior.

Certainly it is possible to learn a great deal about a system's behavior by building and running tests on the system itself. Usually this is not a practical way to approach design and it is not the most efficient. Once a system has been built, it is very expensive to make the changes in the design that might be necessary in order for it to meet performance criteria. Optimization in this situation is very difficult. Scaled models of systems, which presumably represent the behavior of a larger system but on a smaller scale, can also be useful in yielding insight about the system. Unfortunately, these types of models (wind tunnel mockups, for example) incur some of the same disadvantages as full-scale prototypes. They are still very expensive and time-consuming to build and are hard to modify once they have been built.

In order to learn about system behavior without building physical prototypes it is very useful to construct mathematical models of systems. This practice is often called "virtual prototyping" (Puttre, 1992), since the idea is to identify the behavior of a system within a computer environment in order to replace or supplement what can be learned about system behavior from the construction of physical prototypes. These virtual prototypes or models capture the physical behavior of the system in their structure and can be simulated to provide information about the system's response to different inputs and changes in system characteristics. The mathematical models used to represent lumped parameter systems typically occur as sets of ordinary differential equations. Parameters in the equations can easily be changed to reflect design modifications, and unnecessary hardware costs are avoided. Solving the equations numerically for given initial conditions and parameters over a certain time period provides a prediction of the actual response of the system, and if the virtual prototype accurately captures the important aspects of the behavior of the system, it can then be used to see the effect of changes in the parameters or conditions under which the system is operated. In this way the design of the system can be improved and optimized.

Unfortunately, obtaining a set of mathematical equations for a system of realistic complexity is not a very straightforward task and can be prone to error. The large number of variables and parameters involved, along with the complex relations between those variables, make the problem difficult to approach. To make the transformation from physical system to equations easier, various types of graphical modeling systems are used. Graphical modeling can be seen as a "bridge between the mathematics of differential equations and the physics of engineering systems" (Rosenberg and Karnopp, 1983). Graphical models are easier to understand in terms of the connectivity of the various parts of the system model than are the equations. Graphical models are more closely related to

the geometry of the original system. They are appropriate for managing the information in large-scale systems in a hierarchical way. In general, the more removed an engineering model representation is from the geometric form of the physical system, the harder it is for an engineer to understand it intuitively.

Different types of graphical modeling are involved in the process of going from a physical system to a set of mathematical equations. One type is the schematic diagram. Schematic diagrams help to capture the attributes of a system in order to make equation derivation easier, but quite often schematic diagrams are not precise enough in the definition of variables to make the process of equation formulation exact and unique. For instance, in mechanical schematics, often certain assumptions are not made clear in their structure, such as whether friction applies to a given motion or not, or whether heat losses from a component are important and should be considered. It is also difficult to connect schematic models which have been constructed in different engineering disciplines, since the notations are quite different in each. For example, in mechanical schematics certain symbols are commonly used to represent masses, springs, and dampers. In electrical schematics certain symbols represent resistors and other electrical circuit elements. However, the connections between these two different schematic domains are not well defined in either terms. Another common graphical modeling system is the block diagram. Block diagrams clearly indicate the existence of functional relationships among variables in the system, but the structure of the graph does not take into account the physics of the problem in a direct way unless the entire system is signal-based. Block diagrams are closer to mathematical equation models in that they show variables directly as the connectors and represent functional relationships by the nodes (i.e. blocks).

Bond Graphing is a very useful modeling technique which can be used to avoid some of the problems associated with both schematic diagram models and block diagram

[illegible]

models. Bond Graphs are topological in form and therefore provide a close relationship between the physical device and the model, while still providing a clear representation of functional relationships in the model which can be translated directly into mathematical terms. Bond graphs are especially valuable in cases where the system to be considered spans more than one energy domain because they can represent energy transduction in a graphically consistent manner. A brief description of bond graph modeling methodology is included in the next section.

2.1.2 Bond Graph Modeling Methodology

Bond graphing is a structured modeling technique used to represent multi-domain systems as a series of components linked by power bonds. The systems best modeled by bond graphs are lumped parameter systems which can be mathematically modeled as sets of ordinary differential equations. Bond graphs provide a basis for detailed physical modeling of engineering systems in preparation for simulating their behavior. The graphs capture the flow paths for energy within a system in a way which agrees with the underlying physical laws of the components (Karnopp, et al, 1990). Some examples of bond graph models can be seen in chapter 5, where the visualization examples are described. Additional examples appear in a brief introduction to using bond graphs for system modeling included in Appendix A.

Bond graphs support hierarchical decomposition of large-scale systems by dividing them into subcomponents and connections at different levels. Each of the connections represents an energy transfer, or power, at that connection. The power is the product of a pair of variables, denoted effort and flow. If consistent units are used, power units can be equated across different engineering domains (e.g. watts). Energy measured consistently can then be seen to flow through the connectors from one domain to the other. This is one

feature that makes bond graphs useful for modeling multi-domain systems. It keeps the bond graph much closer to the physics of the system than a block diagram, which does not track energy flow directly.

An additional characteristic of bond graphs is that they are highly organized in the representation of information in the model. The bond graph modeling process can be followed in a very structured way, which makes model building clearer and less prone to error and confusion. Bond graphs are also very precise statements of which model effects are included. They make clear all of the assumptions made in the modeling process at the energy level. They do not indicate the functional details of the effects.

At the lowest levels of a bond graph decomposition of a system, a relatively small set of bond graph elements can represent the physical behaviors present in a remarkably broad set of components and effects. Bond graph techniques recognize the similarity of physical behaviors across different energy domains and use this energetic similarity to produce concise, unified models. For example, in electric circuits a linear resistor behaves as a device which drains power (P) from the circuit at a rate equal to the current (i) in the resistor multiplied by the voltage (e) in the resistor ($P = i * e$); correspondingly in a mechanical system a viscous damper also dissipates power from the system and does so at a rate of the velocity (V) multiplied by the force (F) in the damper ($P = V * F$). The constitutive equations for these two effects are the same and bond graphs take advantage of that fact to produce coherent models. Although this example demonstrates linear relationships between variables, non-linear effects can readily be represented by bond graphs also. The process of using bond graphs to model a system often enhances the modeler's knowledge of the system and can also yield insight into the system's behavior, even before simulation has been performed.

The set of standard bond graph elements includes several element or node types. These are shown in Table 1. Source elements supply energy to the system. Passive elements describe observed types of physical behaviors, such as the dissipative effect mentioned above. Two-port elements support the modeling of energy transformation from one domain to another. Junction elements serve to connect these elements together in a way which is consistent with the physical laws governing the behavior of the component. Each element has its own constitutive equations which describe the behavior of that element. These equations can be either linear or non-linear. Once a complete bond graph of a system is constructed using these elements, causality can be added to graphically indicate which variables are inputs and which are outputs within the system. Bond graphs can also be combined with block diagram elements and signals to allow for the study of different types of control structures and their effect on system behavior. A bond graph that integrates control signals through block diagrams with bonds is called a System Graph.

System graphs support the direct formulation of a set of state space equations from the model. State space equations are a useful form for doing system analysis or control design. At this point the equations can be numerically solved and simulations can be run for various test conditions. A critical part of the system analysis process is to take the resulting response data from the numerical simulation and interpret it in a meaningful way. Advantage should be taken of the structure provided by a bond graph model in the display of the numerical results of a simulation, since the bond graph already has a special correspondence to the physical system in the designer's mind.

Table 1. Standard Bond Graph Elements.

Element Type	Element Name	Bond Graph Symbol	Constitutive equations	
			General Form	Linear Form
Sources	Source of Effort	SE $\begin{array}{c} \xrightarrow{E} \\ \xleftarrow{F} \end{array}$	$E = f(\text{time})$	$E = \text{Constant}$
	Source of Flow	SF $\begin{array}{c} \xrightarrow{E} \\ \xleftarrow{F} \end{array}$	$F = f(\text{time})$	$F = \text{Constant}$
Passive One-Ports	Resistance	R $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$E = f(F)$	$E = R * F$
	Capacitance	C $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$E = f(Q)$ $F = dQ/dt$	$E = 1/C * Q$ $F = dQ/dt$
	Inertia	I $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$V = f(P)$ $E = dP/dt$	$V = 1/L * P$ $E = dP/dt$
Two-port transducers	Transformer	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} \text{TF} \begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = n * E2$ $F1 = 1/n * F2$	
	Gyrator	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} \text{GY} \begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = r * F2$ $F1 = 1/r * E2$	
Junction elements	One-Junction	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} 1 \begin{array}{c} \xrightarrow{E3} \\ \xleftarrow{F3} \end{array}$ $\begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$F1 = F1 = F3$ $E1 - E2 - E3 = 0$	
	Zero-Junction	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} 0 \begin{array}{c} \xrightarrow{E3} \\ \xleftarrow{F3} \end{array}$ $\begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = E2 = E3$ $F1 - F2 - F3 = 0$	

2.1.3 Computer Support for Simulation

Computer support for the activities of modeling and simulation is necessary to enable practical design work. The typical activities involved in simulating system response are selecting appropriate parameters for all of the constitutive equations of the bond graph model, then solving for the dynamic response for some selected initial conditions. The purpose of the simulation at first might be to see if the model performance matches the expected performance. This is the model verification stage. Once the model has been shown to behave properly, it can be used to test the system performance's sensitivity to changes in parameters. The response can be optimized according to some performance criteria. A variety of output data types can be used to generate information about the model's behavior, such as time-history data, linearization and eigenvalue data, steady-state values, and sensitivity of these values to system parameter changes.

Several valuable services are provided by computer simulation to make the process of design iteration and optimization more efficient. The first is that computers can be used to remove the tedium of deriving the actual ordinary differential equations sets, and to make construction and modification of models easier. Mistakes can also be avoided which might occur in equation derivation due to the complexity of the problem. With good computer support, the user can concentrate on modeling and design decisions, since the computer can handle the updating of equations as changes are made. Second, computers are useful in numerically solving the mathematical model, since most problems of practical size do not have closed form solutions. Third, one of the most important functions of the computer in system modeling and simulation is in the capability it provides for display and examination of results data. Computers allow the user to explore the data, looking at it from different perspectives so as to gain insight into the system behavior.

Bond graph models lend themselves to algorithmic derivation of the system equations. Therefore computer support for bond graph modeling can be provided and system behavior can be simulated effectively at the detailed numerical level. For example, the ENPORT software accepts bond graphs and block diagrams as components in an integrated modeling environment (Rosenberg, 1993). ENPORT makes construction and modification of system graphs a straightforward graphical process, as well as supporting the equation derivation and solution processes. ENPORT can also be used to display the results data in X-Y plots of variables versus time. Display of time-averaged power variables is directly on the bond graph as color and line-width variation of the bond structure (Zhou, 1988). This may not be adequate if transient phenomena are considered, which is quite often the case in complex system design. It is also possible to pass the equations from ENPORT in symbolic form to other software that can interpret the equations; this is another level of manipulation that can lead to insight.

2.2 Scientific Visualization

2.2.1 Description of the Field

Scientific visualization, also referred to as data visualization, is emerging as a means to help answer the need for clearer ways of presenting response data to designers. As ever more powerful computers and measuring instruments generate ever more data, we face a need for more effective ways to interpret and use that data. One definition of scientific visualization is "using computer-generated graphics to help us understand and visually clarify the relationships inherent in data" (Rosenblum and Brown, 1992). Another is that scientific visualization "uses computer generated images to help scientists extract knowledge and understanding from experimental or simulated data" (Nielsen, 1993). A National Science Foundation report was recently published which encouraged engineers and

scientists to develop new visualization methods to promote both research and education (McCormick, DeFanti and Brown, 1987). NSF also recently awarded \$14.6 million to establish the National Science and Technology Center for Computer Graphics and Scientific Visualization, illustrating just how much importance is placed on progress in this area. The research reported upon here is an attempt to meet this challenge in the specific area of dynamic engineering systems design.

Scientific visualization addresses the problem of data interpretation by combining several different computer-related disciplines. Scientific visualization depends on computer expertise relating to graphics and information processing, as well as on expertise related to the different scientific domains in which visualization is used. There is also an artistic component that involves aesthetics, and a biological component involving theories of human perception.

How humans perceive things and what is meant by understanding are issues that are not entirely defined at this time. The field of human perception attempts to explore these issues by observation and experimentation. This field relates directly to scientific visualization because knowing how humans relate to data and the physical world around them is a prerequisite to creating the most effective design and simulation environments. Like a computer, a human undergoes a specific process used to internalize information. Unlike a computer, this process is somewhat hard to identify. If some aspects of this process are examined, information can be organized in such a way as to take the best advantage of human information processing capabilities.

It has been shown that data which are organized in a structured fashion are understood and remembered better than data presented in an unorganized manner (Haber and Wilkinson, 1982). It has also been shown that humans are capable of comprehending

more complex concepts visually than through words and numbers alone (Haber and Wilkinson, 1982). The visual system provides a way to organize even non-pictorial data in a geometric structure. Various studies have shown that the natural way for people to relate to data is pictorially. Visual data is remembered better than words or numbers (Standing, Conezio, and Haber, 1970), and our visual systems automatically attempt to interpret flat pictures as if they are real geometries in three dimensions.

Another facet of human perception is that humans tend to remember and relate to data better when they have control over their manipulation of the data. This is why user-driven interactive visualization is a goal to be striven for. There are also certain ways of presenting data which distort the human perceptual process and can be used to trick the user. Although there is no motivation to distort data in an engineering modeling problem, since the goal is accurate understanding of system behavior, care must still be taken to avoid unintentionally presenting data in a way which might be misinterpreted. Obviously, a good design and simulation environment should avoid any possibility of confusion.

For engineering modeling and simulation, scientific visualization provides a way to make interesting features of the simulated behavior stand out rather than be obscured by numerical data. The challenge of applying the concepts of scientific visualization in a design context is to create computer-aided visualization environments which help engineers to comprehend the physical significance of their simulation models' behavior. The idea of visualization can also be applied in engineering education, where visual thinking and mental imagery need to be stressed in order to promote better understanding of engineering fundamentals.

Scientific visualization concepts can be used to support better understanding of bond graph model analysis by providing two types of capabilities. These are illustrated in

Figure 4 where the connection between the designer's model, the bond graph model, and the visual display of model behavior is highlighted. The first capability involves presentation of data obtained from simulations by relating the response directly to the bond graph structure (Path Y). The second capability increases the correspondence between the bond graph model and the designer's models, and therefore the physical system, by associating the bond graph with the system geometry in three dimensions (Path X).

2.2.2 Current Tools and Research Areas

The field of scientific visualization is used in many different scientific domains and in a variety of ways. Applications are found in disciplines as diverse as medicine, meteorology, engineering, architecture, computer programming, physics, chemistry, statistics, and mathematics. These applications range from the simple display of two-dimensional scatter plots to complex three-dimensional mappings of data to geometry.

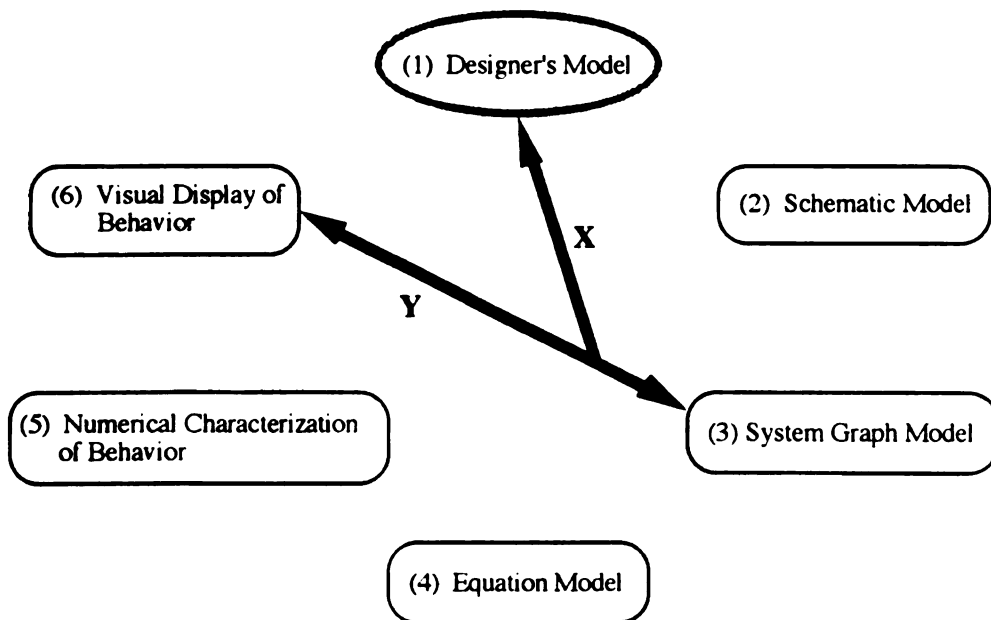


Figure 4. Scientific Visualization and the Design Process.

One of the major areas of current development in scientific visualization is the display of continuous or sampled data in two or three dimensions. This visualization process involves a series of transformations which provide a mapping between raw data and certain graphical element attributes. Some examples of programs which support the display of general distributed data values are: 1) PV-Wave (Schmalzel, 1992), which includes 2-D line and scatter plots, 3-D contour and surface plots, as well as table and animation data views; 2) Data Visualizer by Wavefront Technologies (Brittain, 1990), which can handle distributed fully structured as well as unstructured data; 3) AVS, a commercial module-based visualization system (Upson, 1989); 4) Visage, which supports distributed visualization (IEEE Visualization, 1992); and 5) the equation solvers like MATHEMATICA and ANSYS previously mentioned. Such programs require the user to decide on a visualization method and to direct how the raw data will be displayed. It was decided that these stand-alone general visualization softwares were not sufficient for the required bond graph design environment although they are complementary. The structure supplied by the bond graph model should be exploited for understanding as fully as possible.

Another area which is being extensively studied is distributed system visualization known as scenic or quasi-scenic rendering (Becker and Cleveland, 1991). The goal of this type of visualization is to map information obtained from experiments or simulations onto their associated distributed geometric surfaces or volumes. This covers areas such as volume visualization of medical data (Fuchs, et al, 1989) and computational fluid dynamics (Giles and Haimes, 1990). Similar techniques are being applied to stresses and strains in engineering mechanics (Haber, 1990).

Although distributed parameter systems, which in a modeling context involve the solution of partial differential equations, have been and are being extensively explored by

scientific visualization techniques, less has been done in the area of spatially lumped or finite dimensional systems, which involve the solution of time dependent ordinary differential equations. Imaging methods are needed in this area to provide as much detailed information to designers as possible within a global context that corresponds well to the graphical model and the physical problem.

For this research an area of application for scientific visualization principles has been identified which has not been explored up to this time, namely the application to bond graph modeling and display of simulation data. The structure of this type of visualization is lumped, rather than distributed over a volume, since the positions of nodes and connectors have been defined, and the goal is to use this structure to do modeling that relates to physical systems and to display data in a meaningful way. The display can be thought of as a problem of high-dimensional data, since 3 physical dimensions (x , y , z) are dealt with as well as time and the values of the variables of interest. The topic of display of high dimensional data has not been an area of a lot of research, perhaps because it is so complex. The research that has been done includes the use of iconic representations like Chernoff faces (Chernoff, 1973), and some work which is concerned with showing a dependent variable which is a function of many independent variables (Beshers and Feiner, 1992 and Mihalisin, et al, 1991). The bond graph output display problem is somewhat different in that it is necessary to show the values of many dependent variables (associated with the powers on all the connectors of the bond graph) as a function of time in the context of their position in the bond graph model.

Chapter 3

A Two-Dimensional Bond Graph Visualization Environment (BGVis-2D)

3.1 Environment Specifications

The goal of this visualization environment is to relate the response data calculated by a solution of the system equations obtained from the bond graph model to the structure of the bond graph model. The response is usually determined by a numerical solution of the equations, but could be obtained from a closed-form solution in the case of linear systems. Since the designer presumably understands how the bond graph model relates to the actual physical system under study, this will organize the output data in a way that is highly consistent with the designer's understanding of the physical system. Since bond graphs are designed to capture power paths within a system, some useful quantities to display will be the power and energy values. What is needed is a way to display the instantaneous value of a variable type for each of the bonds in a system graph as an animation in time.

The common method for displaying results from a bond graph model simulation consists of plotting one or a small number of variable values versus time on a two-dimensional screen or hardcopy. Output from block diagram or direct mathematical equation model solutions are displayed in this same way, indicating that the structure of the bond graph itself is not being utilized in any way to aid in the organization and presentation of the output data. This approach also requires that the user choose the variables of importance for display. This implies some prior knowledge or expectation of

which elements in the system need to be checked, which may not always be obvious. Displaying a larger number of variables in an organized fashion would allow the significant aspects of the global system behavior to become clear. Then relevant variables can be identified and plotted if the user requires more detailed information.

One existing possibility of taking advantage of the bond graph structure for displaying data is the option of displaying time-averaged variables in a static display as line width and color variation on the bond graph (Zhou, 1988). This type of presentation is very helpful in separating sections of the graph which demonstrate high versus low averages of variables, which might lead to model reduction, but it is not necessarily adequate for dealing with the interpretation of transient system response. The Zhou method of average value presentation can be used as a base to extend the display of variable values to a dynamic animation in time.

The visualization environment should be smoothly integrated as part of a data post-processor into a bond graph model building and solving environment. As much control of the display as possible should be given to the user to allow him or her to interact with the data in ways that are most meaningful. The user interface should, of course, be clear and easy to understand and manipulate.

3.2 Design of the Two-Dimensional Visualization Environment

3.2.1 Features Selected for Implementation

The variable types chosen for potential display on the bond graph are the power variables, effort and flow, as well as the power variable itself, and also the energy variables displacement and momentum, as well as the energy variable itself. Effort (E)

and flow (F) variables for each bond over every time stage in the time range of solution are obtained from the simulation and stored in the results file. Power (P) variables for each bond are calculated in the dynamic display postprocessor by multiplying the effort and flow on each bond for each time stage. The momentum (P) and displacement (Q) variables are not stored as a result of the simulation, except for those that are state variables. In order to calculate all of the P and Q variable values over the solution time range, the fact that effort and momentum and flow and displacement are related by the integral relationships $P = \int E dt + P_0$, and $Q = \int F dt + Q_0$, respectively, was used. A trapezoidal integration routine was implemented to calculate the momentum variable from the effort variables for each time step, and the same procedure was used to calculate the displacement variable from the flow variables for each time step. The following equation defines the trapezoidal integration.

$$\int_a^b x(t) dt \approx \frac{b-a}{2n} \left[x(t_0) + 2x(t_1) + 2x(t_2) + \dots + 2x(t_{n-1}) + x(t_n) \right]$$

For the integrations, all of the initial values for displacements and momentums were taken to be zero. Instantaneous energy variables were determined by integrating the product of the flow and effort on each bond for each time step.

Once the variable values over all the bonds for the time range of interest are available, each variable type is checked for global maximum and minimum values, and magnitude ranges are determined. This allows each variable type to be scaled consistently over all bonds. Three types of scaling options are available: linear, logarithmic and absolute logarithmic. The logarithmic scaling options normally are useful when values of a variable type extend over several orders of magnitude. Logarithmic scaling can only be used if all variable values for the variable type to be displayed are greater than zero. The dynamic display checks for this condition, and prints an error if an attempt is made to scale negative or zero-valued data logarithmically.

Absolute logarithmic scaling uses only the absolute magnitudes of the values, where the negative sign on the value is indicated by the arrow direction or color change. Thus the problem of negative numbers is avoided. Zero values are set to the minimum magnitude value for the variable type to avoid the error associated with taking the logarithm of zero.

Bond Coding. The two primary modes for time varying display of bond related variables are line width variation of each bond and color variation of each bond. This method can be thought of as "flow" display, as opposed to "level" display which is described under the heading of Node Coding. Line width variation and color variation methods can be used independently, allowing two different variable types to be displayed at once, or they can represent the same variable type, thus providing reinforcing display variations for the same variable type.

Ten levels of difference variation are provided for the line width modulating feature. The choice of this number was affected by the geometry of which the display device is capable. A maximum line width was determined by the relationship of bonds to nodes in the presentation of the graph, while the minimum line width was determined by the minimum line width allowed in the line drawing routine. After some experimentation, it was determined that ten gradations of line width between the maximum and the minimum provide the user with variation sufficient to detect meaningful differences in variable values. Negative values for a given variable are indicated by a shift in the arrow placement on the bond for linear and absolute logarithmic scaling methods. This is an especially natural way to think about power flow, since by definition power flow on a bond is positive in the direction of the half arrow.

Ten levels of color variation are also provided. This number could be increased for more detailed data display since human perception of color variation is more sensitive than perception of geometry differences. When line width and color are used to show the same variable type, only one set of colors, shaded from purple to red, is used, since the arrow directions indicate negative results. When color and line width are used to represent separate variables, two sets of color are used, the same as above for positive values, and a set shaded from green to blue for negative values. The two distinct color sets allow the negative versus positive data to show up very clearly. Another possibility is to use the VBGYOR (rainbow) spectrum to show value level change, but it is not clear that this color set is always intuitive for people in comprehending the corresponding levels, and the position of a zero-valued level is not obvious. For each time stage of the animation, the bonds are redrawn with the appropriate width corresponding to the variable value type selected for line-width and the appropriate color corresponding to the variable type selected for color.

Node Coding. During a results display the nodes of the graph are drawn once in the colors originally defined by the model builder relating to energy domain: white for general, blue for mechanical, green for electrical, purple for hydraulic, and so on, as defined in the ENPORT model building module (Rosenberg, 1992).

A secondary visualization tool that is supplied in the BGVis-2D tool, associated with nodes or "levels" rather than bonds or "flows", is a glyph symbol located at each C, I and R node in the graph. This method can be used by itself or in combination with the previously described "flow" type display. A glyph is a geometric symbol that represents information at a given position. The glyph is based on a thermometer motif and shows the amount of energy stored or dissipated in that element at a given time. The energy increase or decrease is shown by an increase or decrease of the line height in the

thermometer-style glyph. The glyph technique might also be extended to include display of other variable types that can be viewed as "levels", such as displacement.

The net energy transfer to any node in a bond graph can be determined by integrating the instantaneous power values on each of the bonds connected to that node and summing the results. The C and I elements are known as energy storage elements because the amount of energy stored in the element can be determined from the net energy transfer plus the addition of any initial energy already stored in the node. The user is allowed to supply a numerical value for the initial energy stored in each C and I element. The amount of energy dissipated in an R element is determined directly from the net energy transfer. The glyph display at these nodes can be useful for the designer trying to understand where energy is being accumulated and released throughout the system globally during a simulation.

Signal Coding. Signals are not included in the dynamic display of power, energy, effort, flow, momentum, or displacement variables, since the power flow along a signal is by definition equal to zero, and because there is no consistent way to indicate to which variable type the information carried by a signal belongs. If bond graph builders were encouraged to label signal values with the first letter of the label corresponding to the variable type (E, F, P, or Q), then a method might be set up to include the appropriate signals when a given variable type is displayed. At this point, signal variable values can be displayed independently from bond variables by choosing to display a "signal" variable type.

A key is provided to indicate the correspondence between line widths and colors and numerical value levels. An example of the key is shown in Figure 5. Each line width is drawn in the first column, and the range of numerical values it represents is indicated

above and below it. In the second column, each color is demonstrated, both negative and positive, and the numerical values corresponding to that color are printed. In the case where no variable is chosen to be displayed with line width variation, the line widths are drawn without numerical labels. Similarly if no variable is chosen for color variation, the colors are still shown, but are also unlabeled. The key is updated whenever new variable types are chosen for display as either line width or color variation. At the bottom of the key, the value of the time for which the data is currently being displayed is shown. This value is updated whenever the time step displayed is changed.

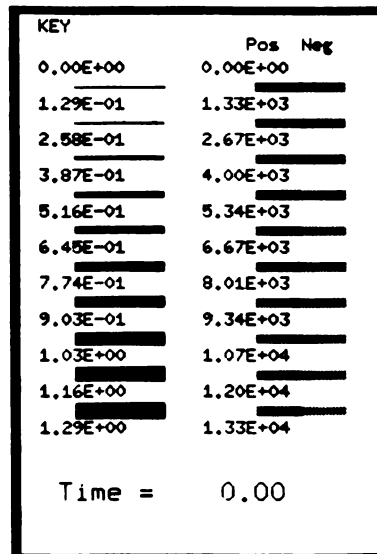


Figure 5. Dynamic Display Key to Widths and Colors.

Time Plots. The graphical variation methods described above are intended to be qualitative in nature, dealing with relationships between the values of variables at different locations within the entire bond graph. At some point it may become useful to zoom in on variable values associated with a particular bond. This function has been provided by allowing the user to select a bond in the dynamic display, whereupon the user is allowed to choose one of the bond variables for display in a plot as a function of time. An even more precise display might focus on choosing a particular point in time from the time plot for a particular variable and obtaining numerical values for variables in table form. This function has not yet been implemented in the dynamic display environment, but a similar result can be obtained by using the "table" function provided outside of the dynamic display environment. Time plots of several chosen variables related to different bonds in the graph can also still be constructed by using the facility provided in the original ENPORT postprocessor.

Animation Control. A user may desire to display the results of a simulation over the entire time frame used in the solution of the differential equations or over some subset of that time. The choice of time subsets for the simulation data is provided by the ENPORT postprocessor. The determination of ranges for each variable type is performed over this chosen time frame.

The user also requires a way to control the rate at which the animation is presented. Time controls are provided for this task. Several options have been implemented. The user is allowed to run the display from the current time to a later time specified by the user at a constant speed (seconds per frame) that is input by the user. During this constant speed animation, no user input is allowed. In order to account for a user who might not want to proceed through the animation at a fixed rate, single time step functions are provided. These functions allow users to proceed with the animation at

their own pace by selecting a button to move forward or backward one step. A function is also provided to allow users to quickly go to a specified time stage without having to step through all the intermediate stages.

View Control. A user of the BGVis-2D environment might also wish to display only a portion of a bond graph model if the model is so large that the display will be cluttered. The ability to pan through the graph in both the x and y directions is provided by scrollbars attached to the graph viewing window. It is also possible to zoom in to or out from a graph view.

3.2.2 Platform Selected for Implementation

The bond graph modeling program ENPORT was chosen as the vehicle for implementation of the environment characteristics described above. ENPORT was described in the previous chapter as a software environment which accepts bond graphs and block diagrams as components in an integrated modeling environment, makes construction and modification of system graphs a straightforward graphical process, and supports the equation derivation and solution processes. ENPORT was chosen because it is a well-documented and verified software program, and because the software code was available in a convenient setting for modification and addition of visualization functionality.

ENPORT is currently available in three different formats. An earlier version is written in FORTRAN and runs on mainframe computers which run the VMS operating system. At the time this research took place, the VMS operating system was being gradually phased out. The most common currently distributed version of ENPORT is written in FORTRAN and runs on MS-DOS based personal computers. An experimental

version of ENPORT is implemented in a UNIX environment and is programmed in the C computer language, taking advantage of the X Window System user interface. The UNIX version of ENPORT was chosen for the addition of the new display technologies of this research for several reasons. The mouse-driven X Windows format provided better user interface characteristics when compared to the other versions, which are more menu-driven. What is more important, the higher processing requirements of the graphics display anticipated, especially for the 3-D capability, required the power of a workstation. It is planned that the display enhancements generated by this research will eventually be carried over into the PC environment, especially since 486 and Pentium processors are becoming more widely used by engineers.

The implementation of this research was performed on a Sun Sparc-10 workstation running the UNIX operating system. The ENPORT version used was the 1.0 β version running under UNIX. All additional code was written in the C computer language, and the Xt Intrinsics functions, along with the Athena widget set, were used to support the user interface functions (Nye and O'Reilly, 1990). Documentation and the computer code generated in the course of the project are included in Appendix B of this report. The enhancements to the bond graph modeling environment were added as part of a postprocessor module of ENPORT.

Examples of how the BGVis-2D environment described in this chapter is actually used for visualization purposes are described in chapter 5. The figures provided in that chapter provide a rough idea of a process that is inherently dynamic and interactive. There is a videotape available from the author directly or from the Mechanical Engineering Department at Michigan State University which provides a clearer indication of how the designed bond graph visualization environment actually performs.

3.2.3 User's Guide

The following section can be used as a reference for running the software implementation of the BGVis-2D tool. Each of the available functions is described in the order in which it would most likely be performed. Pictures of the user interfaces are included.

Starting the Postprocessor. The ENPORT program's postprocessor window is started by typing the name of the postprocessor program, which is "BGVis-2D". This interface can be used to load in simulation results files, provide 2-D plots of selected variables versus time, specify time intervals of interest, and start the animation module. A picture of the control window is shown in Figure 6 for reference. Exiting the ENPORT postprocessor can be executed by clicking on the "Exit" button at the appropriate time.

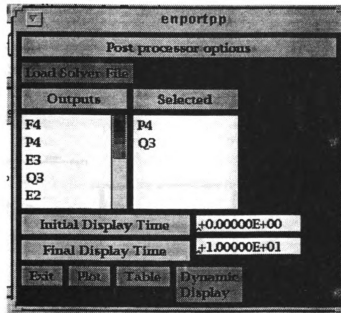


Figure 6. ENPORT Postprocessor Module Control Window.

Loading an Output File. Clicking on the "Load" button will bring up the directory structure window which will allow the user to select an ENPORT simulation output file (suffix .S2P) previously generated by the ENPORT solver module. Clicking on the file name will then cause that file to be loaded into the postprocessor.

Selecting a Time Interval. The default time limits shown in the initial and final time boxes are taken as the beginning and ending times of the simulation (see Figure 6). If some subset of that time interval is desired for study, new time limits can be entered into the boxes by simply typing them in using the keyboard.

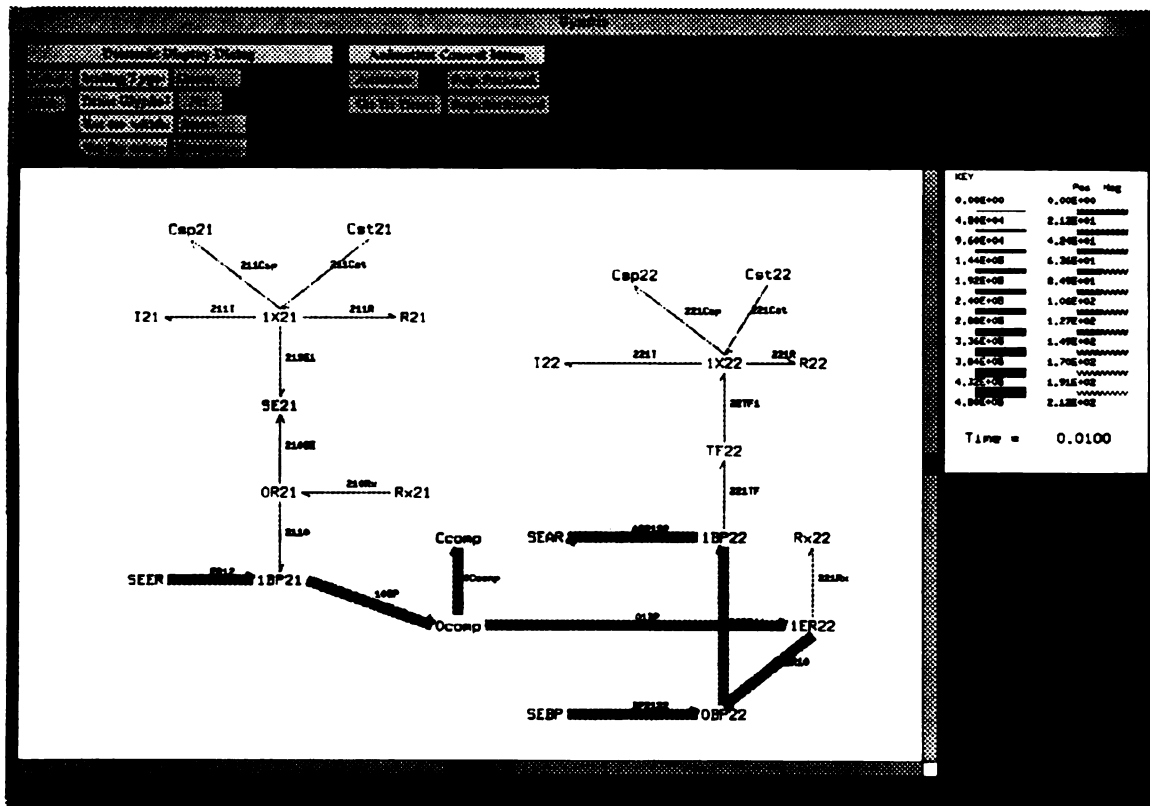


Figure 7. Dynamic Display Module Control Window.

Entering the Dynamic Display Module. Clicking on the button labeled "Dynamic Display" will initialize the Dynamic Display Dialog. A picture of this control window is shown in Figure 7 for reference.

Loading the model file. Clicking the "Load" button at the top of the menu area at the far left will bring up the same file selection window as in the postprocessor. This window can be used to select the ENPORT model file (suffix .ENP) to be loaded. This file contains the system graph as constructed in the ENPORT model build module. The system graph will then be drawn in the animation viewport, scaled so that the model fits in the window. Determination of output variable ranges occurs at the time of model file loading.

Using the Animation Viewport. The large window in the dynamic display dialog is the canvas on which the system graph is drawn. There are two functions included related to controlling the appearance of the graph. The first allows the user to shrink or increase the size of the model by clicking the "Zoom" button at the bottom left of the dialog. Doing so will bring up a window into which a chosen scale factor can be typed via the keyboard. The second involves using the scrollbars at the bottom and to the right of the window to pan through the model.

Choosing variables for display. The choice for which of the seven variable types (power, effort, flow, energy, momentum, displacement, signal, none) to display for line width and which one to display for color are made by mouse selection from pull-down menus provided at the top of the dynamic display control window (see Figure 7). Clicking and holding in the box to the right of the menu label will cause the pulldown menu to pop up, and the mouse can be dragged down and released on the appropriate variable option. The user can choose whether or not to display the glyphs during an

animation by using the “Show glyphs?” pulldown menu also at the top of the dynamic display window.

Choosing a Scaling Method. The program allows three choices for scaling of variables: linear, logarithmic, and absolute logarithmic. The choice of which to use is governed by the pull-down menu obtained when the button next to the "Scaling Choice" label is clicked and dragged down to the appropriate choice and then released. If negative values of the chosen variables are present when logarithmic scaling is chosen, an error message will be shown.

Controlling the Animation. The controls for running the dynamic display are included in the box at the top right of the dynamic display dialog labeled "Animation Control Menu" (See Figure 7). When the "Animate" button is chosen, the dynamic display is drawn for each time increment from the current time to the time specified by the user at a speed (seconds per frame) that is input by the user. The fastest rate available is 1 second per frame and the user can choose any multiple of that as the approximate rate of display by typing into a popup window. During the time this animation is being performed, the display updates automatically, and no interference by the user is allowed. In order to account for a user who might want to control the animation by moving one frame at a time, “Step forward” and “Step backward” functions are provided. These functions allow the animation to proceed either forward or backward by one time increment when the button is clicked. If the current timestep is the lower time limit, and "Step backward" is chosen, no action is taken. Similarly, if the current timestep is the upper limit, no action is taken. A “Go to timestep:” function is also provided to allow the animation to be redrawn at the specified time stage without having to step through all the intermediate stages. Clicking on this button will pop up a box into which the desired time value can be entered by the user.

Interpreting the Data Key. The smaller viewport window on the right is provided to show the numerical scale values of the displayed variables that correspond to each color and line width possibility. The current time stage value is also displayed at the bottom of this window.

Obtaining a Time Plot for a Bond. If a detailed plot of a variable associated with a single bond is desired, it can be obtained by clicking on the bond's name in the animation viewport. The user is allowed to specify which variable type will be displayed (power, effort, flow, energy, momentum, displacement, signal). This pops up the plot window, with the selected variable for that bond displayed as a function of time. Time plots of several variables can be displayed by returning to the ENPORT postprocessor control window and using the "Plot" button.

Leaving the Dynamic Display Module. Click on the "Exit" button in the menu area at the far left to close the window and exit the dynamic display module. This will return the user to the postprocessor where a different results file can be located, or the program can be exited.

Chapter 4

A Three-Dimensional Bond Graph Visualization Environment (BGVis-3D)

4.1 Environment Specifications

As was shown in the previous chapter, there are definite advantages to be gained from relating the response data from a simulation directly to the bond graph. These advantages are extended with the ability to display bond graphs in a three-dimensional environment and to manipulate them as 3-D geometric objects. The basic goal of the three-dimensional bond graph modeling environment is to make the viewing of power pathways or other variable values easier and to make interpretation of the model behavior clearer. An example of this type of thinking in the area of chemistry/physics is the visualization of complex molecular structures. This type of visualization is generally conducted for the purpose of interpretation of molecular behavior (Moffat, 1991). In a similar manner, more insight into a bond graph model can be gained if the designer can interact with it in such a way that it can be visualized in three dimensions and the viewing process can be under the modeler's control. The BGVis-3D environment allows for viewing of the 3-D graph from different angles and from a range of distances. The user is able to position the graph in the way that makes its structure and behavior the most clear. The same specifications related to controlling the data presentation in the animation that were presented for the 2-D situation also apply here.

The intention of the environment is also to allow the user to display a bond graph model which corresponds closely, allowing for scale, with the three-dimensional

geometry of the physical system being designed. Bond graphs are usually described as a topological modeling type, meaning that the connectivity of the graph does not generally relate to the geometry of the physical object. This can be an advantage, since the subcomponents of the bond graph model can be moved in whatever way seems suitable without changing the character of the model, as long as the correct connectivity is maintained. The disadvantage of a topological model is that it does not encourage visualization of the geometry of the physical system in the modeling process. The ability to use a three-dimensional geometric bond graph gives the model a well-defined spatial relationship to the physical object, allowing the designer to understand the model behavior in light of the actual geometry of the system. The environment allows for animation of variable types on the bonds in much the same way as for the 2-D environment, but the results can be even more clearly related to the physical system in the 3-D environment.

4.2 Design of the Three-Dimensional Visualization Environment

4.2.1 Features Selected for Implementation

The bond graph is displayed in the 3-D environment with spheres as nodes and cylindrical solids with cone-shaped arrows as bonds. Although standard bond graph convention calls for a half-arrow to indicate bond direction, full arrows are used in the BGVis-3D display. The primary reason for this choice is that any arrow that does not extend out in all directions from the cylindrical bond might not be seen if the graph is viewed from a particular orientation. To avoid any ambiguity in interpretation of connector type, signals are displayed as rectangular solids, distinguishing them from cylindrically shaped bonds. Bond graph nodes are drawn as spheres, while block diagram elements are drawn as rectangular solids. The nodes are drawn in colors corresponding

to their power domains, while the connectors have changing colors depending on the variable values being shown.

The model is presented in a perspective projection, and lighting and shading parameters are specified to make the 3-D geometry look as real as possible. Direct trade-offs need to be made between quality of the display surfaces and the timely performance of display updating. After much experimentation, it was decided to maintain a reasonable display quality despite long draw times, based on the assumption that the performance problem could be reduced in the future with the use of a workstation with greater processing power.

Manipulation of the 3-D model is provided by valuator on the screen. A valuator consists of a box with a sliding handle whose position can be controlled by dragging the handle with the mouse. The position of the handle corresponds to a particular value of a viewing parameter. One valuator controls the size of the field of view, providing for a zooming in/out function. The others control the angular position about the x, y, and z axes of the model in the workstation by changing the view position in the x, y, and z directions.

The same variable types are available for display as in the 2-D environment (power, effort, flow, energy, momentum, displacement, signal). Data loading, calculation of momentum and displacement values, and scaling functions are also the same. The variation method provided for showing values is color change on each bond. Bond width was not used as a variation technique, since 3-D perspective drawing of the graph causes parts of the model that are further away to be shown smaller than parts of the model at the front. Changing bond widths to represent information would cause erroneous conclusions due to the perspective effect. Ten levels of color are provided based on a

VBGYOR (rainbow) type spectrum with violet as the low value and red as the high. Reference bond directions are reversed in the display to show negative variable values.

Functions for controlling the animation with respect to time are provided which are similar in nature to those of the 2-D environment. Animation at a constant speed and step-type functions are provided within the 3-D framework by choosing buttons in a selector window. Each time the animation is updated the nodes are redrawn in the neutral color, while the bonds are redrawn in the appropriate colors corresponding to the chosen variable value. A key is also provided to index colors with numerical value levels.

Currently the BGVis-3D environment can be used for display of 3-D bond graph representations and not for graphical construction of the models. Two-dimensional model files can be changed to three-dimensional model files by using a text editor and manually typing in z coordinate values. It is anticipated that a three-dimensional model builder could be constructed using techniques directly corresponding to the current ENPORT two-dimensional model builder. Since the intent of this project was to show the value of enhanced simulation results display, it was decided to leave 3-D model building capabilities to a future time.

4.2.2 Platform Selected for Implementation

The implementation of this research was also performed on a Sun Sparc-10 workstation running the UNIX operating system. The 3-D display post-processor is implemented as a stand-alone module that can access ENPORT (1.0 β version) library routines necessary for file reading and any other necessary utilities. Three-dimensional models can be loaded from ENPORT model files which have been modified to include a third coordinate. To support this modification, the basic data structure defined in

ENPORT to store node and connector data was altered. This also led to modifications to the file reading/writing routines. Documentation and the computer code generated in the course of this phase of the research can be found in Appendix B. An example illustrating the utility of the 3-D display environment can be found in Chapter 5.

In order to build a 3-D modeling environment with the characteristics described above, it was necessary to either find or build programming software capable of performing the 3-D graphics tasks. The fact that developing computer graphics code for allowing 3-D display of ball and stick type structure is very time consuming and the desire to make portable code led to the use of a graphics standard to support the modeling effort. The PHIGS graphics standard (Kasper and Arns, 1993) was used to supply the capability for controlling and drawing 3-D geometries, including lighting and shading effects. A description of PHIGS can be found in the next section.

4.2.3 Description of the PHIGS Graphics Standard

A computer graphics standard is a set of computer graphics subroutines which has been established as a basis for performing computer graphics operations. It has generally been designed to be as portable as possible, allowing the applications using the standard to be used on different computer systems and to be useful in a broad range of graphics application domains. The standard is also designed to be independent of computer programming language used.

Several different graphics standards have come into existence since the late 1960's, when development of standards was begun. Among the first was the Graphical Kernel System (GKS), which was specified to address 2-D display of computer graphics. GKS was also later used to support 3-D display, and GKS-3D was published in 1988

(Int'l Standard, GKS-3D, 1988). Another standard was developed and presented around the same time which supports both the modeling and viewing functions of computer graphics. That standard is called the Programmer's Hierarchical Interactive Graphics System (PHIGS), and this was the standard chosen for use in creating the 3-D bond graph display environment (Int'l Standard, PHIGS, 1988). PHIGS is described as "a set of functions for computer graphics programming in environments requiring rapid modification of graphical data that describes geometrically related objects" (Arnold and Duce, 1990). The 3-D bond graph environment falls under this category of applications and thus the PHIGS standard was considered useful for developing the software. An addition to the original standard called PHIGS PLUS was also defined later to provide higher level capabilities for lighting and shading calculations (Int'l Standard, PHIGS PLUS, 1992). This standard is also used in this work along with PHIGS. Some of the basic characteristics of PHIGS and how it is used in designing graphics applications will be described here.

Since PHIGS is defined independently of programming language, it was necessary to use the standard-specified C binding for PHIGS, which defines how the functionality of PHIGS is tied to the C programming language. All programming done in the course of designing the 3-D bond graph environment was written in C.

An important feature of PHIGS, and one on which the standard is based, is the concept of a *workstation*. PHIGS uses this term to mean "an abstract collection of hardware and software with graphics capabilities" (Kasper and Ams, 1993). A particular workstation is implemented by connections to physical output and input devices. There are some functions of PHIGS which are workstation-independent, i.e. designed to be portable between all hardware systems, and some functions which are workstation-dependent.

The functionality of PHIGS allows for creation of device-independent graphical models by defining graphical data units called *structures*. These structures consist of hierarchically arranged element primitives or building blocks along with their attributes. The structures are stored and can be accessed by their application-specified names. Editing of these structures after initial creation is also allowed. Once the graphical model has been constructed, PHIGS provides routines defining the viewing parameters for the model. Finally, the model is posted to a workstation for display.

In the course of graphics modeling and viewing, it is necessary for the model to go through a pipeline of coordinate transformations. A diagram of the PHIGS supported modeling transformation pipeline is shown in Figure 8. The initial model is described in a modeling coordinate system, which can be chosen for convenience in constructing individual pieces of a model. A world coordinate system is defined which allows individual pieces to be assembled into a larger model. Thus model subcomponents can be constructed in a local coordinate system and then be scaled, rotated, and translated before placement in the global modeling space.

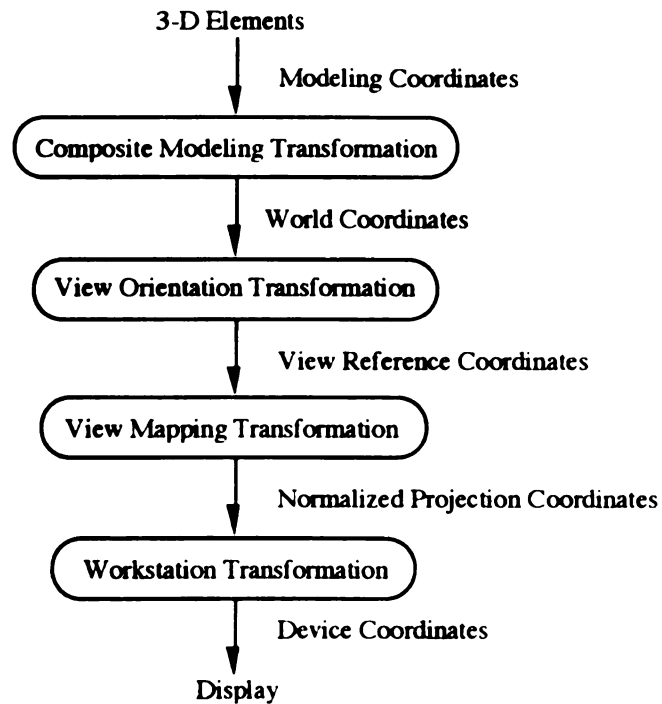


Figure 8. PHIGS Modeling Transformation Pipeline.

A view reference coordinate system can be chosen to select the direction from which the model will be viewed. The model in view reference coordinates is then normalized in preparation for display on a particular workstation. The final transformation takes place changing normalized projection coordinates to device coordinates, which are the actual coordinates used to display the model on a particular graphical output device. PHIGS provides routines which support all of the functions described in a way which ensures that the user does not have to deal explicitly with the mathematics, primarily matrix identification and multiplication, involved in the transformations. It thus provides three-dimensional graphical processing capabilities at a relatively high level which can easily be integrated into an application.

4.2.4 User's Guide

The following section can be used as reference for running the 3-D dynamic display environment. Each of the functions available is described under the appropriate heading.

Starting the 3-D Display Module. The 3-D Display Postprocessor can be loaded by typing the name "enport3D". This window provides the functions of loading of model and data files, specifying the time range of the display, selecting the scaling type and variable to display, and opening the PHIGS display window. A picture of the 3-D postprocessor control window is shown in Figure 9. Exiting the 3-D postprocessor can be executed by clicking on the "Quit" button.

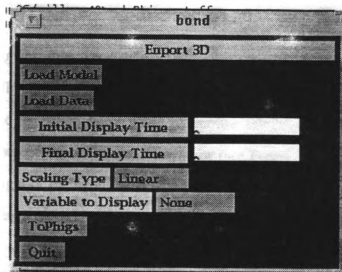


Figure 9. 3-D Postprocessor Control Window.

Loading an Output File. Clicking on the "Load Data" button will bring up the directory structure window which will allow the user to select an ENPORT simulation output file (suffix .S2P) previously generated by the ENPORT solver module. Clicking on the file name will then cause that file to be loaded into the 3-D postprocessor.

Loading a Model File. Bond graph model data can be loaded into the application using the "Load Model" button in the same way as for the data file. The selection window can be used to select the z-modified ENPORT model file (suffix .ENP) to be loaded. This file contains the system graph as constructed by the ENPORT model build module, but modified to include a third coordinate (z) for all of the model elements. Determination of all output variable ranges occurs at this time.

Selecting a Time Interval. The default time limits shown in the initial and final time boxes are taken as the beginning and ending times of the simulation (see Figure 9). If some subset of that time interval is desired for study, new time limits can be entered into the boxes by simply typing them in using the keyboard.

Choosing variables for display. The choice for which of the seven variable types (power, effort, flow, energy, momentum, displacement, signal, none) to display for color variation on the 3-D model are made by mouse selection from a pull-down menu provided near the bottom of the control window (see Figure 9). Clicking and holding in the box to the right of the menu label will cause the pull down menu to pop up, and the mouse can be dragged down and released on the appropriate variable option.

Choosing a Scaling Method. The program allows three choices for scaling of variables: linear, logarithmic, and absolute logarithmic. The choice of which to use is governed by the pull-down menu obtained when the button next to the "Scaling Type"

label is clicked and dragged down to the appropriate choice and then released. If negative values of the chosen variables are present when logarithmic scaling is chosen, an error message will be shown.

Entering the 3-D PHIGS Display Module. The model can be displayed by choosing the “ToPhigs” option once the model and data files have been loaded. A PHIGS workstation is opened, and the bond graph is displayed there in three dimensions. The workstation in this case is an X window. A picture of the 3-D display control window is shown in Figure 10 for reference. The graph is displayed with spheres as nodes and rectangular solids with arrows as connectors. The model is presented in a perspective projection and lighting and shading parameters are specified to make the 3-D geometry look as real as possible.

Manipulation of the 3-D Model. Manipulation of the 3-D model is provided by valuator on the screen. Four valuator are present. One controls the size of the field of view, providing for a zooming in and out function. The other three control angular position of the model in the workstation - one each providing rotation about the x, y, and z axes respectively. Each of these valuator can be worked by dragging the button to different positions along the track.

Controlling the Animation. The controls for running the 3-D animation are included in a selector window popped up along with the valuator by the PHIGS workstation. The same buttons are provided as in the 2-D environment. See section 3.2.3 for details of how they work.

[illegible]

Figure 10. The 3-D Postprocessor Display Window.

Chapter 5

Visualization Examples Using BGVis-2D

5.1 Automobile Modeling Example

In the field of automotive engineering one of the critical tasks is to design automobile suspensions for good performance, for example to provide reasonable settling times and limited motions for the car chassis and passenger due to a displacement disturbance from a rough road. As an example to demonstrate the capabilities of the two-dimensional bond graph visualization tool (BGVis-2D), the three-dimensional vehicle illustrated in Figure 11 was chosen for study. This particular example was chosen for two reasons. First, because the vehicle is a real life engineering system design problem. Second, because the system is straightforward enough for experienced engineers to form a conception of how the system will perform under different inputs. The purpose of the BGVis-2D environment is to provide insight into the behavior of an engineering system. The benefits of using BGVis-2D are clearly demonstrated by comparing the visualized response against the expected response.

A schematic diagram of the car chassis unit and suspensions with seat/passenger is shown in Figure 12. The schematic model functions as follows. The car chassis unit is treated as a rigid body which is allowed to translate (bounce), in the vertical direction (Z axis), and rotate about the Y (pitch) axis and the X (roll) axis. For simplicity the angular motions are assumed to be small. An independent mechanical suspension system is attached at each corner of this rigid body. The corners thus experience different vertical (Z-axis) displacements depending on the position and orientation of the chassis. The

suspensions consist of a spring and damper connected in parallel to the axle. A wheel is mounted to each axle and maintains contact with the ground. The input to the system is a vertical velocity applied to each wheel induced by the car traveling over a specified ground profile at a constant linear speed in the horizontal direction (X axis). A seat/driver are included in the model, connected through a compliance and damping effect to a point in the front half of the chassis. One of the primary responses of interest is the position of the seat/passenger in the car, which is allowed to vary in the Z (vertical) direction. The following section will consider a three degree-of-freedom bounce/pitch/roll (BPR) model of the vehicle dynamics. A two-dimensional bond graph representation will be used. A three-dimensional bond graph representation of the BPR model is described in chapter 7.

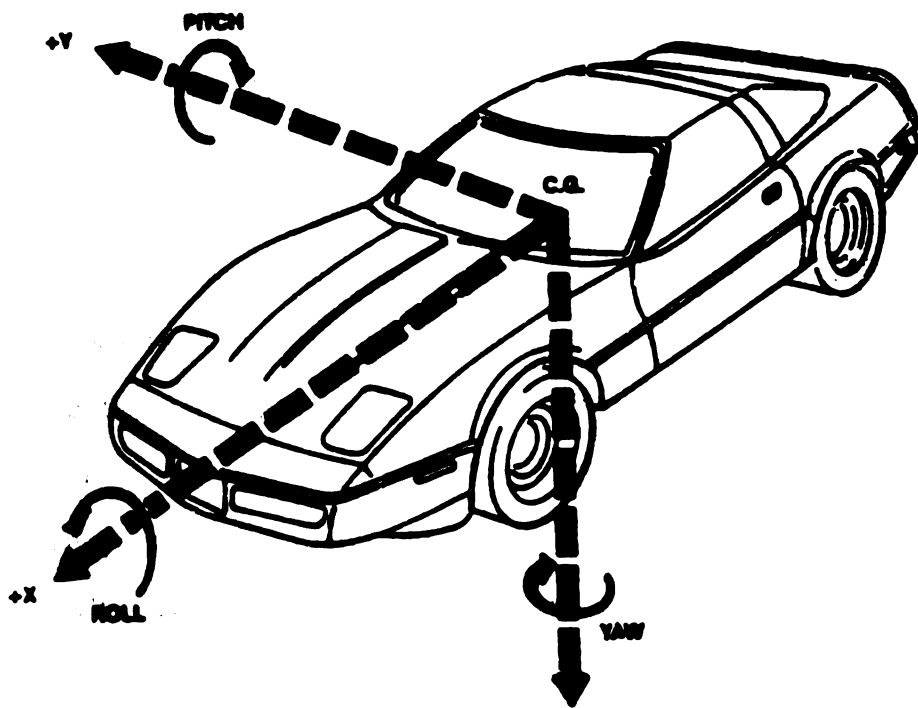


Figure 11. Vehicle System Illustration.

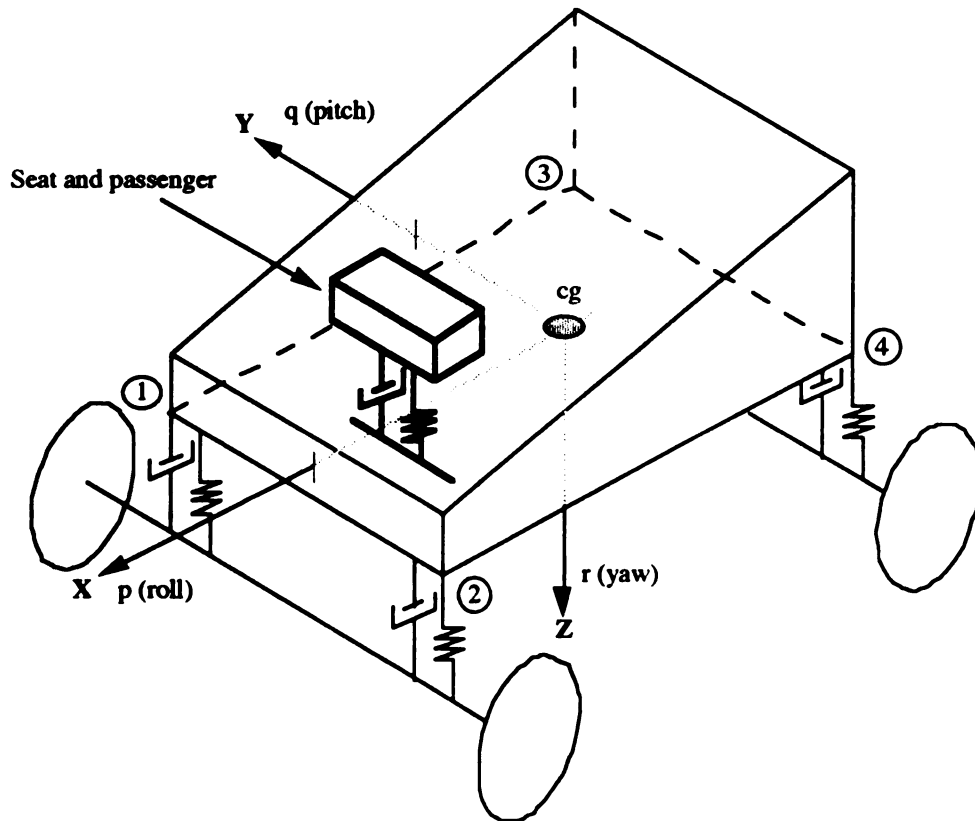


Figure 12. Schematic of Car Chassis Unit and Suspensions with Seat.

Bond Graph Model Description. The BPR model of the vehicle can be thought of as consisting of several major sections, or macros, which are connected by power bonds. Figure 13 shows the macro bond graph of the BPR car model. The seat unit is connected to the rigid chassis unit by a rigid connection labeled **CS** on the graph. Each of the suspensions is connected to a corner of the car via a rigid connection. **RF** stands for the right front corner, **LF** for left front, and **RR** and **LR** for the right and left rear corners, respectively. Each of the suspensions also maintains contact with the ground. Viewing the model as a macro bond graph representation is very useful if overall system response is of concern, rather than individual responses occurring within each subcomponent. A macro construction feature is currently not available in the workstation version of

ENPORT, but it is intended as an addition to the functionality of the program in the near future. Adding macro viewing capability to the BGVis-2D tool would assist in assessing the behavior of systems, especially in very complex systems with large numbers of bonds.

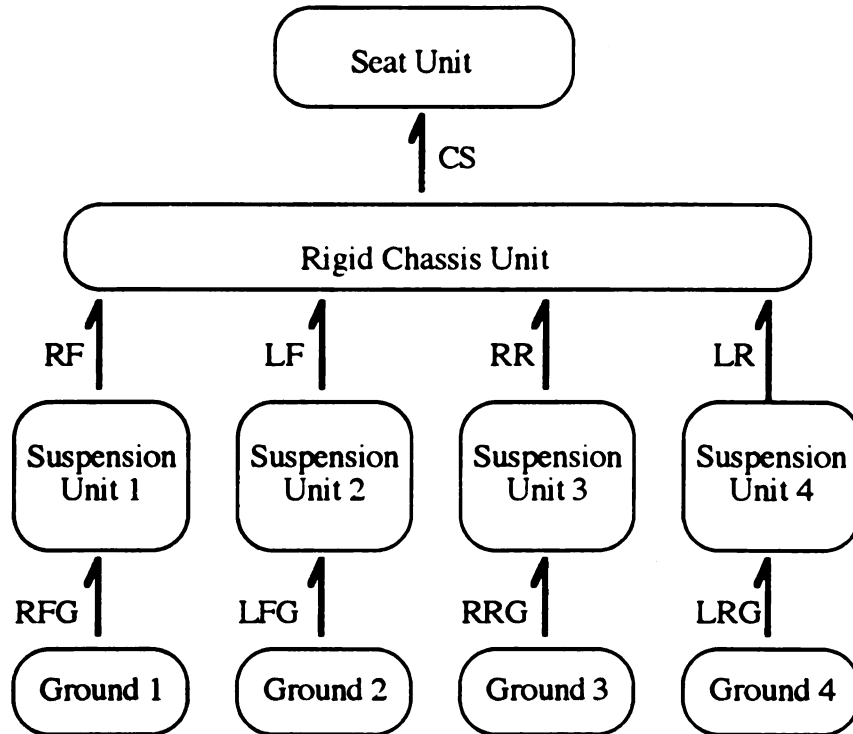


Figure 13. Macro Bond Graph of BPR Model.

For the purpose of simulating the behavior of the BPR model, each of the macros was expanded in detail using basic bond graph elements. The bond graph representations of the seat unit, rigid chassis unit, and suspension unit will each be shown and explained. The 2-D bond graph representation of the seat unit is shown in Figure 14. Node **1Z6** represents the vertical motion of the seat unit. The **SE6** node includes the input force on the seat due to gravity and the **IM6** node includes the inertia effect of the seat unit mass.

RD and **CK** are the dissipation and compliance effects of the coupling between the seat and the point on the rigid body chassis to which it is attached.

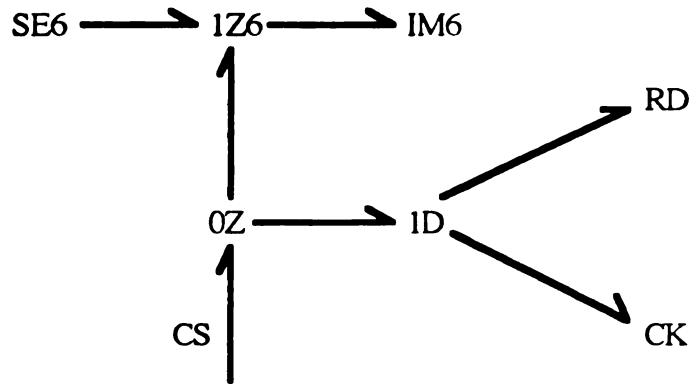


Figure 14. 2-D Bond Graph Representation of Seat Unit

The rigid chassis unit bond graph model is shown in Figure 15. The **TF** nodes and their connectors represent the geometric constraints satisfied by the rigid body. The **1W** junction represents the translational motion of the rigid body chassis in the Z direction, while **1P** is the roll motion (about the X axis), and **1Q** is the pitch motion (about the Y axis). The **0Z18** junction serves to calculate the position of the point on the rigid chassis to which the seat unit is attached. **0Z17**, **0Z11**, **0Z15**, and **0Z13** each determine the position of a corner of the rigid chassis to which a suspension is attached. All velocities and forces are considered positive in the downward direction. All rotational quantities are taken positive in the directions indicated on the schematic of Figure 12.

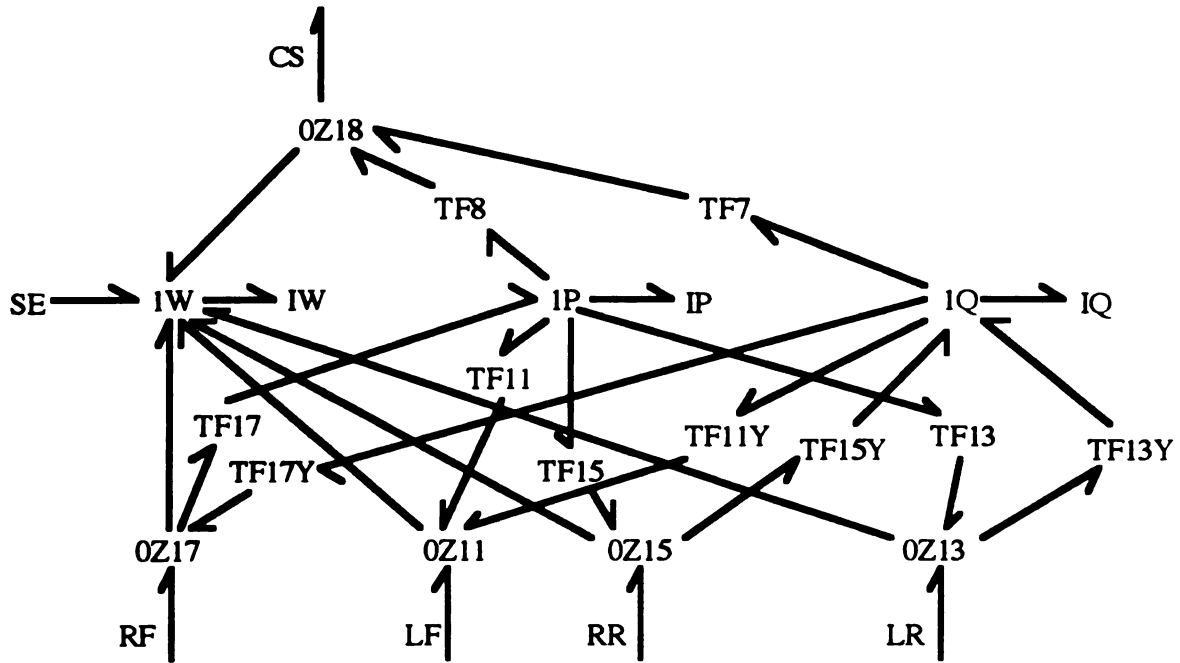


Figure 15. 2-D Bond Graph Representation of Rigid Chassis Unit.

A detailed schematic of one of the independent suspension, axle, and wheel systems which attaches to a corner of the chassis is shown in Figure 16 along with the corresponding bond graph. The stiffness and damping of each suspension are represented in this model with mechanical elements (**CS1** and **RS1**), but for an industrial application the suspension unit could be modeled as an electronic active suspension containing electrical and hydraulic components as well as mechanical elements. With bond connections between the suspension elements at different corners of the car, non-independent suspensions like roll bars also could be modeled and analyzed. The axles and wheels are modeled in this example with inertia (**IW1**), resistance (**RW1**), and compliance (**CW1**) effects. The schematic shows the coordinate systems used to construct the bond graph.

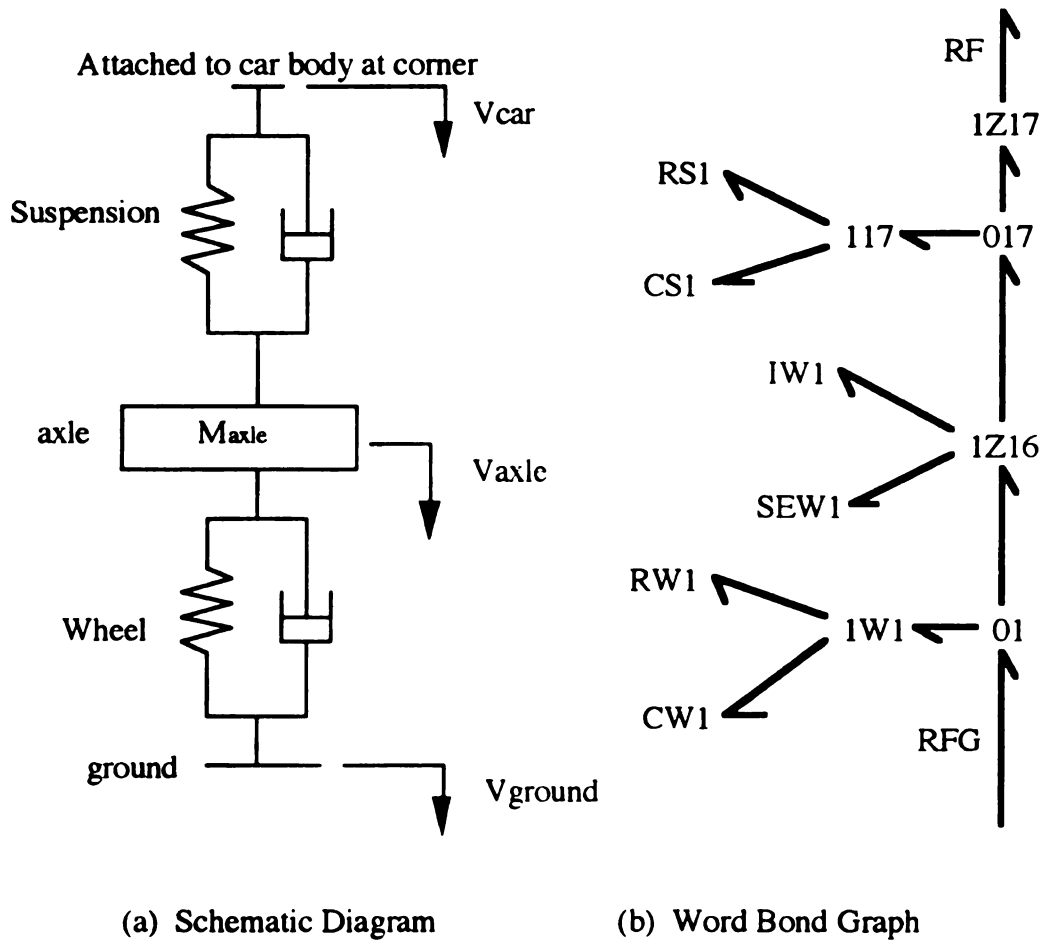


Figure 16. Detail of Single Suspension Assembly

A two-dimensional bond graph of the entire three degree-of-freedom BPR vehicle model is shown in Figure 17. This model is formed by joining the sections described previously, connected as they were in the macro bond graph of Figure 13. One simplification was made in the full model, which eliminated the **1Z** node of the seat unit. Each vertical branch in the full BPR model bond graph represents a suspension and wheel series. Branch 1 is the front/right suspension, branch 2 is the front/left, while branches 3 and 4 are the rear/right and rear/left respectively (corresponding to the schematic in Figure 12). The ground connections are represented by source of flow nodes. A vertical velocity is imposed on the wheel which is induced by the interaction of the forward

horizontal velocity (assumed to be constant) and the shape of the ground position profile that each wheel experiences. More details on the velocity input modeling can be found in the next section. Table 2 contains a summary of the equations used for this model along with their parameter values.

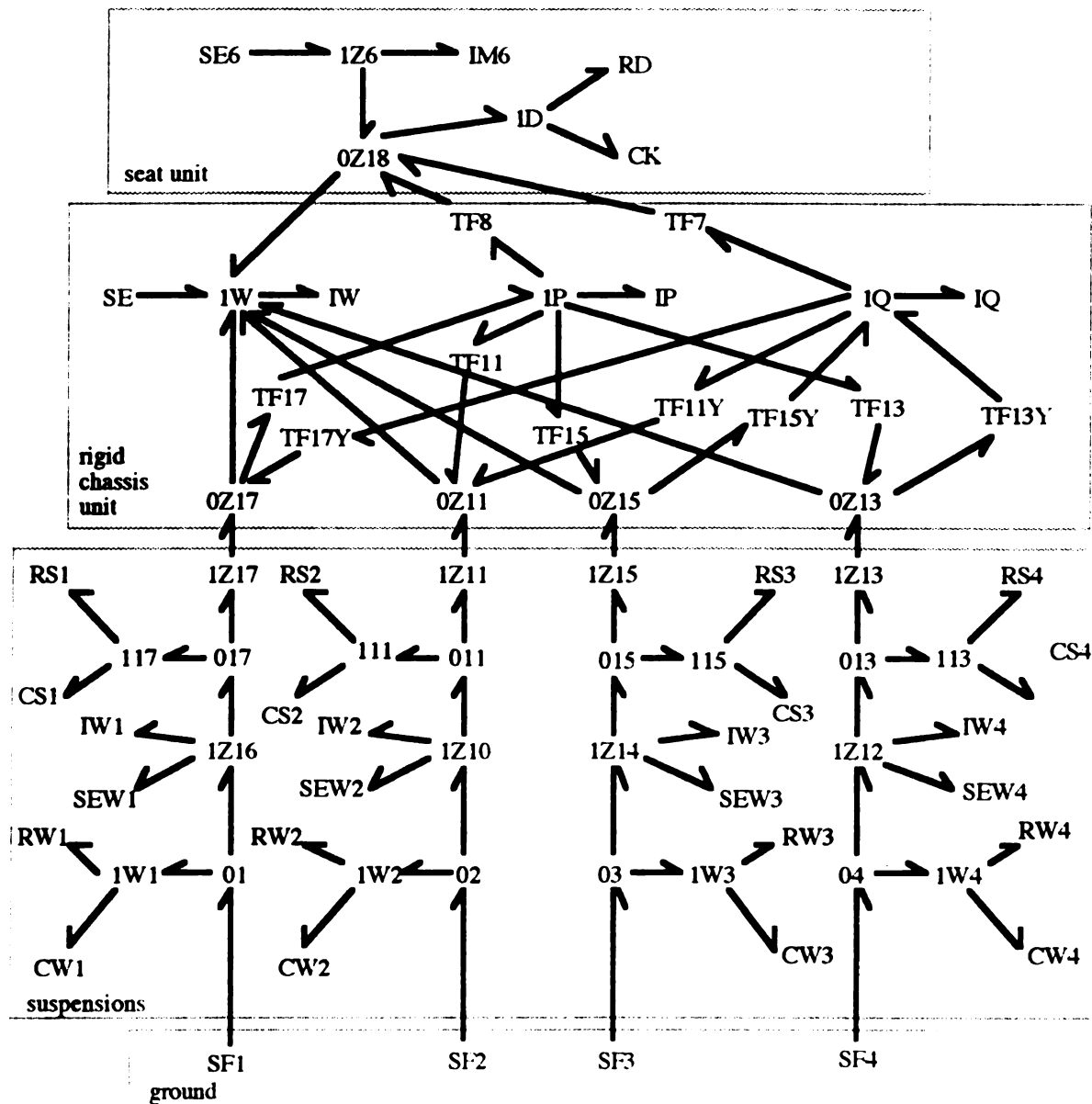
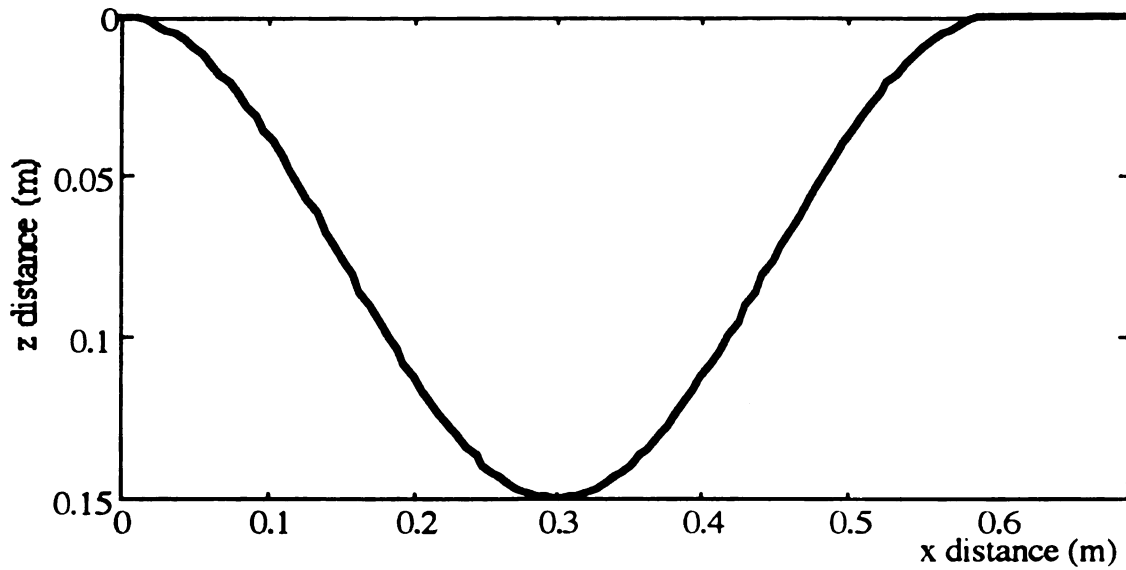


Figure 17. Two-Dimensional Bond Graph of Car Model.

Table 2. Car Body and Suspension Model Equations and Parameters

Element	Equation	Parameter
SF1	$F(t) = \text{pothole function}$	delay = 0.10 sec
SF3	$F(t) = \text{pothole w/ delay}$	delay = 0.29 sec
SF2, SF4	$F(t) = V \quad (\text{CON})$	$V = 0.0 \text{ m/sec}$
CW1, CW2, CW3, CW4	$E(t) = k * Q(t) \quad (\text{GAIN})$	$k = 310,866 \text{ N/m}$
RW1, RW2, RW3, RW4	$E(t) = R * F(t) \quad (\text{GAIN})$	$R = 10.0 \text{ N*sec/m}$
SEW1, SEW2, SEW3, SEW4	$E(t) = W \quad (\text{CON})$	$W = 441 \text{ N}$
IW1, IW2, IW3, IW4	$F(t) = (1/L) * P(t) \quad (\text{ATT})$	$L = 45 \text{ kg}$
CS1, CS2, CS3, CS4	$E(t) = k * Q(t) \quad (\text{GAIN})$	$k = 55,232 \text{ N/m}$
RS1, RS2, RS3, RS4	$E(t) = R * F(t) \quad (\text{GAIN})$	$R = 100.0 \text{ N*sec/m}$
TF11Y, TF13Y, TF15Y, TF17Y	$V(t) = m * \omega(t) \quad (\text{GAIN})$	$m = 1.25 \text{ m}$
TF11, TF13, TF15, TF17	$V(t) = m * \omega(t) \quad (\text{GAIN})$	$m = 0.75 \text{ m}$
IQ	$F(t) = (1/L) * P(t) \quad (\text{ATT})$	$L = 821 \text{ m}^2\text{kg}$
IP	$F(t) = (1/L) * P(t) \quad (\text{ATT})$	$L = 368 \text{ m}^2\text{kg}$
IW	$F(t) = (1/L) * P(t) \quad (\text{ATT})$	$L = 1360 \text{ kg}$
SE	$E(t) = W \quad (\text{CON})$	$W = 13342 \text{ N}$
RD	$E(t) = R * F(t) \quad (\text{GAIN})$	$R = 100 \text{ N*sec/m}$
CK	$E(t) = k * Q(t) \quad (\text{GAIN})$	$k = 27,000 \text{ N/m}$
IM6	$F(t) = (1/L) * P(t) \quad (\text{ATT})$	$L = 70 \text{ kg}$
SE6	$E(t) = W \quad (\text{CON})$	$W = 687 \text{ N}$
TF7	$V(t) = m * \omega(t) \quad (\text{GAIN})$	$m = 0.76 \text{ m}$
TF8	$V(t) = m * \omega(t) \quad (\text{GAIN})$	$m = 0.51 \text{ m}$

System Response and Assessment. The bond graph model of the car body and suspensions was subjected to an input consisting of an encounter with a relatively large pothole in the road. The pothole was modeled as a function following a sinusoidal profile. The vertical velocity induced by this function was determined and applied to the right front tire at the time the pothole was encountered. The same velocity function is experienced by the rear tire, but after a time delay which is dependent on the horizontal velocity of the automobile (for this example, approximately 35 mph). Figure 18 illustrates the mathematical modeling of the bump input. The ENPORT model file and function definitions for this example are included in Appendix C, along with the initial conditions used for the simulation.

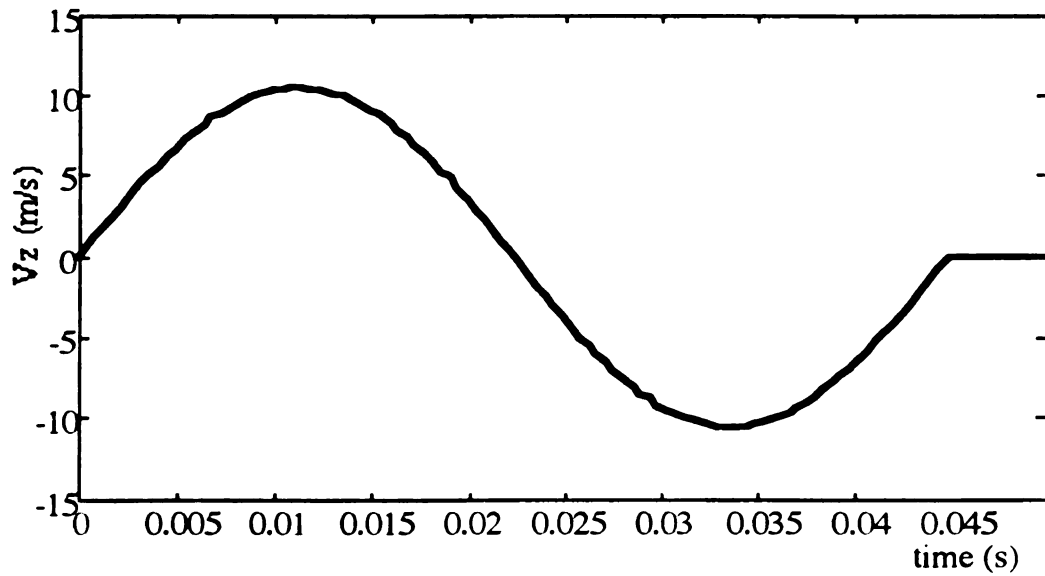


Pothole Profile: $z = -0.075 \cdot \cos\left(\frac{10}{3} \cdot \pi \cdot x\right) + 0.075$
for $0 < x < 0.6$

$$V_x = 13.42 \text{ m/s}$$

$$V_z = dz/dt = dz/dx \cdot dx/dt = dz/dx \cdot V_x$$

$$t = x/V_x$$



Vertical Flow Source: $V_z = (3.355) \cdot \pi \cdot \sin\left(\frac{10}{3} \cdot \pi \cdot 13.42 \cdot t\right)$

Time delay for back wheel: $t = \text{length of car} / V_x = (2.5 \text{ m}) / (13.42 \text{ m/s}) = 0.19 \text{ s}$

Figure 18. Mathematical Model of Pothole Source of Flow.

Table 3 shows a tabulation of the response values for the suspension spring deflections of the system (Q42, Q45, Q48, Q51) to the pothole input over a time range of 0.5 second. This response was obtained by setting the initial values for the displacements to their equilibrium values (see Appendix C) and solving for the response using the Runge-Kutta integration routine in ENPORT. It is obvious that this presentation of response as a table of data is difficult to interpret in terms of the geometry of the model. Figure 19 shows a time plot of the variables mentioned above, along with the position of the seat and passenger with respect to the chassis (Q86) for the same 0.5 second time period. It also takes considerable effort to relate this presentation of variable values back to the bond graph and the schematic, and to visualize their meaning.

Table 3. Response Data from Pothole Simulation.

TIME	Q42	Q45	Q48	Q51
+0.00000E+00	-6.32700E-02	-6.75100E-02	-5.95000E-02	-6.37200E-02
+2.00000E-02	-6.32700E-02	-6.75100E-02	-5.94900E-02	-6.37200E-02
+4.00000E-02	-6.32700E-02	-6.75100E-02	-5.95000E-02	-6.37200E-02
+6.00000E-02	-6.32700E-02	-6.75100E-02	-5.95000E-02	-6.37200E-02
+8.00000E-02	-6.32700E-02	-6.75100E-02	-5.95000E-02	-6.37200E-02
+1.00000E-01	-6.32700E-02	-6.75100E-02	-5.95000E-02	-6.37200E-02
+1.20000E-01	+5.83600E-02	-7.65300E-02	-5.42400E-02	-5.49800E-02
+1.40000E-01	-9.18300E-02	-9.63000E-02	-5.52400E-02	-5.29700E-02
+1.60000E-01	-1.06320E-01	-8.11500E-02	-6.99100E-02	-5.58100E-02
+1.80000E-01	-7.15300E-02	-6.96200E-02	-8.10200E-02	-7.12200E-02
+1.99990E-01	-6.49200E-02	-5.80100E-02	-7.16900E-02	-6.34800E-02
+2.19990E-01	-7.08100E-02	-6.32900E-02	-6.48000E-02	-5.75100E-02
+2.39990E-01	-6.73500E-02	-6.37000E-02	-6.29300E-02	-5.90300E-02
+2.59990E-01	-6.50300E-02	-6.33800E-02	-6.24500E-02	-6.09600E-02
+2.79990E-01	-6.42400E-02	-6.50500E-02	-6.05500E-02	-6.12200E-02
+2.99990E-01	-6.28600E-02	-6.53400E-02	-5.24500E-02	-6.23800E-02
+3.20000E-01	-5.23800E-02	-4.79700E-02	+2.82600E-02	-8.70100E-02
+3.40000E-01	-6.08600E-02	-5.95300E-02	-1.18340E-01	-8.62400E-02
+3.60000E-01	-8.29300E-02	-7.05100E-02	-7.65100E-02	-7.30800E-02
+3.80010E-01	-7.84300E-02	-7.23900E-02	-5.98400E-02	-6.04500E-02
+4.00010E-01	-6.93600E-02	-6.31900E-02	-6.25300E-02	-5.44100E-02
+4.20010E-01	-6.52000E-02	-6.16800E-02	-6.32900E-02	-6.04900E-02
+4.40020E-01	-6.48400E-02	-6.42800E-02	-6.01700E-02	-5.93500E-02
+4.60020E-01	-6.35400E-02	-6.49500E-02	-5.91200E-02	-6.06400E-02
+4.80020E-01	-6.22200E-02	-6.56500E-02	-5.85800E-02	-6.19900E-02
+5.00030E-01	-6.18000E-02	-6.67300E-02	-5.77300E-02	-6.26100E-02

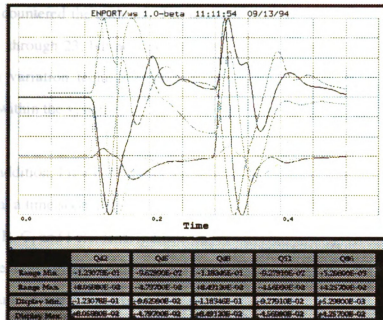


Figure 19. Response Data from Pothole Simulation

Response Using the BGVis-2D Tool. Several figures are shown to illustrate the assessment of response of the BPR vehicle model to the pothole using the enhanced bond graph display environment. The variable type selected for line width display in these figures is power. The first (Figure 20) shows the car at its equilibrium position before the bump is encountered. Since at this equilibrium point the parts of the vehicle are not undergoing any vertical velocity, powers are all equal to zero and all of the bonds are drawn at their minimum width. Figure 21 shows the response of the system at the time just after the front/right tire has encountered the pothole. Power can be seen to be flowing up branch 1 (right/front) into the **CW1** element as the tire responds to the downward velocity. In Figure 22, power flows further up the suspension branch to be dissipated by the suspension's viscous damping (**RS1**). Figure 23 shows that the power

input to the model by the pothole is nearly damped out by this damping effect at a later time stage. In Figures 24 through 26, responses are shown at a time after the rear/right tire has encountered the pothole. The response follows the same pattern as occurred in Figures 21 through 23, but now power can be seen to be transmitted up branch 3. The line width variation technique lends itself well to the visualization of "flow" type quantities within the system.

In addition, Figure 27 shows a display of energy variable values using the "level" indicators at a time soon after the front tire encountering the pothole. The glyph symbols next to the R, C, and I elements indicate that at this particular moment in the course of the animation energy is stored kinetically in the I element for the wheel/axle (**IW1**), while a small amount of energy has been dissipated in the suspension (**RS1**). Eventually, the majority of the energy input to the system by the pothole is damped out through the action of the R elements in suspensions 1 and 3. The "flow" and "level" type indicators could also be used together to enhance the engineer's understanding of how the system functions.

For the purpose of checking exactly how much relative displacement is experienced by the seat unit during the simulation, bond 88 was selected with the mouse, and the time plot of displacement shown in Figure 28 was obtained. The time plot shows a small change in the absolute position of the seat unit mass during the course of the simulation for the chosen model parameter values. It can be concluded that although less than one tenth of the maximum power transmitted to the wheel ever reaches the car chassis and passenger (as shown by Figures 20 through 26), there is still a small amount of deflection experienced by the passenger. These visualization techniques give a qualitative idea of how the vehicle reacts to a pothole in the road, which would allow tuning of the suspensions to further isolate the road surface from the passenger motion.

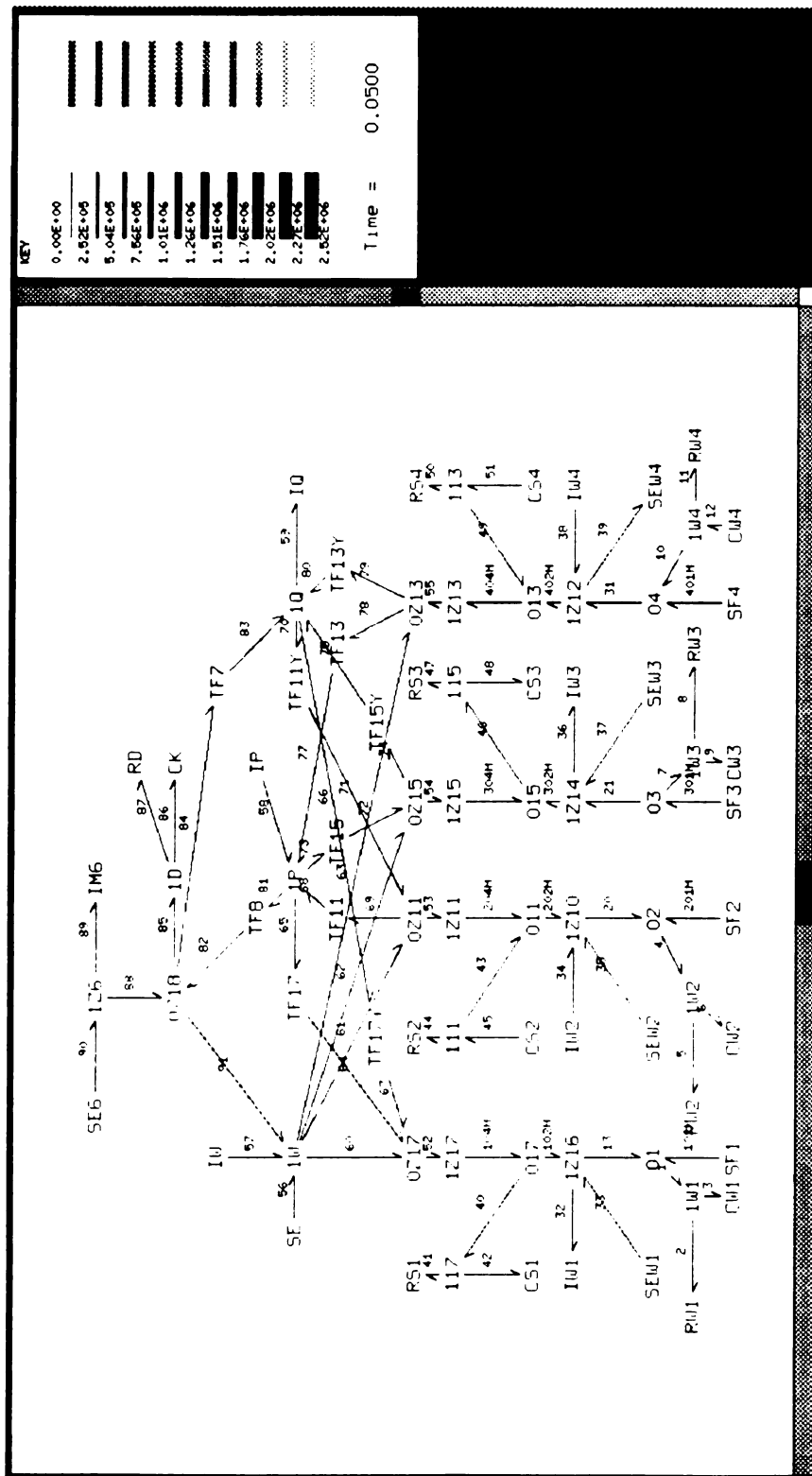


Figure 20. Power Variable Response for BPR Model (Time = 0.050).

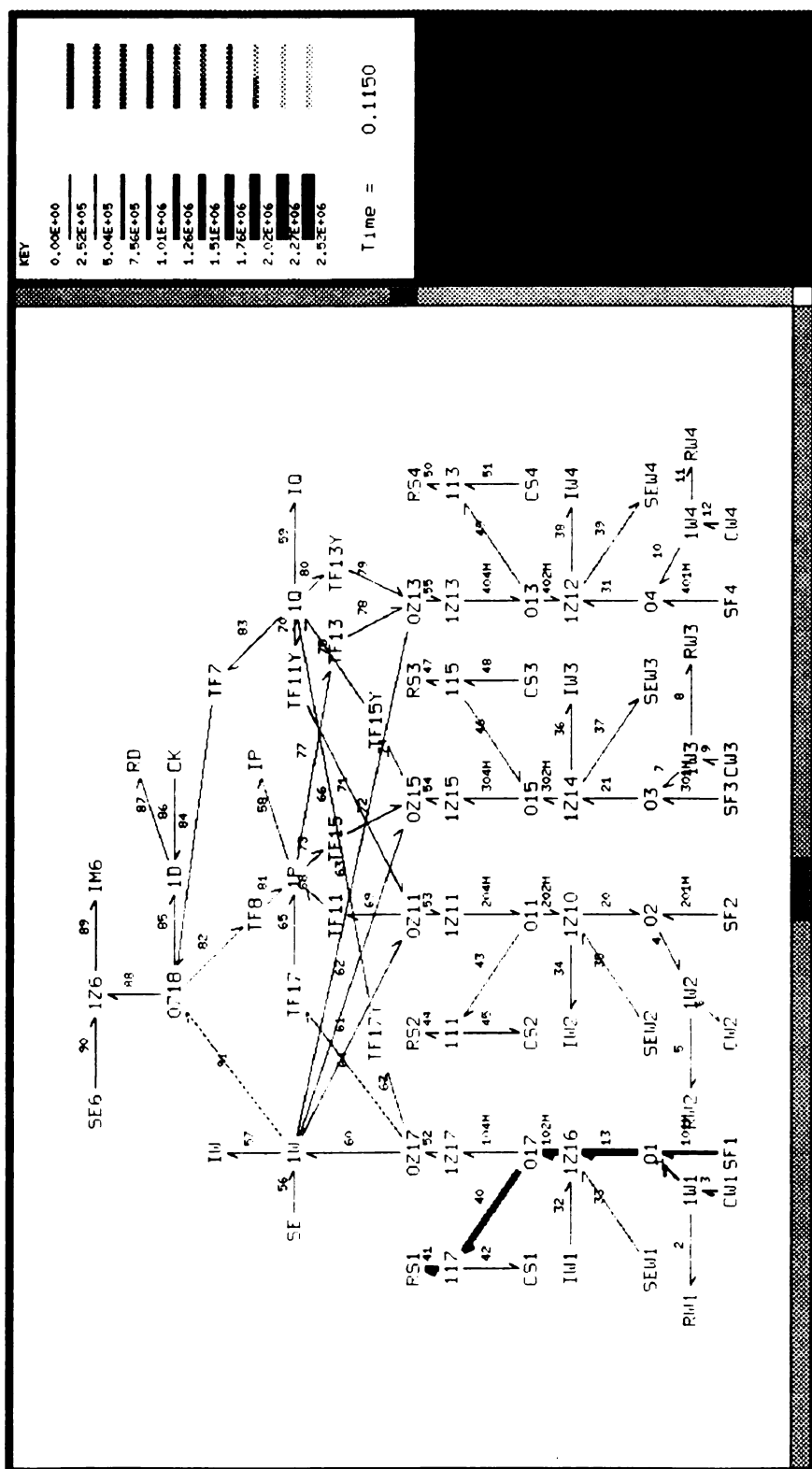


Figure 22. Power Variable Response for BPR Model (Time = 0.115).

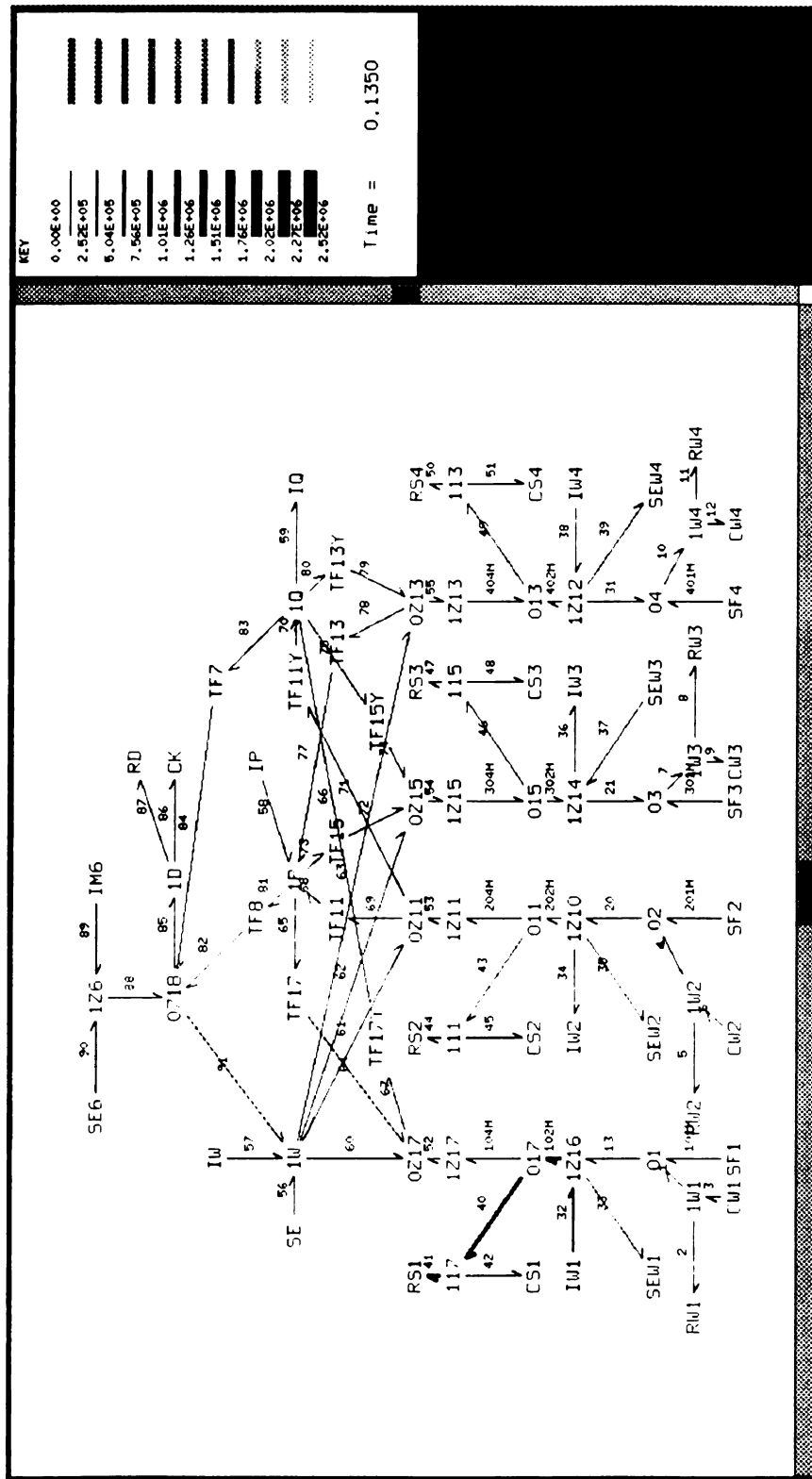


Figure 23. Power Variable Response for BPR Model (Time = 0.135).

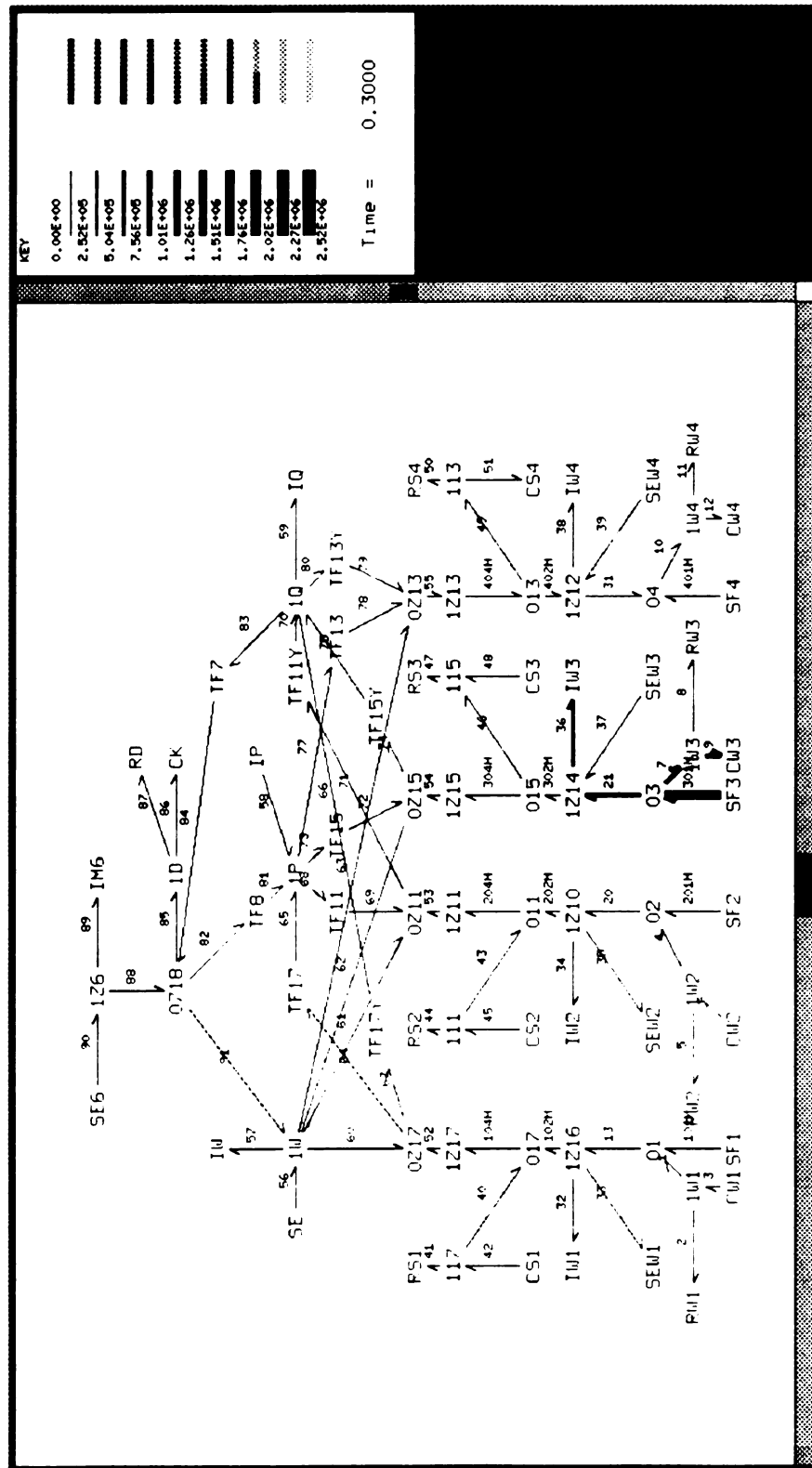


Figure 24. Power Variable Response for BPR Model (Time = 0.300).

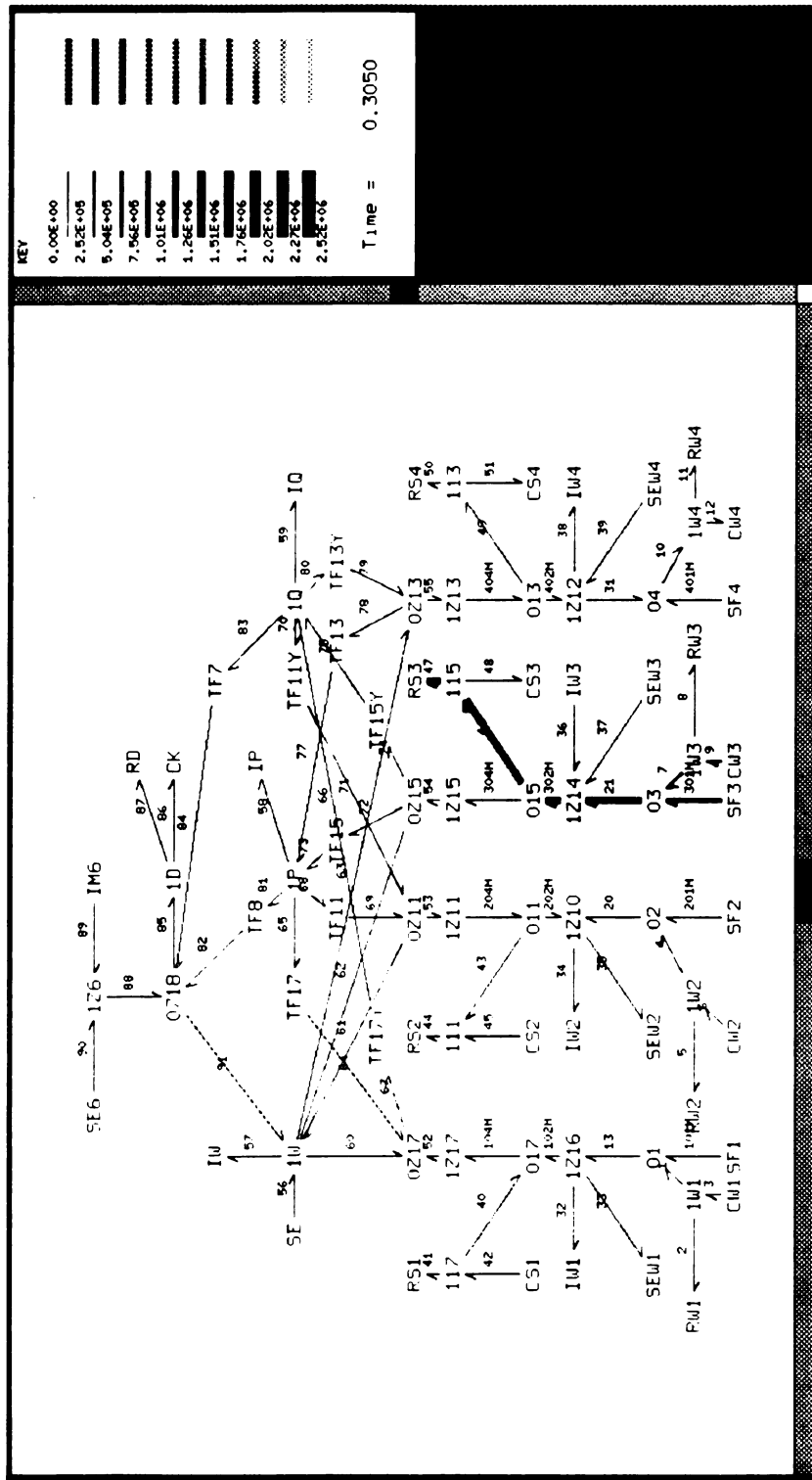


Figure 25. Power Variable Response for BPR Model (Time = 0.305).

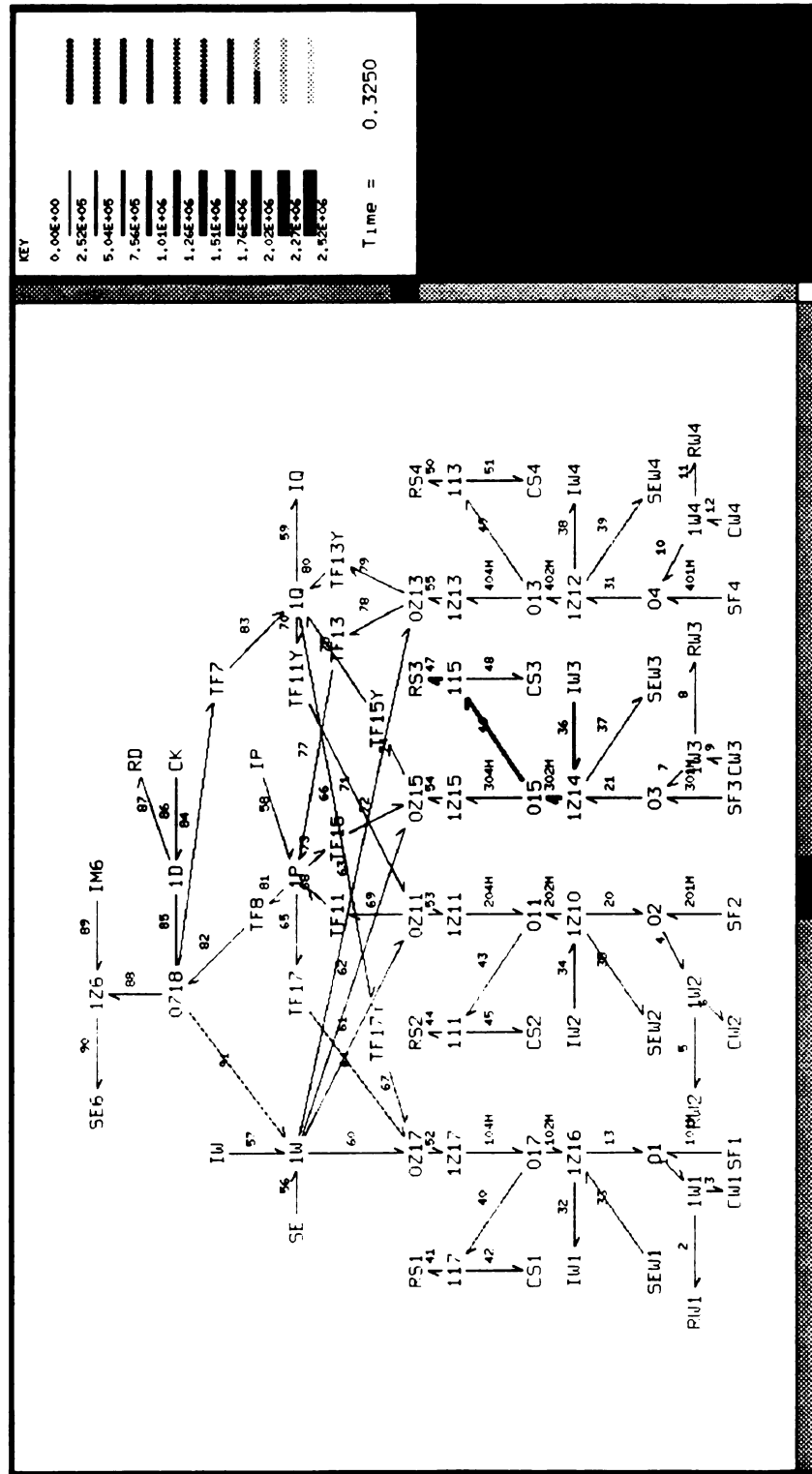


Figure 26. Power Variable Response for BPR Model (Time = 0.325).

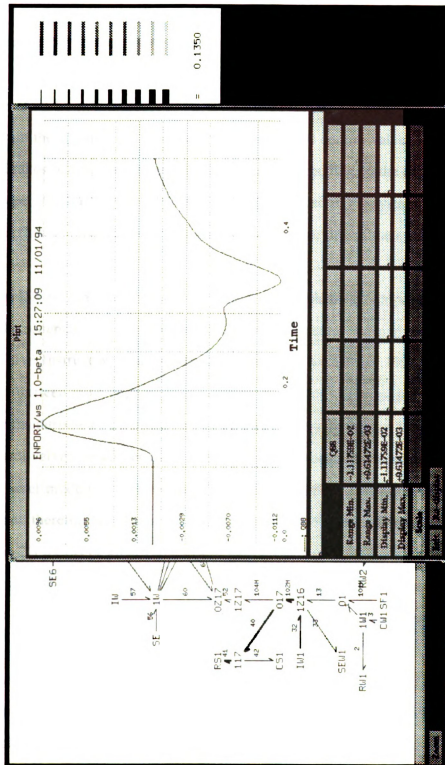


Figure 28. Time Plot of Displacement Obtained by Selecting Q88.

5.2 Railcar Air Brake Switching System Example

The DB-60 Control Valve is part of a power braking system used to stop railway train cars. The braking system consists of pressurized air brakes which provide friction to the wheels of each car to decrease its speed, a brake pipe containing pressurized air which extends from the engine to all of the cars, and the DB-60 control valve which directs air to the air brakes from storage reservoirs as necessary depending on the engineer's control of brake pipe pressure. The storage tanks are also included in the system, and the switch controls air flow to these reservoirs for charging and recharging of the system as well.

The DB-60 is a combination of hydraulic and mechanical components. Figure 29 presents the schematic diagram of the railcar airbrake control valve given in the DB-60 Control Valve Instruction Manual ME-21 (New York Air Brake, 1991). The manual consists of thirteen three-page-wide color plates and twenty-eight pages of explanatory text all for the purpose of educating an engineer about how the system functions. The air brake control valve was chosen as an example for this research because it is a complex, industrial level modeling problem involving moving mechanical parts and pneumatic circuitry, and therefore illustrates the necessity and effectiveness of better visualization. The BGVis-2D tool is of great value in supporting the construction and verification of a working model of the air brake system. A model which has been validated against actual test results would be a very useful tool for testing proposed design improvements.

Model Description. The construction of a bond graph model for a system as complex as the control valve is not an easy task. The approach taken was to hierarchically decompose the overall system into successively smaller and more detailed subcomponents and their connections. The key activity involved in this modeling decomposition is the identification of the ports (input/outputs) of the system as a whole

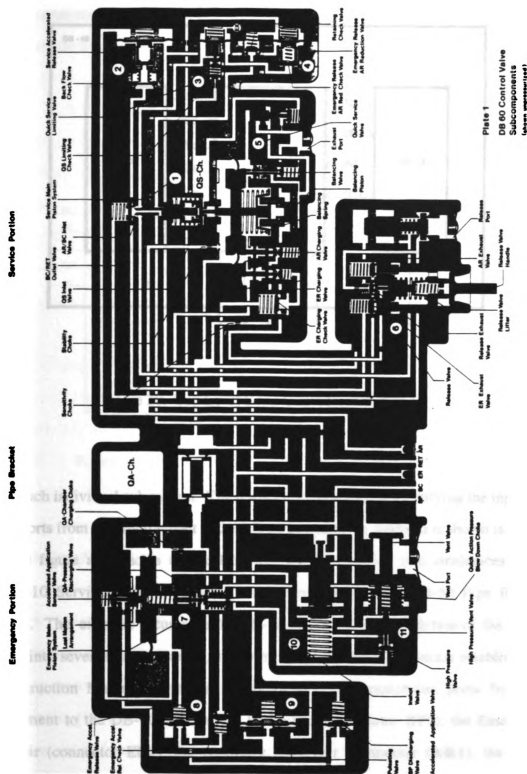


Figure 29. Schematic of the Air Brake Control Valve Switch.

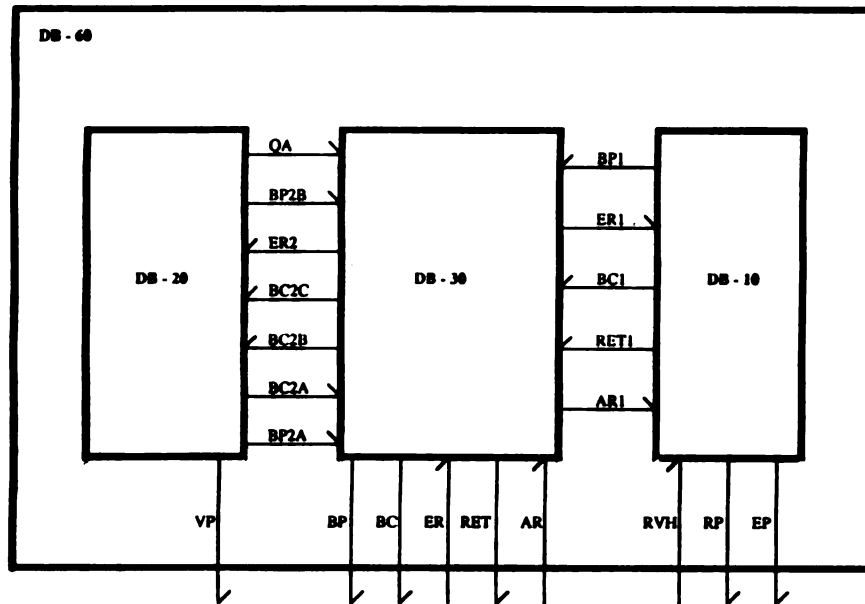


Figure 30. Top Level Bond Graph Decomposition.

and of each individual sub-system. The result of the first step of identifying the input and output ports from the environment to the DB-60 device being modeled is shown in Figure 30. This figure also shows the breakdown of the main device into subdevices called the DB-10 Service Portion, DB-20 Emergency Portion, and DB-30 Pipe Bracket modules. This example focuses on the DB-10 module. The reticulation of the DB-10 module into seven subcomponents, following the numbering convention established in the Instruction Manual, is shown in Figure 31. The pneumatic ports from the environment to the DB-10 unit are the Brake Pipe (connector **BP1**), the Emergency Reservoir (connector **ER1**), the Auxiliary Reservoir (connector (**AR1**), the Brake Cylinder (connector **BC1**), and the Retainer (**RET1**). Connectors **RP** and **EP** are ports to the atmosphere and **RVH** is a mechanical port. The figure shows sub-devices numbered

0 through 6 and their interconnections. Due to the complexity of the system, only those subdevices necessary to the functioning of the different modes of operation under study in this example were modeled. These are described in the following paragraphs.

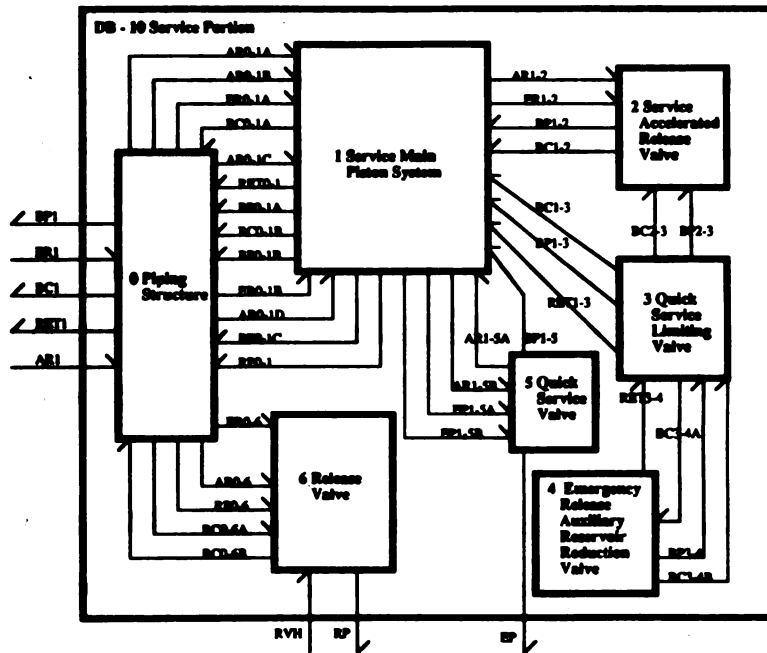


Figure 31. Bond Graph Decomposition of DB-10 Module.

Modes of operation. The DB-10 module is designed to perform many different functions depending on brake pipe (BP) pressure and pressure levels in the storage reservoirs. Several have been identified for modeling. When the storage reservoirs are empty, an increase in BP pressure causes charging of the auxiliary reservoir (AR) and emergency reservoir (ER) from the BP. Charging involves primarily the Service Main Piston System (1) (see Figure 29). In a standard braking procedure, a decrease in BP pressure causes air from the AR to be directed to the brake cylinder (BC) to stop the train. Standard braking also primarily involves the Service Main Piston. After braking is completed, BC air is exhausted to end brake application, and ER air may be fed into the

BP to support recharge of the system. Recharging occurs with assistance from the Service Accelerated Release Valve (2) as well as the Main Piston. At any time, a manual lever can be used to exhaust the AR and ER to the atmosphere. Manual release involves the Release Valve (6) in combination with the Main Piston. These subdevices of the model were modeled appropriately and correctly connected to each other. Equations were set up for each of the elements using reasonably estimated parameters. With the parameters set, simulations of the different modes of behavior of the device were completed. Before presenting the entire system model, a modeling and simulation exercise involving the Service Accelerated Release Valve (2) will first be described. The behavior of this subsystem is also visualized before moving on to the full Air Brake Valve System model.

Check Valve Modeling. To verify the concepts involved in creating bond graph models of pneumatic/mechanical devices, especially modeling of check valve (CV) behavior, a bond graph model of the Service Accelerated Release Valve (2) only was created. A detailed schematic of the subcomponent is shown in Figure 32. The bond graph model for this device is shown in Figure 33. The device consists of a Back Flow Check Valve (2-1) and a diaphragm actuated valve (2-2) which together control flow from the emergency reservoir to the brake pipe.

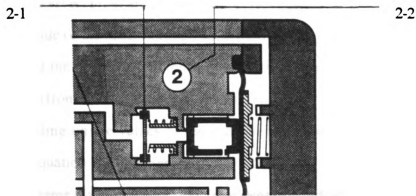


Figure 32. Schematic of the Service Accelerated Release Valve (2).

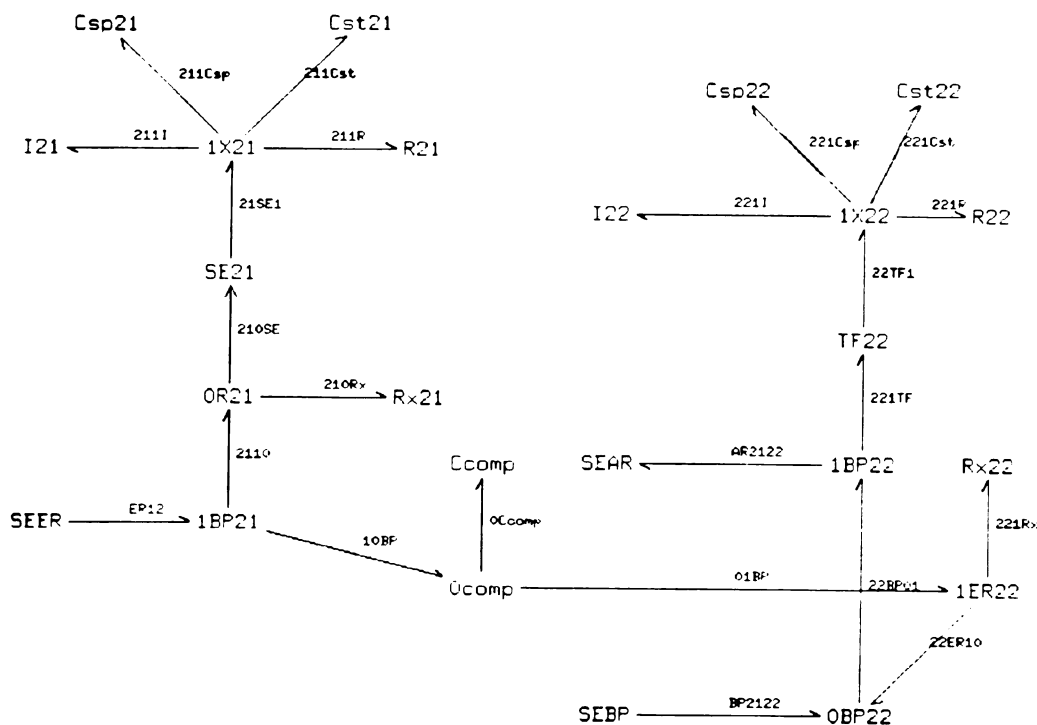


Figure 33. Bond Graph of the Service Accelerated Release Valve.

The CV is modeled as a mass which is allowed to translate in the X direction (**1X21**). Forces on this mass which affect its motion are 1) the force due to the spring (**Csp21**); 2) a force due to the stop when it is encountered by the mass (**Cst21**); 3) a force due to viscous friction (**R21**); 4) an inertia force (**I21**); (5) and a force due to the differing pressures on each side of the mass (through bond **21SE1**). When the displacement of the CV is positive, and the pressure difference over the CV is positive, there will be flow through the valve (from **SEER** to **0comp**) which can be calculated by a non-linear relationship depending on the values of the pressure difference and the valve opening dimension. This equation is implemented by the **Rx21** node. The diaphragm actuated valve works in the same manner, where flow from **0comp** to **SEBP** is only allowed if the pressure difference over the diaphragm overcomes the spring force causing the valve to

open. Therefore, in order for flow to occur from the ER through to the BP, the AR pressure must be lower than the BP (to open the diaphragm actuated valve), and the BP pressure must be lower than the ER pressure (to open the check valve). This sub-component thus provides ER air to the BP for recharging when the AR pressure is low, for example after a standard braking application.

A simulation of the Service Accelerated Release Valve has been performed to verify the basic behavior of the model. Parameters were estimated to achieve reasonable results. Details of the parameters and equations used for the bond graph elements shown in Figure 33 can be found in Appendix C.

Verification of the expected functioning of the component can be easily seen by using the BGVis-2D display of the power variable type on the bond graph using line width variation. Figures 34 and 35 show two different time stages during the course of the animation. In Figure 34, at time = 0.5 it can be clearly seen that no power transfer is occurring in the device (minimum line widths for all bonds), since the AR pressure is low, but BP and ER pressures are equal. Figure 35 is used to illustrate the use of color as well as line width variation by the BGVis-2D tool. In Figure 35, at time = 0.76, the AR pressure is still low, and the BP pressure has experienced a smaller decrease. Color variation on the bonds shows pressure on bond **ER12** is greater than pressure on bond **BP2122** which is greater than pressure on bond **AR2122**. The line widths show that there is power flow through the valve (large widths for bonds all along the path from **SEBP** to **SEER**). The arrow directions indicate the direction of flow. Thus the model behaves as intended. In addition, Figures 36 and 37 show the flow variable type displayed with line width. At time $t = 0.21$ seconds, the AR pressure is decreasing, causing the diaphragm to move. This velocity is shown by the large line widths connected to the **1x22** node in Figure 36. At time $t = 0.6$ seconds (Figure 37), the BP pressure is decreasing. This

causes the check valve to move into the open position (larger bond widths on the **1x21** node), while the diaphragm (**1x22**) moves back somewhat toward its initial position since the pressure difference between AR and BP has decreased. Arrow direction changes on the bonds connected to node **1x22** show the occurrence of negative flow. The visualization provides a qualitative analysis of what is going on in the device without having to select individual bond values for display of time plots.

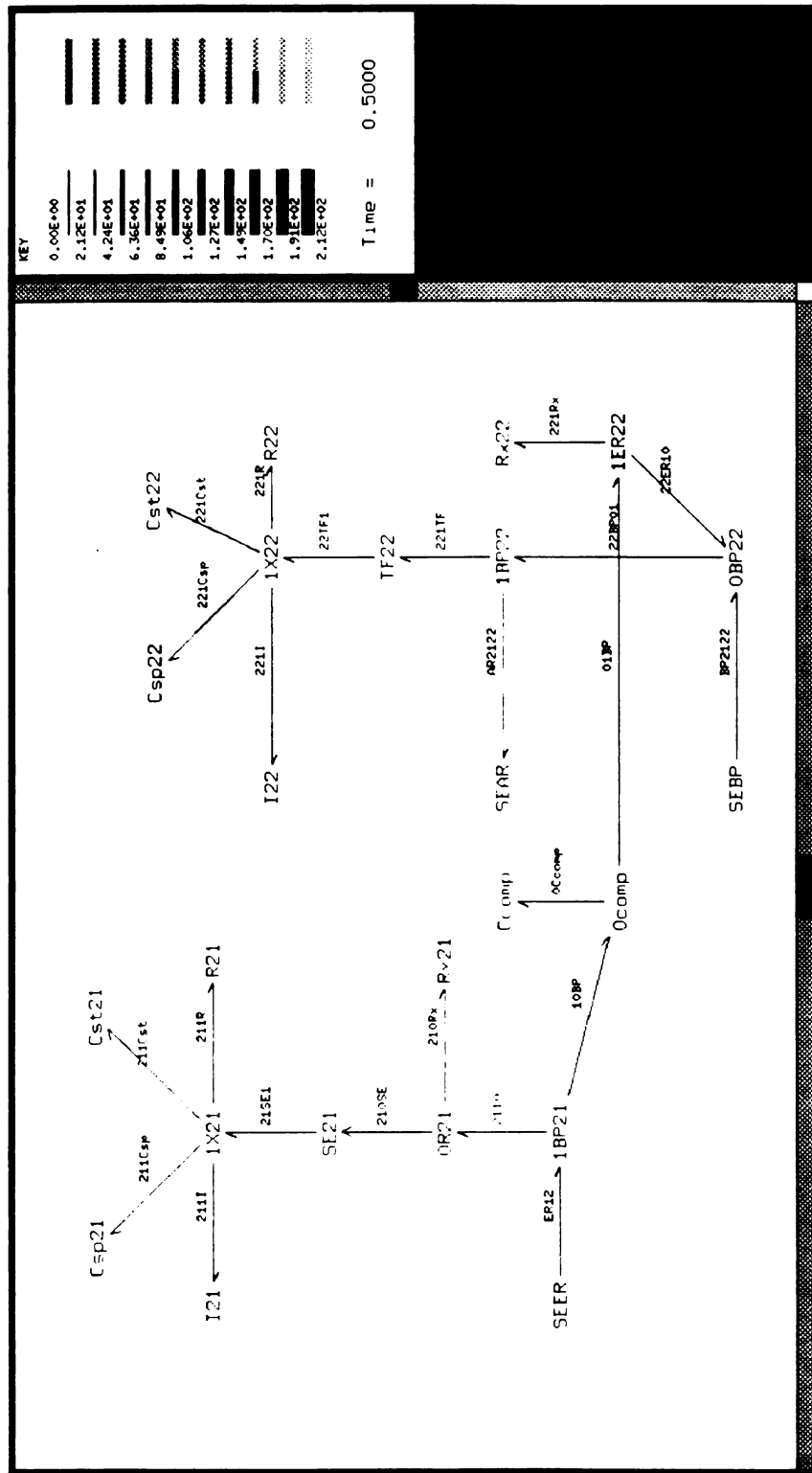


Figure 34. Power Variable Display for Check Valve (Time = 0.5).

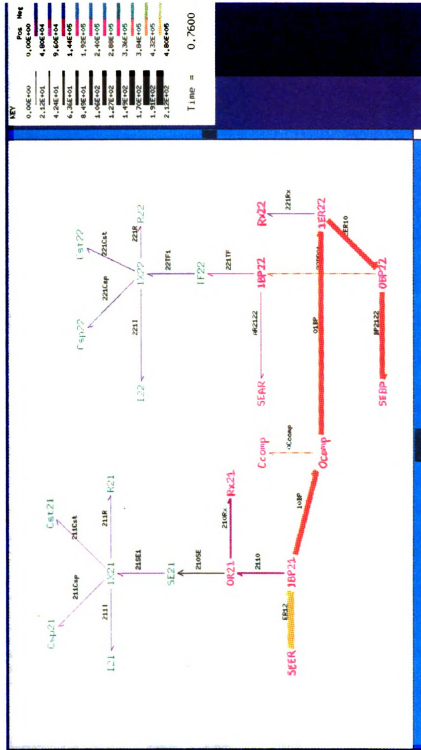


Figure 35. Power and Effort Variable Display for Check Valve (Time = 0.76).

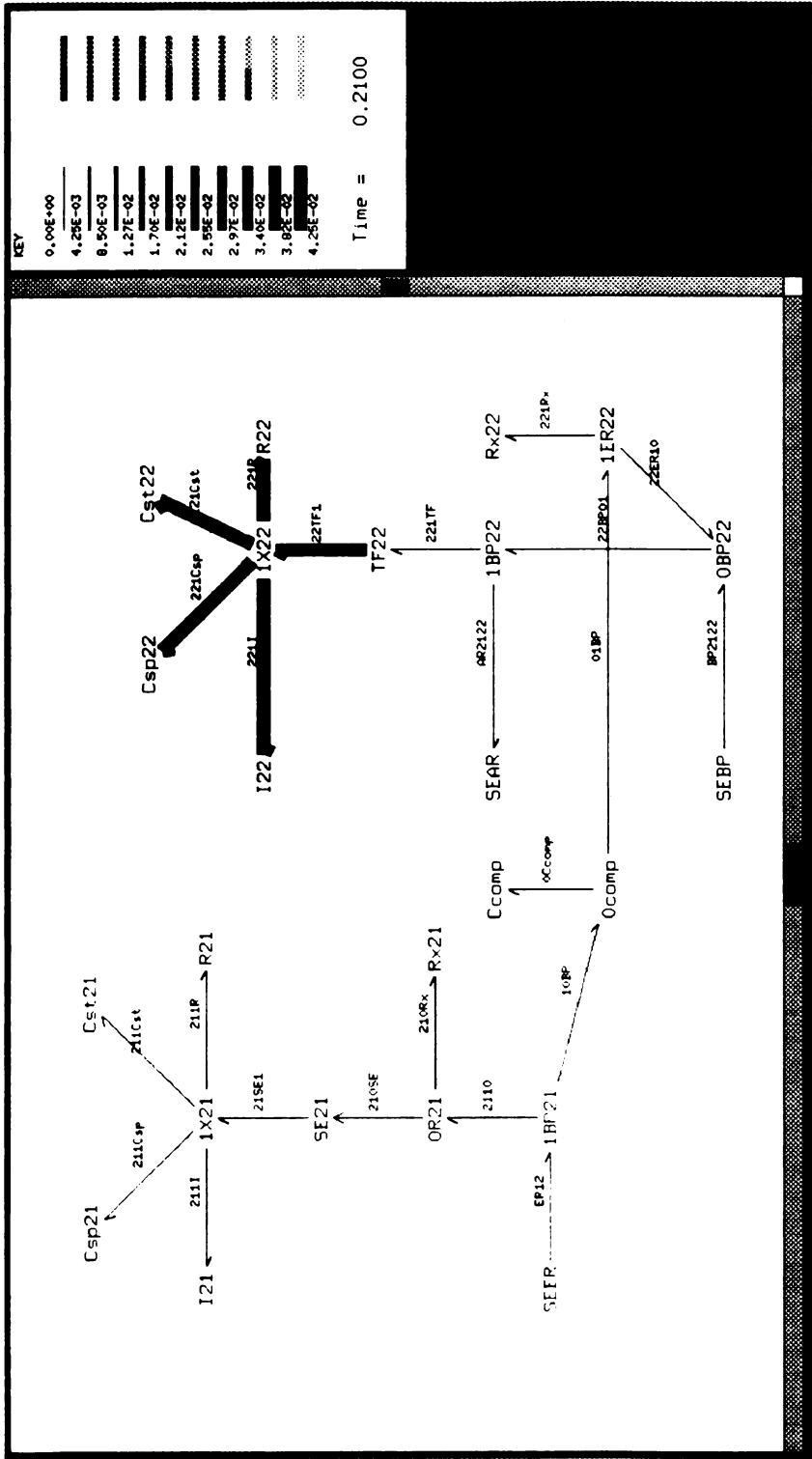


Figure 36. Flow Variable Display for Check Valve (Time = 0.21).

Figure 37. Flow Variable Display for Check Valve (Time = 0.6).

Main Piston Modeling. With regard to the overall air brake control valve modeling task, the objective was to model the modes of behavior of the device described previously using ENPORT. The modes of operation that were simulated and visualized in this example were: 1) initial charging of the auxiliary and emergency reservoirs, 2) standard braking where air from the auxiliary reservoir is directed to the brake cylinder, 3) secondary recharging where brake cylinder pressure is released and air from the emergency reservoir is directed to the brake pipe, and 4) manual release of auxiliary and emergency reservoir pressure. A bond graph of the parts of the Main Piston System (1), Service Accelerated Release Valve (2), and the Manual Release Valve (6), which are involved in the modes of operation of the device described previously is shown in Figure 38. This bond graph models the ER and the AR each as a C type storage element (**car** and **cer**). The BP pressure is the input to the system (**sebp**). The velocity of the diaphragm of the main piston is represented as node **1x**. The forces on the node include a spring, seat, viscous damping, and inertia as in the previous check valve example. Also included is a force generated when the diaphragm comes in contact with the ER charging valve stem (**1x3**), a force generated when the diaphragm comes in contact with the AR charging valve stem (**1x2**), and a force generated in the opposite direction when the diaphragm comes in contact with the BC inlet valve (**1x5**). The hydraulic flow rate through these valves is determined from the valve opening dimension in elements **rx**, **rx2** and **rx5** respectively. The pressure difference across the diaphragm is calculated at node **1tf** and the force on the diaphragm is determined by multiplying the pressure difference by the diaphragm area at the **tf** node. The **0bp** node is the pressure above the diaphragm, while the **0ar** node is the pressure below the diaphragm. These areas are connected through several valve and choke pathways to the reservoirs and input pressures. The ER Charging Check Valve is part of the Main Piston sub-assembly and is included with node **1x4** and associated nodes as shown in the box on the graph. The Service Accelerated Release Valve is included at the top of the bond graph with appropriate connections to

reservoirs and inputs. The Manual Release sub-component is modeled by the **lrel** junction. A mechanical force can be input to this element via the **serel** node, and the position of the release valve controls the volume flow rate through the **rxar** and **rxer** elements.

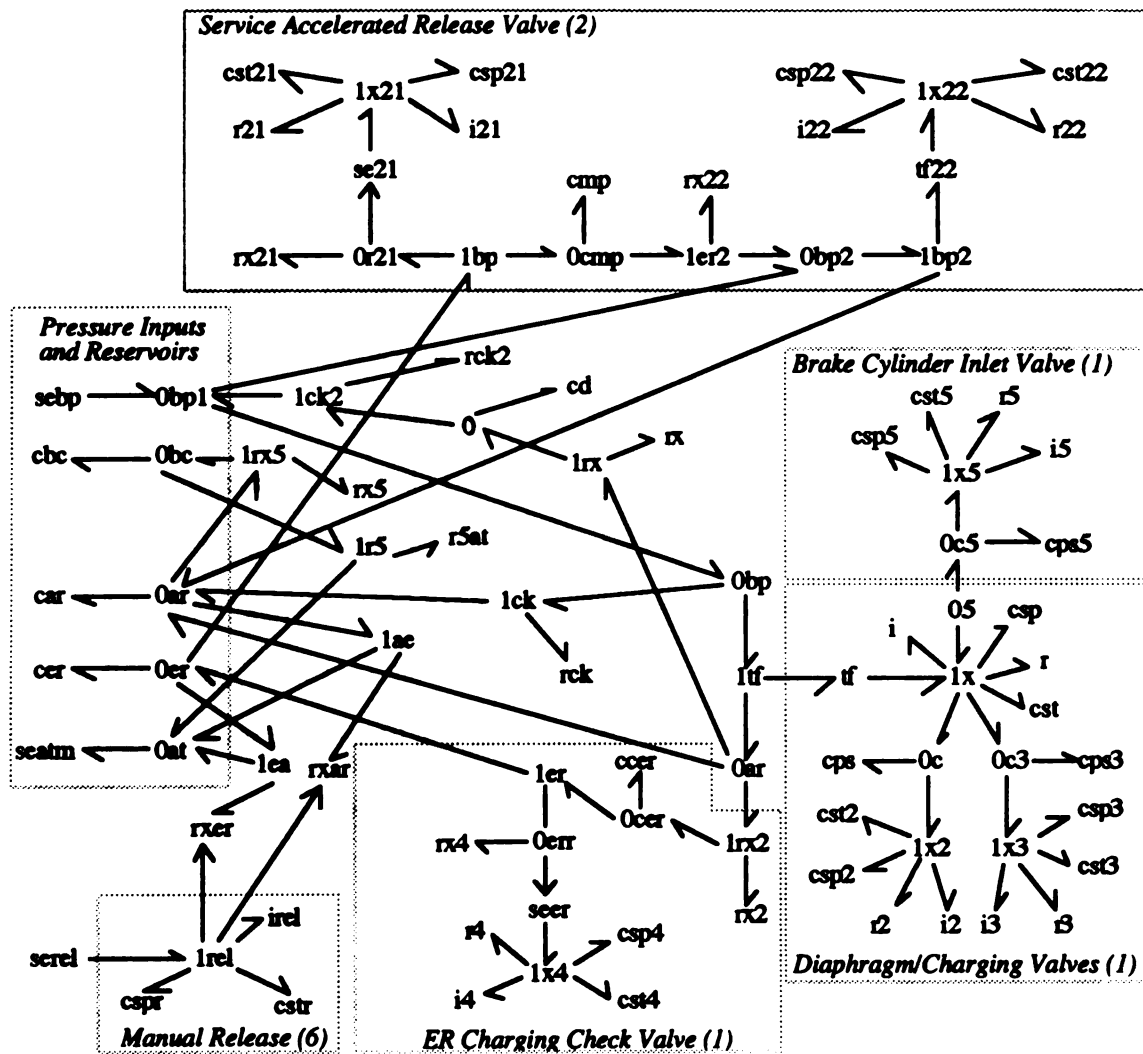


Figure 38. Bond Graph of the Main Piston System and Reservoirs.

Mode 1: Initial Charging of the Reservoirs. The results of the simulations of the four modes were visualized using the BGVis-2D tool in order to show how the tool helps in making the behavior of the system easier to comprehend. The first mode is initial charging of the AR and ER from the BP. Charging of the system is accomplished by a linear increase in BP pressure supplied as an input. When the BP pressure is increased, it is expected that the pressure in the storage reservoirs will also experience an increase in pressure due to flow of air from the BP through the main piston system. Figures 39 through 42 show this response. Power is shown as a variable type using line width. At time = 0, there is no flow, and all initial pressures are set to zero. Initial displacements for all of the mechanical valves are set to their equilibrium values (see Appendix C). In Figure 39, the pressure in the BP has increased, which causes flow to start through the main piston, which causes the pressure in the AR to increase. Figure 40 shows the system at a later time where the BP pressure has become high enough to start flow into the ER. Figure 41 shows the response as the BP pressure nears its maximum. Air is still flowing into both tanks and power is higher due to the higher pressures and flow rates involved. Figure 42 shows the response after the brake pipe has reached its maximum pressure level, and is now remaining steady. Since there is less pressure difference across the diaphragm, flow into the reservoirs decreases. Eventually all pressures are equal, so all check valves are closed and no more flow in the sub-assembly occurs. The manual calls this situation charging lap position.

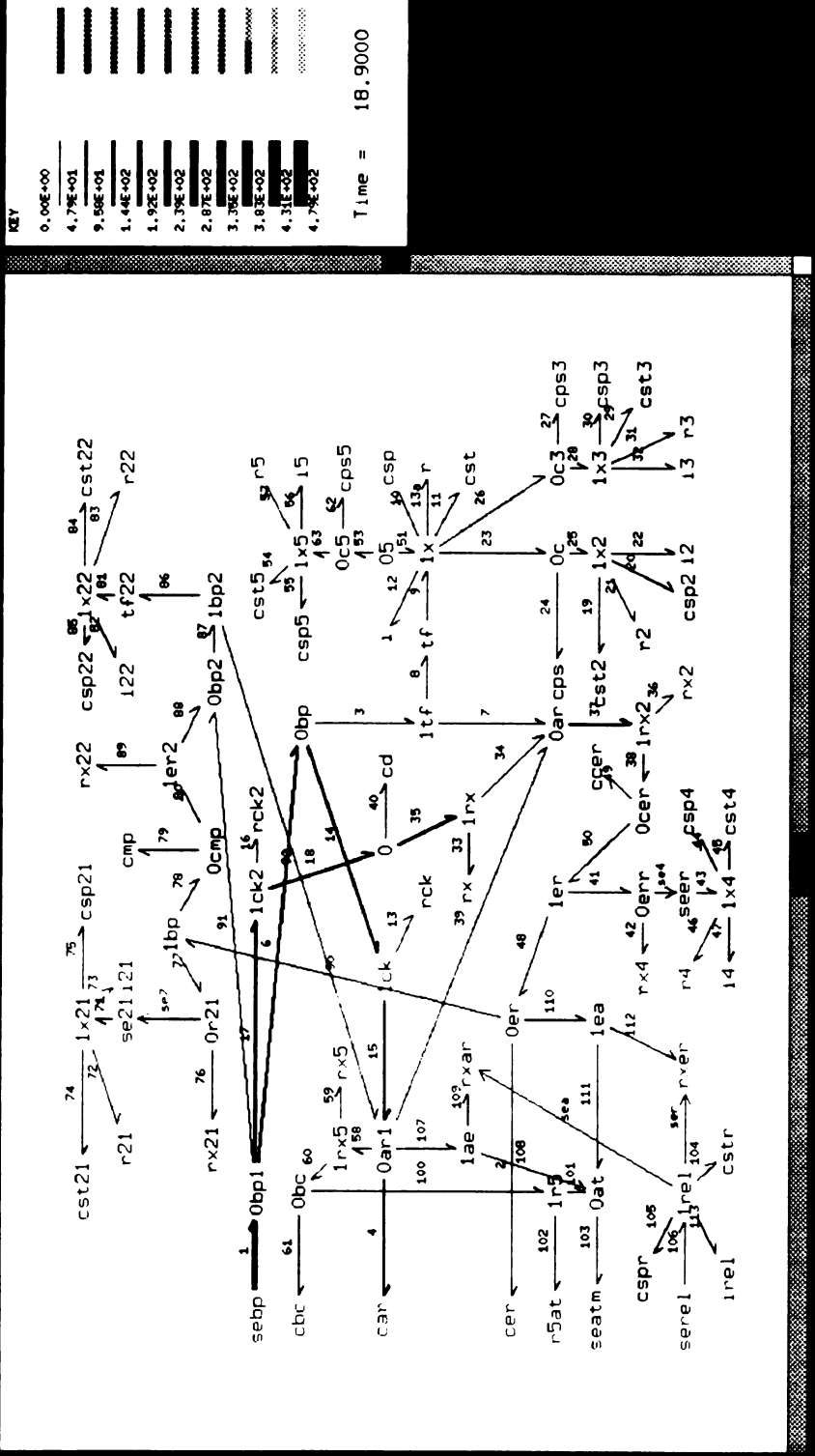


Figure 39. Power Response for Air Brake Model Charging Mode (Time = 18.9).

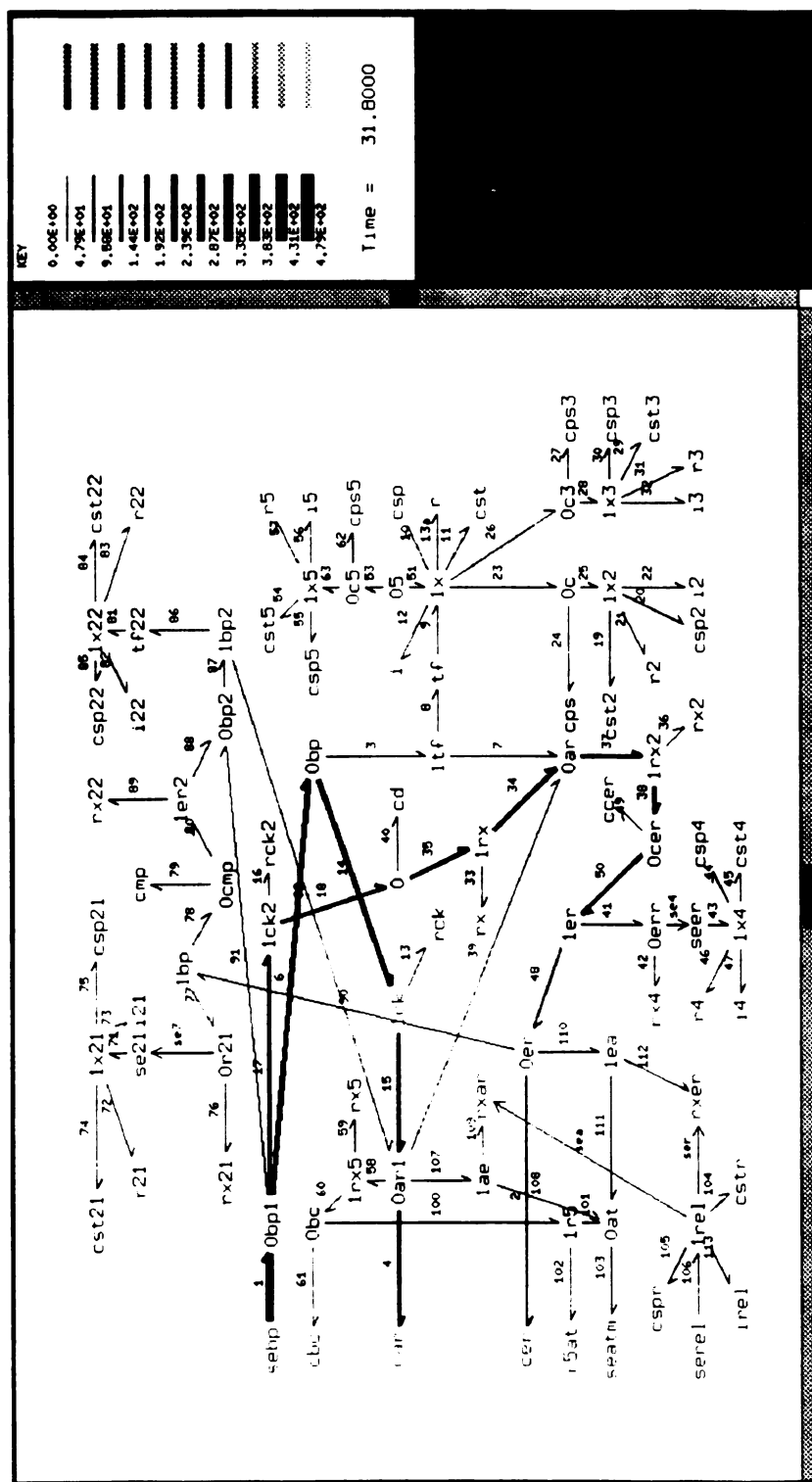


Figure 40. Power Response for Air Brake Model Charging Mode (Time = 31.8).

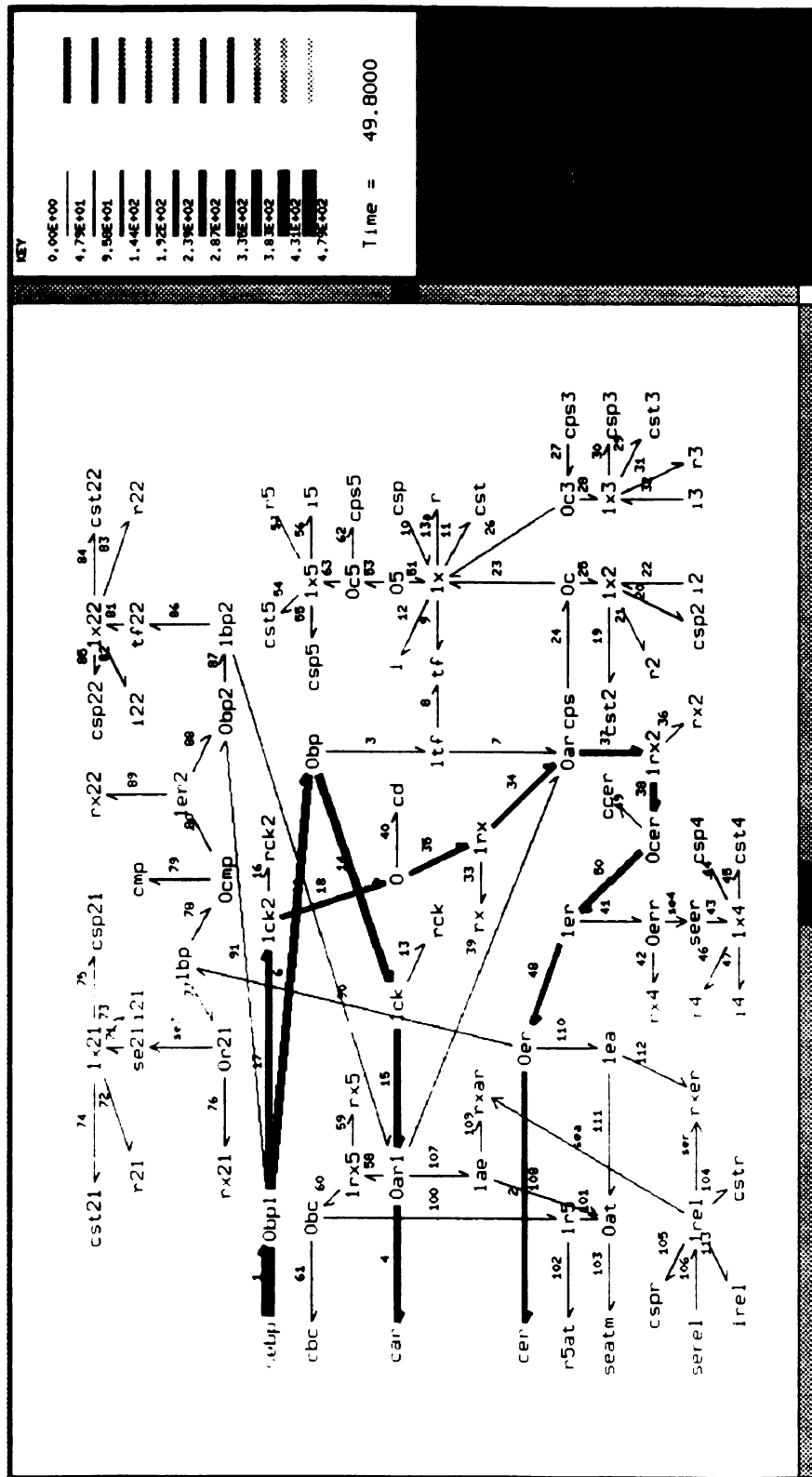


Figure 41. Power Response for Air Brake Model Charging Mode (Time = 49.8).

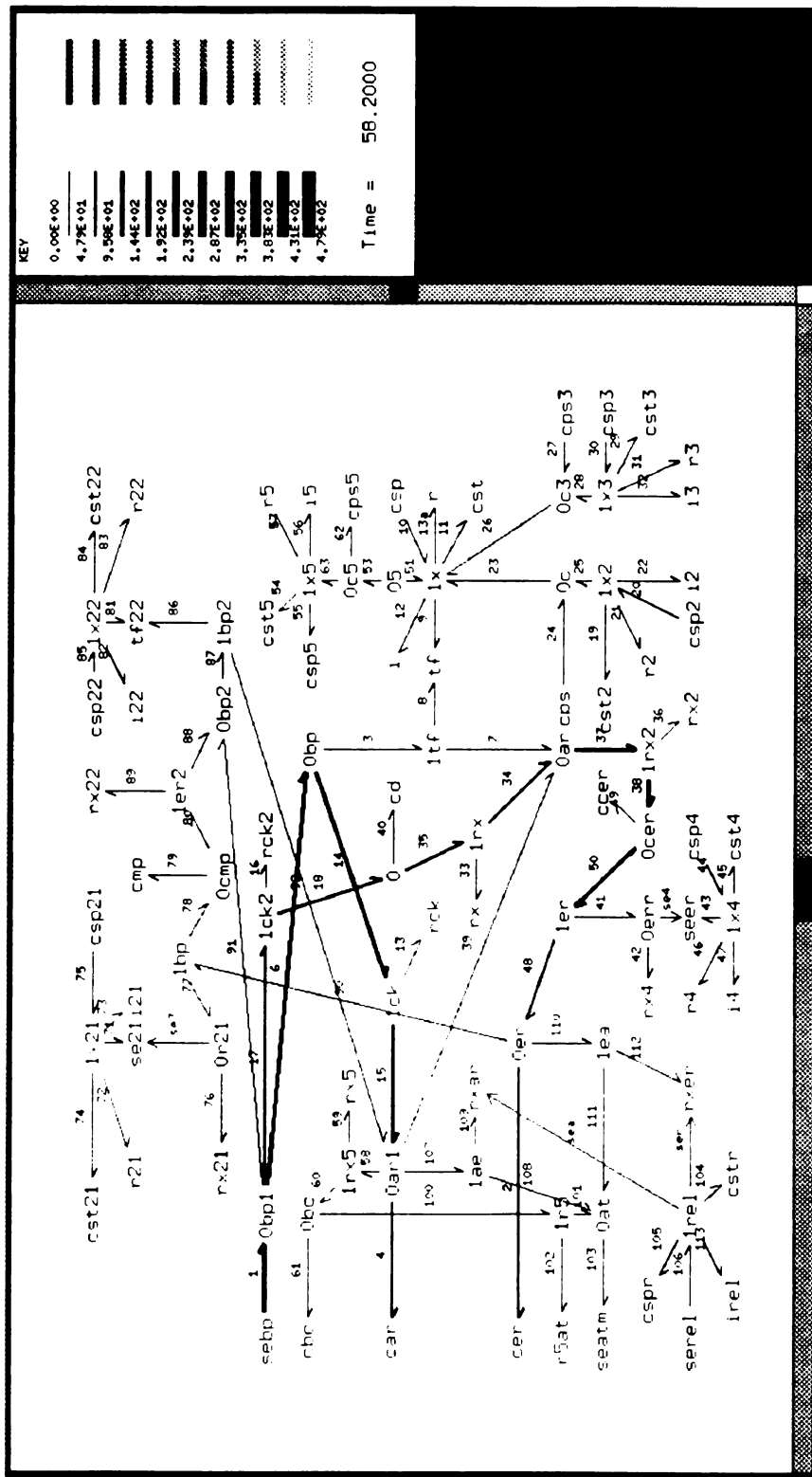


Figure 42. Power Response for Air Brake Model Charging Mode (Time = 58.2).

Standard braking. The main piston system is also involved in sending air from the AR to the BC when the BP pressure is decreased, if AR and ER pressures are high. This mode is known as standard braking and is shown in Figures 43 through 45. The system starts out in a fully charged state (charging lap position) with the initial conditions for nodes **car** and **cer** set to a high value (see Appendix C for values). Power is shown as the variable type using line width once again. In Figure 43, the pressure in the BP has decreased, which causes the main diaphragm to shift in the opposite direction from when charging, opening the BC inlet valve. Power can be seen to be flowing into node **cbc** and also some power is lost to the BP as AR pressure tries to equalize with BP pressure. Figure 44 shows the system when BP pressure has stabilized at a lower value. The AR is still losing energy to the BP. Figure 45 shows flow decreasing to the point where the system will once again be in lap position, but at a lower overall pressure.

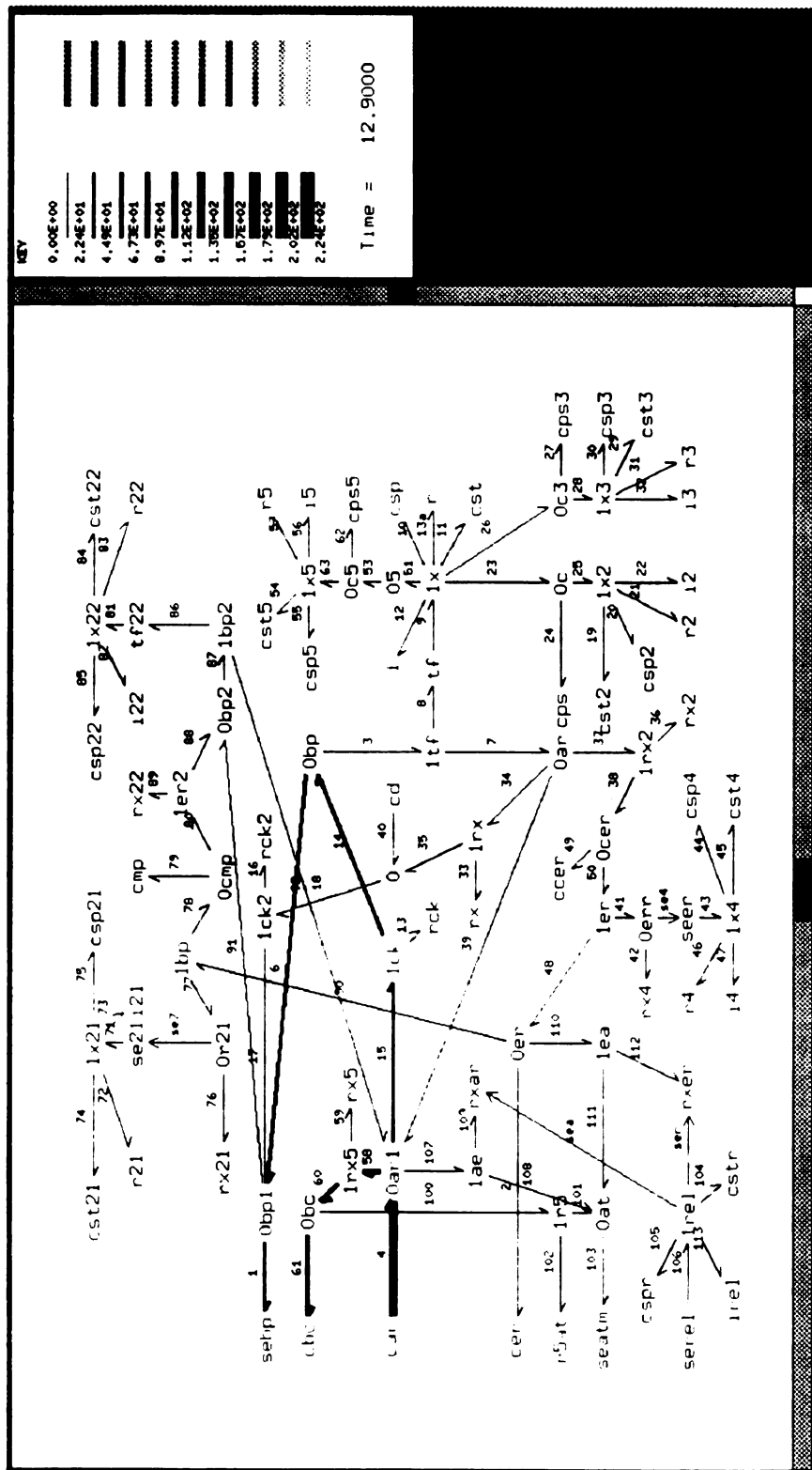


Figure 43. Power Response for Air Brake Model Braking Mode (Time = 12.9).

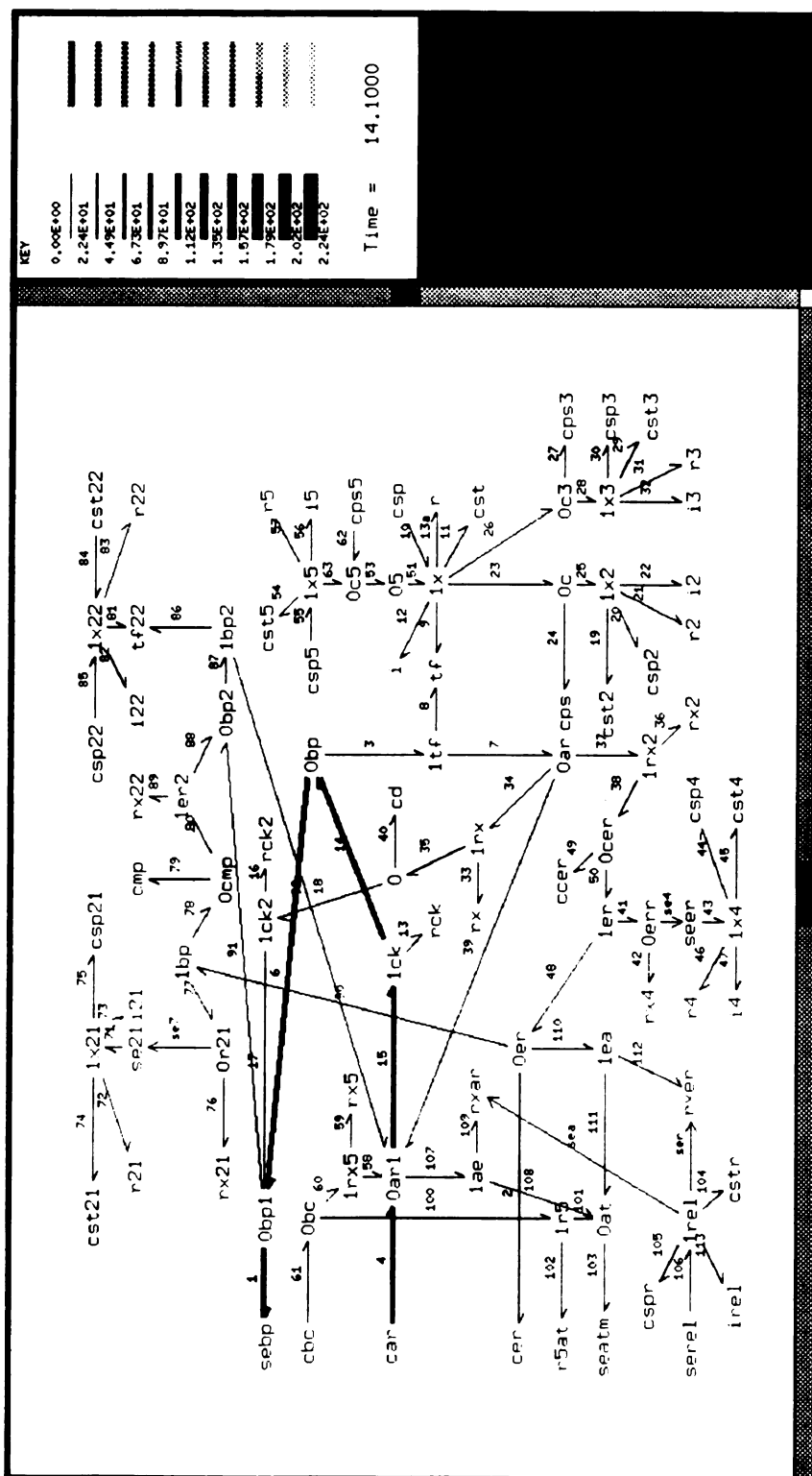


Figure 44. Power Response for Air Brake Model Braking Mode (Time = 14.1).

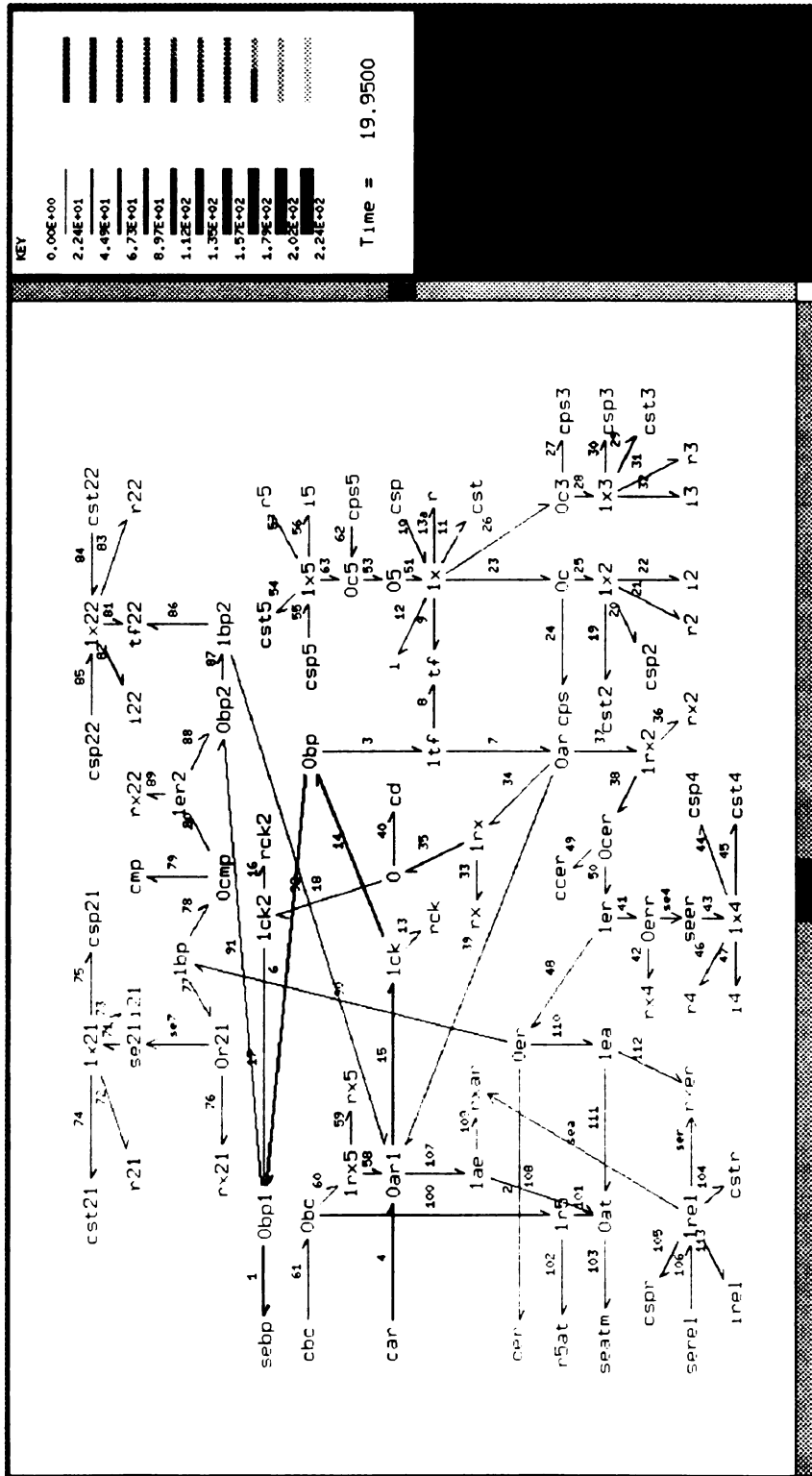


Figure 45. Power Response for Air Brake Model Braking Mode (Time = 19.95).

Recharging. After a standard braking application, when the system is in service lap position, an increase in BP pressure should cause release of BC pressure to the atmosphere to end the braking action. Also, the AR should be recharged from the BP to ready the system for the next braking action. The Service Accelerated Release Valve contributes to this recharging initially by sending ER air into the BP for AR recharging. Figures 46 through 49 show accelerated recharge with the flow variable shown as line width variation. In Figure 46, the pressure in the BP (**sebp**) is starting to increase, which causes flow from **cbc** to **seatm**. In this figure the wider line widths around the **1x21** node also indicate that the check valve of the Service Accelerated Release Valve is moving to a new position to allow flow through the valve. Figure 47 shows the system at a later time when the flow out of the BC has increased and a small amount of flow can be seen to occur from ER to BP through the Service Accelerated Release Valve. Figure 48 shows the response as the BP pressure continues to increase. The BC is still emptying while air is starting to flow from the BP to recharge the AR. Finally, Figure 49 shows the response after the BP has steadied out at its maximum pressure level. Small amounts of flow are still occurring. Soon the system returns to lap position.

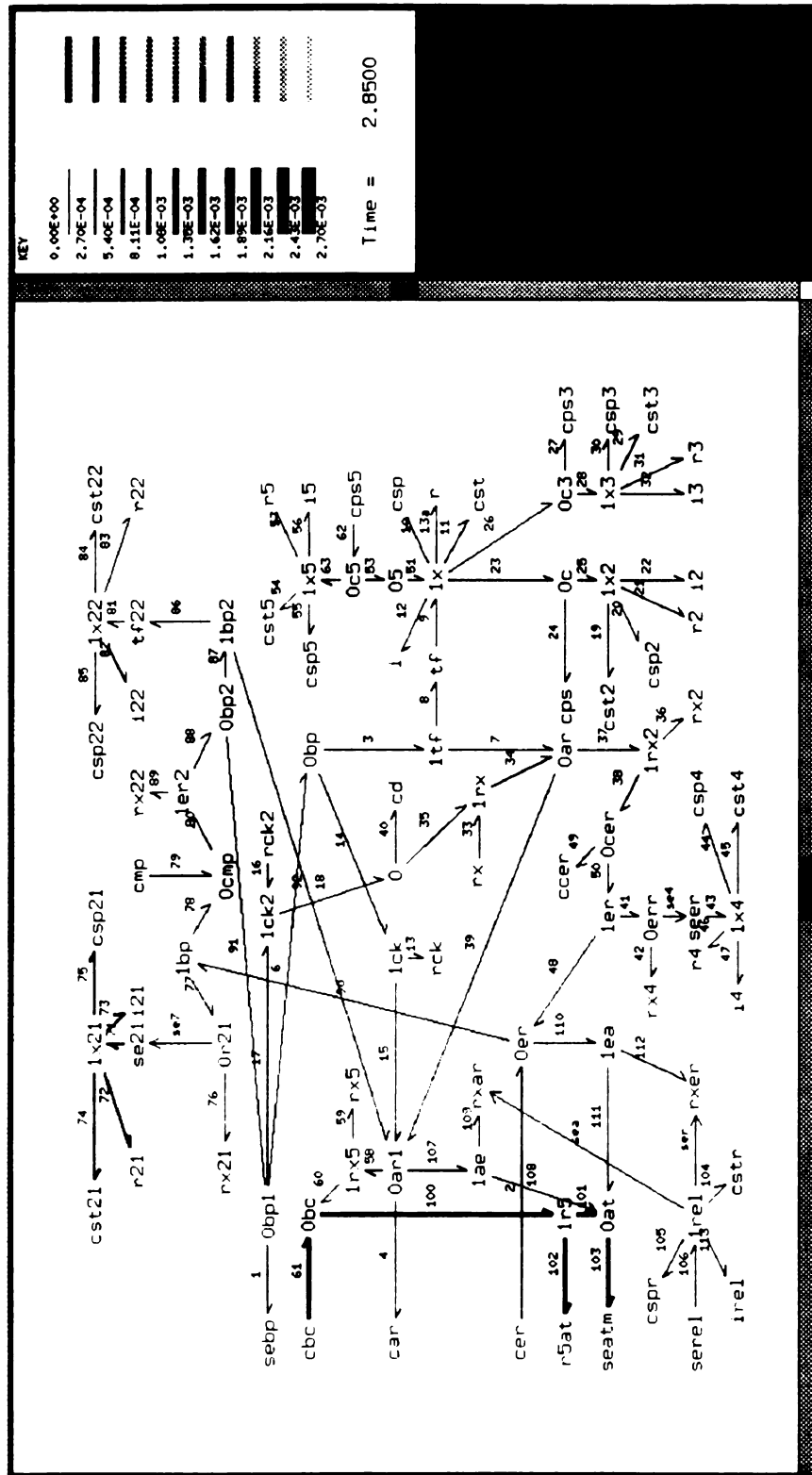


Figure 46. Flow Response for Air Brake Model Recharging Mode (Time = 2.85).

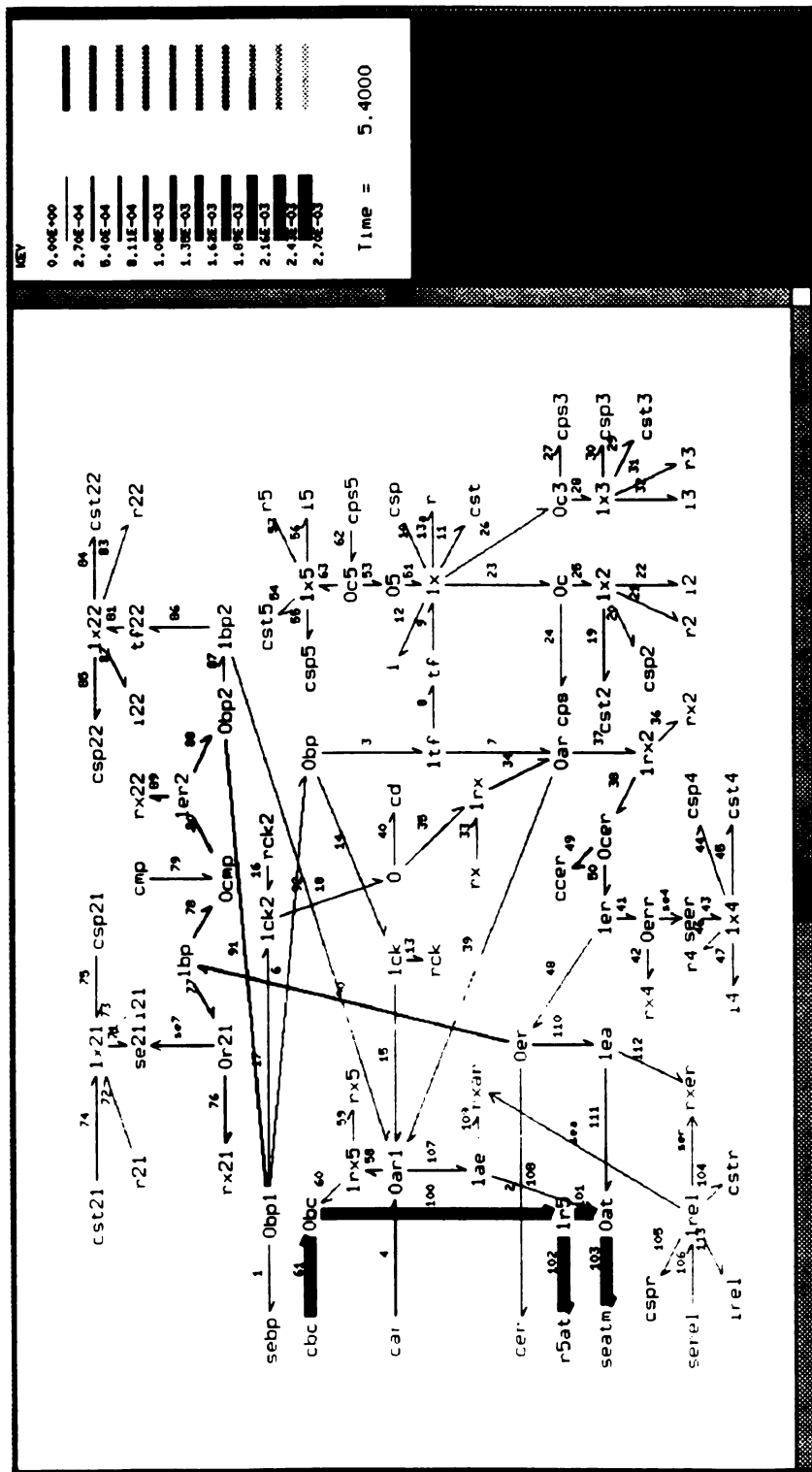


Figure 47. Flow Response for Air Brake Model Recharging Mode (Time = 5.4).

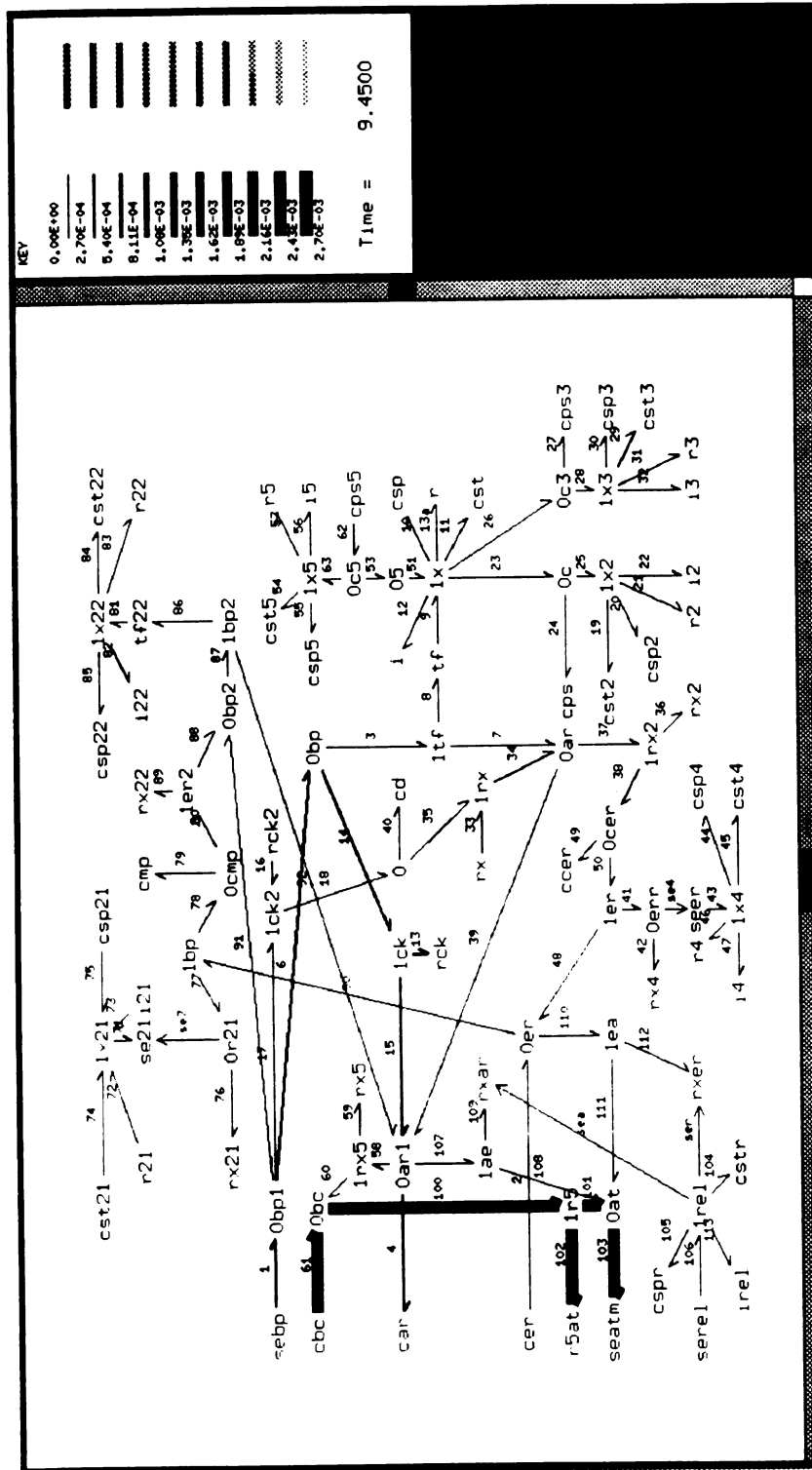


Figure 48. Flow Response for Air Brake Model Recharging Mode (Time = 9.45).

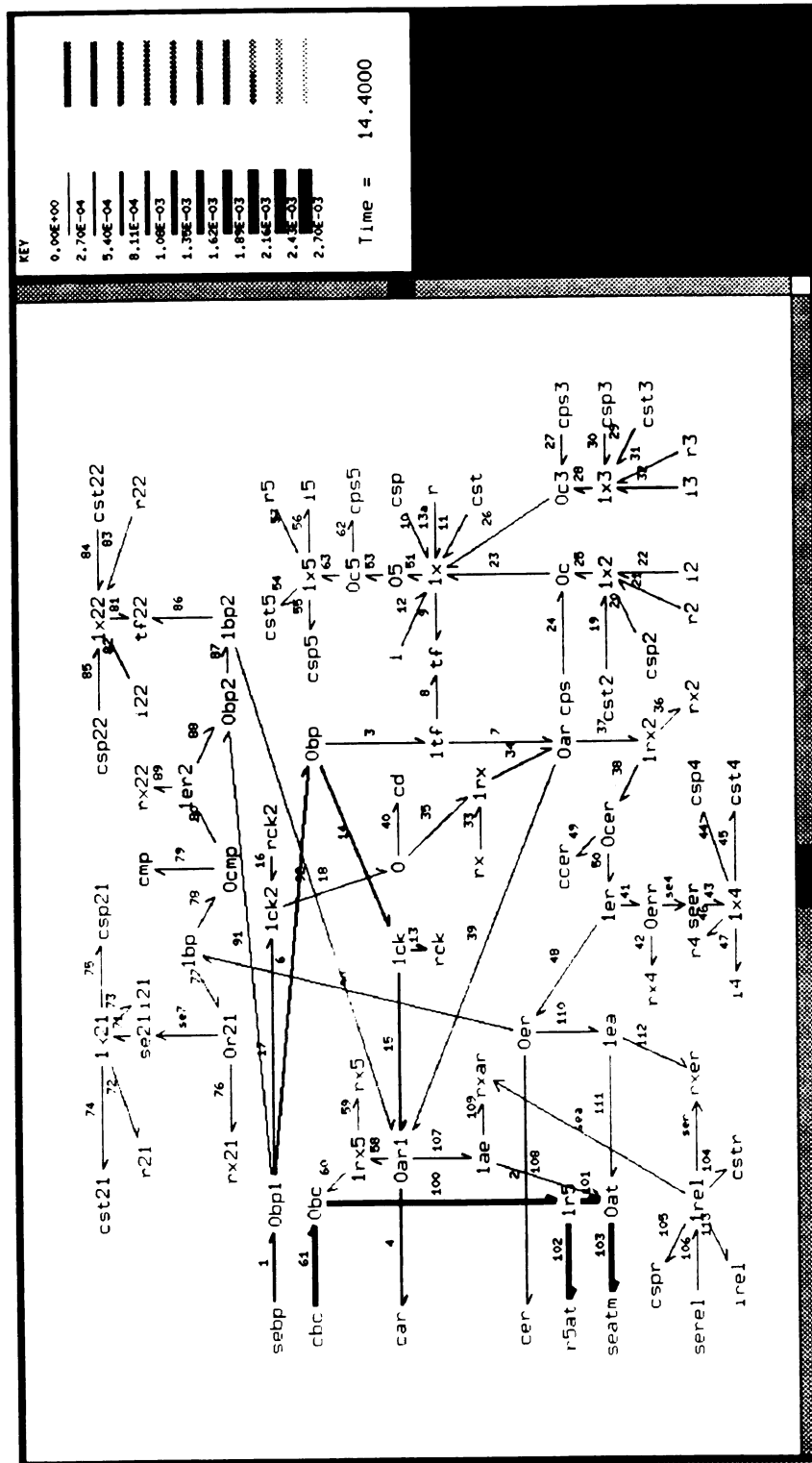


Figure 49. Flow Response for Air Brake Model Recharging Mode (Time = 14.4).

Manual release. From any lap position the option should always be available to manually release the pressure from the AR and the ER. When AR pressure decreases this will also cause release of the BC, assuming that BP pressure remains fixed. Figure 50 shows this occurrence. The power variable is displayed with line width variation. The force input **serel** is increased by a set amount to cause the manual release valve to move up from its valve seat, creating a flow path from ER and AR to atmosphere. It can be seen on the graph that power is high along the paths from **car** and **cer** to **seatm**. Nearly all of this power is dissipated in the flow resistances (**rxar** and **rxer**). The power is low on the bonds connected to **0at** because the atmospheric pressure is a very small value compared to AR and ER pressures. A small flow also occurs along the path from **cbe** to **seatm**. In Figure 51, AR and ER pressures have decreased to the point that flow into the system from BP starts to occur. If the system were held in this position with the manual release valve held open, pressures would all stabilize to that of the BP, and a constant flow would occur from BP out to atmosphere.

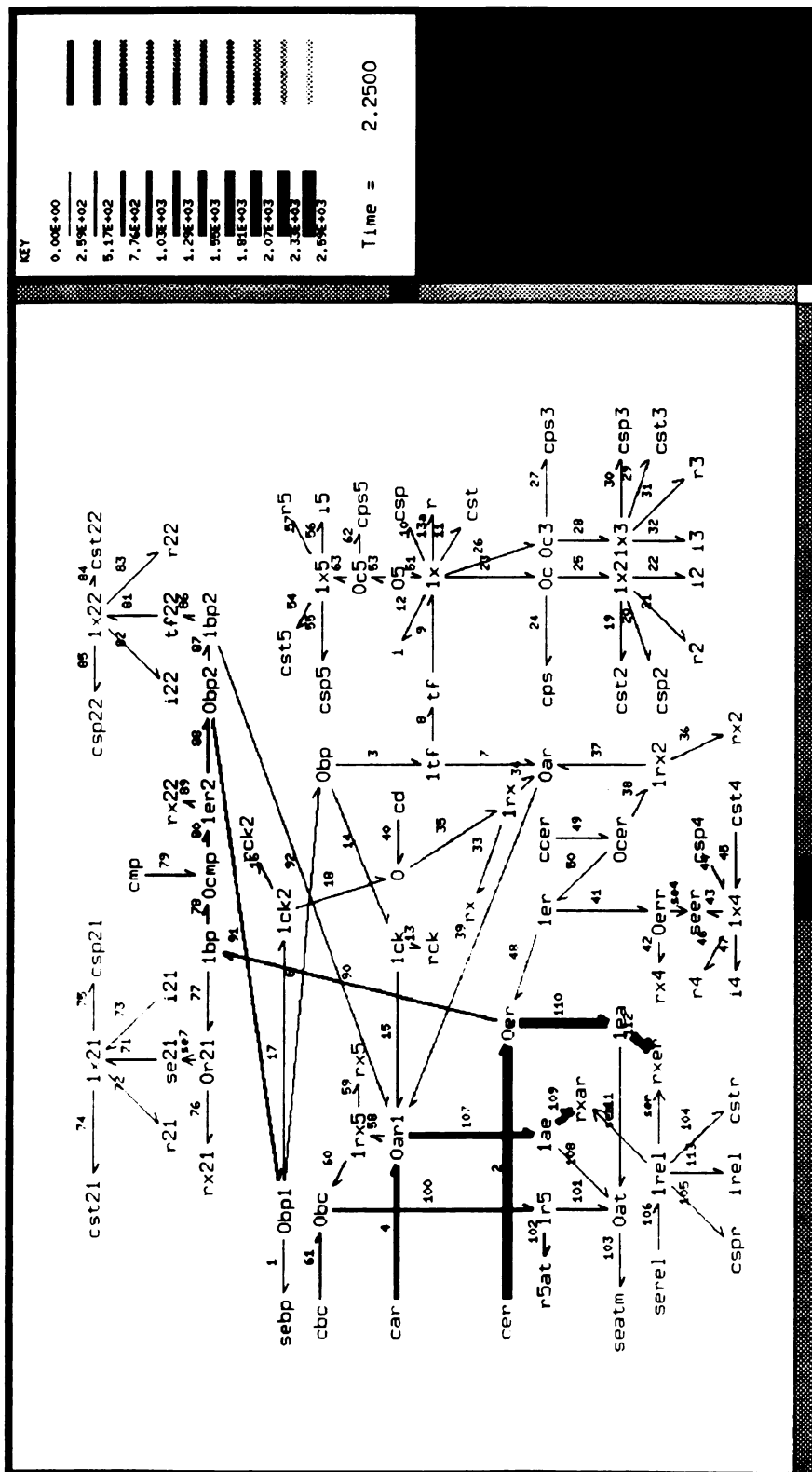


Figure 50. Power Response for Air Brake Model Release Mode (Time = 2.25).

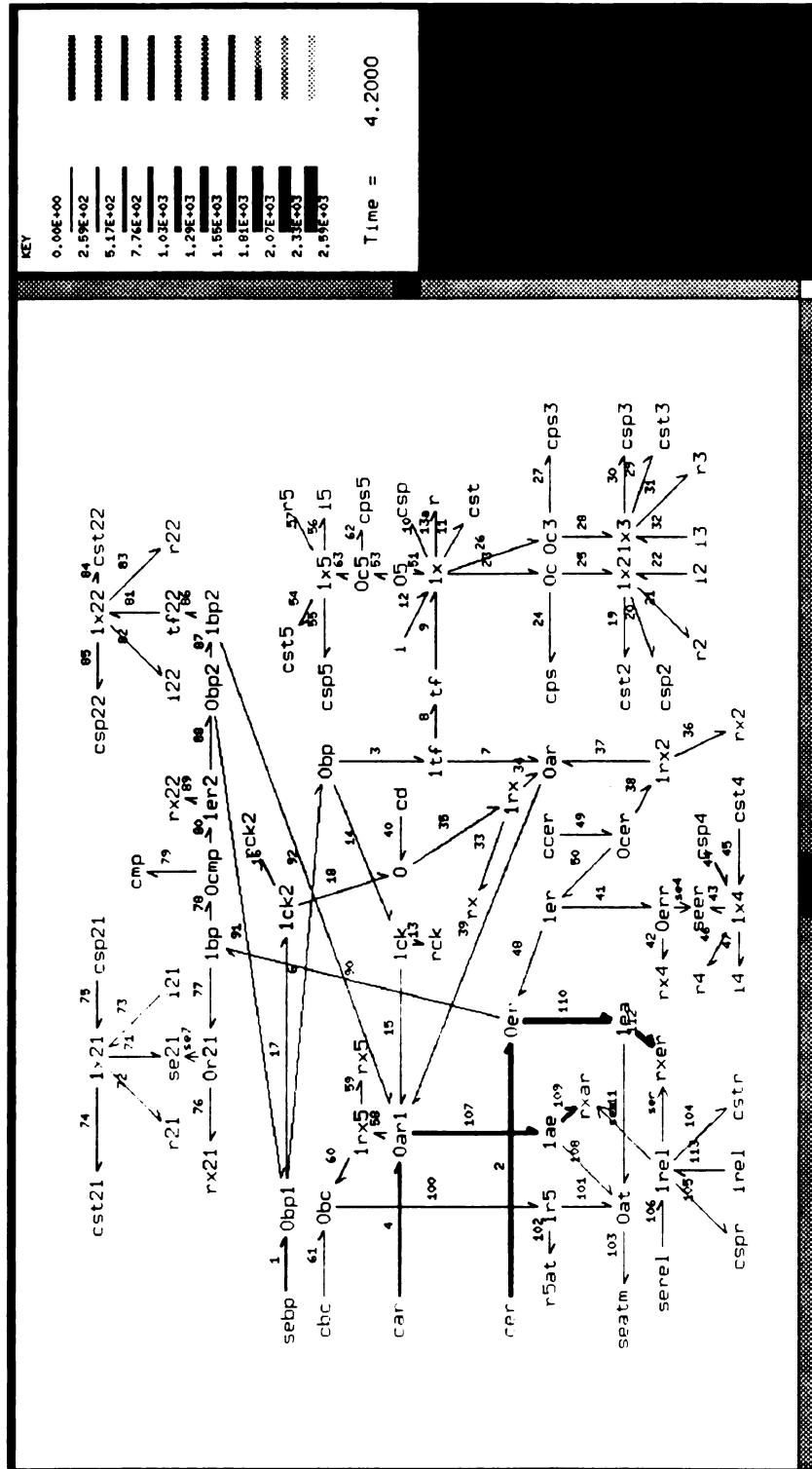


Figure 51. Power Response for Air Brake Model Release Mode (Time = 4.2).

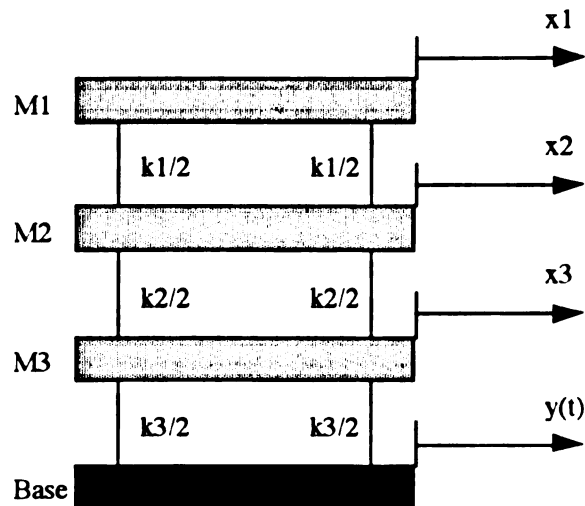


Figure 52. Schematic of Mechanical Three DOF System.

5.3 Modal Analysis Example

In addition to applying visualization techniques to industrial scale systems design problems, there is also usefulness in extending the concepts to educational problems, in order to aid students in their understanding of basic phenomena. One of these educational applications of visualization is modal analysis in vibrating systems. Mathematical models of systems with several degrees of freedom can be used to determine system natural frequencies and mode shapes. The mode shape is a description of how a system vibrates when it is operated at or near one of its natural frequencies. The BGVis-2D tool can be useful in helping students to "picture" the different mode shapes of a system and understand the actual time response better.

An example three degree of freedom (DOF) mechanical system used to illustrate how the dynamic display can show mode shapes is shown in Figure 52. In this system three masses are connected together with linear springs. Each mass is allowed to translate in the x direction. The system is connected to a base, which can be used as a

source of motion for the system, but for this example which will involve natural vibrations, the base is fixed in place. A bond graph of this system is shown in Figure 53. Each I element along with the corresponding I element represents the velocity of a mass. The C elements model the springs between each mass. All of the elements are linear, with the masses of the bodies 1, 2, and 3 equal to 1 kg, 2 kg, and 2 kg, respectively. The C element stiffnesses are 1 N/m for C1, 1 N/m for C2, and 2 N/m for C3. The bond graph is set up to conform to the geometry of the actual system, to aid in the visualization. The model file including the equations for this example can be found in Appendix C.

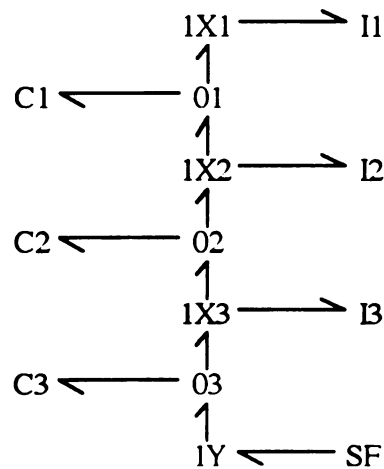


Figure 53. Bond Graph of Mechanical Three DOF System.

ENPORT was used to determine the three unique eigenvalues (natural frequencies) and eigenvectors (mode shapes) of the mechanical three DOF system. The eigenvalue and eigenvector values calculated are shown in Table 4. When an eigenvector is used as the initial condition for a simulation of the natural response, the system experiences the natural mode of vibration corresponding to that eigenvector. Simulations of the time response for the three different modes of this system were completed and the results were visualized using the BGVis-2D display environment.

Table 4. Eigenvalues/vectors of Mechanical Three DOF System.

Mode	Natural Frequency	Eigenvector Q4	Q5	Q6
1	0.437 rad/s	-0.2185	-0.5721	-0.3536
2	1.144 rad/s	0.2390	-0.4781	0.4781
3	1.414 rad/s	0.3742	-0.2312	-0.6054

Figure 54 shows the response of the flow variable type displayed at a time when the flows are near their maximum. Since the response is sinusoidal, the displacements of the masses are also sinusoidal functions with the same frequency and amplitudes all scaled by the natural frequency. The line widths on bonds 1, 2, and 3 illustrate the first mode shape for the system. During the course of the time response the overall values rise and fall, but the relationship between the displacements of masses 1, 2, and 3 stays the same. Mass 1 experiences the greatest displacement, with mass 2 experiencing a somewhat less displacement in the same direction, and mass 3 has the least displacement also in the same direction. Watching the levels of the glyph symbols allows the student to track how energy is shifted between kinetic energy (I elements) and stored energy in the springs (C elements) during the course of the animation. Figure 55 shows the second mode shape. In this mode masses 2 and 3 move together, but with the displacement of mass 2 at about half the value of mass 3, while mass 1 moves in the opposite direction with a displacement value near that of mass 3. Figure 56 shows the third mode shape. In this mode masses 1 and 3 experience equal displacements in the same direction, while the displacement of mass 2 is equal and opposite. The advantage of using the dynamic display would be even greater for systems with more degrees of freedom and for systems involving energy domains other than mechanical translation where mode shapes are not as easily visualizable (e.g. electrical).

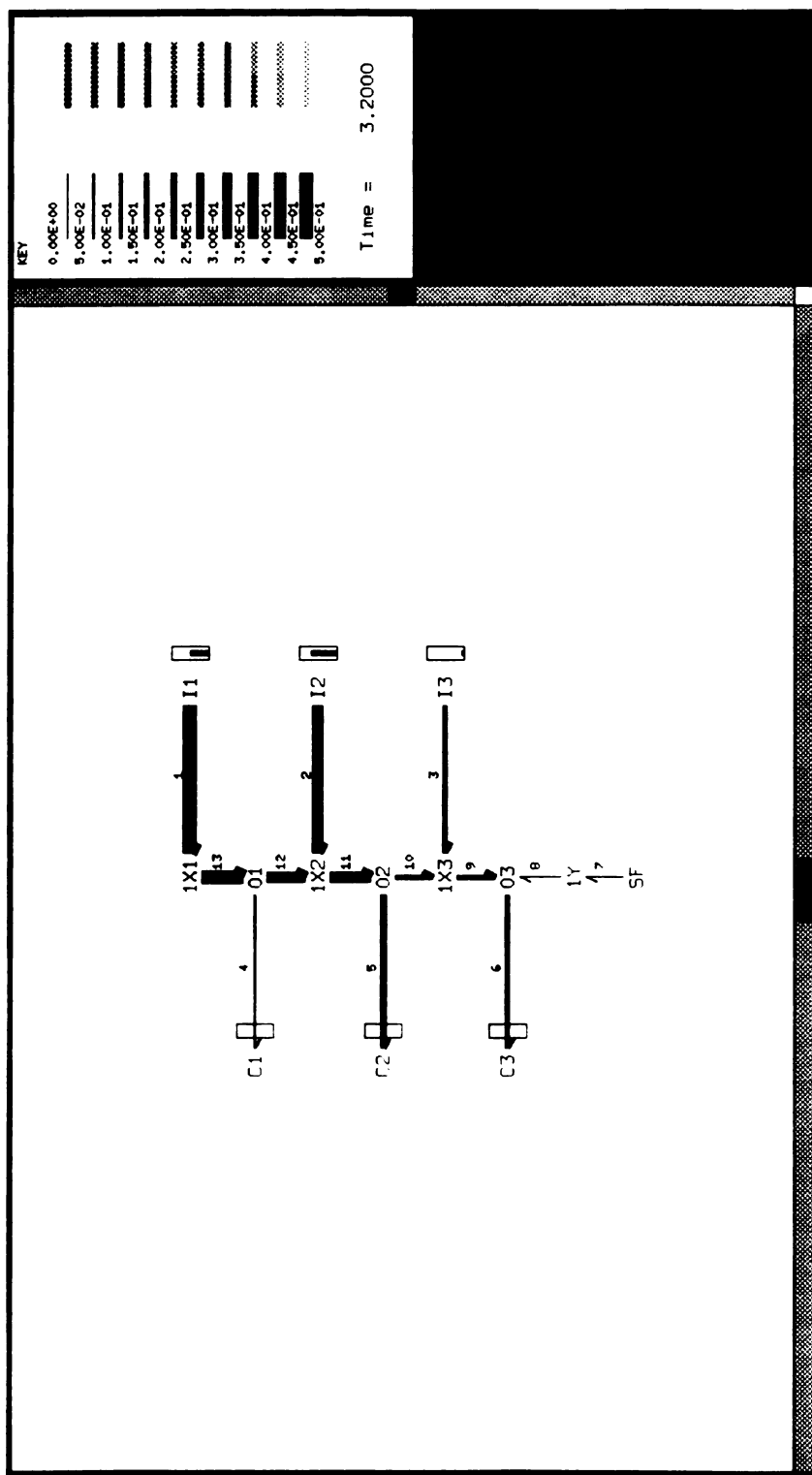
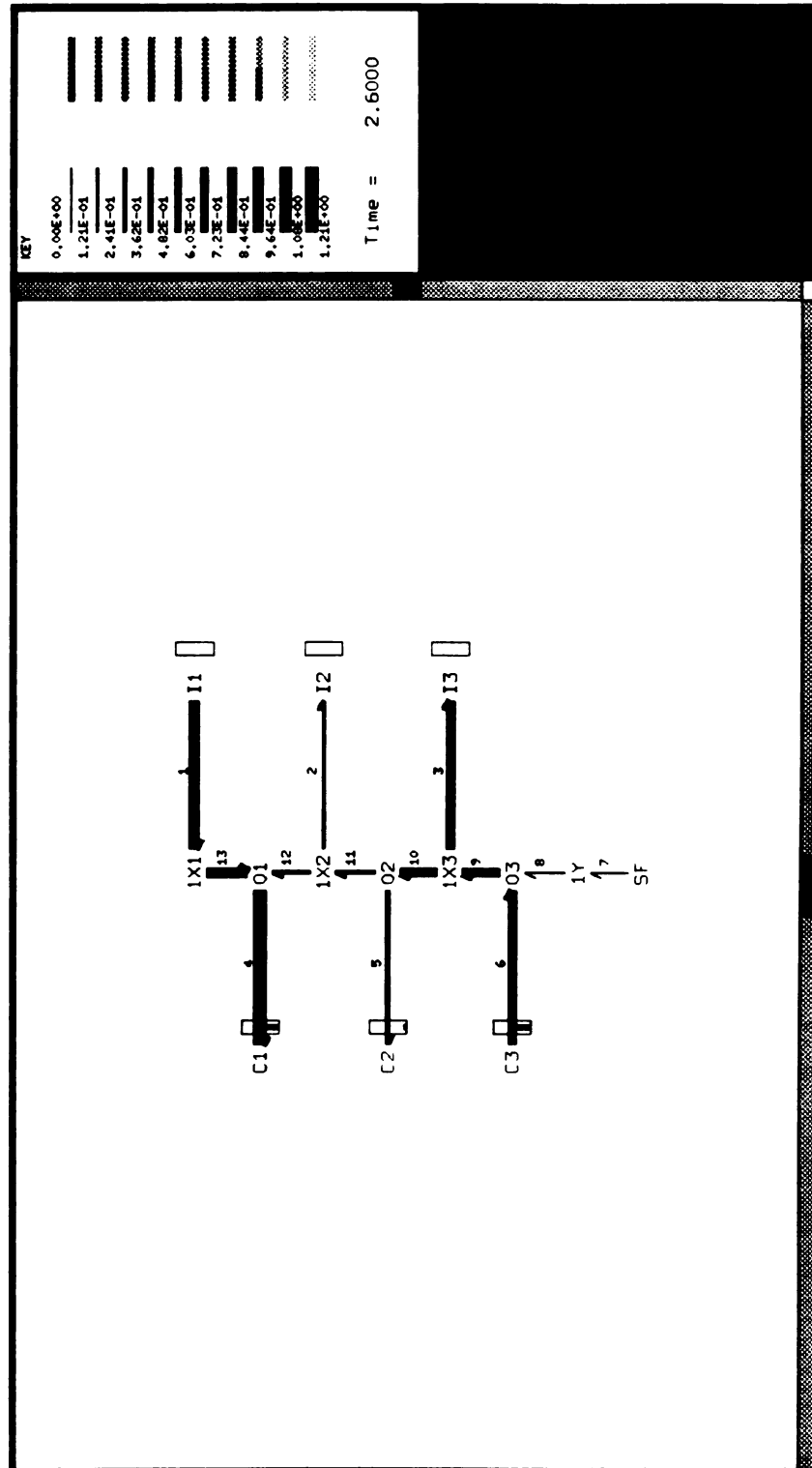


Figure 54. Flow Variable Display Showing First Mode Shape.



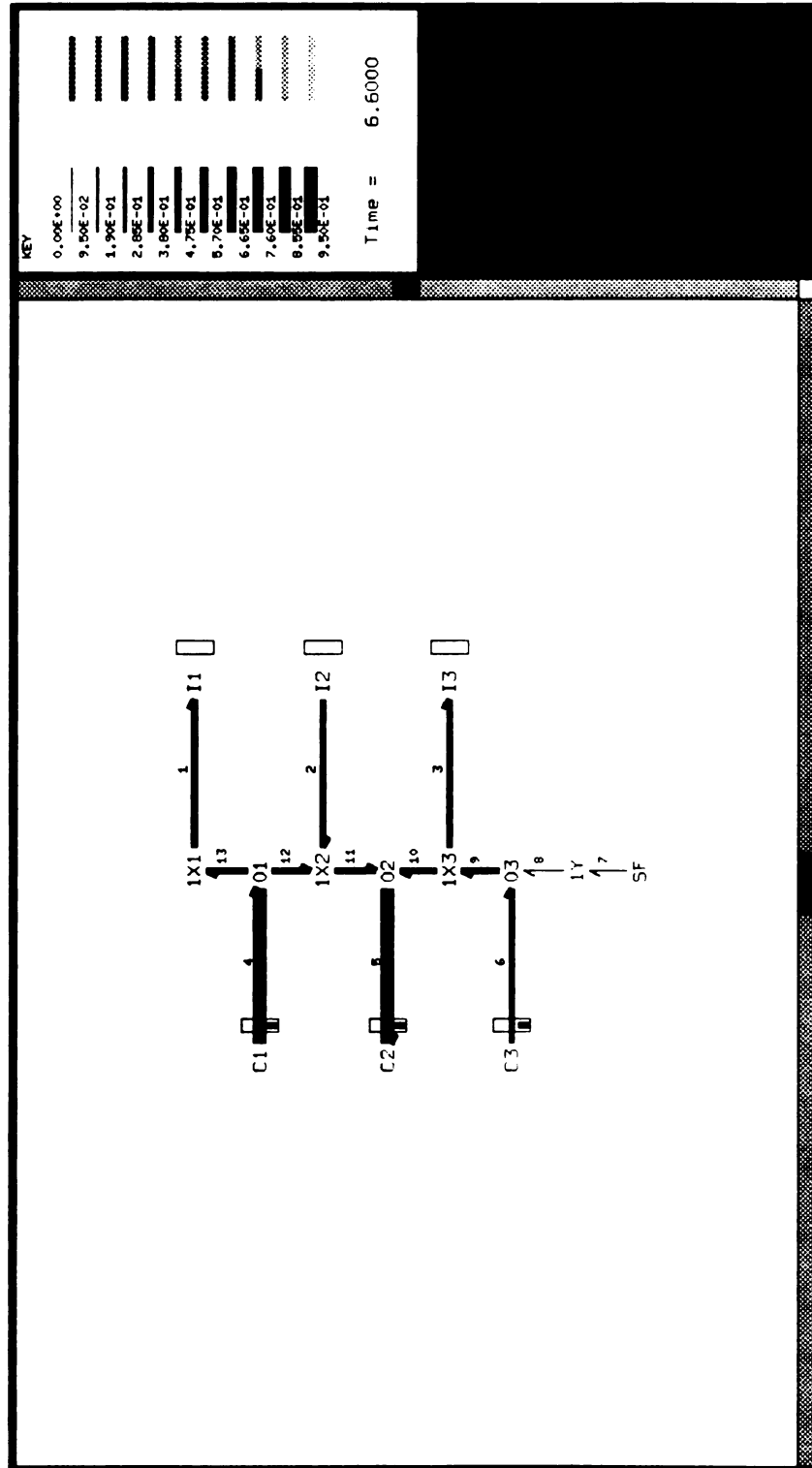


Figure 56. Displacement Variable Display Showing Third Mode Shape.

Chapter 6

Visualization Example Using BGVis-3D

The three degree-of-freedom BPR car and suspensions bond graph model representation was extended to three dimensions using the three-dimensional bond graph display environment (BGVis-3D). The 3-D bond graph representation of the BPR model is shown in Figure 57. The 3-D representation can be seen to have the same connectivity as the 2-D version, but clearly the 3-D representation corresponds more closely to the geometry of the schematic (refer to Figure 12). The suspension branches and the rotation/translation axes are much easier to associate with their corresponding locations in the schematic, so much so that it may no longer be necessary to keep the schematic nearby for constant reference.

Figures 58 through 61 show the same power response obtained from the model of chapter 5 with a pothole input visualized in the 3-D environment. Color variation is used to show the power values on the bonds. Figure 58 shows the expansion stage of the right/front branch at the beginning of the pothole. Figure 59 shows a later time stage when energy is being damped out of the front branch after the front tire has passed the pothole. Figure 60 shows the effect of the rear tire encountering the pothole and Figure 61 shows the damping out of this energy in branch 3. Combining the response data with a 3-D representation of the bond graph model lessens the amount of work required by the modeler to visualize the response with respect to the schematic and therefore the physical system itself.

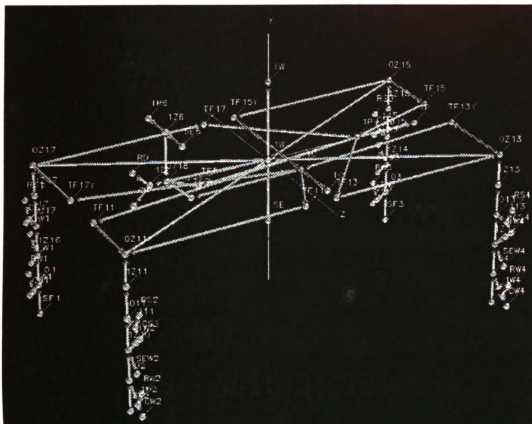


Figure 57. 3-D Bond Graph Representation of Car and Suspensions Model.

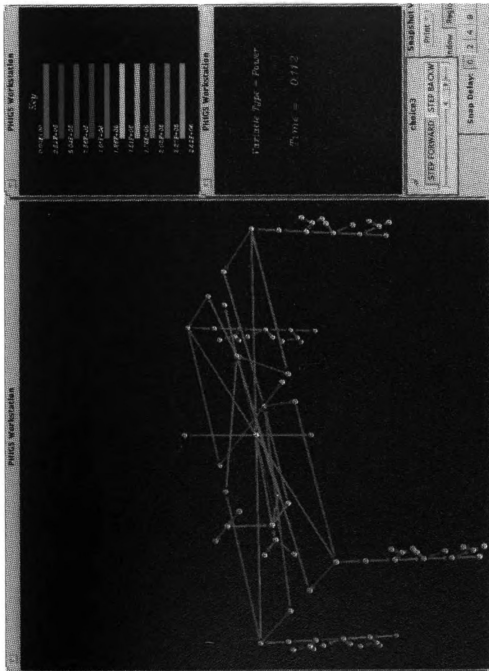


Figure 58. 3-D Dynamic Display of Response to Pothole Input (Time = 0.112).

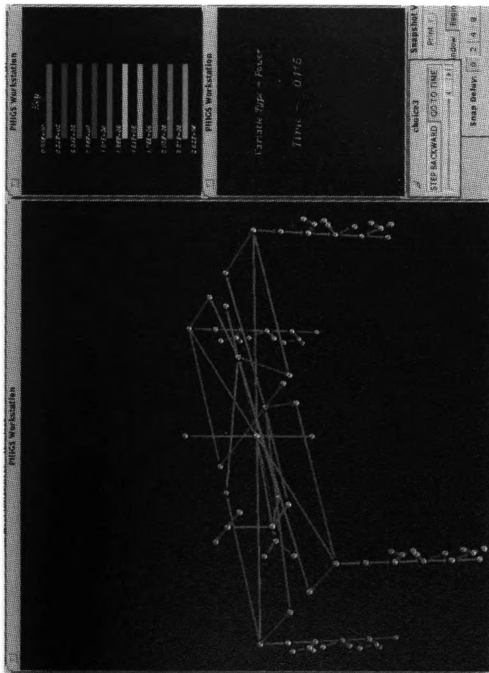


Figure 59. 3-D Dynamic Display of Response to Pothole Input (Time = 0.115).

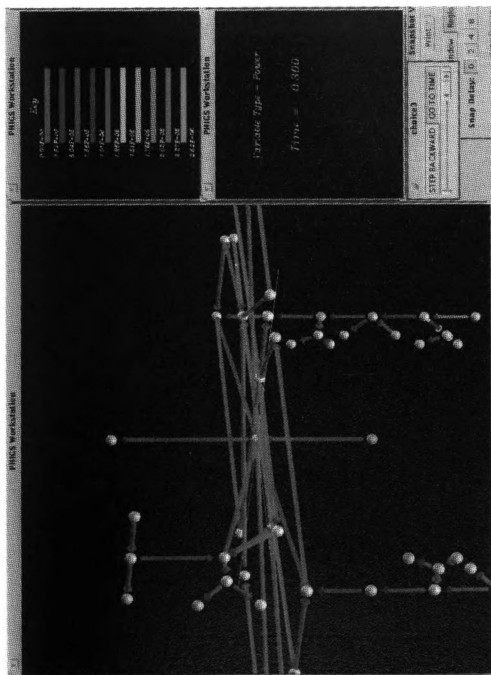


Figure 60. 3-D Dynamic Display of Response to Pothole Input (Time = 0.300).

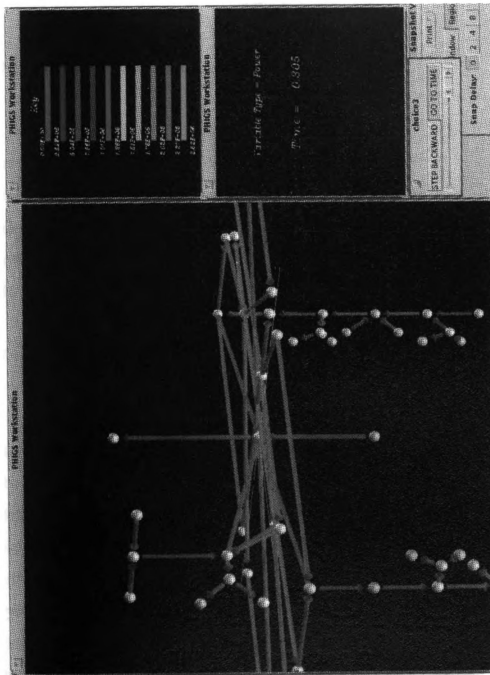


Figure 61. 3-D Dynamic Display of Response to Pothole Input (Time = 0.305).

Chapter 7

Conclusions

7.1 Summary of Contributions

The research work completed has made a significant contribution to the use of bond graphs in modeling and design of large-scale systems. The principal research objective of designing and implementing an improved design environment for engineering systems based on the use of bond graph methods as a structured modeling technique, which makes the interpretation of dynamic system response easier for designers, has been achieved.

7.1.1 Two-Dimensional Visualization

The BGVis-2D environment provides the dynamic link between model response data and bond graph structure which is necessary to take advantage of the knowledge of the system that the bond graph supplies to the engineer. This link between simulation data and bond graph model makes the visualization of response more meaningful. This is especially useful in the process of gaining insight about model behavior. The environment is also very valuable in assisting in the task of explaining complicated system behavior to those who are not familiar with an existing system model. The visualization provides a way to communicate how the system is behaving in a basically qualitative way.

The primary functions implemented in the BGVis-2D display environment were:

- 1) loading of bond graph model files into the data post-processor;
- 2) integration of effort and flow variables with respect to time to obtain the net energy variables explicitly;
- 3) three different scaling options for data;
- 4) bond color and width variation for a display of bond graph variables including power as if they were flowing on the graph;
- 5) glyph display for a level-type display of associated energy values in the energy storing and dissipating elements of the bond graph;
- 6) time controls necessary to control the display of the animation at the user's discretion.

Two practical engineering system examples, an automobile in bounce-pitch-roll mode and a railcar airbrake switching system, were developed to illustrate the use of the BGVis-2D tool.

7.1.2 Three-Dimensional Visualization

The BGVis-3D environment provides a stronger link between the bond graph and the physical system by allowing display of three-dimensional representations of bond graphs. Matching key points in the bond graph and physical schematic further enhances the link between the physical system and the model behavior. The computer graphics standard PHIGS was used to implement the functions of the graphical display. A method for translating bond graphs into three-dimensional objects using spheres for nodes and cylinders with arrows for bonds was established.

The primary functions implemented in the BGVis-3D display environment, in addition to some of those mentioned above in the 2-D environment description, were: 1) loading of 3-D bond graph model files and data files into the environment; 2) control of the viewing parameters, including changing the angle of view and the distance from which the 3-D bond graph is viewed 3) color variation to show values of bond graph

variables including power and energy on the graph. The automobile example was extended to three dimensions to show the capabilities of the BGVis-3D environment.

7.2 Areas for Further Research

Several areas have been identified as possibilities for future research. An overall goal of future research might be working toward what has been called "interactive quantitative visualization" or "computational steering" (Peskin, et al, 1991). The basic idea behind these concepts is to graphically link the model with results during computation so the designer can make decisions about the quality of the model as the computation progresses. The suggested future research areas are grouped into two categories, the first relating primarily to the BGVis-2D environment, the second to the BGVis-3D environment.

Regarding the two-dimensional bond graph visualization, the following areas warrant further attention.

1. The integration of macro-modeling capabilities and library storage of sub-components with the new display capabilities would be welcome. In this way large systems could be expressed in bond graph form without having to see all the individual details of each subcomponent. Elements in close geometric proximity could be grouped to make the power animation clearer.

2. Additional scaling options for variables could be implemented to show behavior in the clearest way possible. Scaling of variables by energy domain might be helpful to distinguish between variable levels which vary greatly over the different energy domains. Power and energy can be directly equated across the domains if

appropriate units are used, but individual variables have their own units in each domain. For example, considering the behavior of the airbrake switching system, the efforts in the hydraulic sections are numerically large, while the forces on the mechanical parts are very small in comparison. If all the efforts are scaled together, the relative values of forces on different mechanical components is obscured. If the mechanical efforts could be scaled independently of the hydraulic efforts, better relative comparisons could be made.

3. The capability for including multibonds in the graph should be added to the ENPORT modeling software. A multibond is a concise way to represent a natural grouping of single bonds (Breedveld, 1985). Although multibonds do not necessarily aid in helping bond graphs to correspond to natural geometry at a detailed level, they can sometimes be of great help in simplifying graphs at the macro level.

4. It would be desirable to show in the animation not only information related to each specific instant in time, but to include some time history information as well. In this way still prints or photographs of different time stages would show power data related to several previous and following time stages, allowing for better interpretation of dynamic response without having to view an entire animation sequence.

5. It would be useful to transfer the functions of the 2-D environment described in this thesis from the UNIX version of ENPORT to the more widely disseminated PC-DOS version. More powerful PC's such as Pentium and Power-PC based machines make the animation realizable in the PC environment and would therefore make it available to many more engineers to aid in their system design work.

6. The files used to store the simulation results data are often very large due to the fact that all efforts and flows for all bonds over all time stages is stored along with the state variables. A certain amount of this information is redundant due to bond graph constraints. For example, all of the efforts on bonds connected to the same zero-junction are constrained to be equal. It might be useful to establish a format for output data files which compresses the files by eliminating all redundant information. Reconstruction of the data could be performed in the visualization postprocessor.

In addition to the 2-D list, the following suggestions for future research apply for the three-dimensional bond graph visualization.

1. Since the goal of this research was to show the effectiveness of display of simulated behavior data on a 3-D bond graph representation, the activity of 3-D bond graph construction was ignored. In the future it would be useful to develop a 3-D model builder along the lines of the current 2-D model builder

2. To further enhance the correspondence of 3-D bond graph representations to physical systems or schematics it would be useful to develop the concept of anchor points in the bond graph. These points would be specifically intended to refer directly to specified locations in the system. The ability to place and manipulate these anchor point elements would allow the engineer to scale the model with the physical system, as in a CAD drawing environment.

3. In the interest of providing as much information to a designer as possible in a given display, it would be useful to develop an animation of the physical motions of the mechanically translating bodies in the system using bond graph element anchor points in the bond graph display environment. Multibody dynamic mechanical system modelers

like ADAMS already use this technique with rigid body models to enhance interpretation of simulation results, and adding this type of display to bond graph models would also be useful.

APPENDICES

Appendix A

A Brief Introduction to Bond Graph Modeling

A.1 Bond Graph Variables and Node Types

In Chapter 2 of this report, the purpose of bond graph modeling was stated and a general description of bond/system graphs was given. The purpose of this appendix is to provide a few more details about system graphs and the way they are constructed.

A bond graph is a graphical representation of the physical behavior of a system leading to an explicit mathematical model involving system variables. A bond represents a power transfer connection which appears at a port, or place where energy is transferred to or from a device. The structure of a bond graph takes advantage of the fact that physical variables in many engineering domains appear as related pairs whose product is power. These power variables are denoted as **effort** and **flow**. Related to the power variables, two other types of variables are identified which are of use in determining energies. These are called **momentum** and **displacement**. Depending upon which energy domain the power transfer occurs in, the generic variables effort, flow, momentum, and displacement can be assigned to different physical quantities. Table A-1 lists the variable names and units for several common engineering energy domains.

A given system can be separated into subcomponents connected at their ports to other subcomponents via bonds. This decomposition can be done at various levels, until at the most precise level, physical behaviors of a subcomponent can be described with a relatively small set of bond graph elements (node types) and their associated constitutive

equations. The set of standard bond graph modeling elements was shown in Table 2 of Chapter 2 and is repeated here as Table A-2. These elements can be used to describe behaviors found in most engineering system domains.

Bond graphs can also be augmented by the addition of block diagram elements and signals. In this way information flow as well as energy flow can be accounted for and control systems can be modeled along with the actual system under study. Signals are different from bonds, since signals by definition carry no power, only information. Signals are represented graphically by using a double arrow rather than the single arrow used on a bond. Some examples of block diagram elements are shown in Figure A-3.

Table A-1. Variables in Various Energy Domains.

Variable	General	Mechanical Translation	Mechanical Rotation	Electrical	Hydraulic
Effort	$e(t)$	F, force (Newton)	τ , torque (Newton*meter)	e, voltage (Volt)	P, pressure (Pascal)
Flow	$f(t)$	V, velocity (meter/second)	ω , angular vel. (radian/second)	i, current (Ampere)	Q, volume flow rate (meter ³ /second)
Momen- tum	$p = \int e \, dt$	P, momentum (Newton*second)	H, angular mom. (Newton*m*sec)	λ , flux link. (Weber)	P_p , integral of press. (Pascal*second)
Displace- ment	$q = \int f \, dt$	X, distance (meter)	θ , angle (radian)	q, charge (Coulomb)	V, volume (meter ³)
Power	$P(t) = e(t)f(t)$	$F(t)V(t)$ (Watt)	$\tau(t)\omega(t)$ (Watt)	$e(t)i(t)$ (Watt)	$P(t)Q(t)$ (Watt)
Energy	$E(p) = \int f \, dp$ (Joule) $E(q) = \int e \, dq$ (Joule)	$\int V \, dP$, kinetic (Joule) $\int F \, dX$, potential (Joule)	$\int \omega \, dH$, kinetic (Joule) $\int \tau \, d\theta$, potential (Joule)	$\int i \, d\lambda$, magnetic (J) $\int e \, dq$, electric (J)	$\int Q \, dP_p$, kinetic (Joule) $\int P \, dV$, potential (Joule)

Table A-2. Standard Bond Graph Elements.

Element Type	Element Name	Bond Graph Symbol	Constitutive equations	
			General Form	Linear Form
Sources	Source of Effort	SE $\begin{array}{c} \xrightarrow{E} \\ \xleftarrow{F} \end{array}$	$E = f(\text{time})$	$E = \text{Constant}$
	Source of Flow	SF $\begin{array}{c} \xrightarrow{E} \\ \xleftarrow{F} \end{array}$	$F = f(\text{time})$	$F = \text{Constant}$
Passive One-Ports	Resistance	R $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$E = f(F)$	$E = R * F$
	Capacitance	C $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$E = f(Q)$ $F = dQ/dt$	$E = 1/C * Q$ $F = dQ/dt$
	Inertia	I $\begin{array}{c} \xleftarrow{E} \\ \xrightarrow{F} \end{array}$	$V = f(P)$ $E = dP/dt$	$V = 1/L * P$ $E = dP/dt$
Two-port transducers	Transformer	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} \text{TF} \begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = n * E2$ $F1 = 1/n * F2$	
	Gyrator	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} \text{GY} \begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = r * F2$ $F1 = 1/r * E2$	
Junction elements	One-Junction	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} 1 \begin{array}{c} \xrightarrow{E3} \\ \xleftarrow{F3} \end{array}$ $\begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$F1 = F1 = F3$ $E1 - E2 - E3 = 0$	
	Zero-Junction	$\begin{array}{c} \xrightarrow{E1} \\ \xleftarrow{F1} \end{array} 0 \begin{array}{c} \xrightarrow{E3} \\ \xleftarrow{F3} \end{array}$ $\begin{array}{c} \xrightarrow{E2} \\ \xleftarrow{F2} \end{array}$	$E1 = E2 = E3$ $F1 - F2 - F3 = 0$	

Table A-3. Block Diagram Elements.

Element Name	Symbol	Constitutive equation
Signal Source	$\boxed{\text{SRC}} \xrightarrow{Y}$	$Y = f(\text{time})$
Signal Sink	$\xrightarrow{U} \boxed{\text{SINK}}$	No output equation
Gain	$\xrightarrow{U} \boxed{K} \xrightarrow{Y}$	$Y = K * U$
Integrator	$\xrightarrow{U} \boxed{\text{INT}} \xrightarrow{Y}$	$Y = \int U dt$
Function	$\xrightarrow{U} \boxed{\text{FCN}} \xrightarrow{Y}$	$Y = f(U)$
Transfer Function	$\xrightarrow{U} \boxed{T(s)} \xrightarrow{Y}$	$Y = T(s) * U$
Summer	$\begin{array}{c} \xrightarrow{U1} \\ \xrightarrow{U2} \\ \xrightarrow{U3} \end{array} \boxed{\text{SUM}} \xrightarrow{Y}$	$Y = U1 + U2 + U3$
Distributor	$\xrightarrow{U} \boxed{\text{DIST}} \begin{array}{c} \xrightarrow{Y1} \\ \xrightarrow{Y2} \\ \xrightarrow{Y3} \end{array}$	$Y1 = U, Y2 = U, Y3 = U$

A.2 Standard Procedures for Modeling

Because bond graphs have been developed with the goal of allowing structured hierarchical decomposition of a system, straightforward methods can be used to translate schematic diagrams into bond graph models. The methods differ depending on the energy domain being modeled, and several of these methods for handling common domains will be detailed in the following sections, along with simple examples. The general steps for obtaining a bond graph from a schematic are: 1) establish a coordinate system and unique system variables for the system to be modeled; 2) follow the appropriate procedure to obtain bond graph model with appropriate signs on bonds; and 3) define causality to establish input/output relationships within model in order to prepare for algorithmic equation derivation. This section outlines the establishment of coordinate systems and modeling procedures for mechanical, electrical, and hydraulic systems. Causality considerations will be discussed in a later section.

Mechanical Translation and Rotation. The procedure for obtaining a bond graph model of a mechanical translational system consists of the following steps. First, identify all unique velocities in the system (flows) and choose the positive directions for each velocity and its associated force (effort). The positive power direction occurs when both of the previous quantities are positive. Next, establish a 1-junction for each unique velocity. Attach necessary inertia elements to their appropriate 1-junctions, along with sources of force or velocity. Finally, connect in compliance and damping effects with 0-junctions in between the appropriate 1-junctions.

Figure A-1 shows a simple mechanical system schematic and its bond graph representation. The 1-junction indicates that a common velocity (V) exists for several different effects, and that the forces generated by these effects on the mass must add up to

zero. The effects suggested by the schematic are the input force on the body (Se), the compliance effect of the spring (C), the resistance effect of the damper (R), and the force due to the inertia of the body (I). Mechanical rotational systems follow the same procedure, but linear velocities are replaced by angular velocities (flows) and forces are replaced by torques (efforts).

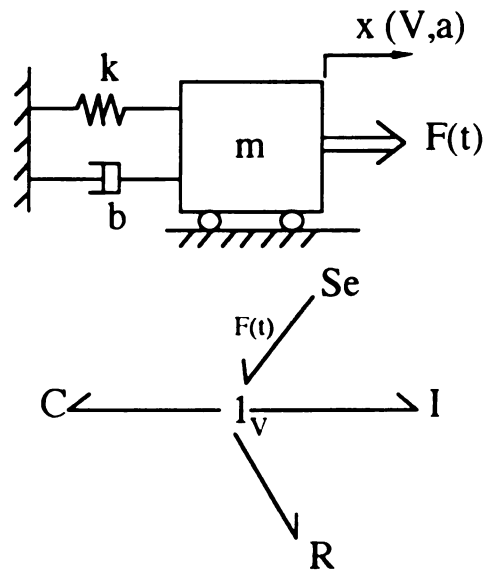


Figure A-1. Mechanical Translation Example.

Electrical Networks. There are two different ways to approach electrical model building. The first is to use 1-junctions (common current) for electrical elements in series and 0-junctions (common voltage) for electrical elements in parallel, and construct the entire model by observation. Another procedure, called the circuit-construction method, is more straightforward. The procedure starts with establishing positive directions for currents in the circuit. Next, all unique voltages are identified, and a 0-junction is established for each unique voltage. All other circuit elements are connected into this structure with 1-junctions between appropriate voltages. It is finally necessary to choose

one voltage as ground, eliminate that 0-junction, and simplify the graph as much as possible.

A simple circuit example is shown in Figure A-2. The 1-junction represents the common current (i) in the circuit, with all the voltage drops around the loop adding to zero. The effects indicated in the schematic are the voltage source (Se), the electrical resistor (R), the electrical capacitor (C), and the electrical inductance (I).

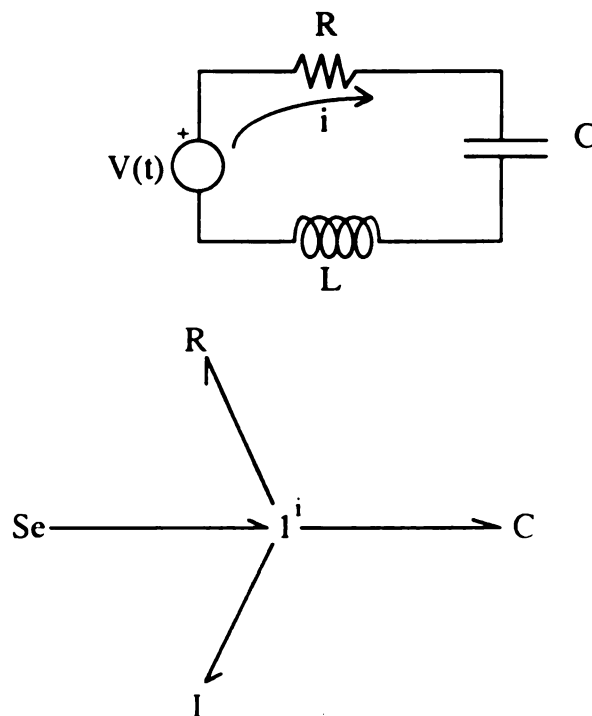


Figure A-2. Electrical Example.

Hydraulic Circuits. Bond graph models can also be used to represent hydraulic/pneumatic circuits. The procedure used is similar to that for electrical circuits, but currents are replaced by volume flow rates (flows) and voltages are replaced by

pressures (efforts). First, the positive directions for volume flow rates (Q 's) should be established. Next, 0-junctions should be used to represent each unique pressure in the system. Storage (C) elements can be placed on their appropriate 0-junctions, and all other hydraulic elements can be added in with 1-junctions connected between the appropriate 0-junctions.

Figure A-3 shows a hydraulic pump system example. The pump itself transforms mechanical power (t, w) to hydraulic power (P, Q). There are two unique pressures in the system: the inlet pressure (0_a), and the outlet pressure (0_b). The 1-junction determines the difference between the two pressures and relates that value to the mechanical input through the transformer (TF).

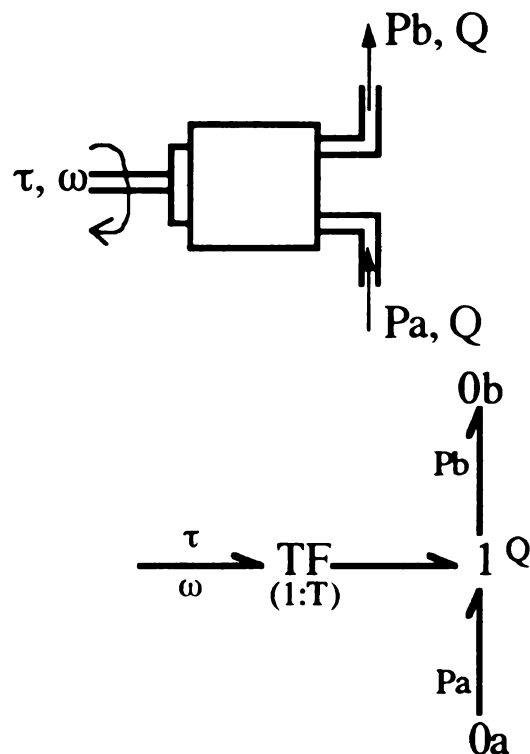
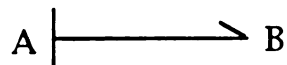


Figure A-3. Hydraulic Circuit Example.

A.3 Causality Assignment

Once a system bond graph has been constructed, it is necessary to establish a system of input and output relationships in order to make the composition of the constitutive equations of the elements into a set of system differential equations more direct. Applying causality provides a structured way to develop system differential equations. A causal stroke on each bond is used to indicate in which direction the flow is the input, and therefore in which direction the effort is an output. With the causal stroke placed on the left for the following elements



it can be inferred that the flow variable is the input to element B, and conversely, the effort is the output from element B. From the point of view of element A, effort is the input, and flow is the output. Different element types have different restrictions placed upon where the causal strokes can be placed. Table A-4 lists the available causalities for each element type.

Table A-4. Causality Possibilities for Bond Graph Elements.

Source Elements	Se ———	Sf ———
Passive Elements	I ———	Integral causality
	C ———	Integral causality
	R ———	R ———
Two-port Elements	——— TF ———	——— TF ———
	——— GY ———	——— GY ———
Junction Elements		

A.4 Multi-Domain Example

In order to show the methodology of bond graph modeling for multi-domain systems, an example is presented here. Each part of the system in a given domain is modeled using the previously mentioned methods. The connections between domains are provided by transformers or modulated source elements. The example system is the Radar Pedestal Position Control System shown in Figure A-4. The system consists of a radar pedestal which supports a radar dish which is allowed to rotate about the z-axis. In order for the radar to track objects accurately, it is necessary to carefully control the position of the pedestal by means of a motor supplying a torque and angular velocity through a shaft and gearing system. A partial decomposition of the system into subcomponents is shown in Figure A-5.

A complete system graph is shown in Figure A-6. In the complete system graph element names ending in E are part of the electrical circuit driving the motor. Specifically, IE and RE are the inductance and resistance of the motor circuit respectively. Element names ending in M are part of the mechanical part of the system, including a shaft with compliance and gearing system for speed reduction. Specifically, IM and RM are the inertia and damping in the motor, CS is the shaft compliance, TF represents the speed reducing gears, IP is the load inertia, and RP is the bearing friction of the load. The block diagram elements provide the control action to be applied to the circuit controlling the motor speed. The variable w is the angular speed of the pedestal, and θ is the position of the pedestal, calculated by integrating the speed signal. Causality is also shown on the diagram. The system's equations have been derived using the computer program ENPORT, and the response of this system to a step increase is shown in Figure A-7.

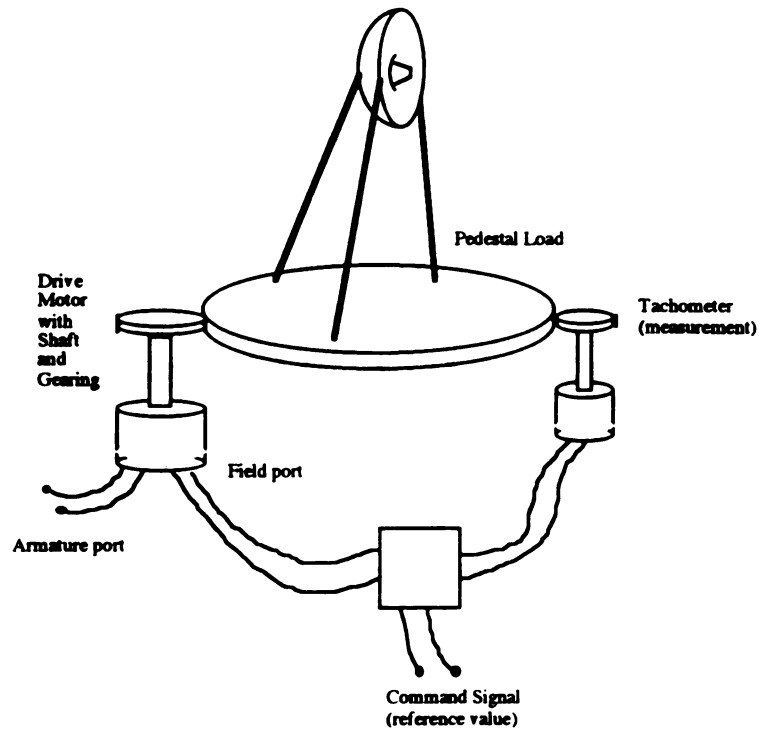


Figure A-4. Schematic of the Radar Pedestal Control System.

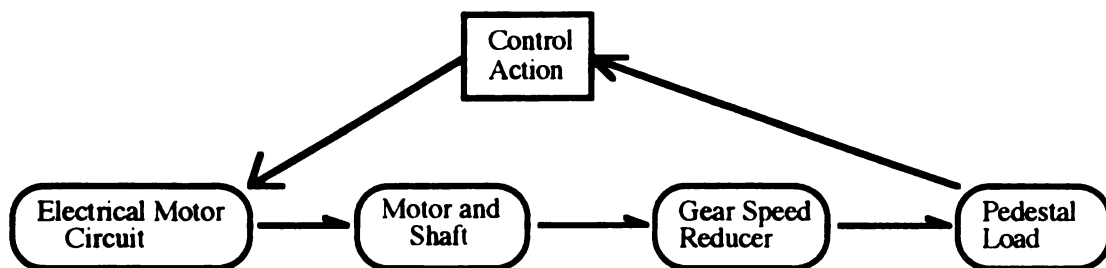


Figure A-5. Section (macro) bond graph as illustration of hierarchical decomposition.

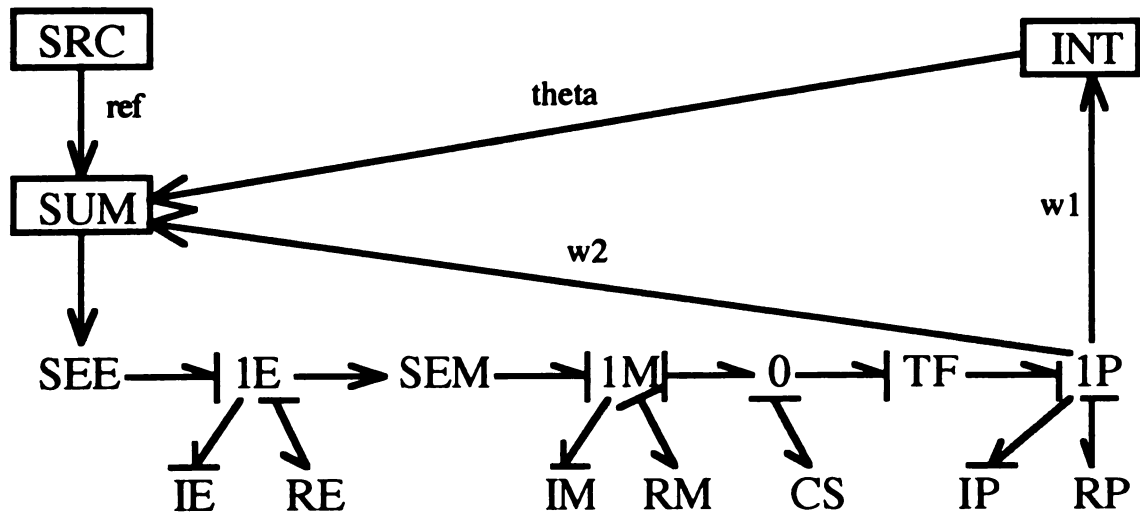


Figure A-6. Bond Graph of the Radar Pedestal System.

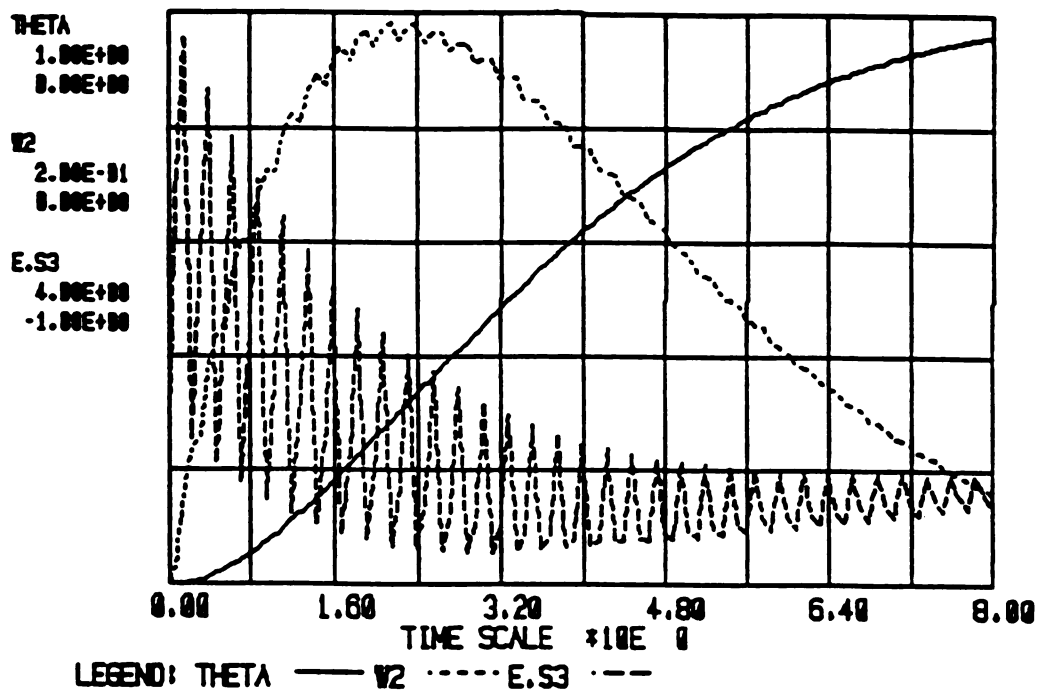


Figure A-7. Response to a Step Input.

Appendix B

Program Code

B.1 Two-Dimensional Environment

The software developed for this application is stored in a file called **dyndis.c**. It is necessary to compile this along with the ENPORT library routines stored in **libenp.a** and the file **mainpp.c** which contains the main program. What follows is a listing of the C language code used to implement the 2-D visualization environment (BGVis-2D).

```

dyndis.c          Tue Nov 8 08:46:57 1994          1
/* FILE dyndis.c -----
 *
 * Description of file contents
 *
 * This file contains all of the functions necessary for setting up
 * and subsequently using the post processor dynamic display dialog.
 *
 * Programming history (most recent change to least)
 *
 * 93 July through 94 Aug Original file written by Gayle E. Ermer
 *
 * .....
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <command.h>
#include "postproc.h"
#include "version.h"
#include "rv.h"
#include "mhzsol.h"
#include "drawarea.h"
#include "graph.h"
#include "canvase.h"
#include "database.h"
#include "widget.h"
#include "mode.h"
#include "build.h"
#include "arraydia.h"
#include "ynwidget.h"
#include "listselw.h"
#include "widgettr.h"
#include "solver.h"
#include "paramtr.h"
#include "qcurve1.h"

#define BORDER 3
#define ANNOT_LEN 10
#define CAUSE_LEN 6
#define RAD TO DEG(a) ((a)*180.0/M_PI)
#define SPREAD (M_PI/6.0)
#define ZOOM_FPERCENT 10
#define ZOOM_IN (ZOOM_FPERCENT)
#define ZOOM_OUT (-ZOOM_FPERCENT)
#define ERROR 0
#define GRAY 1
#define flip_across y(a) ((a) >= 0.0 ? M_PI - (a) : - M_PI - (a))
#define flip_across x(a) (-(a))
#define ItmAutoscale "autoscale"

enum VarTypeNames { POWER, EFFORT, FLOW, DISPLACEMENT, MOMENTUM, ENERGY,
                    SIGNAL, NONE };

enum ScaleNames {LIN, LOG, ABSELOG};

enum GlyphNames {NO, YES};

enum DisplayNames {DPILOT_CANCEL, DPILOT_REFRESH, DPILOT_NEW_BUTTONS};

enum { PS_PAUSE, PS_COMP, PS_STEPP, PS_STEPP, PS_CANCEL,
       PS_NEW_BUTTONS };

/* External Variables ****/

extern ModeType *first Mode;
extern ConnectorType *first Connector;
extern int ModDefont, ConDefont;
extern OC go[];
extern XFontStruct *font[];

```

```

extern int num stages;
extern WidgetClass graphWidgetClass;
extern int grid snap;
extern int grid visible, grid size;
extern int ModDefont, ConDefont;
extern int grid visible, grid size;
extern int show causality;
extern real spinDialogInitTimeVal;
extern real spinDialogFinalTimeVal;
extern double zoom factor, scale factor;
extern real spinAxisName[MAX_SELECTED];
extern real spinAxisName2[MAX_SELECTED];
extern real spinAxisName3[MAX_SELECTED];
extern int spinAxisName4[MAX_SELECTED];
extern int spinAxisName5[MAX_SELECTED];
extern int spinAxisName6[MAX_SELECTED];
extern int our_scale;
extern int spinDialogAfter;

/* Function Prototypes ****/

void Postproc Make Button Cancel();
void Postproc Popup Ddis();
void Postproc Popdown Ddis Dialog();
void Postproc draw();
void Postproc Load file();
void Postproc Load it();
Widget Postproc Init Graph();
Widget Postproc Init Graph Key();
Widget Postproc Init Canvas();
Widget Postproc Init Canvas Key();
void Postproc frame();
void Postproc center scrollbar();
unsigned Postproc width();
unsigned Postproc height();
Dimension Postproc width();
Dimension Postproc height();
void Postproc clear canvas();
void Postproc draw graph();
void Postproc expose();
void Postproc graph();
void Postproc find graph extent();
void Postproc find node();
void Postproc node coords();
void Postproc to screen();
void Postproc scale();
void Postproc zoom();
void Postproc place();
int Postproc in region();
void Postproc connector();
void Postproc arrow head();
void Postproc causality();
void Postproc label coords();
void Postproc label line();
void Postproc snap to grid();
void Postproc snap all points to grid();
void Postproc get();
void Postproc det Line Value();
void Postproc Make Ddis Buttons();
void Postproc Make Ddis Buttons2();
void Postproc Make Scale Buttons();
void Postproc Make Glyph Buttons();
void Postproc Change Var();
void Postproc Change Scale();

```



```

static XTranslations new_actions[] = {
    ("FVType Pop", FVType Pop),
    ("FVType Pop2", FVType Pop2),
    ("SCType Pop", SCType Pop),
    ("CLType Pop", CLType Pop),
};

XTranslations changed_translations;
static char *translations = "Goverride\n\
    <AltDown>: highlight() FVType Pop() MenuPopup(Variable);\n\
static char *translations = \"Goverride\n\
    <AltDown>: highlight() FVType Pop2() MenuPopup(Variable2);\n\
static char *translations = \"Goverride\n\
    <AltDown>: highlight() SCType Pop() MenuPopup(Scaling);\n\
static char *translations = \"Goverride\n\
    <AltDown>: highlight() CLType Pop() MenuPopup(Glyph);\n\
static char translations2[] = \"Goverride\n\
    Ctrl-Key-J: end-of-line()\n\
    Ctrl-Key-M: end-of-line()\n\
    Ctrl-Key-N: end-of-line()\n\
    Ctrl-Key-O: beginning-of-line()\n\
    Ctrl-Key-P: beginning-of-line()\n\
    Ctrl-Key-Q: beginning-of-line()\n\
    Meta-Key-I: beginning-of-line()\n\
    Meta-Key-J: end-of-line()\n\
    Meta-Key-K: end-of-line()\n\
    Meta-Key-L: beginning-of-line()\n\
    <Key>Down: end-of-line()\n\
    <Key>Up: beginning-of-line()\n\
    <Key>Return: end-of-line()\n\
XTranslations changed_bindings;

XtTranslations(new_actions, XtNumber(new_actions));

/* Create the popupshell */
ppddisDialogPopupsShell =
    XtCreateManagedWidget("Dyndis", transientShellWidgetClass,
        parent, NULL, 0);

i = 0;
XtSetArg(args[i], XtMLabel, "Dyndis"); ++i;
XtSetArg(args[i], XtBorderWidth, 0); ++i;
XtSetValues(ppddisDialogPopupsShell, args, i);

/* Create the form to hold the btnx and graphspace */
ppddisDialogForm =
    XtCreateManagedWidget("ppddisDialogForm", formWidgetClass,
        ppddisDialogPopupsShell, NULL, 0);

i = 0;
XtSetArg(args[i], XtBorderWidth, 0); ++i;
XtSetArg(args[i], XtMLabel, "Dyndis"); ++i;
XtSetValues(ppddisDialogForm, args, i);

/* Create the btnx to hold the command buttons */
ppddisBtnx = XtCreateManagedWidget("ppddisBtnx", formWidgetClass,
        ppddisDialogForm, NULL, 0);

i = 0;
XtSetArg(args[i], XtMLabel, "Dyndis"); ++i;
XtSetArg(args[i], XtMLabel, "Dyndis"); ++i;
XtSetValues(ppddisBtnx, args, i);

```



```

XtSetValues(ppDialogButton[DDIS DRAW], args, 1);
XtAddCallback(ppDialogButton[DDIS DRAW], XtCallback, NULL);

/* go to time button */
ppDialogButton[DDIS OTIME] =
    XtCreateManagedWidget("ppDialogButton[DDIS OTIME]",
        commandWidgetClass, ppDialogButtonBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Go To Time: "); i++;
XtSetArg(args[i], XtNfromField, NULL); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS DRAW]); i++;
XtSetValues(ppDialogButton[DDIS OTIME], args, 1);
XtAddCallback(ppDialogButton[DDIS OTIME], XtCallback,
    Postproc GoTo Time, NULL);

/* stop forward button */
ppDialogButton[DDIS STEPP] =
    XtCreateManagedWidget("ppDialogButton[DDIS DRAW]",
        commandWidgetClass, ppDialogButtonBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Stop Forward"); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS OTIME]); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS STEPP]); i++;
XtSetValues(ppDialogButton[DDIS STEPP], args, 1);
XtAddCallback(ppDialogButton[DDIS STEPP], XtCallback,
    Postproc Stop Forward, NULL);

/* stop back button */
ppDialogButton[DDIS STEPB] =
    XtCreateManagedWidget("ppDialogButton[DDIS STEPB]",
        commandWidgetClass, ppDialogButtonBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Stop Backward"); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS OTIME]); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS STEPB]); i++;
XtSetValues(ppDialogButton[DDIS STEPB], args, 1);
XtAddCallback(ppDialogButton[DDIS STEPB], XtCallback,
    Postproc Stop Backward, NULL);

/* Create an exit button */
ppDialogButton[DDIS CANCEL] =
    XtCreateManagedWidget("ppDialogButton[DDIS CANCEL]",
        commandWidgetClass, ppDialogBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Exit "); i++;
XtSetArg(args[i], XtNfromField, NULL); i++;
XtSetArg(args[i], XtNfromField, ppDialogButton[DDIS LOAD]); i++;
XtSetValues(ppDialogButton[DDIS CANCEL], args, 1);
XtAddCallback(ppDialogButton[DDIS CANCEL], XtCallback,
    Postproc Dis Button Cancel, NULL);

/* Initialize the main viewing widgets */
ppDialogView = Postproc Init Graph(ppDialogForm);
XtSetValues(ppDialogView, XtNfromField, ppDialogBox, NULL);
XtSetValues(ppDialogView, XtNwidth, 900, NULL);
XtSetValues(ppDialogView, XtNheight, 600, NULL);

```

```

/* Create a zoom button */
ppDialogButton[DDIS ZOOM] =
    XtCreateManagedWidget("ppDialogButton[DDIS ZOOM]",
        commandWidgetClass, ppDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Zoom "); i++;
XtSetArg(args[i], XtNfromField, NULL); i++;
XtSetArg(args[i], XtNfromField, ppDialogView); i++;
XtSetValues(ppDialogButton[DDIS ZOOM], args, 1);
XtAddCallback(ppDialogButton[DDIS ZOOM], XtCallback,
    Zoom Window, NULL);

/* Initialize the key widgets */
ppDialogKey = Postproc Init Graph Key(ppDialogForm);
XtSetValues(ppDialogKey, XtNfromField, ppDialogBox, NULL);
XtSetValues(ppDialogKey, XtNfromField, ppDialogView);
XtSetValues(ppDialogKey, XtNwidth, 200, NULL);
XtSetValues(ppDialogKey, XtNheight, 300, NULL);

/* Create a scaling choice option */
ppScalingChoice = XtCreateManagedWidget("ppScalingChoice", labelWidgetClass,
    ppDialogBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Scaling Type:"); i++;
XtSetArg(args[i], XtNorderWidth, 0); i++;
XtSetArg(args[i], XtNfromField, ppDialogLabel); i++;
XtSetArg(args[i], XtNfromField, ppDialogLabel); i++;
XtSetValues(ppScalingChoice, args, 1);

ppScalingChoiceVal = XtCreateManagedWidget("ppower", commandWidgetClass,
    ppDialogBox, NULL, 0);

changedTranslations = XtParseTranslationTable(translations);
XtOverrideTranslations(ppScalingChoiceVal, changedTranslations);
i = 0;
XtSetArg(args[i], XtNlabel, "Linear "); i++;
XtSetArg(args[i], XtNorderWidth, 0); i++;
XtSetArg(args[i], XtNfromField, ppScalingChoice); i++;
XtSetArg(args[i], XtNfromField, ppDialogLabel); i++;
XtSetArg(args[i], XtNresizeable, TRUE); i++;
XtSetValues(ppScalingChoiceVal, args, 1);

Postproc Make_Scale_Butons();

/* Create the glyph choice option */
ppGlyphChoice = XtCreateManagedWidget("ppGlyphChoice", labelWidgetClass,
    ppDialogBox, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Draw Glyphs?"); i++;
XtSetArg(args[i], XtNorderWidth, 0); i++;
XtSetArg(args[i], XtNfromField, ppScalingChoiceVal); i++;
XtSetArg(args[i], XtNfromField, ppDialogLabel); i++;
XtSetValues(ppGlyphChoice, args, 1);

ppGlyphChoiceVal = XtCreateManagedWidget("ppower", commandWidgetClass,
    ppDialogBox, NULL, 0);

```

```

changed translations = XParseTranslationTable(translations);
XtOverridetranslations(ppGlyphChoiceVal, changed translations);

i = 0;
XtSetArg(args[i], XtN"label", "Mo "); i++;
XtSetArg(args[i], XtN"orderWidth", 0); i++;
XtSetArg(args[i], XtN"orderForis", ppGlyphChoice); i++;
XtSetArg(args[i], XtN"rowVert", ppScalingChoiceVal); i++;
XtSetArg(args[i], XtN"resizable", TRUE); i++;
XtSetValues(ppGlyphChoiceVal, args, 1);

Postproc Make Glyph Buttons();

/* Create the variable choice menu for width */
ppChoiceVarWidth = XCreateManagedWidget ("ChoiceVarWidth", labelWidgetClass,
ppDialogCbx, NULL, 0);

i = 0;
XtSetArg(args[i], XtN"label", "Var for width"); i++;
XtSetArg(args[i], XtN"orderWidth", 0); i++;
XtSetArg(args[i], XtN"orderForis", ppGlyphChoiceVal); i++;
XtSetArg(args[i], XtN"rowVert", ppDialogButton[DDIS_LOAD]); i++;
XtSetValues(ppChoiceVarWidth, args, 1);

ppVarTypeWidth = XCreateManagedWidget ("Mpower", commandWidgetClass,
ppDialogCbx, NULL, 0);

changed translations = XParseTranslationTable(translations);
XtOverridetranslations(ppVarTypeWidth, changed translations);

i = 0;
XtSetArg(args[i], XtN"label", "None "); i++;
XtSetArg(args[i], XtN"orderWidth", 0); i++;
XtSetArg(args[i], XtN"orderForis", ppChoiceVarWidth); i++;
XtSetArg(args[i], XtN"rowVert", ppGlyphChoiceVal); i++;
XtSetArg(args[i], XtN"resizable", TRUE); i++;
XtSetValues(ppVarTypeWidth, args, 1);

Postproc Make Ddis Buttons();

/* Create the variable choice menu for color */
ppChoiceVarColor = XCreateManagedWidget ("ChoiceVarColor", labelWidgetClass,
ppDialogCbx, NULL, 0);

i = 0;
XtSetArg(args[i], XtN"label", "Var for color"); i++;
XtSetArg(args[i], XtN"orderWidth", 0); i++;
XtSetArg(args[i], XtN"rowVert", ppVarTypeWidth); i++;
XtSetArg(args[i], XtN"orderForis", ppDialogButton[DDIS_LOAD]); i++;
XtSetValues(ppChoiceVarColor, args, 1);

ppVarTypeColor = XCreateManagedWidget ("Mpower", commandWidgetClass,
ppDialogCbx, NULL, 0);

changed translations = XParseTranslationTable(translations);
XtOverridetranslations(ppVarTypeColor, changed translations);

i = 0;
XtSetArg(args[i], XtN"label", "None "); i++;
XtSetArg(args[i], XtN"orderWidth", 0); i++;
XtSetArg(args[i], XtN"orderForis", ppChoiceVarColor); i++;
XtSetArg(args[i], XtN"rowVert", ppVarTypeWidth); i++;
XtSetArg(args[i], XtN"resizable", TRUE); i++;
XtSetValues(ppVarTypeColor, args, 1);

Postproc Make Ddis Buttons2();

/* Initialise the popup for initial glyph values */
Init Glyph ICs(topLevel);

/* Allocate Colors and QC's */
Postproc Set Colors();
Postproc Set QC's();

Postproc Set QC's Key();

/* Initialise the zoom and plot popups */
PMake Zoom Window();

Postproc Init Plot Select();
Postproc Init Ddis Plot Dialog(parent);

/* FUNCTION Postproc Make Glyph Buttons()
* Purpose
*   Initialise the glyph choice menu
*   (called from Postproc Init Ddis Dialog)
* Inputs
*   None
* Outputs
*   None
* Programming History
*   1994 Jan, Gayle E. Ermer, Original function written
*/

void Postproc Make Glyph Buttons()
{
    int i;
    Arg args[10];
    Widget button[2];
    extern Widget ppChoicePopupShell, topLevel;
    extern Widget ppChoiceCbx;
    extern int Glyph Var;

    static char *translations = "Override\n\
-Study: MenuPopdown(Glyph)";
    static char *trans2 = "Override\n\
-Study: set() notify() unset()";
    XTranslations changed translations;

    /* Create the Popup Shell */
    ppChoicePopupShell = XCreatePopupShell("Glyph",
transientShellWidgetClass,
topLevel, NULL, 0);

    changed translations = XParseTranslationTable(translations);
    XOverrideTranslations(ppChoicePopupShell, changed translations);

    /* Create the menu box */
}

```



```

XtSetArg(args[1], XtNfromOrig, NULL); i++;
XtSetArg(args[1], XtNlabel, "Absolute Log"); i++;
XtSetArg(args[1], XtNresizable, FALSE); i++;
XtSetValues(button[2], args, i);
XtAddCallback(button[2], XtMCallback, Change Scale, (int *) ABSLOG);

changed translations = XtParseTranslationTable(trans2);
for (i = 0; i < 3; i++)
    XtOverrideTranslations(button[i], changed translations);

return;
}

/* FUNCTION Postproc Make Dds Buttons()
 * Purpose
 *   Initialise the width variable choice menu
 *   (called from Postproc Init_Dds_Dialog)
 * Inputs
 *   None
 * Outputs
 *   None
 * Programming History
 *   1993 Sept, Gayle E. Ermer, Original function written
 */

void Postproc Make Dds Buttons()
{
    int i;
    Arg args[10];
    Widget button[6];
    extern Widget ppChoicePopupShell, toplevel,
    extern int Cur_Var;

    static char *translations = "Override\n\
    <BtnUp>; MenuPopdown(Variable)";
    static char *trans2 = "Override\n\
    <BtnUp>; set() notify() unset()";
    XtTranslations changed translations;

    /* Create the Popup Shell */
    ppChoicePopupShell = XtCreatePopupShell("Variable", transientShellWidgetClass,
    toplevel, NULL, 0);

    changed translations = XtParseTranslationTable(translations);
    XtOverrideTranslations(ppChoicePopupShell, changed translations);

    /* Create the menu box */

    ppChoiceBox = XtCreateManagedWidget("ChoiceBox", formWidgetClass,
    ppChoicePopupShell, NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 2); i++;
    XtSetArg(args[1], XtNfromOrig, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetValues(ppChoiceBox, args, i);

    /* Create the buttons */

```

```

i = 0;
XtSetArg(args[i], XtBorderWidth, 0); i++;
XtSetArg(args[i], XtBorderVert, Pbutton[6]); i++;
XtSetArg(args[i], XtBorderHoriz, NULL); i++;
XtSetArg(args[i], XtBorder, "Energy"); i++;
XtSetArg(args[i], XtResizable, FALSE); i++;
XtSetValues(Pbutton[5], args, i);
XtAddCallback(Pbutton[5], XtCallback, Change Var, (int *) ENERGY);

Pbutton[6] = XtCreateManagedWidget("button", commandWidgetClass,
ppChoiceBox,
NULL, 0);

i = 0;
XtSetArg(args[i], XtBorderWidth, 0); i++;
XtSetArg(args[i], XtBorderVert, Pbutton[5]); i++;
XtSetArg(args[i], XtBorderHoriz, NULL); i++;
XtSetArg(args[i], XtBorder, "Signal"); i++;
XtSetArg(args[i], XtResizable, FALSE); i++;
XtSetValues(Pbutton[6], args, i);
XtAddCallback(Pbutton[6], XtCallback, Change Var, (int *) SIGNAL);

Pbutton[7] = XtCreateManagedWidget("button", commandWidgetClass,
ppChoiceBox,
NULL, 0);

i = 0;
XtSetArg(args[i], XtBorderWidth, 0); i++;
XtSetArg(args[i], XtBorderVert, Pbutton[6]); i++;
XtSetArg(args[i], XtBorderHoriz, NULL); i++;
XtSetArg(args[i], XtBorder, "None"); i++;
XtSetArg(args[i], XtResizable, FALSE); i++;
XtSetValues(Pbutton[7], args, i);
XtAddCallback(Pbutton[7], XtCallback, Change Var, (int *) NONE);

changed_translations = XtParseTranslationTable(trans2);
for (i = 0; i < 6; i++)
XtOverrideTranslations(Pbutton[i], changed_translations);

return;
}

/* FUNCTION Postproc Make Ddis Buttons2()
* Purpose
* Initialize the color variable choice menu
* (called from Postproc Init Ddis Dialog)
* Inputs
* None
* Outputs
* None
* Programming History
* 1993 Sept, Gayle E. Ermer, Original function written
*/

void Postproc_Make_Ddis_Buttons2()
{
    int i;
    Arg args[10];
    Widget Pbutton2[8];
    extern Widget ppChoicePopUpShell2, topLevel;
    extern Widget ppChoiceBox2;

    extern int Cur Var2;

    static char *translations = "Override\n\
<BtnUp>: MenuPopDown(Variables2)";
    static char *trans2 = "Override\n\
<BtnUp>: Notify() unset()";
    XtTranslations changed_translations;

    /*** Create the PopUp Shell ****/

    ppChoicePopUpShell2 = XtCreatePopUpShell("Variable2",
transientShellWidgetClass,
topLevel, NULL, 0);

    changed_translations = XtParseTranslationTable(translations);
    XtOverrideTranslations(ppChoicePopUpShell2, changed_translations);

    /*** Create the menu box ****/

    ppChoiceBox2 = XtCreateManagedWidget("ChoiceBox2", formWidgetClass,
ppChoicePopUpShell2, NULL, 0);

    i = 0;
    XtSetArg(args[i], XtBorderWidth, 1); i++;
    XtSetArg(args[i], XtBorderHoriz, NULL); i++;
    XtSetArg(args[i], XtResizable, FALSE); i++;
    XtSetValues(ppChoiceBox2, args, i);

    /*** Create the menu buttons ****/

    Cur Var2 = NONE;

    Pbutton2[0] = XtCreateManagedWidget("button", commandWidgetClass,
ppChoiceBox2,
NULL, 0);

    i = 0;
    XtSetArg(args[i], XtBorderWidth, 0); i++;
    XtSetArg(args[i], XtBorderVert, NULL); i++;
    XtSetArg(args[i], XtBorderHoriz, NULL); i++;
    XtSetArg(args[i], XtBorder, "Power"); i++;
    XtSetArg(args[i], XtResizable, FALSE); i++;
    XtSetValues(Pbutton2[0], args, i);
    XtAddCallback(Pbutton2[0], XtCallback, Change Var Color, (int *) POWER);

    Pbutton2[1] = XtCreateManagedWidget("button", commandWidgetClass,
ppChoiceBox2,
NULL, 0);

    i = 0;
    XtSetArg(args[i], XtBorderWidth, 0); i++;
    XtSetArg(args[i], XtBorderVert, Pbutton2[0]); i++;
    XtSetArg(args[i], XtBorderHoriz, NULL); i++;
    XtSetArg(args[i], XtBorder, "Effort"); i++;
    XtSetArg(args[i], XtResizable, FALSE); i++;
    XtSetValues(Pbutton2[1], args, i);
    XtAddCallback(Pbutton2[1], XtCallback, Change Var Color, (int *) EFFORT);

    Pbutton2[2] = XtCreateManagedWidget("button", commandWidgetClass,
ppChoiceBox2,
NULL, 0);

    i = 0;
    XtSetArg(args[i], XtBorderWidth, 0); i++;
    XtSetArg(args[i], XtBorderVert, Pbutton2[1]); i++;
    XtSetArg(args[i], XtBorderHoriz, NULL); i++;
    XtSetArg(args[i], XtBorder, "Flow"); i++;
    XtSetArg(args[i], XtResizable, FALSE); i++;
    XtSetValues(Pbutton2[2], args, i);
    XtAddCallback(Pbutton2[2], XtCallback, Change Var Color, (int *) FLOW);

```

dyndis.c Tue Nov 8 08:46:57 1994 10

```

XtSetValues(Pbutton2[2], args, 1);
XtAddCallback(Pbutton2[2], XtCallback, Change Var Color, (int *) FLOW);

Pbutton2[3] = XtCreateManagedWidget("button", commandWidgetClass,
    ppChoiceBox2,
    NULL, 0);

i = 0;
XtSetArg(args[1], XtBorderWidth, 0); i++;
XtSetArg(args[1], XtFromVert, Pbutton2[2]); i++;
XtSetArg(args[1], XtFromMoris, NULL); i++;
XtSetArg(args[1], XtLabel, " Displacement "); i++;
XtSetArg(args[1], XtResizable, FALSE); i++;
XtSetValues(Pbutton2[3], args, 1);
XtAddCallback(Pbutton2[3], XtCallback, Change Var Color,
    (int *) DISPLACEMENT);

Pbutton2[4] = XtCreateManagedWidget("button", commandWidgetClass,
    ppChoiceBox2,
    NULL, 0);

i = 0;
XtSetArg(args[1], XtBorderWidth, 0); i++;
XtSetArg(args[1], XtFromVert, Pbutton2[3]); i++;
XtSetArg(args[1], XtFromMoris, NULL); i++;
XtSetArg(args[1], XtLabel, " Momentum "); i++;
XtSetArg(args[1], XtResizable, FALSE); i++;
XtSetValues(Pbutton2[4], args, 1);
XtAddCallback(Pbutton2[4], XtCallback, Change Var Color, (int *) MOMENTUM);

Pbutton2[5] = XtCreateManagedWidget("button", commandWidgetClass,
    ppChoiceBox2,
    NULL, 0);

i = 0;
XtSetArg(args[1], XtBorderWidth, 0); i++;
XtSetArg(args[1], XtFromVert, Pbutton2[4]); i++;
XtSetArg(args[1], XtFromMoris, NULL); i++;
XtSetArg(args[1], XtLabel, " Energy "); i++;
XtSetArg(args[1], XtResizable, FALSE); i++;
XtSetValues(Pbutton2[5], args, 1);
XtAddCallback(Pbutton2[5], XtCallback, Change Var Color, (int *) ENERGY);

Pbutton2[6] = XtCreateManagedWidget("button", commandWidgetClass,
    ppChoiceBox2,
    NULL, 0);

i = 0;
XtSetArg(args[1], XtBorderWidth, 0); i++;
XtSetArg(args[1], XtFromVert, Pbutton2[5]); i++;
XtSetArg(args[1], XtFromMoris, NULL); i++;
XtSetArg(args[1], XtLabel, " Signal "); i++;
XtSetArg(args[1], XtResizable, FALSE); i++;
XtSetValues(Pbutton2[6], args, 1);
XtAddCallback(Pbutton2[6], XtCallback, Change Var Color, (int *) SIGNAL);

Pbutton2[7] = XtCreateManagedWidget("button", commandWidgetClass,
    ppChoiceBox2,
    NULL, 0);

i = 0;
XtSetArg(args[1], XtBorderWidth, 0); i++;
XtSetArg(args[1], XtFromVert, Pbutton2[6]); i++;
XtSetArg(args[1], XtFromMoris, NULL); i++;
XtSetArg(args[1], XtLabel, " None "); i++;
XtSetArg(args[1], XtResizable, FALSE); i++;
XtSetValues(Pbutton2[7], args, 1);
XtAddCallback(Pbutton2[7], XtCallback, Change Var Color, (int *) NONE);

changed translations = XtParseTranslationTable(trans2);

for (i = 0; i < 9; i++)
    XtOverrideTranslations(Pbutton2[i], changed translations);

return;
}

/* FUNCTION Postproc Init Plot Select
 * Purpose
 * Initialize the variable choice menu for plotting
 * (called from Postproc Init Dds Dialog)
 * Inputs
 * None
 * Outputs
 * None
 * Programming History
 * 1994 July, Gayle E. Ermer, Original function written
 */

void Postproc Init Plot Select()
{
    extern Widget toplevel;
    int i;
    Arg args[10];
    Widget PlotButton[8];
    extern int Plot Var;

    static char *translations = "Override\
    <BtnUp>: MenuPopdown(Variable2);\
    static char *trans2 = "Override\
    <BtnUp>: notify() unset();\
    XtTranslations changed translations;

    /* Create the Popup Shell */
    ppPlotSelectShell = XtCreatePopupShell("PlotSelect",
        transientShellWidgetClass,
        toplevel, NULL, 0);

    changed translations = XtParseTranslationTable(translations);
    XtOverrideTranslations(ppPlotSelectShell, changed translations);

    /* Create the menu box */
    ppPlotSelectBox = XtCreateManagedWidget("ChoiceBox", formWidgetClass,
        ppPlotSelectShell, NULL, 0);

    i = 0;
    XtSetArg(args[1], XtBorderWidth, 1); i++;
    XtSetArg(args[1], XtFromMoris, NULL); i++;
    XtSetArg(args[1], XtResizable, FALSE); i++;
    XtSetValues(ppPlotSelectBox, args, 1);

    /* Create the menu buttons */
    Plot Var = POWER;

    PlotButton[0] = XtCreateManagedWidget("button", commandWidgetClass,
        ppPlotSelectBox,
        NULL, 0);

    i = 0;

```



```

*/
void Change_Plot_Select2(int variable)
{
    extern Widget toplevel;
    extern int Plot_Var;
    extern int Cur_Stage;
    extern int Init_timestep;
    char *heading[] = {
        "Power",
        "Effort",
        "Flow",
        "Displacement",
        "Momentum",
        "Energy",
        "Signal"
    };
    Arg args[2];
    Plot_Var = variable;
    XtPopdown(ppPlotSelectShell);
    Postproc_Main_Button_DDlg_Plot(pickcon);
    return;
}

/* FUNCTION Change_glyph
 * Purpose
 *   Change the glyph when the menu box is used
 * Inputs
 *   Widget w - the calling widget
 * Outputs
 *   None
 * Programming History
 *   1994 Jan, Gayle E. Ermer, Function transposed to dyndis file
 */
void Change_glyph(w, client_data, call_data)
Widget w;
caddr_t call_data, client_data;
{
    int variable = (int) client_data;
    Change_Glyph(variable);
    return;
}

/* FUNCTION Change_glyph2
 * Purpose
 *   Change the heading in the menu box
 *   (called by Change_glyph)
 * Inputs
 *   int variable - the value of the new variable
 * Outputs

```


dynidis.c Tue Nov 8 08:46:57 1994 13

```

EnergyICShell = XtCreatePopUpShell ("EnergyICShell",
    transientShellWidgetClass,
    parent, args, 1);

Energy_init = XtCreateManagedWidget ("Energy_init", formWidgetClass,
    EnergyICShell, NULL, 0);

i = 0;
XtSetArg(args[i], XtNfromVert, 100); i++;
XtSetArg(args[i], XtNfromHoris, NULL); i++;
XtSetValues(Energy_init, args, i);

/* Create the label */
label = XtCreateManagedWidget ("candlabel", labelWidgetClass,
    Energy_init, NULL, 0);

i = 0;
XtSetArg(args[i], XtNfromVert, label); i++;
XtSetArg(args[i], XtNfromHoris, NULL); i++;
editenergy = XtCreateManagedWidget ("candedit", asciiTextWidgetClass,
    Energy_init, args, i);

/* Create the view window */
SetTextWidget(editenergy);
energy_viewer = XtCreateManagedWidget ("candview", viewportWidgetClass,
    Energy_init, NULL, 0);

i = 0;
XtSetArg(args[i], XtNfromVert, editenergy); i++;
XtSetArg(args[i], XtNfromHoris, NULL); i++;
XtSetValues(energy_viewer, args, i);

/* update button */
i = 0;
XtSetArg(args[i], XtNfromVert, label); i++;
XtSetArg(args[i], XtNfromHoris, energy_viewer); i++;
XtSetArg(args[i], XtNresponsive, FALSE); i++;
update = XtCreateManagedWidget ("candupdate", commandWidgetClass,
    Energy_init, args, i);

XtAddCallback(update, XtCallbackProc) update energy, NULL);

/* editing box */
energy_box = XtCreateManagedWidget ("candbox", listWidgetClass,
    energy_viewer, NULL, 0);
XtAddCallback(energy_box, XtCallbackProc) edit energy, NULL);

/* ok button */
i = 0;
XtSetArg(args[i], XtNfromVert, energy_viewer); i++;
XtSetArg(args[i], XtNfromHoris, NULL); i++;
ok_button = XtCreateManagedWidget ("ok", commandWidgetClass,
    Energy_init, args, i);
XtAddCallback(ok_button, XtCallbackProc) accept energy, NULL);

/* cancel button */
i = 0;
XtSetArg(args[i], XtNfromVert, energy_viewer); i++;
XtSetArg(args[i], XtNfromHoris, ok_button); i++;
cancel = XtCreateManagedWidget ("cancel", commandWidgetClass,
    Energy_init, args, i);
XtAddCallback(cancel, XtCallbackProc) cancel energy, NULL);

save energy vals();
}

/* FUNCTION Change_Scale
 * Purpose
 * Change the scale when the menu box is used
 * Inputs
 * Widget w - the calling widget
 * Outputs
 * None
 * Programming History
 * 1993 Dec, Gayle E. Ermer
 */
void Change_Scale(w, client_data, call_data)
Widget w;
caddr_t call_data, client_data;
{
    int variable = (int) client_data;
    Change_Scaled(variable);
    return;
}

/* FUNCTION Change_Scaled
 * Purpose
 * Change the heading in the menu box - called by ChangeScale
 * (called by Change_Scale)
 * Inputs
 * int variable - the value of the new variable
 * Outputs
 * None
 * Programming History
 * 1993 Sept, Gayle E. Ermer, Original function written
 */
void Change_Scaled(int variable)
{
    extern int Scale_Var;
    extern Widget psScalingChoiceVal;
    extern int Cur_Stage;
    extern int init_timestep;
    char *heading[] = {
        "Linear",
        "Logarithmic",
        "Absolute Log",
    };
    Arg args[2];

```



```

char *heading[] = {
    "Power",
    "Effort",
    "Flow",
    "Displacement",
    "Momentum",
    "Energy",
    "Signal",
    "None"
};

Arg args[2];

Cur_Var2 = variable;

XtSetArg(args[0], XtNlabel, heading[variable]);
XtSetValues(ppVarTypeColor, args, 1);

Cur_Stage = init_timestep;

return;
}

/* FUNCTION PVTType Pop
 * Purpose
 * Popup the width variable choice menu in the appropriate location
 * Inputs
 * Widget w - calling widget
 * Outputs
 * None
 * Programming History
 * 1993 Sept, Gayle E. Ermer, Function adapted to dyndis file
 */
/*ARGUSED*/
void PVTType Pop2(w, event)
Widget w;
XtEvent *event;
{
    extern Widget ppChoicePopupShell2;
    extern Widget ppVarTypeColor;
    Position x, y, xlr, ylr, height;
    int i;
    Arg args[5];

    i = 0;
    XtSetArg(args[i], XtNheight, &height); i++;
    XtSetValues(ppVarTypeColor, args, 1);

    xlr = 0;
    ylr = height;

    XtTranslateCoords(ppVarTypeColor, xlr, ylr, &x, &y);

    i = 0;
    XtSetArg(args[i], XtNx, x); i++;
    XtSetArg(args[i], XtNy, y); i++;
    XtSetValues(ppChoicePopupShell2, args, 1);

    return;
}

/* FUNCTION OutType Pop
 * Purpose
 * Popup the glyph choice menu in the appropriate location
 * Inputs
 * Widget w - the calling widget
 * Outputs
 * None
 * Programming History
 * 1993 Dec, Gayle E. Ermer, Function adapted to dyndis file
 */
/*ARGUSED*/
void OutType Pop(w, event)
Widget w;

```

```

    XButtonEvent *event;

    extern Widget ppPolybChoiceVal;
    extern Widget ppChoicePopupShell;
    Position x, y, xlr, ylr, height;
    int i;
    Arg arg[5];

    i = 0;
    XtSetArg(arg[i], XtMHeight, &height); i++;
    XtGetValues(ppPolybChoiceVal, arg, i);

    xlr = 0;
    ylr = height;

    XtTranslateCoords(ppPolybChoiceVal, xlr, ylr, &x, &y);

    i = 0;
    XtSetArg(arg[i], XtMx, x); i++;
    XtSetArg(arg[i], XtMy, y); i++;
    XtGetValues(ppChoicePopupShell, arg, i);

    return;
}

/* FUNCTION sCType Pop
 *
 * Purpose
 *   Pop up the scaling choice menu in the appropriate location
 *
 * Inputs
 *   Widget w - the calling widget
 *
 * Outputs
 *   None
 *
 * Programming History
 *   1993 Dec, Gayle E. Ermar, Function adapted to dyndis file
 *
 */
/*ACQUIRED*/
void sCType Pop(w, event)
Widget w;
XButtonEvent *event;
{
    extern Widget ppScalingChoiceVal;
    extern Widget ppChoicePopupShell;
    Position x, y, xlr, ylr, height;
    int i;
    Arg arg[5];

    i = 0;
    XtSetArg(arg[i], XtMHeight, &height); i++;
    XtGetValues(ppScalingChoiceVal, arg, i);

    xlr = 0;
    ylr = height;

    XtTranslateCoords(ppScalingChoiceVal, xlr, ylr, &x, &y);

    i = 0;
    XtSetArg(arg[i], XtMx, x); i++;
    XtSetArg(arg[i], XtMy, y); i++;
    XtGetValues(ppChoicePopupShell, arg, i);
}

return;
}

/* FUNCTION Postproc Main Button Ddis
 *
 * Purpose
 *   Puts up the Ddis dialog when the "Dyn Dis" button is pressed
 *
 * Inputs
 *   parent (the parent widget of the dynamic display dialog)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 June, Gayle E. Ermar, Function transposed to dyndis file
 *
 */
void Postproc Main Button Ddis(Widget w, caddr t junk, caddr t junk2)
{
    extern Widget toplvl;

    if (!Postproc Main Dialog Check()) {
        return;
    }
    Postproc Popup Ddis();
}

/* FUNCTION Postproc Popup Ddis
 *
 * Purpose
 *   Pops up the Ddis dialog
 *
 * Inputs
 *   none
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 June, Gayle E. Ermar, Original function written
 *
 */
void Postproc Popup Ddis()
{
    extern Widget toplvl;
    real tmaxr, tminr, tmaxy, tminy;

    CenterPop(toplvl, appDdisDialogPopupShell);
}

/* FUNCTION Postproc Ddis.Button.Cancel
 *
 * Purpose
 *   Cancels the Ddis Dialog
 *
 * Inputs
 *   parent (the parent widget of the ddis dialog)
 *
 * Outputs
 *   none
 *
 */

```



```

/* FUNCTION Postproc_Arrange Signals
 * Purpose
 * Arrange and find the ranges of the signal variables
 * (called from Postproc_Load It)
 * Inputs
 * None
 * Outputs
 * None
 * Programming History
 * 24 Jun Original function written by G. Ermar
 */

void Postproc_Arrange_Signals(MWidget w)
{
    ModeType *traverseModes;
    ConnectorType *traverseCons;
    int ondx;
    char *ptr;
    char *ptring;
    char Temp[13];
    int iindx,i, j, tok, stage;
    extern int num_stages;
    int num_conns;
    real value;
    real ranges, maxs, mins, minranges;
    real datas;
    extern real min, max, range, minrang;
    extern real sdata[200][300];
    extern int Cur_Stage;
    extern int num_stages, init_timestep, final_timestep;
    extern real ppmainDialogInitTimeVal, ppmainDialogFinalTimeVal;

    /* arrange signal data in array */
    traverseCons = First_Connector->Next;
    i = 0;
    while (traverseCons) {
        for (ondx = 0; ondx < seqlb2 1.novv; ++ondx) {
            ptring = traverseCons->name;
            ptr = seqlb1 1.oval * (12 + ondx);
            sprintf(Temp, "%s", nohars (ptr, 12), ptr);
            tok = strump(ptring, Temp);
            if (tok == 0) {
                for (stage = init_timestep; stage <= final_timestep; ++stage) {
                    Postproc_Get_Value(spos0bk 1.rslit(ondx-MAXIND), stage, &value);
                    sdata[i][stage] = value;
                }
            }
            traverseCons = traverseCons->Next; ++i;
        }
    }

    /* find ranges and max/min */
    ranges = 0.0;
    minranges = 999999999.0;
    num_conns = (seqlb2 1.novv - sclab2 1.nxi) / 2;

    for (stage = 0; stage < num_stages; stage++) {
        for (iindx = 0; iindx < num_conns; iindx++) {
            datas = sdata[iindx][stage];
            if (datas < 0.0) datas = -datas;
            if (datas > ranges) ranges = datas;
        }
        range = ranges;
    }

    for (stage = 0; stage < num_stages; stage++) {
        for (iindx = 0; iindx < num_conns; iindx++) {
            datas = sdata[iindx][stage];
            if (datas == 0.0) minranges = minranges;
            if (datas < 0.0) datas = -datas;
            if (datas < minranges && datas != 0.0) minranges = datas;
        }
        minrang = minranges;
    }

    maxs = -999999999.0;
    mins = 999999999.0;

    for (stage = 0; stage < num_stages; stage++) {
        for (iindx = 0; iindx < num_conns; iindx++) {
            datas = sdata[iindx][stage];
            if (datas > maxs) maxs = datas;
            if (datas < mins) mins = datas;
        }
        max = maxs;
        min = mins;
    }

    /* FUNCTION Postproc_Find_Data_Ranges
     * Purpose
     * Find the ranges of the effort, flow, and power variables
     * (called from Postproc_Arrange_Data)
     * Inputs
     * None
     * Outputs
     * None
     * Programming History
     * 24 Mar Original function written by G. Ermar
     */

    void Postproc_Find_Data_Ranges()
    {
        extern real ppdata[200][300][2];
        extern int num_stages, new_num_stages;
        int stage, iindx, num_conns;
        real datas, dataf, datap, maxs, maxf, maxp, mins, minf, minp;
        extern real prange, srange, frange;
        extern real pairng, emirng, fairng;

        maxs = maxf = maxp = 0.0;
        mins = minf = minp = 999999999.0;
        num_conns = (seqlb2 1.novv - sclab2 1.nxi) / 2;

        /* find maximum magnitude */

```



```

    ppdata[index][0][1] = 0.0;
    traverseCons = traverseCons->Next; ++index;
}

/* Second stage ****/
sum = 0;
traverseCons = First Connector->Next;
index = 0;
while (traverseCons) {
    sum = ppdata[index][0][1] + ppdata[index][1][1];
    ppdata[index][1][1] = sum * deltat / 2.0;
    traverseCons = traverseCons->Next; ++index;
}

/* Rest of stages ****/
for (stage = 2; stage < new_num_stages; stage++) {
    sum = 0;
    traverseCons = First Connector->Next;
    index = 0;
    while (traverseCons) {
        sum = ppdata[index][0][0] + ppdata[index][0][1]
            + ppdata[index][stage][0] + ppdata[index][stage][1];
        for (j = 1; j < stage; j++) {
            sum = sum + 2.0 * ppdata[index][j][0] + ppdata[index][j][1];
        }
        ppdata[index][stage][2] = (((real) stage * deltat)
            / (2.0 * (real) (stage))) * sum;
        traverseCons = traverseCons->Next; ++index;
    }
}

/* Calculate energies from efforts and flows ****/
/* First stage ****/
traverseCons = First Connector->Next;
index = 0;
while (traverseCons) {
    ppdata[index][0][2] = 0.0;
    traverseCons = traverseCons->Next; ++index;
}

/* Second stage ****/
sum = 0;
traverseCons = First Connector->Next;
index = 0;
while (traverseCons) {
    sum = ppdata[index][0][0] + ppdata[index][0][1]
        + ppdata[index][1][0] + ppdata[index][1][1];
    ppdata[index][1][2] = sum * deltat / 2.0;
    traverseCons = traverseCons->Next; ++index;
}

/* Rest of stages ****/
for (stage = 2; stage < new_num_stages; stage++) {
    sum = 0;
    traverseCons = First Connector->Next;
    index = 0;
    while (traverseCons) {
        sum = ppdata[index][0][0] + ppdata[index][0][1]
            + ppdata[index][stage][0] + ppdata[index][stage][1];
        for (j = 1; j < stage; j++) {
            sum = sum + 2.0 * ppdata[index][j][0] + ppdata[index][j][1];
        }
        ppdata[index][stage][2] = (((real) stage * deltat)
            / (2.0 * (real) (stage))) * sum;
        traverseCons = traverseCons->Next; ++index;
    }
}

/* find maximum magnitudes */
void Postproc Int Ranges()
{
    extern real ppdata[200][300][3];
    extern int num_stages, new_num_stages;
    int stage, index, num_cons;
    real datap, dataq, dataa, maxp, maxq, maxa, minp, ming, minn;
    extern real pmaxrange, qrange, brange;
    extern real pminrange, qminrange, uainrange;
    maxp = maxq = maxa = 0.0;
    ming = minq = minn = 999999999.0;
    num_cons = (seqlen2 1.novv - seqlen2 1.nxi) / 2;
}

```



```

    timestep = Cur_Stage;
    if (timestep >= final_timestep) {
    }
    else {
        Postproc_Draw_Key(w);
        Postproc_Draw(w);
        Cur_Stage = Cur_Stage + 1;
    }
}

/* FUNCTION Postproc_Step_Backward
 *
 * Purpose
 *     Executes previous step of the animation
 *
 * Inputs
 *     parent (the parent widget of the Ddis dialog)
 *
 * Outputs
 *     none
 *
 * Programming History
 *     93 Dec Original function written by G. Ermar
 */
void Postproc_Step_Backward(Widget w)
{
    extern int Cur_Stage;
    extern int timestep;
    extern int init_timestep;
    int flag;

    flag = Postproc_Check_Log();
    if (flag < 0) return;

    timestep = Cur_Stage - 2;

    if (timestep < init_timestep) {
    }
    else {
        Postproc_Draw_Key(w);
        Postproc_Draw(w);
        Cur_Stage = Cur_Stage - 1;
    }
}

/* FUNCTION Postproc_Goto_Time
 *
 * Purpose
 *     Go to new frame and draw
 *
 * Inputs
 *     parent (the parent widget of the Ddis dialog)
 *
 * Outputs
 *     none
 *
 * Programming History
 *     93 Sept Original function written by G. Ermar
 */
void Postproc_Goto_Time(Widget w)
{
    extern int timestep;
    extern int new_num_stages, init_timestep, final_timestep;
    extern int Scale_Var;
    extern int Cur_Var, Cur_Var2;
    extern int Cur_Stage;
    int delay;
    real end_time;
    int end_stage;
    int i, length;
    Arg args[5];
    char *time;

    void Postproc_Start_Animate(Widget w)
    {
        extern int timestep;
        extern int new_num_stages, init_timestep, final_timestep;
        extern int Scale_Var;
        extern int Cur_Var, Cur_Var2;
        extern int Cur_Stage;
        int delay;
        real end_time;
        int end_stage;
        int i, length;
        Arg args[5];
        char *time;
    }
}

```

```

char *end_string;
Display "diop;
extern real ppmainDialogFinalTimeVal, ppmainDialogInitTimeVal;
XEvent event;
int flag;

/*** Check for log of negative # problem ****/
flag = Postproc Check Log();
if (flag < 0) return;

/*** Draw Key ****/
Postproc Draw Key(w);

/*** Get final time ****/
GetDialog("Enter Final Time: ", &end_string);
end_time = atof(end_string);

if (end_time < ppmainDialogInitTimeVal ||
    end_time > ppmainDialogFinalTimeVal) {
}
else {
    end_stage = (end_time - setlb4.1.tim) / setlb4.1.dttstr;
}

/*** Deal with Time Increment ****/
GetDialog("Enter Time per Frame in seconds: ", &time);
asctoi(&time, &delay);

/*** Start time loop ****/
for (timestage = Cur Stage;
    timestage <= end_stage; ++timestage) {

/*** Draw the graph ****/
Postproc Draw(w);

/*** Wait ****/
sleep(delay);

}
Cur Stage = end_stage;
return;
}

/* FUNCTION Postproc Check Log()
* Purpose
* Check for negative numbers in output
* Inputs
* none
* Outputs
* none
* Programming History
char time[] = "KEY";

```

```

char fontMG[] = "os Mag";
char fontP[10];
char *fontm = "x10";
char *fontm2 = "x10";
XFontStruct *font info;
real labelrange, labelvalue, rangemin;
real max, min;
real delta, exp;

/* Clear the key area */
XCclearArea(XtDisplay(ppDicCanvasKey), XtWindow(ppDicCanvasKey),
            0, 0, 200, 100, False);
XFlush(XtDisplay(ppDicCanvasKey));

/* Label the box as KEY */
values.foreground = BlackPixel(
    XtDisplay(ppDicCanvasKey),
    DefaultScreen(XtDisplay(ppDicCanvasKey)));
XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
           GCForeground, values);
font info = XGCFont(XtDisplay(ppDicCanvasKey), fontm);
nlen = strlen(fontm);
values.font = font info->fid;
XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex, OCFont, values);
XDrawString(XtDisplay(ppDicCanvasKey), XtWindow(ppDicCanvasKey),
            gcmaintex, 10, 10, fontm, nlen);

/* Draw varying line widths */
values.foreground = BlackPixel(
    XtDisplay(ppDicCanvasKey),
    DefaultScreen(XtDisplay(ppDicCanvasKey)));
XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
           GCForeground, values);
for (i = 0; i < 10; i++) {
    values.line width = i+1;
    XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
               GCLineWidth, values);
    XDrawLine(XtDisplay(ppDicCanvasKey), XtWindow(ppDicCanvasKey),
              gcmaintex, 10, i*20 + 40, 80, i*20 + 40);
}

/* Draw different colors with constant line widths */
values.line width = 5;
XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
           GCLineWidth, values);
for (i = 0; i < 10; i++) {
    values.foreground = Pset Color((real) (i*0.1 + 0.001), 0);
    XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
               GCForeground, values);
    XDrawLine(XtDisplay(ppDicCanvasKey), XtWindow(ppDicCanvasKey),
              gcmaintex, 110, i*20 + 40, 155, i*20 + 40);
    values.foreground = Pset Color((real) (i*0.1 + 0.001), 1);
    XChangeGC(XtDisplay(ppDicCanvasKey), gcmaintex,
               GCForeground, values);
    XDrawLine(XtDisplay(ppDicCanvasKey), XtWindow(ppDicCanvasKey),
              gcmaintex, 155, i*20 + 40, 180, i*20 + 40);
}

/* Draw labels for line width variable */

```

```

values foreground = BlackPixel (
    XDisplay(ppDdcCanvasKey));
DefaultScreen(XtDisplay(ppDdcCanvasKey));
if (Cur Var == NAME) values foreground = WhitePixel (
    XtDisplay(ppDdcCanvasKey));
DefaultScreen(XtDisplay(ppDdcCanvasKey));
XChangeGC(XtDisplay(ppDdcCanvasKey), gcAnimate,
    OCForeground, &values);
font info2 = XLocQueryFont(XtDisplay(ppDdcCanvasKey), fontm2);
values.font = font info2->fid;
XChangeGC(XtDisplay(ppDdcCanvasKey), gcAnimate, OCFont, &values);

switch (Cur Var) {
    case BPPOINT: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case FLOW: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case POWER: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case DISPLACEMENT: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case MOMENTUM: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case SIGNAL: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
    case ENERGY: labelrange = &range; max = &max; min = &min;
        rangemin = &rangemin; break;
}

if (Scale Var == LOG) delta = (log10(max) - log10(min)) / 10;
if (Scale Var == ABSLOG) delta = (log10(labelrange) - log10(rangemin)) / 10;

for (i = 0; i < 11; i++) {
    if (Scale Var == LIN) {
        labelvalue = i*labelrange*0.1;
    }
    else if (Scale Var == LOG) {
        exp = i*delta + log10(min);
        labelvalue = pow(10.0, exp);
    }
    else {
        exp = i*delta + log10(rangemin);
        labelvalue = pow(10.0, exp);
    }
    sprintf(TEMP, "%5.2g", labelvalue);
    nlen = strlen(TEMP);
    XDrawString(XtDisplay(ppDdcCanvasKey), XtWindow(ppDdcCanvasKey),
        gcAnimate, 10, i*20 + 31, TEMP, nlen);
}

/** Draw labels for color variable ****/

values foreground = BlackPixel (
    XDisplay(ppDdcCanvasKey));
DefaultScreen(XtDisplay(ppDdcCanvasKey));
if (Cur Var2 == NAME) values foreground = WhitePixel (
    XtDisplay(ppDdcCanvasKey));
DefaultScreen(XtDisplay(ppDdcCanvasKey));
XChangeGC(XtDisplay(ppDdcCanvasKey), gcAnimate,
    OCForeground, &values);
font info2 = XLocQueryFont(XtDisplay(ppDdcCanvasKey), fontm2);
values.font = font info2->fid;
XChangeGC(XtDisplay(ppDdcCanvasKey), gcAnimate, OCFont, &values);
nlen = strlen(PMSRGB);
XDrawString(XtDisplay(ppDdcCanvasKey), XtWindow(ppDdcCanvasKey),

```

```

    gcaimstatf, 125, 20, PORMOD, nlen);
    switch (Cur Var2) {
        case EFFORT: labelrange = orange; max = emax; min = emin;
            rangemin = eminrng; break;
        case FLOW: labelrange = fringe; max = fmax; min = fmin;
            rangemin = fminrng; break;
        case POWER: labelrange = prange; max = pmx; min = pmn;
            rangemin = pmrng; break;
        case DISPLACEMENT: labelrange = grange; max = gmax; min = gmin;
            rangemin = gminrng; break;
        case HUMIDITY: labelrange = purange; max = pmx; min = pmn;
            rangemin = pmrng; break;
        case SSGMIL: labelrange = orange; max = smax; min = smn;
            rangemin = smrng; break;
        case HUMIDITY: labelrange = orange; max = smax; min = smn;
            rangemin = smrng; break;
    }
    for (i = 0; i < 11; i++) {
        if (Scale Var == LIN) {
            labelvalue = i*labelrange+0.1;
        }
        else if (Scale Var == LOG10) {
            exp = i*delta + log10(rangemin);
            labelvalue = pow(10.0, exp);
        }
        else {
            exp = i*delta + log10(min);
            labelvalue = pow(10.0, exp);
        }
        sprintf(TMP, "%5.2E", labelvalue);
        nlen = strlen(TMP);
        XDrawString(XtDisplay(ppDdcCanvasKey), XtWindow(ppDdcCanvasKey),
            gcaimstatf, 100, i*20 + 33, TMP, nlen);
    }
}

/*... Flush to Draw everything ....*/
XFlush(XtDisplay(ppDdcCanvasKey));
}

/* FUNCTION Postproc draw
 *
 * Purpose
 *   Updates the graph
 *
 * Inputs
 *   parent (the parent widget of the Ddc dialog)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   93 August Original function written by G. Erner
 *
 */
void Postproc_draw(Widget w)
{
    extern int Cur Var, Cur Var2;
    extern real orange, fringe, prange, grange, purange, nrange, orange;
    extern real ppdata[200][300][2];
    extern real sdata[200][300];
    extern real ppsdat[200][300][3];

```



```

        switch_arrow = 1;
        value = -value;
    }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == FLOW) {
        value = ppdata[index][timestage][1];
        value = value / frange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == POWER) {
        value = ppdata[index][timestage][0];
        value = ppdata[index][timestage][1];
        value = value * fvalue;
        value = value / prange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == MOM) {
        value = 0;
        switch_arrow = 0;
    }
    if (Cur_Var == SIGNAL) {
        value = 0;
        switch_arrow = 0;
    }
    if (Cur_Var == DISPLACEMENT) {
        value = ppdata[index][timestage][1];
        value = value / grange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == MOMENTUM) {
        value = ppdata[index][timestage][0];
        value = value / prange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == ENERGY) {
        value = ppdata[index][timestage][2];
        value = value / arange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == REPORT) {
        value = ppdata[index][timestage][0];
        value = value / arange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == FLOW) {
        value = ppdata[index][timestage][1];
        value = value / frange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == POWER) {
        value = ppdata[index][timestage][0];
        value = ppdata[index][timestage][1];
        value = value * ofvalue;
        value = value / prange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == MOM) {
        value = 0;
        switch_arrow = 0;
    }
    if (Cur_Var2 == SIGNAL) {
        value = 0;
        switch_arrow = 0;
    }
    if (Cur_Var2 == DISPLACEMENT) {
        value = ppdata[index][timestage][1];
        value = value / grange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == MOMENTUM) {
        value = ppdata[index][timestage][0];
        value = value / prange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var2 == ENERGY) {
        value = ppdata[index][timestage][2];
        value = value / arange;
        if (value < 0) {
            switch_arrow = 1;
            value = -value;
        }
    }
    else switch_arrow = 0;
    }
    if (Cur_Var == Cur_Var2) switch_arrow = 0;
    if (Scale_Var == LIN) {
        values.foreground = Fgct_Color(cvalue, switch_arrow);
        values.line_width = Fgct_gcvalue;
    }
}

```

```

if (Scale Var == LOG) {
    switch (Cur Var2) {
        case EFFORT: value = ovalue * orange; break;
        case FLOW: value = ovalue * fringe; break;
        case POWER: value = ovalue * grange; break;
        case DISPLACEMENT: value = ovalue * grange; break;
        case WORKSTON: value = ovalue * parange; break;
        case ENERGY: value = ovalue * arange; break;
    }
    values.foreground = Pset_Color_Log(value, switch_arrow);
    switch (Cur Var) {
        case EFFORT: value = value * orange; break;
        case FLOW: value = value * fringe; break;
        case POWER: value = value * grange; break;
        case DISPLACEMENT: value = value * grange; break;
        case WORKSTON: value = value * parange; break;
        case ENERGY: value = value * arange; break;
    }
    values.line_width = Pset_gg_Log(value, switch_arrow);
    switch_arrow = 0;
}
if (Scale Var == ABSLOG) {
    switch (Cur Var2) {
        case EFFORT: ovalue = ovalue * orange; break;
        case FLOW: ovalue = ovalue * fringe; break;
        case POWER: ovalue = ovalue * grange; break;
        case DISPLACEMENT: ovalue = ovalue * grange; break;
        case WORKSTON: ovalue = ovalue * parange; break;
        case ENERGY: ovalue = ovalue * arange; break;
    }
    values.foreground = Pset_Color_Abs(ovalue, switch_arrow);
    switch (Cur Var) {
        case EFFORT: value = value * orange; break;
        case FLOW: value = value * fringe; break;
        case POWER: value = value * grange; break;
        case DISPLACEMENT: value = value * grange; break;
        case WORKSTON: value = value * parange; break;
        case ENERGY: value = value * arange; break;
    }
    values.line_width = Pset_gg_Abs(value, switch_arrow);
}
if (Cur Var2 == NONE || Cur Var2 == SIGNAL)
    values.foreground = BlackPixel(
        XDisplay(ppdDialogDraw),
        DefaultScreen(XDisplay(ppdDialogDraw)));
if ((Cur Var == NONE || Cur Var == SIGNAL) &&
    (Cur Var2 != NONE || Cur Var2 != SIGNAL))
    values.line_width = 4;
if ((Cur Var == NONE || Cur Var == SIGNAL) &&
    (Cur Var2 == NONE || Cur Var2 == SIGNAL))
    values.line_width = 1;
XChangeGC(XDisplay(ppdDialogDraw), gcolimate,
    GCLineWidth, &values);
XChangeGC(XDisplay(ppdDialogDraw), gcolimate,
    GCForeground, &values);
Pdraw_connector(ppdDialogDraw, traverseCons,
    gcolimate, switch_arrow);
}
traverseCons = traverseCons->next; ++index;
}
/* Draw glyphs */
/* go through and check node types */
traverseModes = First_Mode->next;

```

```

ctr = 0;
while (traverseModes) {
    Type = traverseModes->type;
    if (Type == R_MODE) {
        Postproc Get Cons on Mode(traverseModes, &numind, &indices);
        sum = 0;
        for (n = 0; n < numind; n++) {
            index = indices[n];
            glyphvalue = ppdist[index][timestage][2];
            sum = sum + glyphvalue;
        }
        glyphlength = Postproc Get Glyph Length(sum);
        if (Glyph Var == 1) {
            Postproc Draw Glyph(ppdDialogDraw, traverseModes, glyphlength);
        }
    }
    if (Type == C_MODE || Type == I_MODE) {
        Postproc Get Cons on Mode(traverseModes, &numind, &indices);
        sum = 0;
        for (n = 0; n < numind; n++) {
            index = indices[n];
            glyphvalue = ppdist[index][timestage][2];
            sum = sum + glyphvalue;
        }
        sum = sum + Init Glyph Val(ctr);
        glyphlength = Postproc Get Glyph Length(sum);
        if (Glyph Var == 1) {
            Postproc Draw Glyph(ppdDialogDraw, traverseModes, glyphlength);
        }
        ctr++;
    }
    traverseModes = traverseModes->next;
}
/* Flush to draw everything */
XFlush(XDisplay(ppdDialogDraw));
return;
}
/* FUNCTION Postproc Get Cons on Mode
 * Purpose
 * Find the connectors on a node
 * Inputs
 * Mode
 * Outputs
 * None
 * Programming History
 * 1993 Dec, Gayle E. Ermer, Function adapted from enport
 */
void Postproc Get Cons on Mode(ModeType *node, int *num, int *ind)
{
    int i, icon;
    C on M Type *nodeconst;
    ConnectorType *const;
    ConnectorType *traverseCons;
    char *string, *pstring;
    int tok;
}

```



```

i = 0;
modconopt = mode->c on mode;
if (modconopt) modconopt = modconopt->next;
while (modconopt) {
    conopt = modconopt->on mode;
    string = conopt->name;
    traverseCons = FirstConnector->next;
    icon = 0;
    while (traverseCons) {
        petring = traverseCons->name;
        tok = string(petring, string);
        if (tok == 0) {
            *(ind+i) = icon;
        }
        traverseCons = traverseCons->next; ++icon;
    }
    modconopt = modconopt->next;
    i++;
}
*sum = i;
}

/* FUNCTION Postproc_Get Glyph Length
 *
 * Purpose
 * Find the length of line in glyph symbol
 *
 * Inputs
 * double gvalue - energy value of glyph
 *
 * Outputs
 * None
 *
 * Programming History
 * 1993 Dec, Gayle E. Ermer, Original function written
 */

```

```

unsigned Postproc_Get Glyph_Length(double gvalue)
{

```

```

    int i;
    extern real glyphrange, glyphmin, glyphmax;
    int glyphlength;

    if (gvalue > (0.95 * glyphrange + glyphmin)) {
        glyphlength = 19;
    }
    else if (gvalue > (0.9 * glyphrange + glyphmin)) {
        glyphlength = 18;
    }
    else if (gvalue > (0.85 * glyphrange + glyphmin)) {
        glyphlength = 17;
    }
    else if (gvalue > (0.8 * glyphrange + glyphmin)) {
        glyphlength = 16;
    }
    else if (gvalue > (0.75 * glyphrange + glyphmin)) {
        glyphlength = 15;
    }
    else if (gvalue > (0.7 * glyphrange + glyphmin)) {
        glyphlength = 14;
    }
    else if (gvalue > (0.65 * glyphrange + glyphmin)) {
        glyphlength = 13;
    }
    else if (gvalue > (0.6 * glyphrange + glyphmin)) {
        glyphlength = 12;
    }
    else if (gvalue > (0.55 * glyphrange + glyphmin)) {
        glyphlength = 11;
    }
    else if (gvalue > (0.5 * glyphrange + glyphmin)) {
        glyphlength = 10;
    }

```

```

    else if (gvalue > (0.45 * glyphrange + glyphmin)) {
        glyphlength = 9;
    }
    else if (gvalue > (0.4 * glyphrange + glyphmin)) {
        glyphlength = 8;
    }
    else if (gvalue > (0.35 * glyphrange + glyphmin)) {
        glyphlength = 7;
    }
    else if (gvalue > (0.3 * glyphrange + glyphmin)) {
        glyphlength = 6;
    }
    else if (gvalue > (0.25 * glyphrange + glyphmin)) {
        glyphlength = 5;
    }
    else if (gvalue > (0.2 * glyphrange + glyphmin)) {
        glyphlength = 4;
    }
    else if (gvalue > (0.15 * glyphrange + glyphmin)) {
        glyphlength = 3;
    }
    else if (gvalue > (0.1 * glyphrange + glyphmin)) {
        glyphlength = 2;
    }
    else if (gvalue > (0.05 * glyphrange + glyphmin)) {
        glyphlength = 1;
    }
    else glyphlength = 0;

    return glyphlength;
}

/* FUNCTION Postproc_Draw Glyph
 *
 * Purpose
 * Draw a glyph at a C or I node
 *
 * Inputs
 * Widget canvas
 * ModeType *mode - node to draw glyph at
 * int length - length of glyph
 *
 * Outputs
 * None
 *
 * Programming History
 * 1993 Dec, Gayle E. Ermer, Original function written
 */

void Postproc_Draw Glyph(Widget ppcanvas, ModeType *mode,
    int length)
{
    extern OC gc[], getanimate;
    int x, y, xb, yb, width, height;
    int xbottom, xtop, ybottom, ytop, xmid, ymid;
    int xcornerbl, xcornerbr, ycornerbl, ycornerbr;
    int ycornerbl, ycornerbr, xcornerbl, xcornerbr;
    XOCValues values;

    /* Get mode coordinates */
    Pealc node_coords(mode, &x, &y, &xb, &yb, &width, &height);

    /* Find glyph corners */
    xbottom = x + 15;
    ybottom = y - 10;
    xtop = xbottom;
    ytop = ybottom + 10;
    xcornerbl = xbottom - 5;
    ycornerbl = ybottom - 5;

```

```

dyndis.c          Tue Nov  8 08:46:57 1994          30

    zoommrbr = ybottom + 5;
    ytopmrbr = ybottom - 5;
    zoommrbl = stop - 5;
    ytopmrbl = ytop + 5;
    zoommrtr = stop + 5;
    ytopmrtr = ytop + 5;

    /*** Draw lines to make glyph ****/

    values.foreground = blackPixel(XtDisplay(ppocanvas),
        DefaultScreen(XtDisplay(ppocanvas)));
    XChangeGC(XtDisplay(ppocanvas), gc[0], GCForeground, &values);

    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrbl, ytopmrbl, zoommrtr, ytopmrtr);
    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrtr, ytopmrtr, zoommrtr, ytopmrtr);
    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrtr, ytopmrtr, zoommrtr, ytopmrtr);
    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrtr, ytopmrtr, zoommrtr, ytopmrtr);
    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrtr, ytopmrtr, zoommrtr, ytopmrtr);

    values.lineWidth = 5;
    XChangeGC(XtDisplay(ppocanvas), gc[0], GCLineWidth, &values);
    XChangeGC(XtDisplay(ppocanvas), gc[0], GCForeground, &values);
    XDrawLine(XtDisplay(ppocanvas), XtWindow(ppocanvas),
        gc[0], zoommrtr, ytopmrtr, zoommrtr, ytopmrtr);

    /* FUNCTION Postproc_Get_Line_Value
    * Purpose
    * gets the appropriate output value with a sign indication
    * Inputs
    * real data - vector of output data
    * int stage - current timestep
    * Outputs
    * real value - output value, int switch_arrow - sign convention
    * Programming History
    * 1993 Sept, Gayle E. Ermer, Original function written
    */
void Postproc_Get_Line_Value(real data[], int stage, real *value,
    int *switch_arrow)
{
    real datavalue;
    datavalue = data[stage];
    if (datavalue < 0 ) {
        *switch_arrow = 1;
        *value = - datavalue;
    }
    else {
        *value = datavalue;
        *switch_arrow = 0;
    }
}

/* FUNCTION Postproc_Get_Value

```

```

    * Purpose
    * gets the appropriate output value
    * Inputs
    * real data - vector of output data, stage - stage of data needed
    * Outputs
    * real value - output value
    * Programming History
    * 1993 Oct, Gayle E. Ermer, Original function written
    */
void Postproc_Get_Value(real data[], int stage, real *value)
{
    real datavalue;
    datavalue = data[stage];
    *value = datavalue;
}

/* FUNCTION Postproc_Set_Colors
    * Purpose
    * Set up the colors used in the display
    * Inputs
    * None
    * Outputs
    * None
    * Programming History
    * 1993 Oct, Gayle E. Ermer, Original function written
    */
void Postproc_Set_Colors()
{
    extern Widget ppDialDialogDraw;
    int success[10];
    extern XColor Redcolor[1];
    extern XColor Greencolor[1];
    Display *ppDisplay;
    Colormap cmap;
    int i;
    int ppscreen;

    ppDisplay = XtDisplay(ppDialDialogDraw);
    ppscreen = DefaultScreen(ppDisplay);
    cmap = DefaultColormap(ppDisplay, ppscreen);

    Redcolor[0].green = 0;
    Redcolor[0].blue = 60000;
    Redcolor[0].red = 30000;
    success[0] = XAllocColor(ppDisplay, cmap, &Redcolor[0]);
    Redcolor[1].green = 0;
    Redcolor[1].blue = 60000;
    Redcolor[1].red = 40000;
    success[1] = XAllocColor(ppDisplay, cmap, &Redcolor[1]);
    Redcolor[2].green = 0;

```

```

Redcolor[2].blue = 5000;
Redcolor[2].red = 45000;
success[2] = XAllocColor (ppDisplay, cmap, &Redcolor[2]);
Redcolor[3].green = 0;
Redcolor[3].blue = 5000;
Redcolor[3].red = 5000;
success[3] = XAllocColor (ppDisplay, cmap, &Redcolor[3]);
Redcolor[4].green = 0;
Redcolor[4].blue = 5000;
Redcolor[4].red = 5000;
success[4] = XAllocColor (ppDisplay, cmap, &Redcolor[4]);
Redcolor[5].green = 0;
Redcolor[5].blue = 5000;
Redcolor[5].red = 6000;
success[5] = XAllocColor (ppDisplay, cmap, &Redcolor[5]);
Redcolor[6].green = 0;
Redcolor[6].blue = 4000;
Redcolor[6].red = 6500;
success[6] = XAllocColor (ppDisplay, cmap, &Redcolor[6]);
Redcolor[7].green = 2000;
Redcolor[7].blue = 1000;
Redcolor[7].red = 6500;
success[7] = XAllocColor (ppDisplay, cmap, &Redcolor[7]);
Redcolor[8].green = 3500;
Redcolor[8].blue = 1000;
Redcolor[8].red = 6500;
success[8] = XAllocColor (ppDisplay, cmap, &Redcolor[8]);
Redcolor[9].green = 4500;
Redcolor[9].blue = 1000;
Redcolor[9].red = 6500;
success[9] = XAllocColor (ppDisplay, cmap, &Redcolor[9]);
Greencolor[0].green = 2500;
Greencolor[0].blue = 4000;
Greencolor[0].red = 0;
success[0] = XAllocColor (ppDisplay, cmap, &Greencolor[0]);
Greencolor[1].green = 3200;
Greencolor[1].blue = 6200;
Greencolor[1].red = 0;
success[1] = XAllocColor (ppDisplay, cmap, &Greencolor[1]);
Greencolor[2].green = 4000;
Greencolor[2].blue = 5500;
Greencolor[2].red = 0;
success[2] = XAllocColor (ppDisplay, cmap, &Greencolor[2]);
Greencolor[3].green = 4700;
Greencolor[3].blue = 5500;
Greencolor[3].red = 0;
success[3] = XAllocColor (ppDisplay, cmap, &Greencolor[3]);
Greencolor[4].green = 5200;
Greencolor[4].blue = 5000;
Greencolor[4].red = 0;
success[4] = XAllocColor (ppDisplay, cmap, &Greencolor[4]);
Greencolor[5].green = 5200;
Greencolor[5].blue = 4200;
Greencolor[5].red = 0;
success[5] = XAllocColor (ppDisplay, cmap, &Greencolor[5]);
Greencolor[6].green = 5200;
Greencolor[6].blue = 2700;
Greencolor[6].red = 0;
success[6] = XAllocColor (ppDisplay, cmap, &Greencolor[6]);
Greencolor[7].green = 6000;
Greencolor[7].blue = 1000;
Greencolor[7].red = 1000;
success[7] = XAllocColor (ppDisplay, cmap, &Greencolor[7]);
Greencolor[8].green = 6000;

```

```

Greencolor[8].blue = 1000;
Greencolor[8].red = 1000;
success[8] = XAllocColor (ppDisplay, cmap, &Greencolor[8]);
Greencolor[9].green = 6500;
Greencolor[9].blue = 4000;
Greencolor[9].red = 2000;
success[9] = XAllocColor (ppDisplay, cmap, &Greencolor[9]);
}
/* FUNCTION Postproc Set OCs
 *
 * Purpose
 *   Initialises the OC to be used for the dynamic display
 *
 * Inputs
 *   None
 *
 * Outputs
 *   None
 *
 * Programming History
 *   1993 Aug, Gayle E. Ermer, Original function written
 *
 */
void Postproc Set OCs()
{
    extern Widget ppDialogDraw;
    Display *ppDisplay;
    extern OC gcAnimate;
    Window ppWin;
    int ppGreen;
    XGCValues initValues;

    ppDisplay = XtDisplay(ppDialogDraw);
    ppWin = DefaultRootWindow(ppDisplay);
    ppScreen = DefaultScreen(ppDisplay);

    initValues.foreground = 1;
    initValues.background = 0;
    initValues.lineWidth = 0;
    initValues.lineStyle = LineSolid;
    initValues.joinStyle = JoinMiter;

    gcAnimate = XCreateGC(ppDisplay, ppWin, (GCForeground | GCBackground
        | GCLineWidth |
        GCLineStyle | GCJoinStyle), &initValues);
}
/* FUNCTION Postproc Set OCs Key
 *
 * Purpose
 *   Initialises the OC for the key
 *
 * Inputs
 *   None
 *
 * Outputs
 *   None
 *
 * Programming History
 *   1993 Aug, Gayle E. Ermer, Original function written
 *
 */

```

```

/,
void Postproc Set OCs Key()
{
    extern Widget ppDdcCanvasKey;
    Display *ppDisplay;
    extern OC gchminated;
    Window ppWin;
    int ppScreen;
    XOCValues InitValues;

    ppDisplay = XDisplay(ppDdcCanvasKey);
    ppWin = DefaultRootWindow(ppDisplay);
    ppScreen = DefaultScreen(ppDisplay);

    InitValues.foreground = 1;
    InitValues.background = 0;
    InitValues.line_width = 0;
    InitValues.line_style = LineSolid;
    InitValues.join_style = JoinMiter;

    gchminated = XCreateOC(ppDisplay, ppWin, (OCForeground | OCBackground
        | OCLineWidth |
        | OCLineStyle | OCJoinStyle), &InitValues);
}

/* FUNCTION Pset.go
 * Purpose
 *   Determines which line width number should be used to draw the
 *   connector
 * Inputs
 *   value (the value of the variable)
 * Outputs
 *   gowidth (the width of the line)
 * Programming History
 *   1993 Aug, Gayle E. Ermer, Original function written
 */
unsigned Pset.go(double value)
{
    int gowidth;

    if (value < 0.1)
        gowidth = 1;
    else if (value < 0.2)
        gowidth = 2;
    else if (value < 0.3)
        gowidth = 3;
    else if (value < 0.4)
        gowidth = 4;
    else if (value < 0.5)
        gowidth = 5;
    else if (value < 0.6)
        gowidth = 6;
    else if (value < 0.7)
        gowidth = 7;
    else if (value < 0.8)
        gowidth = 8;
    else if (value < 0.9)
        gowidth = 9;
    else
        gowidth = 10;

    return gowidth;
}

/* FUNCTION Pset.go Log
 * Purpose
 *   Determines which line width number should be used to draw the
 *   connector for log scaling
 * Inputs
 *   value (the value of the variable)
 * Outputs
 *   gowidth (the width of the line)
 * Programming History
 *   1993 Dec, Gayle E. Ermer, Original function written
 */
unsigned Pset.go Log(double value, int switch arrow)
{
    int gowidth;
    double max, min, delta;
    extern Cur Var;
    extern real emax, emin,
        fmax, fmin,
        pmax, pmin,
        qmax, qmin,
        rmax, rmin,
        smax, smax;

    switch (Cur Var) {
        case EFFORT: max = emax; min = emin; break;
        case FLOW: max = fmax; min = fmin; break;
        case POWER: max = pmax; min = pmin; break;
        case DISPLACEMENT: max = qmax; min = qmin; break;
        case MOMENTUM: max = rmax; min = rmin; break;
        case SIGNAL: max = smax; min = smax; break;
        case UNKNOWN: max = rmax; min = rmin; break;
    }

    if (min <= 0.0) {
        /* Print Error Message */
        return 0;
    }

    delta = (log10(max) - log10(min)) / 10;

    if (log10(value) < delta * 1 + log10(min))
        gowidth = 1;
    else if (log10(value) < delta * 2 + log10(min))
        gowidth = 2;
    else if (log10(value) < delta * 3 + log10(min))
        gowidth = 3;
    else if (log10(value) < delta * 4 + log10(min))
        gowidth = 4;
    else if (log10(value) < delta * 5 + log10(min))
        gowidth = 5;
    else if (log10(value) < delta * 6 + log10(min))
        gowidth = 6;
    else if (log10(value) < delta * 7 + log10(min))
        gowidth = 7;
    else if (log10(value) < delta * 8 + log10(min))
        gowidth = 8;
    else if (log10(value) < delta * 9 + log10(min))
        gowidth = 9;
    else
        gowidth = 10;

    return gowidth;
}

```

```

        gowidth = 6;
    else if (log10(value) < delta * 7 + log10(min))
        gowidth = 7;
    else if (log10(value) < delta * 8 + log10(min))
        gowidth = 8;
    else if (log10(value) < delta * 9 + log10(min))
        gowidth = 9;
    else
        gowidth = 10;
    return gowidth;
}

/* FUNCTION Pset go_Abs
 *
 * Purpose
 *   Determines which line width number should be used to draw the
 *   commentor for abelog scaling
 *
 * Inputs
 *   value (the value of the variable)
 *
 * Outputs
 *   gowidth (the width of the line)
 *
 * Programming History
 *   1994 Jun, Gayle E. Ermar, Original function written
 */
unsigned Pset go_Abs(double value, int switch_arrow)
{
    int gowidth;
    double max, min, delta;
    extern Cur_Var;
    extern real emainrg, fmainrg, pmainrg, gmainrg, pmainrg, mainrg, emainrg;
    extern real orange, frange, prange, grange, parange, arange, srange;

    switch (Cur_Var) {
        case EFFORT: max = orange; min = emainrg; break;
        case FLOW: max = frange; min = fmainrg; break;
        case POWER: max = prange; min = pmainrg; break;
        case DISPLACEMENT: max = grange; min = gmainrg; break;
        case MOMENTUM: max = parange; min = pmainrg; break;
        case SLOWDOWN: max = arange; min = amainrg; break;
        case EMISSION: max = srange; min = smainrg; break;
    }

    if (min <= 0.0) {
        /* Print Error Message */
        return 0;
    }

    delta = (log10(max) - log10(min)) / 10;

    if (value == 0.0) value = min;

    if (log10(value) < delta * 1 + log10(min))
        gowidth = 1;
    else if (log10(value) < delta * 2 + log10(min))
        gowidth = 2;
    else if (log10(value) < delta * 3 + log10(min))
        gowidth = 3;
    else if (log10(value) < delta * 4 + log10(min))
        gowidth = 4;

```

```

    else if (log10(value) < delta * 5 + log10(min))
        gowidth = 5;
    else if (log10(value) < delta * 6 + log10(min))
        gowidth = 6;
    else if (log10(value) < delta * 7 + log10(min))
        gowidth = 7;
    else if (log10(value) < delta * 8 + log10(min))
        gowidth = 8;
    else if (log10(value) < delta * 9 + log10(min))
        gowidth = 9;
    else
        gowidth = 10;
    return gowidth;
}

/* FUNCTION Pset Color_Abs
 *
 * Purpose
 *   Get the abelog color to display for a given output value
 *
 * Inputs
 *   value - the output value, switch_arrow - sign indication
 *
 * Outputs
 *   Color number
 *
 * Programming History
 *   1994 Jun, Gayle E. Ermar, Original function written
 */
int Pset Color_Abs(double value, int switch_arrow)
{
    int pcolor;
    extern IColor Redcolor[];
    extern IColor Greencolor[];
    extern Widget PPDDisDialogDraw;

    double max, min, delta;
    extern Cur_Var2;
    extern real orange, frange, prange, grange, parange, arange, srange;
    extern real emainrg, fmainrg, pmainrg, gmainrg, pmainrg, mainrg, emainrg;

    switch (Cur_Var2) {
        case EFFORT: max = orange; min = emainrg; break;
        case FLOW: max = frange; min = fmainrg; break;
        case POWER: max = prange; min = pmainrg; break;
        case DISPLACEMENT: max = grange; min = gmainrg; break;
        case MOMENTUM: max = parange; min = pmainrg; break;
        case SLOWDOWN: max = arange; min = amainrg; break;
        case EMISSION: max = srange; min = smainrg; break;
    }

    if (min <= 0.0) {
        /* Print Error Message */
        return BlackPixel(XtDisplay(ppDDisDialogDraw),
            DefaultScreen(XtDisplay(ppDDisDialogDraw)));
    }

    if (value == 0.0) value = min;

    delta = (log10(max) - log10(min)) / 10;

    if (log10(value) < delta * 1 + log10(min)) {

```

```

switch (switch_arrow) {
    case 0:
        gccolor = Redcolor[0].p1mal, break;
    case 1:
        gccolor = Greencolor[0].p1mal, break;
}
}
else if (log10(value) < delta * 2 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[1].p1mal, break;
        case 1:
            gccolor = Greencolor[1].p1mal, break;
    }
}
else if (log10(value) < delta * 3 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[2].p1mal, break;
        case 1:
            gccolor = Greencolor[2].p1mal, break;
    }
}
else if (log10(value) < delta * 4 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[3].p1mal, break;
        case 1:
            gccolor = Greencolor[3].p1mal, break;
    }
}
else if (log10(value) < delta * 5 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[4].p1mal, break;
        case 1:
            gccolor = Greencolor[4].p1mal, break;
    }
}
else if (log10(value) < delta * 6 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[5].p1mal, break;
        case 1:
            gccolor = Greencolor[5].p1mal, break;
    }
}
else if (log10(value) < delta * 7 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[6].p1mal, break;
        case 1:
            gccolor = Greencolor[6].p1mal, break;
    }
}
else if (log10(value) < delta * 8 + log10(min)) {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[7].p1mal, break;
        case 1:
            gccolor = Greencolor[7].p1mal, break;
    }
}
else if (log10(value) < delta * 9 + log10(min)) {
    switch (switch_arrow) {

```

```

        case 0:
            gccolor = Redcolor[8].p1mal, break;
        case 1:
            gccolor = Greencolor[8].p1mal, break;
    }
}
else {
    switch (switch_arrow) {
        case 0:
            gccolor = Redcolor[9].p1mal, break;
        case 1:
            gccolor = Greencolor[9].p1mal, break;
    }
}
return gccolor;
}
/* FUNCTION Pset Color_Log
 * Purpose
 * Get the log color to display for a given output value
 * Inputs
 * value - the output value, switch_arrows - sign indication
 * Outputs
 * Color number
 * Programming History
 * 1993 Don, Gayle E. Ermer, Original function written
 */
int Pset_Color_Log(double value, int switch_arrows)
{
    int gccolor;
    extern XColor Redcolor[];
    extern XColor Greencolor[];
    extern Widget ppDialDialogDraw;
    double max, min, delta;
    extern Our_Var2;
    extern real emax, emin,
        fmax, fmin,
        gmax, gmin,
        pmax, pmin,
        nmax, nmin,
        smax, smin;
    switch (Our_Var2) {
        case EFFORT; max = emax; min = emin; break;
        case FLOW; max = fmax; min = fmin; break;
        case POWER; max = pmax; min = pmin; break;
        case DISPLACEMENT; max = gmax; min = gmin; break;
        case MOMENTUM; max = mmax; min = pmin; break;
        case SIGNAL; max = smax; min = smin; break;
        case ENERGY; max = nmax; min = nmin; break;
    }
    if (min <= 0.0) {
        /* Print Error Message */
        return BlackPixel(StdDisplay(ppDialDialogDraw),
            DefaultScreen(StdDisplay(ppDialDialogDraw)));
    }

```

```

    )
    delta = (log10(max) - log10(min)) / 10;
    if (log10(value) < delta * 1 + log10(min)) {
        gccolor = Redcolor[0].pixel;
    }
    else if (log10(value) < delta * 2 + log10(min)) {
        gccolor = Redcolor[1].pixel;
    }
    else if (log10(value) < delta * 3 + log10(min)) {
        gccolor = Redcolor[2].pixel;
    }
    else if (log10(value) < delta * 4 + log10(min)) {
        gccolor = Redcolor[3].pixel;
    }
    else if (log10(value) < delta * 5 + log10(min)) {
        gccolor = Redcolor[4].pixel;
    }
    else if (log10(value) < delta * 6 + log10(min)) {
        gccolor = Redcolor[5].pixel;
    }
    else if (log10(value) < delta * 7 + log10(min)) {
        gccolor = Redcolor[6].pixel;
    }
    else if (log10(value) < delta * 8 + log10(min)) {
        gccolor = Redcolor[7].pixel;
    }
    else if (log10(value) < delta * 9 + log10(min)) {
        gccolor = Redcolor[8].pixel;
    }
    else {
        gccolor = Redcolor[9].pixel;
    }
    return gccolor;
}

/* FUNCTION Post_Color
 *
 * Purpose
 *   Get the appropriate color to display for a given output value
 *
 * Inputs
 *   value - the output value, switch_arrow - sign indication
 *
 * Outputs
 *   Color number
 *
 * Programming History
 *   1993 Oct, Gayle E. Ermer, Original function written
 *
 */

int Post_Color(double value, int switch_arrow)
{
    int gccolor;
    extern XColor Redcolor[];
    extern XColor Greencolor[];

    if (value < 0.1) {
        if (switch_arrow == 1) gccolor = Greencolor[0].pixel;
        if (switch_arrow == 0) gccolor = Redcolor[0].pixel;
    }
    else if (value < 0.2) {

```

dyndis.c Tue Nov 8 08:46:57 1994 36

```

/* FUNCTION Postproc_load_it
 * Purpose
 *   Loads the export model file
 * Inputs
 *   filename of the file to be opened
 * Outputs
 *   none
 * Programming History
 *   1993 July, Gayle E. Ermer, Function adapted to dyndis file
 */

void Postproc_load_it(char *fname)
{
    extern Widget SelectShell;
    extern Widget ppdiDialogDraw;
    extern int Init;
    extern ModeType *First_Mode;
    extern ConnectorType *First_Connector;
    extern int grid_snap;

    int i, ok;

    /* Read in the model file */
    strcpy(filename, fname);
    ok = ReadFile(filename);

    /* Put up the graph */
    if(ok) {
        Postproc_frame();
        Postproc_draw_graph(ppdiDialogDraw);
    }
    else {
        Problem_Pop("Error reading file.");
    }

    /* Sort and scale the data */
    Postproc_Arrange_Data(ppdiDialogDraw);
    Postproc_Signals(ppdiDialogDraw);
    Postproc_Integrate_Data(ppdiDialogDraw);
    Postproc_Find_Maxes_Minus(ppdiDialogDraw);
    Init_energy_array();

    return;
}

/* FUNCTION Postproc_Init_Graph
 * Purpose
 *   Initializes the graph animation window
 * Inputs
 *   form (form widget that governs the viewport)
 * Outputs
 *   none

```

```

 * Programming History
 *   1993 July, Gayle E. Ermer, Function adapted to dyndis file
 */
Widget Postproc_Init_Graph(form)
Widget form;
{
    /* Set up the viewport widget */
    ppdiDialogView = ItVaCreateManagedWidget("canvas viewport",
        viewportWidgetClass, form, NULL);

    /* Set up the canvas widget */
    Postproc_Init_Canvas(ppdiDialogView);
    return ppdiDialogView;
}

/* FUNCTION Postproc_Init_Graph_Key
 * Purpose
 *   Initializes the graph window for key
 * Inputs
 *   form (form widget that governs the viewport)
 * Outputs
 *   none
 * Programming History
 *   1993 Nov, Gayle E. Ermer, Function adapted to dyndis file
 */
Widget Postproc_Init_Graph_Key(form)
Widget form;
{
    /* Set up the viewport widget */
    ppdiDialogKey = ItVaCreateManagedWidget("canvas viewport",
        viewportWidgetClass, form, NULL);

    /* Set up the canvas widget */
    Postproc_Init_Canvas_Key(ppdiDialogKey);
    return ppdiDialogKey;
}

/* FUNCTION Postproc_Init_Canvas
 * Purpose
 *   Initializes the graph canvas
 * Inputs
 *   ppviewport (postprocessor viewport widget)
 * Outputs
 *   none
 * Programming History

```


dyndis.c Tue Nov 8 08:46:57 1994 37

```

 * 1993 July, Gayle E. Ermer, Function adapted to dyndis file
 */
Widget Postproc Init Canvas(Widget pviewport)
{
    int sn;
    Display *dpy;
    Window win;

    true_canvas_width = 16000;
    true_canvas_height = 16000;

    /* Set up the canvas */
    ppddisDialogDraw = XtCreateManagedWidget("canvas",
        canvasWidgetClass, pviewport,
        XtWidth, true_canvas_width,
        XtHeight, true_canvas_height, NULL);

    XtAddCallback(ppddisDialogDraw, XmExposeCallback, Phandle_expose, NULL);
    XtAddCallback(ppddisDialogDraw, XmButtonDragCallback, bi_drag, NULL);
    XtAddCallback(ppddisDialogDraw, XmButtonDownCallback, b1_select, NULL);
    XtAddCallback(ppddisDialogDraw, XmButtonDownCallback, b2_select, NULL);

    dpy = XtDisplay(ppddisDialogDraw);
    win = DefaultRootWindow(dpy);
    sm = DefaultScreen(dpy);

    /* Get the fonts and colors used */
    get_fonts(dpy);
    get_colors(ppddisCanvasKey, dpy, win, sm);

    return ppddisCanvasKey;
}

/* FUNCTION Postproc_frame
 * Purpose
 *   Frame the graph
 * Inputs
 *   none
 * Outputs
 *   none
 * Programming History
 *   1993 July, Gayle E. Ermer, Function adapted from export
 */
void Postproc_frame()
{
    int graph_width, graph_height;
    int minx, maxx, miny, maxy;

    /* Center the graph */
    Postproc_graph(First Mode, First Connector, &graph_width, &graph_height);

    /* Center the scrollbars */
    Postproc_center_scrollbars();

    /* Clear the canvas */
    Postproc_clear_canvas((pcanvas_width() - pviewport_width()) / 2,
        (pcanvas_height() - pviewport_height()) / 2,
        pviewport_width(), pviewport_height());

    /* Set the scale and zoom factors */
    scale_factor = min(
        min((double) pviewport_width() / (double) graph_width,
            (double) pviewport_height() / (double) graph_height),
        1.0);
    scale_factor *= 0.8;
}

```

```

zoom_factor = 1.0;

/* if (grid snap) snap all points to grid(); */

/* FUNCTION Postproc center scrollbars
 *
 * Purpose
 * Centers the scrollbars of the viewport
 *
 * Inputs
 * none
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */
void Postproc center scrollbars()
{
    XawViewportSetCoordinates(ppDialogView,
        (Pcanvas width()-Pviewport width())/2,
        (Pcanvas height()-Pviewport height())/2);
}

/* FUNCTION Pviewport width
 *
 * Purpose
 * Gets the width of the viewport
 *
 * Inputs
 * none
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */
Dimension Pviewport width()
{
    Dimension width;
    XtVaGetValues(ppDialogView, Xthwidth, &width, NULL);
    return width;
}

/* FUNCTION Pviewport height
 *
 * Purpose
 * Gets the height of the viewport
 *
 * Inputs
 * none
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */
Dimension Pviewport height()
{
    Dimension height;
    XtVaGetValues(ppDialogView, Xthheight, &height, NULL);
    return height;
}

/* FUNCTION Pcanvas width
 *
 * Purpose
 * Gets the width of the canvas
 *
 * Inputs
 * none
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */
unsigned Pcanvas width()
{
    return true canvas width;
}

/* FUNCTION Pcanvas height
 *
 * Purpose
 * Gets the height of the canvas
 *
 * Inputs
 * none
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */
unsigned Pcanvas height()
{
    return true canvas height;
}

/* FUNCTION Postproc clear canvas
 *
 * Purpose
 * Clears the canvas area
 *
 * Inputs
 * x, y, width, height of canvas
 *
 * Outputs
 * none
 *
 * Programming History
 * 1993 July, Gayle E. Ermer, Function adapted from export
 */

```

dyndis.c Tue Nov 8 08:46:57 1994 39

```

*      1993 July, Gayle E. Ermar, Function adapted from export
*
void Postproc_clear_canvas(int x, int y, int width, int height)
{
    XClearArea(XtDisplay(ppCanvas), XtWindow(ppCanvas),
               x, y, width, height, False);
}

/* FUNCTION Postproc_draw_graph
* Purpose
* Draws the graph
* Inputs
* ppCanvas (canvas to draw the graph on)
* Outputs
* none
* Programming History
*      1993 Apr, Mark M. Lacey, Original function written
*      1993 July, Gayle E. Ermar, Function transposed to dyndis file
*
void Postproc_draw_graph(Widget ppCanvas)
{
    Dimension w, h;
    int x, y;

    XtGetValues(ppCanvas, XtWidth, w, XtHeight, h, NULL);
    Postproc_clear_canvas(0, 0, w, h);
    if (grid_visible) Pdraw_grid(ppCanvas, 0, 0, w, h);
    Phandle_expose(ppCanvas, NULL, NULL);
}

/* FUNCTION Phandle_expose
* Purpose
* Redraws the graph whenever it is exposed
* Inputs
* ppCanvas (canvas to draw the graph on)
* XtPointer client_data (not used)
* ExposedStruct *e (expose signal)
* Outputs
* none
* Programming History
*      1993 Apr, Mark M. Lacey, Original function written
*      1993 July, Gayle E. Ermar, Function transposed to dyndis file
*
void Phandle_expose(Widget ppCanvas, XtPointer client_data, ExposedStruct *e)
{
    NodeType *traverseNodes;
    ConnectorType *traverseCons;
    XColor values;
    int i, x1, y1, x2, y2;
    XRectangle rect;

    /* change our gc's to use the connector font */
    values.font = font[Connector]--fid;
    for (i = 0; i < GRIDCOLOR+1; ++i)
        XChangeGC(XtDisplay(ppCanvas), gc[i], GCFont, &values);

    /* go through and draw the nodes */
    traverseNodes = FirstNode->Next;
    while (traverseNodes) {
        if (traverseNodes->visible==VISIBLE)
            if (! (i % 11) NodeInRegion(traverseNodes, e->region))
                if (traverseNodes->class == HIGHLIGHT)
                    Pdraw_node(ppCanvas, traverseNodes, gc[HIGHLIGHT]);
                else {
                    Pdraw_node(ppCanvas, traverseNodes, gc[traverseNodes->power]);
                    traverseNodes = traverseNodes->Next;
                }
            }
        }

    /* change our gc's to use the connector font */
    values.font = font[Connector]--fid;
    for (i = 0; i < GRIDCOLOR+1; ++i)
        XChangeGC(XtDisplay(ppCanvas), gc[i], GCFont, &values);

    traverseCons = FirstConnector->Next;
    while (traverseCons) {
        if (traverseCons->visible==VISIBLE)
            if (traverseCons->class == HIGHLIGHT)
                Pdraw_connector(ppCanvas, traverseCons, gc[HIGHLIGHT]);
            else
                Pdraw_connector(ppCanvas, traverseCons, gc[traverseCons->power]);
            traverseCons = traverseCons->Next;
        }
    }

/* FUNCTION Pcenter_graph
* Purpose
* Centers the graph to (0, 0), and returns the width and height.
* Inputs
* *nodes (NodeType structure)
* *connectors (ConnectorType structure)
* *width (viewport width)
* *height (viewport height)
* Outputs
* none
* Programming History
*      1993 Apr, Mark M. Lacey, Original function written
*      1993 July, Gayle E. Ermar, Function transposed to dyndis file
*

```

```

/*
void Pcenter_graph(NodeType *nodes, ConnectorType *connectors, int
    "width, int "height)
{
    int minx,
        miny,
        maxx,
        maxy,
        center_x,
        center_y;
    NodeType *traverseNodes;
    ConnectorType *traverseCons;
    int Point Type *traverseIP;

    /* for each coordinate in the database, find the minimum x/y, and */
    Pfind_graph_extant(nodes, connectors, &minx, &miny, &maxx, &maxy);

    /* find the width, height of the current graph extant */
    "width = maxx - minx;
    "height = maxy - miny;

    /* find the center of the current graph */
    center_x = minx + "width/2;
    center_y = miny + "height/2;

    /* for each coordinate in the database, translate its coordinates
    * newx = oldx - center_x;
    * newy = oldy - center_y;
    */
    traverseNodes = nodes->Next;
    while (traverseNodes) {
        traverseNodes->x = traverseNodes->x - center_x;
        traverseNodes->y = traverseNodes->y - center_y;
        traverseNodes = traverseNodes->Next;
    }

    traverseCons = connectors->Next;
    while (traverseCons) {
        traverseIP = traverseCons->Interior_points->Next;
        while (traverseIP) {
            traverseIP->x = traverseIP->x - center_x;
            traverseIP->y = traverseIP->y - center_y;
            traverseIP = traverseIP->Next;
        }
        traverseCons = traverseCons->Next;
    }

    /* FUNCTION Pdraw_node
    * Purpose
    * draws the node text, centered at its (x,y) coordinates
    * Inputs
    * canvas (canvas widget on which nodes are to be drawn)
    * "node (NodeType structure to draw)
    * groon (graphics context)
    * Outputs
    * none
    * Programming History
    * 1993 Apr, Mark M. Lacey, Original function written
    * 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file
    */
}

```

```

/*
1993 July, Gayle E. Ermer, Function transposed to dyndis file
*/
void Pfind_graph_extant(NodeType *nodes, ConnectorType *connectors, int
    "minx, int "miny, int "maxx, int "maxy)
{
    NodeType *traverseNodes;
    ConnectorType *traverseCons;
    int Point Type *traverseIP;
    traverseNodes = nodes->Next;
    if (!traverseNodes) {
        "minx = 0;
        "miny = 0;
        "maxx = 0;
        "maxy = 0;
        return;
    }
    "minx = traverseNodes->x;
    "miny = traverseNodes->y;
    "maxx = traverseNodes->x;
    "maxy = traverseNodes->y;
    while (traverseNodes) {
        if (traverseNodes->x < "minx) "minx = traverseNodes->x;
        if (traverseNodes->y < "miny) "miny = traverseNodes->y;
        if (traverseNodes->x > "maxx) "maxx = traverseNodes->x;
        if (traverseNodes->y > "maxy) "maxy = traverseNodes->y;
        traverseNodes = traverseNodes->Next;
    }
    traverseCons = connectors->Next;
    while (traverseCons) {
        traverseIP = traverseCons->Interior_points->Next;
        while (traverseIP) {
            if (traverseIP->x < "minx) "minx = traverseIP->x;
            if (traverseIP->y < "miny) "miny = traverseIP->y;
            if (traverseIP->x > "maxx) "maxx = traverseIP->x;
            if (traverseIP->y > "maxy) "maxy = traverseIP->y;
            traverseIP = traverseIP->Next;
        }
        traverseCons = traverseCons->Next;
    }

    /* FUNCTION Pdraw_node
    * Purpose
    * draws the node text, centered at its (x,y) coordinates
    * Inputs
    * canvas (canvas widget on which nodes are to be drawn)
    * "node (NodeType structure to draw)
    * groon (graphics context)
    * Outputs
    * none
    * Programming History
    * 1993 Apr, Mark M. Lacey, Original function written
    * 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file
    */
}

```

dyndis.c Tue Nov 8 08:46:57 1994 41

```

    *
    */
void PDraw_node(Widget canvas, NodeType *node, OC green)
{
    int
    align,
    x,
    y,
    x2,
    y2,
    width,
    height;
    char *name;

    Pnode_node_coords(node, &x, &y, &x2, &y2, &width, &height);

    /* get the node name, and length */
    name = node->name;
    align = strlen(name);

    /* it might be a little wasteful to do this here, but that's */
    /* life...set the font to the current node font */
    XSetFont(XtDisplay(canvas), green, Font[ModeFont]->fid);

    /* draw a string for the node */
    XDrawString(XtDisplay(canvas), XtWindow(canvas),
               green, x, y, name, align);

    if (node->bortype==BOX) {
        XDrawRectangle(XtDisplay(canvas), XtWindow(canvas),
                     green, x2, y2, width, height);
    }

    /* FUNCTION Pnode node_coords
    * Purpose
    * calculate the coordinate to draw the node string at (x, y),
    * along with the coordinates and width of a box that would surround it
    * (xbr, ybr, xbr, hbr)
    * Inputs
    * *node (NodeType structure to draw)
    * Outputs
    * x, y, xbr, ybr, wr, hr (coordinates to use for drawing)
    * Programming History
    * 1993 Apr, Mark M. Lacey, Original function written
    * 1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
    */
void Pnode_node_coords(NodeType *node, int *x, int *y, int *xbr, int
                        *ybr, int *wr, int *hr)
{
    int
    g1, g2,
    dir,
    align,
    x,
    y,
    width,
    height,
    ascent,
    descent;
    XCharStruct
    overall;
    char *name;

    /* get the coordinates of the node */
    x = node->x;
    y = node->y;

    Pdb_to_screen(&x, &y);

    /* get the node name, and length */
    name = node->name;
    align = strlen(name);

    XTextExtents(Font[ModeFont], name, align, &dir, &g1, &g2, &overall);
    ascent = overall.ascent;
    descent = overall.descent;
    width = overall.width;
    height = ascent+descent;

    *x = x-width/2;
    *y = y+height/2;

    *xbr = x-width/2-BORDER;
    *ybr = y+height/2+BORDER;
    *wr = width+2*BORDER;
    *hr = height+2*BORDER;
}

/* FUNCTION Pdb to screen
* Purpose
* Converts database coordinates to screen coordinates by scaling,
* scaling, and then placing a point.
* Inputs
* none
* Outputs
* x, y (coordinates)
* Programming History
* 1993 Apr, Mark M. Lacey, Original function written
* 1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
*/
void Pdb_to_screen(int *x, int *y)
{
    Proom(x, y);
    Pnode(x, y);
    Pplace(x, y);
}

/* FUNCTION Proom
* Purpose
* Multiplies the coordinates of a point by the current room factor.
* Inputs
* none
* Outputs

```

```

    * x, y (coordinates)
    *
    * Programming History
    *   1993 Apr, Mark M. Lacey, Original function written
    *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
    */

void Pzoom(int *x, int *y)
{
    *x = *x * zoom.factor;
    *y = *y * zoom.factor;
}

/* FUNCTION Pscale
 *
 * Purpose
 *   Multiplies the coordinates of a point by the current scale factor.
 *   The scale factor is determined when we frame the graph. It can
 *   never be greater than 1.
 *
 * Inputs
 *   none
 *
 * Outputs
 *   x, y (coordinates)
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 */

void Pscale(int *x, int *y)
{
    *x = *x * scale.factor;
    *y = *y * scale.factor;
}

/* FUNCTION Pplace
 *
 * Purpose
 *   Places a point into the canvas, relative to center. In other
 *   words, if a canvas is 1000x1000 in size, the coordinate (10, 10)
 *   would be placed at (510, 490) (490 since we have to negate the y to
 *   flip it --- the canvas has a different coordinate system than our
 *   graph.)
 *
 * Inputs
 *   none
 *
 * Outputs
 *   x, y (coordinates)
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 */

void Pplace(int *x, int *y)
{
    *x = *x + Pcanvas.width()/2;
    *y = -*y + Pcanvas.height()/2; /* -y since we need to flip graph */
}

```

```

/* FUNCTION Pdraw grid
 *
 * Purpose
 *   Draws the grid
 *
 * Inputs
 *   canvas (canvas widget to draw on)
 *   startx, starty, endx, endy (starting and ending coordinates)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 */

void Pdraw grid(Widget canvas, int startx, int starty, int endx, int endy)
{
    int x, y;

    Penup to grid(startx, starty);
    Pset to grid(endx, endy);

    for (x = startx; x <= endx; x += grid.size)
        XDrawLine(XtDisplay(canvas), XtWindow(canvas), gc(GRIDCOLOR),
                x, starty, x, endy);
    for (y = starty; y <= endy; y += grid.size)
        XDrawLine(XtDisplay(canvas), XtWindow(canvas), gc(GRIDCOLOR),
                startx, y, endx, y);
}

/* FUNCTION Pnode.in_region
 *
 * Purpose
 *   returns 1 if some portion of the node is in the region --- 0
 *   otherwise
 *
 * Inputs
 *   *node (Node type structure)
 *   region (Region structure are to check)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 */

int Pnode.in_region(NodeType *node, Region region)
{
    int x,
        y,
        dx,
        dy,
        height,
        width;

    Pcalc node.coords(&node, &x, &y, &dx, &dy, &width, &height);
}

```

```

dyndis.c      Tue Nov 8 08:46:57 1994      43

/* determine if it is in the region, and return appropriate value */
return !RectInRegion(region, x1, y1, width,
    height);
}

/* FUNCTION PDraw connector
 *
 * Purpose
 *   draws the connectors, including arrows and causality
 *
 * Inputs
 *   canvas (canvas widget on which nodes are to be drawn)
 *   *connector (ConnectorType structure to draw)
 *   gcrn (graphics context)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 Mar, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Erner, Function transposed to dyndis file
 *
 */

void PDraw connector(Widget canvas, ConnectorType *connector, OC gcrn,
    int switch_arrow)
{
    int Point_Type *ip;
    int
        x1,
        y1,
        x2,
        y2,
        labelx, labely;
    ModeType M;
    extern OC go[];

    /* it might be a little wasteful to do this here, but oh */
    /* well...set the font to the current connector font */
    MacFont(XtDisplay(canvas), gcrn, font[Connector]-fid);

    /* ship first interior point */
    ip = connector->interior_point;
    if (ip) ip = ip->next;

    if (ip == NULL) {
        if (connector->to == NULL) return;
        PDraw_line(connector->from, connector->to, x1, y1, x2, y2);
        PDraw_label_coords(x1, y1, x2, y2, labelx, labely);
        XDrawString(XtDisplay(canvas), XtWindow(canvas),
            gcrn, labelx, labely, connector->name,
            strlen(connector->name));
        XDrawLine(XtDisplay(canvas), XtWindow(canvas),
            gcrn, x1, y1, x2, y2);
        if (show_causality && connector->causemode == CAUSE_ON_FROM)
            PDraw_causality(canvas, x1, y1, x2, y2, gcrn);
    } else {
        /* call, but just IGNORE the return x2 & y2 --- we aren't */
        /* going from node to node, so we are just going to use the */
        /* (x,y) from the interior point for our (x,y) */
        M.x = ip->x1;
        M.y = ip->y1;
    }
}

```

```

M.name = "a";
PDraw_line(connector->from, x1, y1, x2, y2);
x2 = ip->x1;
y2 = ip->y1;
PDraw_to_screen(x2, y2);

PDraw_label_coords(x1, y1, x2, y2, labelx, labely);
XDrawString(XtDisplay(canvas), XtWindow(canvas),
    gcrn, labelx, labely, connector->name,
    strlen(connector->name));
XDrawLine(XtDisplay(canvas), XtWindow(canvas),
    gcrn, x1, y1, x2, y2);

if (show_causality && connector->causemode == CAUSE_ON_FROM)
    PDraw_causality(canvas, x1, y1, x2, y2, gcrn);

while(ip && ip->next) {
    x1 = ip->x1;
    y1 = ip->y1;
    x2 = ip->next->x1;
    y2 = ip->next->y1;
    PDraw_to_screen(x1, y1);
    PDraw_to_screen(x2, y2);

    PDraw_label_coords(x1, y1, x2, y2, labelx, labely);
    XDrawString(XtDisplay(canvas), XtWindow(canvas),
        gcrn, labelx, labely, connector->name,
        strlen(connector->name));
    XDrawLine(XtDisplay(canvas), XtWindow(canvas),
        gcrn, x1, y1, x2, y2);
    ip = ip->next;
}

if (connector->to == NULL) return;
/* as above, we will ignore one of the sets of coords --- this */
/* is time the x1 & y1 */
M.x = ip->x1;
M.y = ip->y1;
M.name = "a";
PDraw_line(x1, y1, connector->to, x2, y2);
x2 = ip->x1;
y2 = ip->y1;
PDraw_to_screen(x1, y1);

PDraw_label_coords(x1, y1, x2, y2, labelx, labely);
XDrawString(XtDisplay(canvas), XtWindow(canvas),
    gcrn, labelx, labely, connector->name,
    strlen(connector->name));
XDrawLine(XtDisplay(canvas), XtWindow(canvas),
    gcrn, x1, y1, x2, y2);
}

/* at this point (x1, y1) & (x2, y2) are the last pair of */
/* coordinates --- they will be used in determining the direction */
/* of the arrow-head(s) */
if (switch_arrow == 1) {
    PDraw_arrow_head(canvas, x2, y2, x1, y1, connector->type, gcrn);
}
}

```

```

else {
    Pdraw_arrow_head(canvas, x1, y1, x2, y2, connector--type, green);
}
if (show_causality && connector--causemode == CAUSE ON TO)
    Pdraw_causality(canvas, x2, y2, x1, y1, green);
}

/* FUNCTION Pdraw_arrow_head
 *
 * Purpose
 *   draws the arrows
 *
 * Inputs
 *   canvas (canvas widget on which nodes are to be drawn)
 *   x1, y1, x2, y2, type (coordinates)
 *   green (graphics context)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 *
 */

void Pdraw_arrow_head(Window canvas, int x1, int y1, int x2, int y2, int
type, OC green)
{
    double dx, dy, slope, angle0, angle1, angle2;
    int arrow_x1, arrow_y1, arrow_x2, arrow_y2;

    dy = y2 - y1;
    dx = x2 - x1;

    if (fabs(dx) != 0.0) {
        slope = dy/dx;
        angle0 = atan(slope);
        if (dx < 0.0)
            angle0 = flip_across_y(angle0);
        else
            if (fabs(angle0) != 0.0)
                angle0 = flip_across_x(angle0);
    } else {
        if (dy > 0) angle0 = -M_PI/2.0;
        else angle0 = M_PI/2.0;
    }

    angle1 = angle0 - M_PI/2.0;
    angle2 = angle0 + M_PI/2.0;

    arrow_x1 = CAUSE_LHM * cos(angle1);
    arrow_y1 = -CAUSE_LHM * sin(angle1);
    arrow_x2 = CAUSE_LHM * cos(angle2);
    arrow_y2 = -CAUSE_LHM * sin(angle2);

    XDrawLine(XtDisplay(canvas), XtWindow(canvas),
green, x1-arrow_x1, y1-arrow_y1, x1-arrow_x2, y1-arrow_y2);
}

/* FUNCTION Pcalc_line
 *
 * Purpose
 *   calculates the positions of the line endpoints
 *
 * Inputs
 *   "n1, "n2 (two ModeType structures)
 *
 * Outputs
 *   x1, y1, x2, y2, type (coordinates)
 *
 * Programming History

```

```

/* FUNCTION Pdraw_causality
 *
 * Purpose
 *   draws the causality
 *
 * Inputs
 *   canvas (canvas widget on which nodes are to be drawn)
 *   x1, y1, x2, y2, type (coordinates)
 *   green (graphics context)
 *
 * Outputs
 *   none
 *
 * Programming History
 *   1993 Apr, Mark M. Lacey, Original function written
 *   1993 Aug, Gayle E. Ermar, Function transposed to dyndis file
 *
 */

void Pdraw_causality(Window canvas, int x1, int y1, int x2, int y2, OC green)
{
    double dx, dy, slope, angle0, angle1, angle2;
    int arrow_x1, arrow_y1, arrow_x2, arrow_y2;

    dy = y2 - y1;
    dx = x2 - x1;

    if (fabs(dx) != 0.0) {
        slope = dy/dx;
        angle0 = atan(slope);
        if (dx < 0.0)
            angle0 = flip_across_y(angle0);
        else
            if (fabs(angle0) != 0.0)
                angle0 = flip_across_x(angle0);
    } else {
        if (dy > 0) angle0 = -M_PI/2.0;
        else angle0 = M_PI/2.0;
    }

    angle1 = angle0 - M_PI/2.0;
    angle2 = angle0 + M_PI/2.0;

    arrow_x1 = CAUSE_LHM * cos(angle1);
    arrow_y1 = -CAUSE_LHM * sin(angle1);
    arrow_x2 = CAUSE_LHM * cos(angle2);
    arrow_y2 = -CAUSE_LHM * sin(angle2);

    XDrawLine(XtDisplay(canvas), XtWindow(canvas),
green, x1-arrow_x1, y1-arrow_y1, x1-arrow_x2, y1-arrow_y2);
}

/* FUNCTION Pcalc_line
 *
 * Purpose
 *   calculates the positions of the line endpoints
 *
 * Inputs
 *   "n1, "n2 (two ModeType structures)
 *
 * Outputs
 *   x1, y1, x2, y2, type (coordinates)
 *
 * Programming History

```



```

* 1993 Apr, Mark M. Lacey, Original function written
* 1993 Aug, Gayle M. Ermar, Function transposed to dyndis file
*/

void Pnode_line(NodeType *n1, NodeType *n2, int *x1, int *y1, int *x2,
               int *y2)
{
    int dxmmy;
    int x1b,
        y1b,
        widthb2,
        heightb2,
        x1b,
        y1b,
        widthb1,
        heightb1;
    double delx, dely;
    double line_slope, box_slope;
    double dx, dy;

    *x1 = n1->x;
    *y1 = n1->y;
    *x2 = n2->x;
    *y2 = n2->y;
    Pdb_to_screen(x1, y1);
    Pdb_to_screen(x2, y2);

    /* calculate new x, y for from box */
    Pnode_node_coords(n1, dxmmy, edummy, ax1b, ay1b, axwidth1, theight1);
    /* calculate new x, y for to box */
    Pnode_node_coords(n2, dxmmy, edummy, ax2b, ay2b, axwidth2, theight2);
    /* if the x's are the same, then the line goes through the TOP or */
    /* the bottom of the box */
    if (*x2 - *x1 == 0) {
        /* bottom of "from" box, top of "to" box */
        *y1 = y1b + height1 + 1;
        *y2 = y2b - 1;
    }
    else {
        /* top of "from" box, bottom of "to" box */
        *y1 = y1b - 1;
        *y2 = y2b + height2 + 1;
    }

    }
    else {
        dy = (double) *y2 - (double) *y1;
        dx = (double) *x2 - (double) *x1;
        line_slope = dy/dx;
        box_slope = ((double) height1/ (double) width1);
        if (line_slope > box_slope || -line_slope > box_slope) {
            /* bottom or else, top */
            if (dy > 0) {
                *y1 = y1b + height1 + 1;
                delx = ((height1/2.0) + 1) / line_slope;
                *x1 = *x1 + delx;
            }
            else {

```

```

                *y1 = y1b - 1;
                delx = ((height1/2.0) + 1) / line_slope;
                *x1 = *x1 - delx;
            }
        }
        else {
            /* right, or else, left */
            if (dx > 0) {
                *x1 = x1b + width1 + 1;
                dely = ((width1/2.0) + 1) * line_slope;
                *y1 = *y1 + dely;
            }
            else {
                *x1 = x1b - 1;
                dely = ((width1/2.0) + 1) * line_slope;
                *y1 = *y1 - dely;
            }
        }
    }

    /* calculate new x, y for to box */
    box_slope = ((double) height2/ (double) width2);
    if (line_slope > box_slope || -line_slope > box_slope) {
        /* bottom, or else, top */
        if (dy < 0) {
            *y2 = y2b + height2 + 1;
            delx = ((height2/2.0) + 1) / line_slope;
            *x2 = *x2 + delx;
        }
        else {
            *y2 = y2b - 1;
            delx = ((height2/2.0) + 1) / line_slope;
            *x2 = *x2 - delx;
        }
    }
    else {
        /* right, or else, left */
        if (dx < 0) {
            *x2 = x2b + width2 + 1;
            dely = ((width2/2.0) + 1) * line_slope;
            *y2 = *y2 + dely;
        }
        else {
            *x2 = x2b - 1;
            dely = ((width2/2.0) + 1) * line_slope;
            *y2 = *y2 - dely;
        }
    }
}

/* FUNCTION Pnode_label_coords
* Purpose
* calculates the positions of the labels
* Inputs
* x1, y1, x2, y2, type (coordinates)
* Outputs
* x, y (label coordinates)
* Programming History
* 1993 Apr, Mark M. Lacey, Original function written

```


dyndis.c Tue Nov 8 08:46:57 1994 47

```

* Purpose: Makes widget which will control Grid Spacing
* (called from Postproc Init_Dds Dialog)
*
* Inputs
* none
*
* Outputs
* none
*
* Programming History
* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file
*/

void Make Zoom Window()
{
    extern Widget toplabel;
    extern Widget ZoomShell;
    extern Widget ZoomLabel;
    extern int ZoomLevel, ZoomChange;
    Widget ZoomForm, Zoom_In, Zoom_Out;
    Widget Zoom_OK, Zoom_Cancel;
    Arg arg(5);
    int i;

    /* Create the popup shell */
    ZoomShell= XtCreatePopupShell("Zoom Dialog",
                                transientShellWidgetClass,
                                toplabel, NULL, 0);

    /* Now create all of the widgets inside of the popup */
    ZoomForm= XtCreateManagedWidget("ZoomForm", formWidgetClass,
                                    ZoomShell, NULL, 0);
    i= 0;
    XtSetArg(arg(1), XtBorderWidth, 2); i++;
    XtSetArg(arg(1), XtVertDistance, 5); i++;
    XtSetArg(arg(1), XtResizable, FALSE); i++;
    XtSetValues(ZoomForm, arg, i);

    /* I'll create the label first, but wait to set its arguments until later
    * Why do this? Well, I'm doing it this way so that I can use the
    * label widget as an argument to the callback. I don't want to set its
    * arguments here so I can place its location to the right of the up/down
    * buttons
    */
    ZoomLabel= XtCreateManagedWidget("Zoom label", labelWidgetClass,
                                    ZoomForm, NULL, 0);

    Zoom_In= XtCreateManagedWidget("Zoom in", commandWidgetClass,
                                    ZoomForm, NULL, 0);
    i= 0;
    XtSetArg(arg(1), XtLabel, "Zoom in "); i++;
    XtSetArg(arg(1), XtBorderWidth, 0); i++;
    XtSetArg(arg(1), XtFromMoris, NULL); i++;
    XtSetArg(arg(1), XtResizable, FALSE); i++;
    XtSetArg(arg(1), XtFromVert, NULL); i++;
    XtSetValues(Zoom_In, arg, i);
    XtAddCallback(Zoom_In, XtCallback, PZoom, (int *) ZOOM_IN);

    Zoom_Out= XtCreateManagedWidget("Zoom out", commandWidgetClass,
                                    ZoomForm, NULL, 0);
    i= 0;
    XtSetArg(arg(1), XtLabel, " "); i++;
    XtSetArg(arg(1), XtBorderWidth, 0); i++;
    XtSetArg(arg(1), XtFromMoris, NULL); i++;
    XtSetArg(arg(1), XtResizable, TRUE); i++;
    XtSetArg(arg(1), XtFromVert, NULL); i++;
    XtSetValues(Zoom_Out, arg, i);
    XtAddCallback(Zoom_Out, XtCallback, PZoom, (int *) ZOOM_OUT);

    /* Now, I'll set the Zoom label widget */
    i= 0;
    XtSetArg(arg(1), XtLabel, " "); i++;
    XtSetArg(arg(1), XtBorderWidth, 0); i++;
    XtSetArg(arg(1), XtFromMoris, NULL); i++;
    XtSetArg(arg(1), XtResizable, TRUE); i++;
    XtSetArg(arg(1), XtFromVert, NULL); i++;
    XtSetValues(ZoomLabel, arg, i);

    Zoom_OK= XtCreateManagedWidget("OK", commandWidgetClass,
                                    ZoomForm, NULL, 0);
    i= 0;
    XtSetArg(arg(1), XtBorderWidth, 2); i++;
    XtSetArg(arg(1), XtLabel, "OK"); i++;
    XtSetArg(arg(1), XtFromMoris, NULL); i++;
    XtSetArg(arg(1), XtFromVert, Zoom_Out); i++;
    XtSetValues(Zoom_OK, arg, i);
    XtAddCallback(Zoom_OK, XtCallback, PChangeZoom, NULL);

    return;
}

/* PZoom */
* Purpose: Changes the zoom
* Inputs
* none
*
* Outputs
* none
*
* Programming History
* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file
*/
void PZoom(w, client data, call data)
Widget w;
caddr_t call_data, client_data;
{
    extern Widget ZoomLabel;
    char str(20);
    Arg arg(1);
    int amount= (int) client_data;
    int zl;
    int err;

    zl = temp_zoom_factor * 100;
    err = zl % ZOOM_PERCENT;
    if (err != 0)
        if (err < 5)
            zl -= zl % ZOOM_PERCENT;
        else
            zl += ZOOM_PERCENT - (zl % ZOOM_PERCENT);
}

```



```

        dx, dy);

        i= 0;
        XtSetArg(arg[i], XtWidth, am*width); i++;
        XtSetArg(arg[i], XtHeight, am*height); i++;
        XtGetValues(ZoomShell, arg, i);

        xm = (Position) ( x - (Position) (m*width/2) );
        ym = (Position) ( y - (Position) (m*height/2) );

        /* Now set these values to the pshell */

        i= 0;
        XtSetArg(arg[i], XtXr, x); i++;
        XtSetArg(arg[i], XtYr, y); i++;
        XtGetValues(ZoomShell, arg, i);

        XtPopUp(ZoomShell, XtGrabExclusive);

        return;

    }

    /* Post_zoom_factor */
    {
        /* Purpose: sets the zoom factor */
        /* Inputs */
        /* none */
        /* Outputs */
        /* none */
        /* Programming History */
        /* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file */
        /* */

        void post_zoom_factor(double d)
        {
            zoom_factor = d;

        }

        /* Post_zoom_factor */
        {
            /* Purpose: gets the zoom factor */
            /* Inputs */
            /* none */
            /* Outputs */
            /* none */
            /* Programming History */
            /* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file */
            /* */

            double pget_zoom_factor()
            {
                return zoom_factor;
            }

            /* Pbl.select */
            {
                /*
                 *
                 */
            }
        }
    }
}

/* Purpose: finds a connector when button is pushed */
/* Inputs */
/* none */
/* Outputs */
/* none */
/* Programming History */
/* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file */
/* */

void pbl_select(Widget canvas, XPointer client_data, XButtonEvent *e)
{
    int x, y;
    extern Widget topLevel;
    extern ModeType *first Mode;
    extern ConnectorType *first Connector;
    extern ConnectorType *pickconn;

    x = e->x;
    y = e->y;

    pickconn = pfind_closest_connector(x, y);

    CenterPop(topLevel, AppPlotSelectShell);

    /* Pfind_closest_connector */
    {
        /* Purpose: finds the closest connector */
        /* Inputs */
        /* int x, y - coordinates of connector */
        /* Outputs */
        /* none */
        /* Programming History */
        /* 1993 Aug, Gayle E. Ermer, Function transposed to dyndis file */
        /* */

        ConnectorType *pfind_closest_connector(int x, int y)
        {
            extern ConnectorType *first Connector;
            int Point_Type *ip;
            int
                x1,
                y1,
                x2,
                y2,
                labelx, labely;
            int dx, dy;
            int mind;
            int disquare;
            ConnectorType *holdconn, *conn;
            ModeType M;

            /* this is 15 pixel radius from the label --- it is not a great method */
            /* for handling this, but it will suffice */
            #define CLOSE_RADIUS 125

            mind = 1000000; /* some big number */
        }
    }
}

```

```

holdconn = NULL;

conn = First.Connector->Next;

/* we need to check each connector, and each one of its interior */
/* points */
while (conn) {
    /* skip first interior point */
    ip = conn->InteriorPoints;
    if (ip) ip = ip->Next;

    if (ip == NULL) {
        Pcalc line(conn->from, conn->to, &x1, &y1, &x2, &y2);
        Pcalc label coords(x1, y1, x2, y2, &labelx, &labely);
        dx = labelx - x1;
        dy = labely - y1;
        dsquare = dx*dx + dy*dy;
        if (dsquare < CLOSE_ENOUGH && dsquare < mind) {
            mind = dsquare;
            holdconn = conn;
        }
    }
    conn = conn->Next;
}
return holdconn;
}

/* FUNCTION Postproc Init Ddis Plot Dialog -----
 *
 * Purpose
 *
 *   Initialises the widgets for the dyndis plot dialog.
 *
 * Inputs
 *
 *   parent      The parent widget of the post processor plot dialog.
 *
 * Outputs
 *
 *   None
 *
 * Programming history (most recent change to least)
 *
 *   92 Aug 17 Original function written by Mark M. Lacey
 *   93 Dec 03 Altered for dynamic display program by Gayle E. Ermer
 */
void Postproc Init Ddis Plot Dialog(Widget parent)
{
    int
    i;
    Arg args[10];
    Widget
    ppPlotDdisDialogForm,
    ppPlotDdisDialogRangeMinLabel,
    ppPlotDdisDialogRangeMaxLabel,
    ppPlotDdisDialogDisplayMinLabel,
    ppPlotDdisDialogDisplayMaxLabel,
    ppPlotDdisDialogScaleLabel,
    Display *theDisplay;
    int theScreen;
    int theScreenDepth;
    Colormap def colormap;
    XColor theXColor, theHardwareColor;
    unsigned long theBlackPixel;
    int theStatus;
    char *colorNames[] = {
        "yellow",

```

```

    "blue",
    "red",
    "green",
    "orange"
);

/* Find out the Display and screen we are dealing with. Also find out if
 * we are dealing with a color or a monochrome system */
theDisplay= XDisplay(parent);
theScreen= XScreenOfDisplay(theDisplay, theScreen);
theScreenDepth= DefaultDepth(theDisplay, theScreen);
theBackPixel= BlackPixel(theDisplay, theScreen);

if (theScreenDepth > 1) /* We are dealing with a color system */
{
    def_colormap= DefaultColormap(theDisplay, theScreen);
    for (i = 0; i < 5; i++)
    {
        theStatus= LookupColor(theDisplay, def_colormap,
                               colorName[i],
                               theBackColor,
                               theHardwareColor);

        if (theStatus != 0)
        {
            theStatus= XAllocColor(theDisplay,
                                   def_colormap,
                                   theHardwareColor);

            if (theStatus != 0)
                GraphColors[i]= theHardwareColor.pixel;
            else
                GraphColors[i]= theBackPixel;
        }
    }
}
else /* We have a monochrome system */
{
    for (i = 0; i < 9; i++)
        GraphColors[i]= theBackPixel;
}

/* Create dialog for plotting
 */
ppPlotDialogPopupShell =
    XtCreatePopupShell("Plot", transientShellWidgetClass,
                      parent, NULL, 0);

i = 0;
XtSetArg(args[i], XtNlabel, "Plot"); ++i;
XtSetArg(args[i], XtNborderWidth, 0); ++i;
XtSetValues(ppPlotDialogPopupShell, args, i);

/* Create a form widget to put the other widgets in
 */
ppPlotDialogForm =
    XtCreateManagedWidget("ppPlotDialogForm",
                          formWidgetClass,
                          ppPlotDialogPopupShell,
                          NULL,
                          0);

i = 0;
XtSetArg(args[i], XtNborderWidth, 0); ++i;
XtSetArg(args[i], XtNresize, FALSE); ++i;
XtSetValues(ppPlotDialogForm, args, i);

/* Create the actual plotting area
 */
ppPlotDialogGraph = (Widget)
    XtAddWidget("ppPlotDialogGraph",
                ppPlotDialogForm,
                graphWidgetClass,
                XtNheight, 400,
                XtNwidth, 650,
                XtNautoscale, FALSE,
                NULL);

/* Create a widget to display their entered title into
 */
ppPlotDialogTitle =
    XtCreateManagedWidget("ppPlotDialogTitle[PLOT_TITLE]",
                          asciiTextWidgetClass,
                          ppPlotDialogForm,
                          NULL,
                          0);

i = 0;
XtSetArg(args[i], XtNwidth, 650); ++i;
XtSetArg(args[i], XtNheight, 400); ++i;
XtSetArg(args[i], XtNresize, FALSE); ++i;
XtSetArg(args[i], XtNfromVert, ppPlotDialogGraph); ++i;
XtSetArg(args[i], XtNfromHoriz, NULL); ++i;
XtSetArg(args[i], XtNtype, XtNasciiString); ++i;
XtSetArg(args[i], XtNeditType, XtNedit); ++i;
XtSetValues(ppPlotDialogTitle, args, i);

/* Create labels for the outputs
 */
for ( ondx = 0; ondx < MAX_SELECTED; ++ondx ) {
    ppPlotDialogOutputLabel[ondx] =
        XtCreateManagedWidget("ppPlotDialogOutputLabel",
                              labelWidgetClass,
                              ppPlotDialogForm,
                              NULL,
                              0);

    i = 0;
    XtSetArg(args[i], XtNlabel, "Output "); ++i;
    if ( ondx == 0 ) {
        XtSetArg(args[i], XtNfromHoriz, NULL); ++i;
        XtSetArg(args[i], XtNborderDistance, 110); ++i;
    }
    else {
        XtSetArg(args[i], XtNfromHoriz,
                  ppPlotDialogOutputLabel[ondx-1]); ++i;
    }
    XtSetArg(args[i], XtNfromVert, ppPlotDialogTitle); ++i;
    XtSetArg(args[i], XtNresize, FALSE); ++i;
    XtSetArg(args[i], XtNwidth, 100); ++i;
    XtSetValues(ppPlotDialogOutputLabel[ondx], args, i);
}

/* Create label for range min
 */

```

```

ppPlotDdsDialogRangeMinMaxLabel =
    XtCreateManagedWidget("ppPlotDialogRangeMinMaxLabel",
        labelWidgetClass,
        ppPlotDdsDialogForm,
        NULL,
        0);

i = 0;
XtSetArg(args[i], XCMFromMoris, "Range Min."); ++i;
XtSetArg(args[i], XCMFromMoris, NULL); ++i;
XtSetArg(args[i], XCMFromVert, ppPlotDdsDialogOutputLabel[0]); ++i;
XtSetArg(args[i], XCMResize, FALSE); ++i;
XtSetArg(args[i], XCMWidth, 100); ++i;
XtSetValues(ppPlotDdsDialogRangeMinMaxLabel, args, i);

/*
 * Create label for range max
 */
ppPlotDdsDialogRangeMaxLabel =
    XtCreateManagedWidget("ppPlotDialogRangeMaxLabel",
        labelWidgetClass,
        ppPlotDdsDialogForm,
        NULL,
        0);

i = 0;
XtSetArg(args[i], XCMLabel, "Range Max."); ++i;
XtSetArg(args[i], XCMFromMoris, NULL); ++i;
XtSetArg(args[i], XCMFromVert, ppPlotDdsDialogRangeMinMaxLabel); ++i;
XtSetArg(args[i], XCMResize, FALSE); ++i;
XtSetArg(args[i], XCMWidth, 100); ++i;
XtSetValues(ppPlotDdsDialogRangeMaxLabel, args, i);

/*
 * Create label for display min
 */
ppPlotDdsDialogDisplayMinLabel =
    XtCreateManagedWidget("ppPlotDialogDisplayMinLabel",
        labelWidgetClass,
        ppPlotDdsDialogForm,
        NULL,
        0);

i = 0;
XtSetArg(args[i], XCMLabel, "Display Min."); ++i;
XtSetArg(args[i], XCMFromMoris, NULL); ++i;
XtSetArg(args[i], XCMFromVert, ppPlotDdsDialogRangeMaxLabel); ++i;
XtSetArg(args[i], XCMResize, FALSE); ++i;
XtSetArg(args[i], XCMWidth, 100); ++i;
XtSetValues(ppPlotDdsDialogDisplayMinLabel, args, i);

/*
 * Create label for display max
 */
ppPlotDdsDialogDisplayMaxLabel =
    XtCreateManagedWidget("ppPlotDialogDisplayMaxLabel",
        labelWidgetClass,
        ppPlotDdsDialogForm,
        NULL,
        0);

i = 0;
XtSetArg(args[i], XCMLabel, "Display Max."); ++i;
XtSetArg(args[i], XCMFromMoris, NULL); ++i;
XtSetArg(args[i], XCMFromVert, ppPlotDdsDialogDisplayMinLabel); ++i;

```


dyndia.c Tue Nov 8 08:46:57 1994 54

```

* Example calling sequence
*
* Postproc Popdown Plot Dialog()
*
* Programming history (most recent change to least)
*
* 22 Aug 17 Original function written by Mark M. Lacey
* 24 Jun Adapted to dyndia by Gayle E. Ermer
*/

void Postproc Popdown_Ddis Plot Dialog()
(
    IfPopdown(ppPlotDdisDialogPopUpShell);
)

/* FUNCTION Postproc Delete Ddis Plot Dialog_Data -----
* Purpose
*
* Delete the data that is used in the post processor plot dialog.
*
* Inputs
*
* none
*
* Outputs
*
* none
*
* Example calling sequence
*
* Postproc Delete Plot Dialog_Data();
*
* Programming history (most recent change to least)
*
* 22 Aug 17 Original function written by Mark M. Lacey
* 24 Jun Adapted to dyndia by Gayle E. Ermer
*/
void Postproc Delete Ddis Plot Dialog_Data()
(
    Postproc Ddis Delete_Graphs();
)

/* FUNCTION Postproc Popup_Ddis Plot -----
* Purpose
*
* Initializes the postprocessor plot data and pops up the plot
* dialog.
*
* Inputs
*
* none
*
* Outputs
*
* none
*
* Example calling sequence
*
* Postproc_Popup_Plot();
*
* Programming history (most recent change to least)
*
* 22 Aug 17 Original function written by Mark M. Lacey
*/
24 Jun Adapted to dyndia by Gayle E. Ermer

void Postproc Popup_Ddis Plot(ConnectorType *conn)
(
    real tmaxx, tminx, tmaxy, tminy;
    extern real ppMinMaxima[MAX_SELECTED];
    extern real ppMinMaxima[MAX_SELECTED];
    extern Widget toplevel;

    Postproc Init Ddis Plot Dialog_Data(conn);
    CenterPop(toplevel, AppPlotDdisDialogPopUpShell);

    Postproc Ddis Plot Dialog Check(conn);
    Postproc_Ddis Add Graphs(conn);

    tmaxx = ppMainDialogInitTimeVal;
    /* times 1.1 so the label is displayed at the end
    had back...
    */
    tmaxx = ppMainDialogFinalTimeVal*1.1;
    tminy = 0.0;
    tmaxy = 1.0;

    graph set bounds(ppPlotDdisDialogGraph, tminx, tmaxx, tminy, tmaxy);
    set our min(ppPlotDdisDialogGraph, ppMinMaxima[our scale]);
    set our max(ppPlotDdisDialogGraph, ppMinMaxima[our scale]);
    graph plot_all(ppPlotDdisDialogGraph);
)

/* FUNCTION Postproc Init Ddis Plot Dialog_Data -----
* Purpose
*
* Initialize the data that is used in the post processor plot
* dialog.
*
* Inputs
*
* none
*
* Outputs
*
* none
*
* Example calling sequence
*
* Postproc Init Ddis Plot Dialog_Data();
*
* Programming history (most recent change to least)
*
* 24 Jun Original function written by Gayle E. Ermer
*/
void Postproc_Init Ddis Plot Dialog_Data(ConnectorType *conn)
(
    char
    Temp[13],
    *tptr;
    Arg
    argv[5];
    int
    i,

```


dyndis.c Tue Nov 8 08:46:57 1994 56

```

    XSetArg(args[1], XtMString, ""); ++i;
    XSetValues(ppPlotDdsDialogDisplayWin[index], args, i);
    XSetValues(ppPlotDdsDialogDisplayWin[index], args, i);
    XSetSensitive(ppPlotDdsDialogScaleButton[index], FALSE);

    i = 0;
    XSetArg(args[1], XtMString, ""); ++i;
    XSetValues(ppPlotDdsDialogScaleButton[index], args, i);
}

/* FUNCTION Postproc Dds Add Graphs -----
*
* Purpose
* Inputs
* Outputs
* Example calling sequence
* Postproc_Add_Graphs()
*
* Programming history (most recent change to least)
* 94 Jun Original function written by Gayle E. Ermer
*/

void Postproc_Dds_Add_Graphs(ConnectorType *conn)
{
    int optr, ns;
    double timedata[300];
    double data[300];
    char name[13], *mptr;
    char top[100];
    int current_time[20];
    int temp;
    extern real ppmMainDialogInitTimeVal;
    extern real ppmMainDialogFinalTimeVal;
    extern int plotindex;
    char temp[21];
    char ename[] = "E";
    char fname[] = "F";
    char pename[] = "P";
    char qname[] = "Q";
    char rname[] = "R";
    char pename[] = "Pow";
    char ename[] = "En";
    char *str;
    int stage;
    int i, tok;
    ConnectorType *traverseCons;
    char *tempname, *curconname;

    /* get the data for TIME, which everything will be plotted against */
    ns = Postproc_Get_Time_Output(seq2 1, nov-1,
        ppmMainDialogInitTimeVal,
        ppmMainDialogFinalTimeVal,
        timedata);

    i = 0;
    traverseCons = First Connector->Next;
    while(traverseCons) {
        tempname = traverseCons->name;
        curconname = conn->name;

        tok = stromp(curconname, tempname);
        if (tok == 0) {
            optr = 1;
            break;
        }
        else {
            i++;
            traverseCons = traverseCons->Next;
        }
    }

    for (stage = init_timestep; stage < final_timestep; stage++) {
        switch (Plot Var) {
            case EFFORT:
                data[stage] = ppdata[optr][stage][0];
                break;
            case FLOW:
                data[stage] = ppdata[optr][stage][1];
                break;
            case MOMENTUM:
                data[stage] = ppdata[optr][stage][0];
                break;
            case DISPLACEMENT:
                data[stage] = ppdata[optr][stage][1];
                break;
            case ENERGY:
                data[stage] = ppdata[optr][stage][2];
                break;
            case POWER:
                data[stage] = ppdata[optr][stage][1] * ppdata[optr][stage][0];
                break;
            case SIGNAL:
                data[stage] = edata[optr][stage];
                break;
        }
    }

    /* get the data for the selected output */
    Postproc_Scale Dds Plot Data(0, ppmMainDialogInitTimeVal,
        ppmMainDialogFinalTimeVal,
        data);

    switch (Plot Var) {
        case EFFORT:
            temp[0] = ename[0];
            temp[1] = ename[1];
            str = conn->name;
            stroat(temp, str);
            break;
        case FLOW:
            temp[0] = fname[0];
            temp[1] = fname[1];
            str = conn->name;
            stroat(temp, str);
            break;
        case MOMENTUM:
            temp[0] = pname[0];
            temp[1] = pname[1];
            str = conn->name;
            stroat(temp, str);
            break;
        case DISPLACEMENT:
            temp[0] = qname[0];
            temp[1] = qname[1];
            str = conn->name;
            stroat(temp, str);
            break;
    }
}

```

```

        strtout(temp, str);
        break;
    case BRIGHT:
        temp[0] = ename[0];
        temp[1] = ename[1];
        temp[2] = ename[2];
        str = conu->name;
        strtout(temp, str);
        break;
    case POWER:
        temp[0] = pname[0];
        temp[1] = pname[1];
        temp[2] = pname[2];
        str = conu->name;
        strtout(temp, str);
        break;
    case SIGNAL:
        temp[0] = ename[0];
        temp[1] = ename[1];
        str = conu->name;
        strtout(temp, str);
        break;
    )
    nptr = atemp[0];
    sprintf(name, "%s", nchars (nptr, 12), nptr);
    time((time t)atemp);
    strttime(current time, sizeof(current time)-1, "%F %D",
        localtime((time t)atemp));
    printf(top, "%s %s", GetVersion(), current time);
    graph add(ppPlotDialogGraph, 1, timedata, data, na,
        name, "time", "", top, GraphColors[1]);
}

/* FUNCTION Postproc Scale Ddis Plot Data -----
* Purpose
* Inputs
* optir Index of output that we want to scale data for.
* itime Initial time to scan from.
* ftime Final time to scan to.
* data Original data.
* Outputs
* data The scaled data.
* Example calling sequence
* Postproc Scale Plot Data(optir, itime, ftime, data);
* Programming history (most recent change to least)
* 92 Aug 17 Original function written by Mark M. Lacey
*/
void Postproc Scale Ddis Plot Data(int optir,
    real itime, real ftime, double data[])
{
    int si, sf, endir;
    real urange;
    extern real ppuMinimums[MAX_SELECTED];

    strtout(temp, str);
    break;
    case BRIGHT:
        temp[0] = ename[0];
        temp[1] = ename[1];
        temp[2] = ename[2];
        str = conu->name;
        strtout(temp, str);
        break;
    case POWER:
        temp[0] = pname[0];
        temp[1] = pname[1];
        temp[2] = pname[2];
        str = conu->name;
        strtout(temp, str);
        break;
    case SIGNAL:
        temp[0] = ename[0];
        temp[1] = ename[1];
        str = conu->name;
        strtout(temp, str);
        break;
    )
    nptr = atemp[0];
    sprintf(name, "%s", nchars (nptr, 12), nptr);
    time((time t)atemp);
    strttime(current time, sizeof(current time)-1, "%F %D",
        localtime((time t)atemp));
    printf(top, "%s %s", GetVersion(), current time);
    graph add(ppPlotDialogGraph, 1, timedata, data, na,
        name, "time", "", top, GraphColors[1]);
}

/* FUNCTION Postproc Get Time Output -----
* Purpose
* Get the data for the output indexed by 'ondx'.
* Inputs
* ondx Index of output we would like data for.
* itime Initial time that we would like data for.
* ftime Final time that we would like data for.
* Outputs
* none
* Example calling sequence
* Postproc Get Output(ondx, itime, ftime, data);
* Programming history (most recent change to least)
* 92 Aug 17 Original function written by Mark M. Lacey
*/
int Postproc Get Time Output(int ondx, real itime, real ftime, double
    data[])
{
    int si, sf, endir;

    si = StepIndex(itime);
    sf = StepIndex(ftime);
    for ( endir = si; endir <= sf; ++endir )
        data[ondx-si] = (double) pos0bk i.ralt[ondx+ondx*MAKES];
    return sf-si+1;
}

/* FUNCTION Postproc Get Ddis Output -----
* Purpose
* Get the data for the output indexed by 'ondx'.
* Inputs

```

dyndis.c Tue Nov 8 08:46:57 1994 58

```

* ondx Index of output we would like data for.
* itime Initial time that we would like data for.
* ftime Final time that we would like data for.
*
* Outputs
*
* none
*
* Example calling sequence
*
* Postproc Get Output(ondx, itime, ftime, data);
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
int Postproc_Get_Ddis_Output(int ondx, real itime, real ftime, double
    data[])
(
    int si, sf, sndx;
    extern real ppdata[200][300][2];
    extern real ppdata[200][300][3];
    extern int plotindex;
    FILE *fp;

    si = StopIndex(itime);
    sf = StopIndex(ftime);

    switch (ondx) (
        case 0:
            fp = fopen("Plotdata", "w");
            for (sndx = si; sndx <= sf; ++sndx) (
                data[sndx-si] = (double) ppdata[plotindex][sndx][0];
            )
            break;
        case 1:
            for (sndx = si; sndx <= sf; ++sndx) (
                data[sndx-si] = (double) ppdata[plotindex][sndx][1];
            )
            break;
        case 2:
            for (sndx = si; sndx <= sf; ++sndx) (
                data[sndx-si] = (double) ppdata[plotindex][sndx][0];
            )
            break;
        case 3:
            for (sndx = si; sndx <= sf; ++sndx) (
                data[sndx-si] = (double) ppdata[plotindex][sndx][1];
            )
            break;
        )
    return sf-si+1;
)

/* FUNCTION Postproc_Ddis_Analyse_Data -----
*
* Purpose
*
* Analyzes the data to find the minimum and maximum for each of
* the outputs, and the output with the largest range.
*
* Inputs
*
* list List of output indices that the user selected

```

```

* list Size of 'list'
* itime Initial time to scan from.
* ftime Final time to scan to.
*
* Outputs
*
* none
*
* Example calling sequence
*
* Postproc Analyse Data(list, loise, itime, ftime);
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
void Postproc_Ddis_Analyse_Data(ConnectorType *conn, real itime, real ftime)
(
    int
    ondx, i, tok, stage;
    extern int plotindex;
    real
    lrange;
    real
    emax, gmax, fmax, mmax, pmax, smax, amax,
    emin, gmin, fmin, mmin, pmin, smin, amin;
    real valueq, valueq, valueq, valueq, valueq, valueq, valueq, valueq;
    extern real ppdata[7];
    extern real ppdata[7];
    ConnectorType *traverseCons;
    char *tempname, *ourconnname;
    extern real ppdata[200][300][2];
    extern real ppdata[200][300][3];
    lrange = 0.0;
    pplargestRangePtr = 0;

    emax = gmax = fmax = mmax = pmax = smax = amax = -999999999.0;
    emin = gmin = fmin = mmin = pmin = smin = amin = 999999999.0;

    i = 0;
    traverseCons = First Connector->Next;
    while(traverseCons) (
        tempname = traverseCons->name;
        ourconnname = conn->name;
        tok = strcmp(ourconnname, tempname);
        if (tok == 0) (
            plotindex = i;
            break;
        )
        else (
            i++;
            traverseCons = traverseCons->Next;
        )
    )

    for (stage = 0; stage < num stages; stage++) (
        valueq = ppdata[plotindex][stage][0];
        valueq = ppdata[plotindex][stage][1];
        valueq = ppdata[plotindex][stage][0];
        valueq = ppdata[plotindex][stage][1];
        valueq = ppdata[plotindex][stage][0];
        valueq = ppdata[plotindex][stage][1];
        valueq = ppdata[plotindex][stage][0];
        valueq = ppdata[plotindex][stage][1];
    )

```


dyndis.c Tue Nov 8 08:46:57 1994 60

```

* Inputs
*
* Outputs
*
* none
*
* Example calling sequence
*
* never called explicitly
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
void Postproc_Ddis_Plot_Button_Display(Widget w, oaddr_t junk1, oaddr_t junk2)
{
    real tmaxx, tminx, tmaxy, tminy;
    extern real ppminimums[MAX_SELECTED];
    extern real ppmaximums[MAX_SELECTED];
    extern ConnectorType *piconns;

    if (!Postproc_Ddis_Plot_Dialog_Check(plotconn)) {
        return;
    }
    Postproc_Ddis_Delete_Graphs();
    Postproc_Ddis_Add_Graphs(plotconn);
    tminx = ppminDialogInatTimeVal;
    /* time 1.1 so the label is displayed at the end
    bad hack...
    */
    tmaxx = ppminDialogInatTimeVal+1.1;
    tminy = 0.0;
    tmaxy = 1.0;
    graph_set_bounds(ppPlotDdisDialogGraph, tminx, tmaxx, tminy, tmaxy);
    set_our_min(ppPlotDdisDialogGraph, ppminimums[our_scale]);
    set_our_max(ppPlotDdisDialogGraph, ppmaximums[our_scale]);
    graph_plot_all(ppPlotDdisDialogGraph);
}

/* FUNCTION Postproc_Main_Button_Ddis_Plot -----
*
* Purpose
*
* Called when the plot button from the main dialog box is
* selected. Pops up the plot dialog box.
*
* Inputs
*
* w      The button widget that was pressed.
* junk1      Passed by X, not used by our routine.
* junk2      Passed by X, not used by our routine.
*
* Outputs
*
* none
*
* Example calling sequence
*
* never called explicitly
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
void Postproc_Main_Button_Ddis_Plot(ConnectorType *conn)
{
    Postproc_Popup_Ddis_Plot(conn);
}

/* FUNCTION Postproc_Ddis_Delete_Graphs -----
*
* Purpose
*
* Inputs
*
* none
*
* Outputs
*
* Example calling sequence
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
void Postproc_Ddis_Delete_Graphs()
{
    int ondx;
    for ( ondx = 0; ondx < 6; ++ondx ) {
        graph_delete(ppPlotDdisDialogGraph, ondx+1);
    }
}

/* FUNCTION Postproc_Ddis_Plot_Dialog_Check -----
*
* Purpose
*
* Inputs
*
* none
*
* Outputs
*
* <return>    1 if all is well, 0 otherwise
*
* Example calling sequence
*
* Postproc_Plot_Dialog_Check();
*
* Programming history (most recent change to least)
*
* 92 Aug 17 Original function written by Mark M. Lacey
*/
int Postproc_Ddis_Plot_Dialog_Check(ConnectorType *conn)
{
    int i, ondx;
    Argv argv[5];
    char *Temp;
    char Message[120];
    char *pstr;
    real treal;
    extern real ppminimums[MAX_SELECTED];
    extern real ppmaximums[MAX_SELECTED];

```



```

mptr = conn->name;

oaddr = 0;

i = 0;
XtSetArg(args[1], XtMString, aTemp); ++i;
XtGetValues(ppPlotDialogDisplayWin[conn], args, 1);
if (!asciitof(Temp, appMinimums[conn])) {
    sprintf(Message, "Display minimum for output %s is not a -
    "valid number", nohars_(mptr, 12), mptr);
    Problem Pop(Message);
    return 0;
}

i = 0;
XtSetArg(args[1], XtMString, aTemp); ++i;
XtGetValues(ppPlotDialogDisplayWin[conn], args, 1);
if (!asciitof(Temp, appMaximums[conn])) {
    sprintf(Message, "Display maximum for output %s is not a -
    "valid number", nohars_(mptr, 12), mptr);
    Problem Pop(Message);
    return 0;
}

if (ppMinimums[conn] >= ppMaximums[conn]) {
    sprintf(Message, "Display minimum for output %s must be -
    "less than display maximum", nohars_(mptr, 12),
    mptr);
    Problem Pop(Message);
    return 0;
}

return 1;
}

/* The remaining functions control the glyph display */
/* Jul 94 Original functions written by Gayle E. Erner */

/* update_energy */
void update_energy(Widget w, caddr_t client_data, caddr_t call_data)
{
    XawListReturnStruct *list_ptr;
    String new_val;
    Arg args[10];
    list_ptr = XawListGetCurrent(energy_box);
    XQueryTextWidget(editenergy, new_val);
    XtSetArg(editenergy, XtMString, list_ptr->list_index, new_val);
    update_glyph_list(list_ptr->list_index, new_val);
    XawListHighlight(energy_box);
    XawListTextWidget(editenergy, "");
    XtSetArg(args[0], XtMString, FALSE);
    XtSetValues(w, args, 1);
}

/* update_glyph_list */
void update_glyph_list(int place, char *new_val)

```

```

{
    char old_name[12];
    second(Init Glyph Menu[place], "%s", old_name);
    sprintf(Init Glyph Menu[place], "%-12s %s", old_name, new_val);
    XawListChange(energy_box, Init Glyph Menu, Num Glyphs, 0, TRUE);
}

/* edit energy */
void edit_energy(Widget w, caddr_t client_data, XawListReturnStruct
*call_data)
{
    Arg args[10];
    XawListTextWidget(editenergy, Init Glyph String[call_data->list_index]);
    XtSetArg(args[0], XtMString, TRUE);
    XtSetValues(update, args, 1);
}

/* accept energy */
void accept_energy(Widget w, caddr_t client_data, caddr_t call_data)
{
    save_energy_vals();
    cancel_energy(w, client_data, call_data);
}

/* cancel energy */
void cancel_energy(Widget w, caddr_t client_data, caddr_t call_data)
{
    XtPopdown(MenuZCSbell);
    find_glyph_ranges();
}

/* save_energy_vals */
void save_energy_vals()
{
    float tempfloat;
    int loop_ctr;
    char error_string[255];
    for (loop_ctr = 0; loop_ctr < Num Glyphs; loop_ctr++) {
        if (asciitof(Init Glyph String[loop_ctr], tempfloat)) {
            Init Glyph Val[loop_ctr] = tempfloat;
        }
        else {
            strcpy(error_string, "Error Converting Glyph Values ");
            strcat(error_string, Init Glyph Menu[loop_ctr]);
            Problem Pop(error_string);
        }
    }
}

/* init_energy_array */
void init_energy_array()
{
    char *tempchar;
    ModeType *traverseModes;
    int index, ctr;

```

dyadis.c Tue Nov 8 09:46:57 1994 62

```

int i, time;
Arg args[1];
char tempstring[10];
/* char temp of glyph elements */

index = 0;
traverseables = First Node->Next;
while (traverseables) {
    if (traverseables->type)
        if (Type == C_NODE || Type == I_NODE) {
            Postproc Get Code on Node(traverseables, annuind, kindinds);
            sum = 0;
            for (i = 0; i < annuind; i++) {
                index = indicies[i];
                value = postind[indices[i][2]];
                sum = sum + value;
            }
        }
        if (Type == C_NODE || Type == I_NODE) {
            Postproc Get Code on Node(traverseables, annuind, kindinds);
            sum = 0;
            for (i = 0; i < annuind; i++) {
                index = indicies[i];
                value = postind[indices[i][2]];
                sum = sum + value;
            }
            sum = sum + Init Glyph Value();
            Get();
        }
        if (sum > Max) Max = sum;
        if (sum < Min) Min = sum;
        traverseables = traverseables->Next;
    }
    indices++;
    glyphmin = min;
    glyphmax = max;
    glyphrange = glyphmax - glyphmin;
}
/* End FILE - dyadis.c */

Min, glyph = index;
sum, glyph = index;
Init Glyph Name = (string *) Xmalloc (Num Glyphs * sizeof(string));
Init Glyph String = (string *) Xmalloc (Num Glyphs * sizeof(string));
index = 0;
traverseables = First Node->Next;
while (traverseables) {
    if (traverseables->type == C_NODE || traverseables->type == I_NODE) {
        if (traverseables->type == C_NODE || traverseables->type == I_NODE) {
            sum, glyph = index;
            Init Glyph Name[index] = (string) Xmalloc (MAXLENGTH)*sizeof(char);
            second(tempchar, "%12s", Init Glyph Name[index]);
            tempstring[index] = tempchar;
            Init Glyph Val[index] = 0.0;
            Init Glyph String[index] = (string) Xmalloc (MAXLENGTH)*sizeof(char);
            append(tempstring, Init Glyph Val[index], Init Glyph String[index]);
            append(Init Glyph Name[index], "%12s %s", tempstring,
                Init Glyph String[index]);
            indices++;
        }
        traverseables = traverseables->Next;
    }
}
{
    i = 0;
    Xfree(args[1]);
    Xfree(tempstring, Num Glyphs); free;
    Xfree(temp[1]); Xfree; Init Glyph Name); free;
    Xfree(tempstring, temp, arg, 1);
}
/* End glyph ranges */

void find_glyph_ranges()
{
    extern int new sum steps, init timestep, final timestep;
    real value, range, min, max;
    int i;
    int minind, maxind;
    int type;
    int minind, index[10];
    int sum;
    NodeType *traverseables;
    while (traverseables) {
        if (traverseables->type == C_NODE || traverseables->type == I_NODE) {
            sum = 0;
            for (i = 0; i < annuind; i++) {
                index[i] = indicies[i];
                value = postind[indices[i][2]];
                sum = sum + value;
            }
        }
        if (Type == C_NODE || Type == I_NODE) {
            Postproc Get Code on Node(traverseables, annuind, kindinds);
            sum = 0;
            for (i = 0; i < annuind; i++) {
                index = indicies[i];
                value = postind[indices[i][2]];
                sum = sum + value;
            }
            sum = sum + Init Glyph Value();
            Get();
        }
        if (sum > Max) Max = sum;
        if (sum < Min) Min = sum;
        traverseables = traverseables->Next;
    }
    indices++;
    glyphmin = min;
    glyphmax = max;
    glyphrange = glyphmax - glyphmin;
}

```

```

mainpp.c      Fri Jun 10 10:36:06 1994      1
/*****
 *
 * FILE:
 *
 * mainpp.c
 *
 * PURPOSE:
 *
 * Create and control the Report post-processor section
 *
 * REVISION HISTORY:
 *
 * 1.0      May 16, 1993      Dave Schultz
 *
 * Initial Version
 *
 *****/
/*****
 *
 * Now we need to include all of the X libraries */
#include "common.h"

#include "listsel.h"

/* Any other include files which we need */

#include "stdio.h"
#include "drawres.h"
#include "graph.h"
#include "widgets.h"
#include "mode.h"
#include "build.h"
#include "arraydia.h"
#include "ywidget.h"
#include "common.h"
*****/
/*****
 *
 * main - set everything up and start Export W/S post-processor
 *
 * PARAMETERS:
 *
 *   argc - number of command line arguments
 *
 *   argv - the command line args
 *
 * (NOTE: Currently, Export does not support command line arguments)
 *
 *****/
main(int argc, char *argv[]) {
    int seteps; (double real *);
    extern Widget toplevel, btnbox, file_title;
    int i;
    Arg args[10];

    seteps_(a(sectib4_1.eps));

    Init_info();

    Mode = NORMAL;

    /*
     * Create the toplevel window
     */
    toplevel= XInitialize ("Export - Post-Processor", "Export",
        NULL, 0, &argc, argv);

    /*
     * Initialize the Post Processor Popup dialog
     */
    PostProc_Init_Main_Dialog (toplevel);

```

```

/*
 * We'll probably want a way to do this in the Solver Dialog
 *
 */
XSetArg(args[1], XtLabel, "McName"); i++;
XSetArg(args[1], XtResource, NULL); i++;
XSetArg(args[1], XtResource, NULL); i++;
File_Title = XtCreateManagedWidget("file title",
    labelWidgetClass, toplevel,
    args, 1);

/*
 * Display the application shell
 */
XtRealizeWidget(toplevel);

/*
 * Initialize the file selector
 */
Init_selector(toplevel);

/*
 * Initialize the matrix viewing widgets
 */
Init_Matrix_Editor(toplevel);
Init_Matrix_Viewer(toplevel);

/*****
 *
 * Initialize the Yes No Popup */
/*****
 *
 * Init_YN_Popup(toplevel);

/*****
 *
 * Initialize the Matrix-X popup */
/*****
 *
 * Matrix_Init_Dialog(toplevel);

/*****
 *
 * Initialize the List Selection Widget */
/*****
 *
 * List_Setup(toplevel);

/*
 * Start the main loop
 */
XtMainLoop();

exit(0);

/* End of main() */

/* End of main.c */

```

B.2 Three-Dimensional Environment

The software developed for this application is stored in a file called **enport3d.c**. It is necessary to compile this along with the ENPORT library routines stored in **libenp.a** and the phigs libraries. What follows is a listing of the C language code in the enport3d.c file used to implement the BGVis-3D environment.

```

bond.c          Tue Sep 20 12:10:46 1994          1
/*****
/* file: export3d.o
/* Source code for 3D model display - based on Phigs
/* Programmer: Gayle Ermer
/* Date Begun: 6/93
/* Last Modification:
*****/

/**** Include files (general and export) ****/

#include <phigs.h>
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <graph.h>
#include <database.h>
#include <common.h>
#include <widgetdb.h>
#include <mode.h>
#include <common.h>
#include <listctrl.h>
#include <build.h>
#include <arraydia.h>
#include <ynwidget.h>
#include <drawarea.h>
#include <listselv.h>
#include <postproc.h>
#include <version.h>
#include <time.h>
#include <rv.h>

/**** Defines ****/

#define PI 3.14159265358979
#define deg ".1415926535897932384/180"

#define SetColor(VI,Rd,Gr,BL) VI.type = FMODEL_ROM; \
VI.val.general.x = Rd; \
VI.val.general.y = Gr; \
VI.val.general.z = BL

#define LIGHTSTYLE 1
#define LEDGEFLAG 2

#define WC_MIN -100.0
#define WC_MAX 100.0

#define MIN ((a) < (b)) ? (a) : (b)
#define MAX ((a) > (b)) ? (a) : (b)

#define QUIT -1
#define ERROR 0
#define OKAY 1

enum ScaleName {LIN, LOG, ABSLOC};

```

```

enum VarTypeNames {POWER, EFFORT, FLOW, DISPLACEMENT, MOMENTUM, ENERGY,
SSIGNAL, NONE};

/**** Global Variables ****/

FILE *fp1;
float WorkstaID = 1;
float WorkstaID2 = 2;
float WorkstaID3 = 3;
float Spheredstruc = 2;
float Scene = 3;
float ModLabels = 4;
float ConLabels = 5;
float Keystruc = 6;
float timestruc = 7;
float Squaredstruc = 8;

PowerL FCViolet, FCBlue, FCLtBlue, FCGreen, FCYellowGn,
FCYellow, FCOld, FCRange, FCPink, FCRed;
PowerL PowerGray, PowerGreen, PowerBlue, PowerLtBlue, PowerPurple, PowerYellow;

/**** Viewing parameters ****/
static Pview rep;
static Pview map;
static Ppoint3 vrp = { 0.0, 0.0, 0.0};
static Ppoint3 vup = { 0.0, 1.0, 0.0};
static Pvec3 vpa = { 0.866, -0.5, 0.5};

static char *strings[] = {
"REFRESH",
"QUIT",
"HIDE COM LABELS",
"HIDE MODE LABELS",
"DISPLAY COM LABELS",
"DISPLAY MODE LABELS",
"START ANIMATE",
"STEP FORWARD",
"STEP BACKWARD",
"GO TO TIME"
};

static num_strings = sizeof(strings)/sizeof(char*);

static Pmatrix identity = { 1.0, 0.0, 0.0, 0.0,
0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 1.0, 0.0,
0.0, 0.0, 0.0, 1.0
};

static double radius = 1.0;

Widget MainDialogInitTimeValue, MainDialogFinTimeValue;
Widget MainDialogScale;
Widget MainDialogVariable;
Widget MainDialogForm, MainDialogLabel;
Widget toplevel;

int Our Stage;
int Our Var;
int Scale Var;
int timeStage;
int init timeStage, final timeStage, new num stages;
Widget ChoicePopUpShell, ChoiceBox1;
Widget ChoicePopUpShell, ChoiceBox;
real pdate[200][300][2];

```



```

/* initialize toplevel widget */
toplevel = XtInitialize ("Load Model File Menu", "Report", NULL, 0, argc,
argv);

XtAddActions(new_actions, XtNumber(new_actions));

/* create a form to hold all other widgets */
MainDialogForm = XtCreateManagedWidget("MainDialogForm", formWidgetClass,
toplevel, NULL, 0);
i = 0;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNresize, FALSE); i++;
XtSetValues(MainDialogForm, args, i);

/* create a label for the dialog */
MainDialogLabel = XtCreateManagedWidget("MainDialogLabel",
labelWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Export 3D"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNresize, FALSE); i++;
XtSetArg(args[i], XtNwidth, 300); i++;
XtSetValues(MainDialogLabel, args, i);

/* create the load model file button */
MainDialogButton = XtCreateManagedWidget("MainDialogButton",
commandWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Load Model"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogLabel); i++;
XtSetValues(MainDialogButton, args, i);
XtAddCallback(MainDialogButton, XtCallbackProc, PhisLoad File, NULL);

/* create the load data file button */
MainDialogLoadData = XtCreateManagedWidget("MainDialogLoadData",
commandWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Load Data"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogButton); i++;
XtSetValues(MainDialogLoadData, args, i);
XtAddCallback(MainDialogLoadData, XtCallbackProc, PhisLoad Data, NULL);

/* create the initial time label and edit/display widget */
MainDialogInitTimeLabel = XtCreateManagedWidget("MainDialogInitTimeLabel",
labelWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Initial Display Time"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogLoadData); i++;
XtSetArg(args[i], XtNresize, FALSE); i++;
XtSetArg(args[i], XtNwidth, 160); i++;
XtSetValues(MainDialogInitTimeLabel, args, i);

MainDialogInitTimeValue = XtCreateManagedWidget("MainDialogInitTimeValue",
asciiTextWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Final Display Time"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogInitTimeLabel); i++;
XtSetArg(args[i], XtNresize, FALSE); i++;
XtSetArg(args[i], XtNwidth, 160); i++;
XtSetValues(MainDialogInitTimeLabel, args, i);

MainDialogPinTimeValue = XtCreateManagedWidget("MainDialogPinTimeValue",
asciiTextWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNborderWidth, 1); i++;
XtSetArg(args[i], XtNwidth, 100); i++;
XtSetArg(args[i], XtNtype, XawasciiString); i++;
XtSetArg(args[i], XtNeditType, XawtextEdit); i++;
XtSetArg(args[i], XtNfromVert, MainDialogInitTimeLabel); i++;
XtSetArg(args[i], XtNfromVert, MainDialogLoadData); i++;
XtSetValues(MainDialogPinTimeValue, args, i);

/* create the final time label and edit/display widget */
MainDialogPinTimeLabel = XtCreateManagedWidget("MainDialogPinTimeLabel",
labelWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "PinTimeLabel"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogPinTimeLabel); i++;
XtSetArg(args[i], XtNresize, FALSE); i++;
XtSetArg(args[i], XtNwidth, 160); i++;
XtSetValues(MainDialogPinTimeLabel, args, i);

MainDialogPinTimeValue = XtCreateManagedWidget("MainDialogPinTimeValue",
asciiTextWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNborderWidth, 1); i++;
XtSetArg(args[i], XtNwidth, 100); i++;
XtSetArg(args[i], XtNtype, XawasciiString); i++;
XtSetArg(args[i], XtNeditType, XawtextEdit); i++;
XtSetArg(args[i], XtNfromVert, MainDialogPinTimeLabel); i++;
XtSetArg(args[i], XtNfromVert, MainDialogPinTimeLabel); i++;
XtSetValues(MainDialogPinTimeValue, args, i);

/* create the scaling choice label and option selector */
MainDialogScaleLabel = XtCreateManagedWidget("MainDialogScaleLabel",
labelWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Scaling Type"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogPinTimeLabel); i++;
XtSetValues(MainDialogScaleLabel, args, i);

MainDialogScale = XtCreateManagedWidget("Power",
commandWidgetClass, MainDialogForm, NULL, 0);
changed_translations = XtParseTranslationTable(translations);
XtOverrideTranslations(MainDialogScale, changed_translations);
i = 0;
XtSetArg(args[i], XtNlabel, "Linear"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainDialogScaleLabel); i++;
XtSetArg(args[i], XtNresizeable, TRUE); i++;
XtSetValues(MainDialogScale, args, i);

Postproc Make Scale Buttons();

/* create the variable display choice label and option selector */
MainDialogVariableLabel = XtCreateManagedWidget("MainDialogVariableLabel",
labelWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtNlabel, "Variable to Display"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromVert, MainDialogScaleLabel); i++;
XtSetArg(args[i], XtNresizeable, TRUE); i++;
XtSetValues(MainDialogVariableLabel, args, i);

```

```

bond.c      Tue Sep 20 12:10:46 1994      4

XtSetValues(MainDialogVariableLabel, args, 1);

MainDialogVariable = XtCreateManagedWidget("Power",
      commandWidgetClass, MainDialogForm, NULL, 0);

changed translations = XtParseTranslationTable(translations2);
XtOverrideTranslations(MainDialogVariable, changed translations);

i = 0;
XtSetArg(args[i], XtN"label", "None"); i++;
XtSetArg(args[i], XtNborderWidth, 0); i++;
XtSetArg(args[i], XtNfromLeft, MainDialogVariableLabel); i++;
XtSetArg(args[i], XtNfromVert, MainDialogScaleLabel); i++;
XtSetArg(args[i], XtNisSelectable, TRUE); i++;
XtSetValues(MainDialogVariable, args, i);

PostProc_Make_Dds_Buttons();

/* create the button that pops up the phigs graphics */
MainToPhigs = XtCreateManagedWidget("MainToPhigs",
      commandWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtN"label", "ToPhigs"); i++;
XtSetArg(args[i], XtNfromLeft, NULL); i++;
XtSetArg(args[i], XtNfromVert, NULL); i++;
XtSetValues(MainToPhigs, args, i);
XtAddCallback(MainToPhigs, XtCallbackProc, Main_To_Phigs, NULL);

/* create a quit button */
MainCancel = XtCreateManagedWidget("MainCancel",
      commandWidgetClass, MainDialogForm, NULL, 0);
i = 0;
XtSetArg(args[i], XtN"label", "Quit"); i++;
XtSetArg(args[i], XtNfromLeft, NULL); i++;
XtSetArg(args[i], XtNfromVert, MainToPhigs); i++;
XtSetValues(MainCancel, args, i);
XtAddCallback(MainCancel, XtCallbackProc, Main_Cancel, NULL);

XtRealizeWidget(topLevel);

Init_selector(topLevel);

XtMainLoop();
}

/***** Main_To_Phigs - called when To Phigs Button is picked *****/
/***** Main_To_Phigs(Widget w) *****/

void Main_To_Phigs(Widget w)
{
    int setaps (double real *);
    extern Pint WorkstatID;
    extern Pint ModLabels, ConLabels;
    static Pint Lights[3] = {1, 2, 3};
    Pint list LightsOn, LightsOff;
    Pint list Matrix;
    static Pint3 Loc = {0.0, 0.0, 0.0};
    static Pint3 Shift1;
    static Pint3 Shift2;
    static Pint3 Scale = {1.0, 1.0, 1.0};
    Pint3 Scales;

```

```

Pint Error;
extern Widget topLevel;
Widget MainDialogForm, MainDialogLabel, MainDialogButton;
Widget MainCancel, MainToPhigs;
int i, index;
Arg args[10];
Pint wst, wst2, wst3;
Pedge flag; edge flag = PEDGE OFF;
Pint style; int style = PSTYLE SOLID;
Pint hidden surf = PHIGS ELSUR NONE NONE;
Pint dbl buff = PHIGS BUF DOUBLE;
Pint wst; wst state;
ModeType *traverseModes;
ConnectorType *traverseCons;
ModeType *fromMode, *toMode;
real nodes, nodes2, nodes3;
real cons1, cons2, cons3, cons4, cons5, cons6, cons7;
real centers, center2, center3;
real angles, angle2, angle3;
real length, length2;
real l2y, l2z;
extern ModeType *first Mode;
extern ConnectorType *first Connector;
Ptext align ModLabels;
Ptext align ConLabels;
Ppoint3 labelleft;
Pvec3 labeloffset;
char *ModName;
char *ConName;
FILE *fp;
int scale;
extern real scale factor;
extern real frange, erange, frange, arange, parange, qrange;
int Type;
int Power;
Pcolor Mode Color;
Pcolor Con Color;
int Mode index;

if (!Main Dialog_Check()) {
    return;
}

Arrange Data();
Arrange Signals();
Integrate Data();
Find_Maxes_Min();

set_colors();
init view mapping();

fp = fopen("bonddata", "w");

/* Check max coordinate size - match 1000 to our 5 unit phigs
   screen, unless maxcoord is > 1000 */
scale = Find_Scale();
if ((real) scale <= 1000.0)
    scale_factor = 10.0;
else
    scale_factor = (real) scale / 100.0;

popen phigs((char*) 0, PDEF MEM SIZE);

set up key();
set up time();

```


bond.c Tue Sep 20 12:10:46 1994 6

```

    polose_struct();
    Type = traverseNodes->type;
    if (Type == INT_MOVE) {
        CreateSphere((Pint) Mode.index);
        CreateSquare(Mode.Color);
        polose_struct();
        popen_struct(Scene);
        pscene_struct((Pint) Mode.index);
        polose_struct();
    }
    else {
        popen_struct((Pint) Mode.index);
        CreateSphere(12, 6, Mode.Color);
        polose_struct();
        popen_struct(Scene);
        pscene_struct((Pint) Mode.index);
        polose_struct();
    }
    popen_struct(Scene);
    post_view_init();
    polose_struct();
    traverseNodes = traverseNodes->Next;
    Mode.index++;
}

/* draw the connectors as different sized rectangles */
index = 10;
traverseCons = First Connector->Next;
while (traverseCons) {
    fromNode = traverseCons->from;
    toNode = traverseCons->to;
    cons1 = (real) fromNode->x;
    cons2 = (real) fromNode->y;
    cons1 = (real) fromNode->x;
    cons2 = (real) toNode->x;
    cons1 = (real) toNode->y;
    cons2 = (real) toNode->x;
    ConsLen = (traverseCons->name);
    length = sqrt((cons1-cons2)*(cons1-cons2) +
        (cons1-cons2)*(cons1-cons2)*(cons1-cons2));
    center = (cons1 + cons2) / 2;
    center = (cons1 + cons2) / 2;
    center = (cons1 + cons2) / 2;
    length = (length / scale_factor) - 3.0;
    Shift1.delta_x = center / scale_factor;
    Shift1.delta_y = center / scale_factor;
    Shift1.delta_x = center / scale_factor;
    Shift1.delta_x = center / scale_factor;
    angle = asin((cons2 - center)/(length/2.0)) * 180.0 / PI;
    l2 = sqrt((length/2.0)*(length/2.0) -
        (cons2 - center)*(cons2 - center));
    if (l2 == 0.0) {
        angles = 0.0;
    }
    else {
        angles = asin((cons2 - center)/(l2) * 180.0 / PI;
        if ((cons2 - center) < 0.0) angles = 180.0 - angles;
    }
    Power = traverseCons->power;
    switch (Power) {
        case GENERAL:
            Con Color = PowerGray;
            break;
        case MECHANICAL:
            Con Color = PowerGreen;
            break;
        case ROTATIONAL:
            Con Color = PowerBlue;
            break;
        case ELECTRICAL:
            Con Color = PowerBlue;
            break;
        case HYDRAULIC:
            Con Color = PowerPurple;
            break;
        case THERMAL:
            Con Color = PowerYellow;
            break;
    }
    Type = traverseCons->type;
    if (Type == SIGNAL) {
        Create_Connector(length, index, Con Color);
    }
    else {
        Create_Cylinder(length, index, Con Color);
    }
    popen_struct(Scene);
    Build_tran_Matrix(Alloc, Shift1, 0 deg, -angle deg, angles deg, dScale,
        Error, Matrix);
    post_local_tran_Matrix_PTYPE_REPLACE();
    pscene_struct((Pint) index);
    polose_struct();
    index++;
    traverseCons = traverseCons->Next;
}

wet = phigs vs type create(phigs vs type x tool,
    PHIGS COLOR TABLE SIZE, 0,
    PHIGS X MOV MOVE, PHIGS MOV DOUBLE CMAP,
    PHIGS TOOL X, 10, PHIGS TOOL Y, 10,
    PHIGS TOOL HEIGHT, 750, PHIGS TOOL WIDTH, 750, 0);
popen ws(Workstaid, (void *)NULL, wet);
popen ws(Workstaid, (void *)NULL, wet);
popen ws(Workstaid, (void *)NULL, wet);
wet2 = phigs vs type create(phigs vs type x tool,
    PHIGS COLOR TABLE SIZE, 0,
    PHIGS X MOV MOVE, PHIGS MOV DOUBLE CMAP,
    PHIGS TOOL X, 10, PHIGS TOOL Y, 10,
    PHIGS TOOL HEIGHT, 300, PHIGS TOOL WIDTH, 300, 0);
popen ws(Workstaid2, (void *)NULL, wet2);
popen ws(Workstaid2, (void *)NULL, wet2);
wet3 = phigs vs type create(phigs vs type x tool,
    PHIGS COLOR TABLE SIZE, 0,
    PHIGS X MOV MOVE, PHIGS MOV DOUBLE CMAP,
    PHIGS TOOL X, 10, PHIGS TOOL Y, 10,
    PHIGS TOOL HEIGHT, 300, PHIGS TOOL WIDTH, 300, 0);
popen ws(Workstaid3, (void *)NULL, wet3);
popen ws(Workstaid3, (void *)NULL, wet3);
DefineLightSource(Workstaid);
post_hlwr_mode(Workstaid, PHIGS HLWR MODE ZBUFF);
post disp upd at (Workstaid, POWER WAIT, PHIGS NIVE);
set view_rep();
init_input();

```

```

pPost_struct(WorkatID, Scene, 1.0);
pPost_struct(WorkatID, ModLabels, 1.0);
pPost_struct(WorkatID, ConLabels, 1.0);

pPost_disp_upd_at(1, FORWEN_AAP, FROME_MIVE);
pPost_edit_mode(PEDIT_REPLACE);

for (i = 0; i < (nms - 1); i++) {
    pPost_ws_at(1, nms - i);
    if (ws_state == PMS_ST_WSOPT)
        pPost_disp_upd_at(1, FORWEN_MAIT, FROME_MIVE);
    if (check_input() != QUIT) {
        pPost_ws_at(1, nms - i);
        if (ws_state == PMS_ST_WSOPT)
            pPost_disp_upd_at(1, FORWEN_AAP, FROME_MIVE);
        } else {
            pPost_ws_at(1, nms - i);
            if (ws_state == PMS_ST_WSOPT)
                pPost_ws(WorkatID);
                pPost_ws(WorkatID);
                pPost_ws(WorkatID);
                pPost_ws(1);
                break;
            }
        }
    }

/*****
/**** set up time ****
/****
/****

void set_up_time()
{
    extern Pint timestruc;
    char TEMP1[10];
    Pint Point;
    int BoldnessRef = -4;
    pPost_align_CenterTop = (PMOR_CTR, PVENT_TOP);
    pPost_struct(timestruc);

    switch (Cur Var) {
        case EFFORT: sprintf(TEMP1, "Variable Type = Effort"); break;
        case FLOW: sprintf(TEMP1, "Variable Type = Flow"); break;
        case POWER: sprintf(TEMP1, "Variable Type = Power"); break;
        case MOMENTUM: sprintf(TEMP1, "Variable Type = Momentum"); break;
        case DISPLACEMENT: sprintf(TEMP1, "Variable Type = Displacement"); break;
        case ENERGY: sprintf(TEMP1, "Variable Type = Energy"); break;
        case SIGNAL: sprintf(TEMP1, "Variable Type = Signal"); break;
        case NONE: sprintf(TEMP1, "Variable Type = None"); break;
    }
    pText(aPoint, TEMP1);

    pPost_char_ht(0.04);
    time = actlib4 1.dcttr + (real) timestage + actlib4 1.tin;
    Point.x = 0.5, Point.y = 0.6;
    sprintf(TEMP2, "Time = %7.3f", time);
    pText(aPoint, TEMP2);
    pClose_struct();
}

/****
/**** set up key ****
/****
/****

void set_up_key()
{
    extern int Cur Var;
    extern Pint keystruc;
    extern real orange, frange, prange, grange, parange, arange, strange;
    extern real emining, fmiring, pmiring, gmiring, pmiring, emining;
    extern real emain, fmain, pmain, gmain, pmain, emain;
    extern real emax, fmax, pmax, gmax, pmmax, emax;
    extern Pcolor PCViolet, PCBlue, PCLightBlue, PCGreen, PCYellowDn,
    Pcolor PVOiolet, PBlue, PLightBlue, PGreen, PYellowDn,
    Pcolor Patch Color;
    pPost_align_CenterTop = (PMOR_CTR, PVENT_TOP);
    int BoldnessRef = -4;
    real labelrange, labelvalue;
    char TEMP[10];
    int i;
    Pint Point;
    int pos index;
    real exp, delta, min, max, rangemin;

```

```

bond.c      Tue Sep 20 12:10:46 1994

/** get colors in correct format **/

pViolet.val.general.x = pViolet.direct.rgb.red;
pViolet.val.general.y = pViolet.direct.rgb.green;
pViolet.val.general.z = pViolet.direct.rgb.blue;
pViolet.val.ind = pViolet.ind;
pViolet.type = PMODEL_RGB;
pBlue.val.general.x = pBlue.direct.rgb.red;
pBlue.val.general.y = pBlue.direct.rgb.green;
pBlue.val.general.z = pBlue.direct.rgb.blue;
pBlue.val.ind = pBlue.ind;
pBlue.type = PMODEL_RGB;
pLBlue.val.general.x = pLBlue.direct.rgb.red;
pLBlue.val.general.y = pLBlue.direct.rgb.green;
pLBlue.val.ind = pLBlue.ind;
pLBlue.type = PMODEL_RGB;
pGreen.val.general.x = pGreen.direct.rgb.red;
pGreen.val.general.y = pGreen.direct.rgb.green;
pGreen.val.general.z = pGreen.direct.rgb.blue;
pGreen.val.ind = pGreen.ind;
pGreen.type = PMODEL_RGB;
pYellow.val.general.x = pYellow.direct.rgb.red;
pYellow.val.general.y = pYellow.direct.rgb.green;
pYellow.val.ind = pYellow.ind;
pYellow.type = PMODEL_RGB;
pOld.val.general.x = pOld.direct.rgb.red;
pOld.val.general.y = pOld.direct.rgb.green;
pOld.val.ind = pOld.ind;
pOld.type = PMODEL_RGB;
pOrange.val.general.x = pOrange.direct.rgb.red;
pOrange.val.general.y = pOrange.direct.rgb.green;
pOrange.val.ind = pOrange.ind;
pOrange.type = PMODEL_RGB;
pPink.val.general.x = pPink.direct.rgb.red;
pPink.val.general.y = pPink.direct.rgb.green;
pPink.val.ind = pPink.ind;
pPink.type = PMODEL_RGB;
pRed.val.general.x = pRed.direct.rgb.red;
pRed.val.general.y = pRed.direct.rgb.green;
pRed.val.ind = pRed.ind;
pRed.type = PMODEL_RGB;
pOpen_struct(keystruc);
pset_text_align(SCenterTop);
pset_char_ht(0.03);

/** Write the label ****/
Point.x = 0.5; Point.y = 0.95;
pset(EndPoint, "key");

pset_text_align(SCenterTop);
pset_char_ht(0.03);

switch (Cur Var) {
case EPPOINT: labelrange = orange; max = emax; min = emin;
rangemin = eminar; break;
case FLOW: labelrange = fringe; max = fmax; min = fmin;
rangemin = fairang; break;
case POWER: labelrange = range; max = pmaz; min = pmin;
rangemin = pminar; break;
case DISPLACEMENT: labelrange = qrange; max = qmax; min = qmin;
rangemin = qmar; break;
case MOMENTUM: labelrange = pmarange; max = pmamax; min = pmin;
rangemin = pmamar; break;
case SSIGMA: labelrange = srange; max = smax; min = smin;
rangemin = sminar; break;
case ENERGY: labelrange = erange; max = emax; min = emin;
rangemin = eminar; break;
}

Point.x = 0.3;
pset_char_ht(0.02);

if (Scale Var == LOG) delta = (log10(max) - log10(min)) / 10;
if (Scale Var == ABSLOG) delta = (log10(labelrange) - log10(rangemin)) / 10;

for (i = 0; i < 11; i++) {
if (Scale Var == LIN) {
labelvalue = i * labelrange * 0.1;
}
else if (Scale Var == LOG) {
exp = i * delta * log10(min);
labelvalue = pow(10.0, exp);
}
else {
exp = i * delta * log10(rangemin);
labelvalue = pow(10.0, exp);
}
printf(TMP, "%5.2E", labelvalue);
Point.y = (10-i) * 0.08 + 0.1;
pset(EndPoint, TMP);
}
pclose_struct();

/** Draw the fill areas with the appropriate colors **/
for (pos_index = 0; pos_index < 10; pos_index++) {
switch (pos_index) {
case 0: Patch_Color = pViolet; break;
case 1: Patch_Color = pBlue; break;
case 2: Patch_Color = pLBlue; break;
case 3: Patch_Color = pGreen; break;
case 4: Patch_Color = pYellow; break;
case 5: Patch_Color = pOld; break;
case 6: Patch_Color = pOrange; break;
case 7: Patch_Color = pPink; break;
case 8: Patch_Color = pRed; break;
case 9: Patch_Color = pOpen; break;
}
create_patch(pos_index);
popen_struct(keystruc);
pset_int_color(patch_Color);
pset_int_style(PSTYLE_SOLID);
pset_struct(pos_index+500);
pclose_struct();
}

```

bond.c Tue Sep 20 12:10:46 1994 9

```

    )
    return maxcoord;
}

/***** create patch *****/
/***** Create Cylinder *****/

void create_patch(int pos index)
{
    Ppoint Points[4];
    Ppoint list PointList;
    popen_struct(pos index+500);

    Points[0].x = 0.3;
    Points[0].y = (9-pos_index)*0.08 + 0.12;
    Points[1].x = 0.3;
    Points[1].y = (9-pos_index)*0.08 + 0.15;
    Points[2].x = 0.7;
    Points[2].y = (9-pos_index)*0.08 + 0.15;
    Points[3].x = 0.7;
    Points[3].y = (9-pos_index)*0.08 + 0.12;

    PointList.num_points = 4;
    PointList.points = Points;
    pfill_area(&PointList);
    polose_struct();
}

/***** Find Scale *****/
/***** Find Scale *****/

int Find_Scale()
{
    extern ModeType *First_Mode;
    ModeType *traverseModes;
    int maxcoord;
    int nx, ny, nz;

    traverseModes = First_Mode->Next;
    maxcoord = 0;
    while (traverseModes) {
        nx = traverseModes->x;
        if (nx > 0) {
            if (nx > maxcoord) maxcoord = nx;
        }
        else {
            if (-nx > maxcoord) maxcoord = -nx;
        }
        ny = traverseModes->y;
        if (ny > 0) {
            if (ny > maxcoord) maxcoord = ny;
        }
        else {
            if (-ny > maxcoord) maxcoord = -ny;
        }
        nz = traverseModes->z;
        if (nz > 0) {
            if (nz > maxcoord) maxcoord = nz;
        }
        else {
            if (-nz > maxcoord) maxcoord = -nz;
        }
        traverseModes = traverseModes->Next;
    }
}

    )
    return maxcoord;
}

/***** Create Cylinder *****/
/***** Create Cylinder *****/

void Create_Cylinder(real length, int index, Pcolor PColor)
{
    Pfloat Cyl[3][51][51];
    Pfloat Cone[3][51][51];
    Pint Longitude, Height, I;
    Pint NumLong, NumHeight, ConeNumHeight;
    Pfloat Theta, R;
    Ppoint Vertices[4];
    Pvec3 Norms[4];
    Pfacet data3 Facets;
    Pfacet vdata list3 VertexSet;
    Pfacet vdata arr3 VertexList;
    Pcolor FacetColorRed, FacetColorBlue;
    Ptnorm3 pointnorms[4];
    Pint sum_areas;
    static Prgb Red;
    static Prgb LightBlue;
    real r = 0.4;
    real l2, rc, tempr;
    extern Pint WorksetID;
    static Pint Lights[3] = {1, 2, 3};
    Pint list LightsOn, LightsOff;
    Red.red = 0.7, Red.green = 0.6, Red.blue = 0.6;
    LightBlue.red = 0.5, LightBlue.green = 0.5, LightBlue.blue = 0.7;

    l2 = length / 2.0;
    rc = 2.0 * r;

    popen_struct((Pint) index);

    /***** define the cylinder's data base *****/

    NumLong = 12;
    NumHeight = 6;
    for (Longitude = 0; Longitude <= NumLong; Longitude++) {
        Theta = 360.0 * Longitude / NumLong;
        for (Height = 0; Height <= NumHeight; Height++) {
            R = Height / NumHeight * (length-1.0) - l2;
            Cyl[0][Longitude][Height] = R;
            Cyl[1][Longitude][Height] = r * cos(Theta deg);
            Cyl[2][Longitude][Height] = r * sin(Theta deg);
        }
    }

    /***** create the cylinder out of unicolor patches *****/

    FacetColorRed.ind = 1;
    FacetColorRed.direct.rgb = Red;
    FacetColorBlue.ind = 1;
    FacetColorBlue.direct.rgb = LightBlue;

    peest view ind(1);
    peest face cull mode(PCULL BACKFACE);
    peest int style(PFILL SOLID);
    peest ref1 eqn(PREFL AND DIFF SPEC);
    peest int shad meth(PED COLOUR);

```

bond.c Tue Sep 20 12:10:46 1994 10

```

Method:coneProperties (WorkstatID,
    1.0, 1.0, 1.0,
    1.0, 1.0, 1.0,
    20.0);

LightOn.num into = 3;
LightOn.ints = Lights;
LightOff.num into = 0;
pvec_light_ave_state(slightOn, slightOff);

num_areas = 1;
for (Height = 0; Height < NumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cone[0][Longitude][Height];
        Vertices[0].y = Cone[1][Longitude][Height];
        Vertices[0].z = Cone[2][Longitude][Height];
        Vertices[1].x = Cone[0][Longitude+1][Height];
        Vertices[1].y = Cone[1][Longitude+1][Height];
        Vertices[1].z = Cone[2][Longitude+1][Height];
        Vertices[2].x = Cone[0][Longitude+2][Height];
        Vertices[2].y = Cone[1][Longitude+2][Height];
        Vertices[2].z = Cone[2][Longitude+2][Height];
        Vertices[3].x = Cone[0][Longitude+3][Height];
        Vertices[3].y = Cone[1][Longitude+3][Height];
        Vertices[3].z = Cone[2][Longitude+3][Height];
        Normals[0].delta_x =
            (Cone[1][Longitude][Height])*(Cone[1][Longitude+1][Height]) +
            (Cone[2][Longitude][Height])*(Cone[2][Longitude+1][Height]);
        temp = Normals[0].delta_x + Normals[0].delta_x / temp;
        Normals[0].delta_x = Normals[0].delta_x / temp;
        Normals[0].delta_y = Cone[1][Longitude][Height] / temp;
        Normals[0].delta_z = Cone[2][Longitude][Height] / temp;
        Normals[1].delta_x =
            (Cone[1][Longitude+1][Height])*(Cone[1][Longitude+2][Height]) +
            (Cone[2][Longitude+1][Height])*(Cone[2][Longitude+2][Height]);
        temp = Normals[1].delta_x + Normals[1].delta_x / temp;
        Normals[1].delta_x = Normals[1].delta_x / temp;
        Normals[1].delta_y = Cone[1][Longitude+1][Height] / temp;
        Normals[1].delta_z = Cone[2][Longitude+1][Height] / temp;
        Normals[2].delta_x =
            (Cone[1][Longitude+2][Height])*(Cone[1][Longitude+3][Height]) +
            (Cone[2][Longitude+2][Height])*(Cone[2][Longitude+3][Height]);
        temp = Normals[2].delta_x + Normals[2].delta_x / temp;
        Normals[2].delta_x = Normals[2].delta_x / temp;
        Normals[2].delta_y = Cone[1][Longitude+2][Height] / temp;
        Normals[2].delta_z = Cone[2][Longitude+2][Height] / temp;
        Normals[3].delta_x =
            (Cone[1][Longitude+3][Height])*(Cone[1][Longitude+4][Height]) +
            (Cone[2][Longitude+3][Height])*(Cone[2][Longitude+4][Height]);
        temp = Normals[3].delta_x + Normals[3].delta_x / temp;
        Normals[3].delta_x = Normals[3].delta_x / temp;
        Normals[3].delta_y = Cone[1][Longitude+3][Height] / temp;
        Normals[3].delta_z = Cone[2][Longitude+3][Height] / temp;
        pointnorms[0].point = Vertices[0];
        pointnorms[1].point = Vertices[1];
        pointnorms[2].point = Vertices[2];
        pointnorms[3].point = Vertices[3];
        pointnorms[0].norm = Normals[0];
        pointnorms[1].norm = Normals[1];
        pointnorms[2].norm = Normals[2];
        pointnorms[3].norm = Normals[3];
        VertextList.ptnorms = pointnorms;
        VertextList.vertex_data = VertextList;
        VertextList.num_vertices = 4;
        Facets.colr = PCOLOR;
        pfill area set) data(PFACET COLOUR, PFEDGE NONE, PFVERT COORD NORMAL,
        PMODEL RGB, dFacets, num_areas, NULL, &VertextList);
    }
}

/* create the cone out of unicolor patches */
for (Height = 0; Height < NumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cone[0][Longitude][Height];
        Vertices[0].y = Cone[1][Longitude][Height];
        Vertices[0].z = Cone[2][Longitude][Height];
        Vertices[1].x = Cone[0][Longitude+1][Height];
        Vertices[1].y = Cone[1][Longitude+1][Height];
        Vertices[1].z = Cone[2][Longitude+1][Height];
        Vertices[2].x = Cone[0][Longitude+2][Height];
        Vertices[2].y = Cone[1][Longitude+2][Height];
        Vertices[2].z = Cone[2][Longitude+2][Height];
        Vertices[3].x = Cone[0][Longitude+3][Height];
        Vertices[3].y = Cone[1][Longitude+3][Height];
        Vertices[3].z = Cone[2][Longitude+3][Height];
        Normals[0].delta_x =
            (Cone[1][Longitude][Height])*(Cone[1][Longitude+1][Height]) +
            (Cone[2][Longitude][Height])*(Cone[2][Longitude+1][Height]);
        temp = Normals[0].delta_x + Normals[0].delta_x / temp;
        Normals[0].delta_x = Normals[0].delta_x / temp;
        Normals[0].delta_y = Cone[1][Longitude][Height] / temp;
        Normals[0].delta_z = Cone[2][Longitude][Height] / temp;
        Normals[1].delta_x =
            (Cone[1][Longitude+1][Height])*(Cone[1][Longitude+2][Height]) +
            (Cone[2][Longitude+1][Height])*(Cone[2][Longitude+2][Height]);
        temp = Normals[1].delta_x + Normals[1].delta_x / temp;
        Normals[1].delta_x = Normals[1].delta_x / temp;
        Normals[1].delta_y = Cone[1][Longitude+1][Height] / temp;
        Normals[1].delta_z = Cone[2][Longitude+1][Height] / temp;
        Normals[2].delta_x =
            (Cone[1][Longitude+2][Height])*(Cone[1][Longitude+3][Height]) +
            (Cone[2][Longitude+2][Height])*(Cone[2][Longitude+3][Height]);
        temp = Normals[2].delta_x + Normals[2].delta_x / temp;
        Normals[2].delta_x = Normals[2].delta_x / temp;
        Normals[2].delta_y = Cone[1][Longitude+2][Height] / temp;
        Normals[2].delta_z = Cone[2][Longitude+2][Height] / temp;
        Normals[3].delta_x =
            (Cone[1][Longitude+3][Height])*(Cone[1][Longitude+4][Height]) +
            (Cone[2][Longitude+3][Height])*(Cone[2][Longitude+4][Height]);
        temp = Normals[3].delta_x + Normals[3].delta_x / temp;
        Normals[3].delta_x = Normals[3].delta_x / temp;
        Normals[3].delta_y = Cone[1][Longitude+3][Height] / temp;
        Normals[3].delta_z = Cone[2][Longitude+3][Height] / temp;
        pointnorms[0].point = Vertices[0];
        pointnorms[1].point = Vertices[1];
        pointnorms[2].point = Vertices[2];
        pointnorms[3].point = Vertices[3];
        pointnorms[0].norm = Normals[0];
        pointnorms[1].norm = Normals[1];
        pointnorms[2].norm = Normals[2];
        pointnorms[3].norm = Normals[3];
        VertextList.ptnorms = pointnorms;
        VertextList.vertex_data = VertextList;
        VertextList.num_vertices = 4;
        Facets.colr = PCOLOR;
        pfill area set) data(PFACET COLOUR, PFEDGE NONE, PFVERT COORD NORMAL,
        PMODEL RGB, dFacets, num_areas, NULL, &VertextList);
    }
}

/* create the back surface to the cone */
for (Height = 0; Height < NumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Theta = 360.0 * Longitude / NumLong;
        for (Height = 0; Height < NumHeight; Height++) {
            R = Height / NumHeight;
            Cone[0][Longitude][Height] = R * 12 - 1.0;
            Cone[1][Longitude][Height] =
                R * (1.0 - R) * cos(Theta deg);
            Cone[2][Longitude][Height] =
                R * (1.0 - R) * sin(Theta deg);
        }
    }
}

```

```

for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Vertices[0].x = Cos[0][Longitude+1][0];
    Vertices[0].y = Cos[1][Longitude+1][0];
    Vertices[0].z = Cos[2][Longitude+1][0];
    Vertices[1].x = Cos[0][Longitude][0];
    Vertices[1].y = Cos[1][Longitude][0];
    Vertices[1].z = Cos[2][Longitude][0];
    Vertices[2].x = Cos[0][Longitude+1][NumHeight];
    Vertices[2].y = Cos[1][Longitude+1][NumHeight];
    Vertices[2].z = Cos[2][Longitude+1][NumHeight];
    Vertices[3].x = Cos[0][Longitude][NumHeight];
    Vertices[3].y = Cos[1][Longitude][NumHeight];
    Vertices[3].z = Cos[2][Longitude][NumHeight];
    Vertices[4].x = Cos[0][Longitude+1][NumHeight];
    Vertices[4].y = Cos[1][Longitude+1][NumHeight];
    Vertices[4].z = Cos[2][Longitude+1][NumHeight];
    Norms[0].delta_x = -1.0;
    Norms[0].delta_y = 0.0;
    Norms[0].delta_z = 0.0;
    Norms[1].delta_x = -1.0;
    Norms[1].delta_y = 0.0;
    Norms[1].delta_z = 0.0;
    Norms[2].delta_x = -1.0;
    Norms[2].delta_y = 0.0;
    Norms[2].delta_z = 0.0;
    Norms[3].delta_x = -1.0;
    Norms[3].delta_y = 0.0;
    Norms[3].delta_z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    Verticelist.ptnorms = pointnorms;
    Verticelist.vertices.data = Verticelist;
    Verticelist.num_vertices = 4;
    Facets.colr = FColor;
    pfill_area_set3_data(PFACET_COLOUR, FEDGE_NONE, PVERT_COORD_NORMAL,
        PMODEL_RGB, aFacets, num_areas, NULL, aVertexSet);
}

polose_struct();

/*****
/*****
/***** Recreate Connector *****/
/*****
void Recreate Connector(Pint struono, real length, Poval FColor)
{
    Pfloat Rect[3][13];
    Prect3 Norms[4];
    Ppoint3 Vertices[4];
    Pfloat_data3 Facets;
    Pfloat_vdata_list1 Verticelist;
    Pfloat_vdata_arr3 Verticelist;
    Poval FacetColorRed;
    Pint num_areas;
    Ppntnorm3 pointnorms[4];
    Prgb Red;
    int i;
    real l2;
    FILE *fp;

    fp = fopen("bonddata2", "w");
    popen_struct(struono);
    pempty_struct(struono);
    pset_elem_ptr(0);
    pset_edit_mode(PEDIT_INSERT);

    l2 = length/2.0;

    Rect[0][0] = -l2; /* Top points */
    Rect[1][0] = 0.2;
    Rect[2][0] = 0.2;
    Rect[0][1] = -l2;
    Rect[1][1] = -0.2;
    Rect[2][1] = 0.2;
    Rect[0][2] = l2 - 1.0;
    Rect[1][2] = -0.2;
    Rect[2][2] = 0.2;
    Rect[0][3] = l2 - 1.0;
}

```

bond.c Tue Sep 20 12:10:46 1994 12

```

Rect[1][3] = 0.2;
Rect[2][3] = 0.2;
Rect[0][4] = -12; /* Bottom points */
Rect[1][4] = 0.2;
Rect[2][4] = 0.2;
Rect[0][5] = -12;
Rect[1][5] = 0.2;
Rect[2][5] = 0.2;
Rect[0][6] = 12 - 1.0;
Rect[1][6] = -0.2;
Rect[2][6] = -0.2;
Rect[0][7] = 12 - 1.0;
Rect[1][7] = 0.2;
Rect[2][7] = -0.2;

/* arrow points */
Rect[0][8] = 12 - 1.0;
Rect[1][8] = -0.4;
Rect[2][8] = 0.4;
Rect[0][9] = 12 - 1.0;
Rect[1][9] = 0.4;
Rect[2][9] = 0.4;
Rect[0][10] = 12 - 1.0;
Rect[1][10] = -0.4;
Rect[2][10] = -0.4;
Rect[0][11] = 12 - 1.0;
Rect[1][11] = 0.4;
Rect[2][11] = -0.4;
Rect[0][12] = 12;
Rect[1][12] = 0.0;
Rect[2][12] = 0.0;

num_areas = 1;

/* Set up top side */
i = 0;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[0][1];
Vertices[0].z = Rect[2][1];
Vertices[1].x = Rect[0][1+1];
Vertices[1].y = Rect[1][1+1];
Vertices[1].z = Rect[2][1+1];
Vertices[2].x = Rect[0][1+1];
Vertices[2].y = Rect[1][1+1];
Vertices[2].z = Rect[2][1+1];
Vertices[3].x = Rect[0][1+2];
Vertices[3].y = Rect[1][1+2];
Vertices[3].z = Rect[2][1+2];
Vertices[4].x = Rect[0][1+3];
Vertices[4].y = Rect[1][1+3];
Vertices[4].z = Rect[2][1+3];
Norms[0].delta.x = 0.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = 1.0;
Norms[1].delta.x = 0.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = 1.0;
Norms[2].delta.x = 0.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = 1.0;
Norms[3].delta.x = 0.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = 1.0;
Norms[4].delta.x = 0.0;
Norms[4].delta.y = 0.0;
Norms[4].delta.z = 1.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[4].point = Vertices[4];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
pointnorms[4].norm = Norms[4];
VertexList.ptnorms = pointnorms;
VertexList.vertices = Vertices;
VertexList.num_vertices = 5;
Recolor.col = FColor;
Recolor.col = FColor;
pfill_area.set3 data(PFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, aFacets, num_areas, NULL, aVertexSet);
/* Bottom side */
i = 4;
Vertices[0].x = Rect[0][1+3];
Vertices[0].y = Rect[1][1+3];
Vertices[0].z = Rect[2][1+3];
Vertices[1].x = Rect[0][1+2];
Vertices[1].y = Rect[1][1+2];
Vertices[1].z = Rect[2][1+2];
Vertices[2].x = Rect[0][1+1];
Vertices[2].y = Rect[1][1+1];
Vertices[2].z = Rect[2][1+1];
Vertices[3].x = Rect[0][1+0];
Vertices[3].y = Rect[1][1+0];
Vertices[3].z = Rect[2][1+0];
Norms[0].delta.x = 0.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = -1.0;
Norms[1].delta.x = 0.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = -1.0;
Norms[2].delta.x = 0.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = -1.0;
Norms[3].delta.x = 0.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = -1.0;
Norms[4].delta.x = 0.0;
Norms[4].delta.y = 0.0;
Norms[4].delta.z = -1.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[4].point = Vertices[4];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
pointnorms[4].norm = Norms[4];
VertexList.ptnorms = pointnorms;
VertexList.vertices = Vertices;
VertexList.num_vertices = 5;
Recolor.col = FColor;
Recolor.col = FColor;
pfill_area.set3 data(PFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, aFacets, num_areas, NULL, aVertexSet);
/* Left side */
i = 0;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[0].z = Rect[2][1];
Vertices[1].x = Rect[0][1+1];
Vertices[1].y = Rect[1][1+1];
Vertices[1].z = Rect[2][1+1];
Vertices[2].x = Rect[0][1+2];
Vertices[2].y = Rect[1][1+2];
Vertices[2].z = Rect[2][1+2];
Vertices[3].x = Rect[0][1+3];
Vertices[3].y = Rect[1][1+3];
Vertices[3].z = Rect[2][1+3];
Norms[0].delta.x = 0.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = 1.0;
Norms[1].delta.x = 0.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = 1.0;
Norms[2].delta.x = 0.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = 1.0;
Norms[3].delta.x = 0.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = 1.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[4].point = Vertices[4];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
pointnorms[4].norm = Norms[4];
VertexList.ptnorms = pointnorms;
VertexList.vertices = Vertices;
VertexList.num_vertices = 5;
Recolor.col = FColor;
Recolor.col = FColor;
pfill_area.set3 data(PFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, aFacets, num_areas, NULL, aVertexSet);

```



```

bond.c      Tue Sep 20 12:10:46 1994      13

Norms[0].delta_x = -1.0;
Norms[0].delta_y = 0.0;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = -1.0;
Norms[1].delta_y = 0.0;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = -1.0;
Norms[2].delta_y = 0.0;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = -1.0;
Norms[3].delta_y = 0.0;
Norms[3].delta_z = 0.0;

pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];

VertexList.pnorms = pointnorms;
VertexList.vertex_data = VertexList;
VertexList.num_vertices = 4;
Facelets.colr = FColor;

fill all area set3 data (SPACET COLOUR, PEDGE MONS, FVERT COORD NORMAL,
PRODEL ROB, dFacelets, num_areas, NULL, aVertexSet);

Right side */
i = 2;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[0].z = Rect[2][1];
Vertices[1].x = Rect[0][1+d];
Vertices[1].y = Rect[1][1+d];
Vertices[1].z = Rect[2][1+d];
Vertices[2].x = Rect[0][1+d];
Vertices[2].y = Rect[0][1+d];
Vertices[2].z = Rect[1][1+d];
Vertices[3].x = Rect[0][1+d];
Vertices[3].y = Rect[1][1+d];
Vertices[3].z = Rect[2][1+d];
Norms[0].delta_x = 1.0;
Norms[0].delta_y = 0.0;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = 1.0;
Norms[1].delta_y = 0.0;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = 1.0;
Norms[2].delta_y = 0.0;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = 1.0;
Norms[3].delta_y = 0.0;
Norms[3].delta_z = 0.0;

pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];

VertexList.pnorms = pointnorms;
VertexList.vertex_data = VertexList;
VertexList.num_vertices = 4;
Facelets.colr = FColor;

fill all area set3 data (SPACET_COLOUR, PEDGE MONS, FVERT COORD NORMAL,
PRODEL ROB, dFacelets, num_areas, NULL, aVertexSet);

```

```

Norms[3].delta y = 1.0;
Norms[3].delta x = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Pcset.cclr = KColor;
pfill area set3 data(PFACCT COLOR, PEDGE NORM, PVERT COORD NORMAL,
PFACCT RGB, aFacets, num areas, NULL, aVertexset);

/* Top arrow side */
i = 0;
Vertices[0].x = Rect[0][1+1];
Vertices[0].y = Rect[1][1+1];
Vertices[0].z = Rect[2][1+1];
Vertices[1].x = Rect[0][1];
Vertices[1].y = Rect[1][1];
Vertices[1].z = Rect[2][1];
Vertices[2].x = Rect[0][1+2];
Vertices[2].y = Rect[1][1+2];
Vertices[2].z = Rect[2][1+2];
Vertices[3].x = Rect[0][1+3];
Vertices[3].y = Rect[1][1+3];
Vertices[3].z = Rect[2][1+3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Pcset.cclr = KColor;
pfill area set3 data(PFACCT COLOR, PEDGE NORM, PVERT COORD NORMAL,
PFACCT RGB, aFacets, num areas, NULL, aVertexset);

/* Bottom arrow side */
i = 10;
Vertices[0].x = Rect[0][11];
Vertices[0].y = Rect[1][11];
Vertices[0].z = Rect[2][11];
Vertices[1].x = Rect[0][1+11];
Vertices[1].y = Rect[1][1+11];
Vertices[1].z = Rect[2][1+11];
Vertices[2].x = Rect[0][1+12];
Vertices[2].y = Rect[1][1+12];
Vertices[2].z = Rect[2][1+12];
Vertices[3].x = Rect[0][1+13];
Vertices[3].y = Rect[1][1+13];
Vertices[3].z = Rect[2][1+13];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Pcset.cclr = KColor;
pfill area set3 data(PFACCT COLOR, PEDGE NORM, PVERT COORD NORMAL,
PFACCT RGB, aFacets, num areas, NULL, aVertexset);

```

bond.c Tue Sep 20 12:10:46 1994 15

```

    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertexList.ptnorms = pointnorms;
    VertexSet.vertex_data = VertexList;
    VertexSet.num_vertices = 4;
    Facets.colr = FColor;

    pfill_area.set3 data(PFACET_COLOUR, PFEDGE_NORMS, PVRTX_COORD_NORMAL,
        PFMODEL_RGB, AFacets, num_areas, NULL, &VertexSet);

/* Front arrow side */
1 = 0;
    Vertices[0].x = Rect[0][1];
    Vertices[0].y = Rect[1][1];
    Vertices[0].z = Rect[2][1];
    Vertices[1].x = Rect[0][1+2];
    Vertices[1].y = Rect[1][1+2];
    Vertices[1].z = Rect[2][1+2];
    Vertices[2].x = Rect[0][1+4];
    Vertices[2].y = Rect[1][1+4];
    Vertices[2].z = Rect[2][1+4];
    Vertices[3].x = Rect[0][1+4];
    Vertices[3].y = Rect[1][1+4];
    Vertices[3].z = Rect[2][1+4];
    Norms[0].delta_x = 0.37;
    Norms[0].delta_y = -0.93;
    Norms[0].delta_z = 0.0;
    Norms[1].delta_x = 0.37;
    Norms[1].delta_y = -0.93;
    Norms[1].delta_z = 0.0;
    Norms[2].delta_x = 0.37;
    Norms[2].delta_y = -0.93;
    Norms[2].delta_z = 0.0;
    Norms[3].delta_x = 0.37;
    Norms[3].delta_y = -0.93;
    Norms[3].delta_z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertexList.ptnorms = pointnorms;
    VertexSet.vertex_data = VertexList;
    VertexSet.num_vertices = 4;
    Facets.colr = FColor;

    pfill_area.set3 data(PFACET_COLOUR, PFEDGE_NORMS, PVRTX_COORD_NORMAL,
        PFMODEL_RGB, AFacets, num_areas, NULL, &VertexSet);

/* arrow face up */
1 = 2;
    Vertices[0].x = Rect[0][1+4];
    Vertices[0].y = Rect[1][1+4];
    Vertices[0].z = Rect[2][1+4];
    Vertices[1].x = Rect[0][1+7];
    Vertices[1].y = Rect[1][1+7];
    Vertices[1].z = Rect[2][1+7];
    Vertices[2].x = Rect[0][1+4];
    Vertices[2].y = Rect[1][1+4];
    Vertices[2].z = Rect[2][1+4];
    Vertices[3].x = Rect[0][1];
    Vertices[3].y = Rect[1][1];
    Vertices[3].z = Rect[2][1];
    Norms[0].delta_x = -1.0;
    Norms[0].delta_y = 0.0;
    Norms[0].delta_z = 0.0;
    Norms[1].delta_x = -1.0;
    Norms[1].delta_y = 0.0;
    Norms[1].delta_z = 0.0;
    Norms[2].delta_x = -1.0;
    Norms[2].delta_y = 0.0;
    Norms[2].delta_z = 0.0;
    Norms[3].delta_x = -1.0;
    Norms[3].delta_y = 0.0;
    Norms[3].delta_z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertexList.ptnorms = pointnorms;
    VertexSet.vertex_data = VertexList;
    VertexSet.num_vertices = 4;
    Facets.colr = FColor;

    pfill_area.set3 data(PFACET_COLOUR, PFEDGE_NORMS, PVRTX_COORD_NORMAL,
        PFMODEL_RGB, AFacets, num_areas, NULL, &VertexSet);

/* arrow face down */
1 = 4;
    Vertices[0].x = Rect[0][1];
    Vertices[0].y = Rect[1][1];
    Vertices[0].z = Rect[2][1];
    Vertices[1].x = Rect[0][1+1];
    Vertices[1].y = Rect[1][1+1];
    Vertices[1].z = Rect[2][1+1];
    Vertices[2].x = Rect[0][1+3];
    Vertices[2].y = Rect[1][1+3];
    Vertices[2].z = Rect[2][1+3];
    Vertices[3].x = Rect[0][1+5];
    Vertices[3].y = Rect[1][1+5];
    Vertices[3].z = Rect[2][1+5];
    Vertices[4].x = Rect[0][1+5];
    Vertices[4].y = Rect[1][1+5];
    Vertices[4].z = Rect[2][1+5];
    Vertices[5].x = Rect[0][1+4];
    Vertices[5].y = Rect[1][1+4];
    Vertices[5].z = Rect[2][1+4];
    Norms[0].delta_x = -1.0;
    Norms[0].delta_y = 0.0;
    Norms[0].delta_z = 0.0;
    Norms[1].delta_x = -1.0;
    Norms[1].delta_y = 0.0;
    Norms[1].delta_z = 0.0;
    Norms[2].delta_x = -1.0;
    Norms[2].delta_y = 0.0;
    Norms[2].delta_z = 0.0;
    Norms[3].delta_x = -1.0;
    Norms[3].delta_y = 0.0;
    Norms[3].delta_z = 0.0;
    Norms[4].delta_x = -1.0;
    Norms[4].delta_y = 0.0;
    Norms[4].delta_z = 0.0;
    Norms[5].delta_x = -1.0;
    Norms[5].delta_y = 0.0;
    Norms[5].delta_z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[4].point = Vertices[4];
    pointnorms[5].point = Vertices[5];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    pointnorms[4].norm = Norms[4];
    pointnorms[5].norm = Norms[5];
    VertexList.ptnorms = pointnorms;
    VertexSet.vertex_data = VertexList;
    VertexSet.num_vertices = 6;
    Facets.colr = FColor;

    pfill_area.set3 data(PFACET_COLOUR, PFEDGE_NORMS, PVRTX_COORD_NORMAL,
        PFMODEL_RGB, AFacets, num_areas, NULL, &VertexSet);

```


bond.c Tue Sep 20 12:10:46 1994 17

```

Rect[0][2] = 12;
Rect[1][2] = -0.2;
Rect[2][2] = 0.2;
Rect[0][3] = 12;
Rect[1][3] = 0.2;
Rect[2][3] = -0.2;
Rect[0][4] = 1.0-12; /* Bottom points */
Rect[1][4] = 0.2;
Rect[2][4] = -0.2;
Rect[0][5] = 1.0-12;
Rect[1][5] = -0.2;
Rect[2][5] = 0.2;
Rect[0][6] = 12;
Rect[1][6] = -0.2;
Rect[2][6] = 0.2;
Rect[0][7] = 12;
Rect[1][7] = 0.2;
Rect[2][7] = -0.2;

/* arrow points */
Rect[0][0] = 1.0-12;
Rect[1][0] = -0.4;
Rect[2][0] = 0.4;
Rect[0][8] = 1.0-12;
Rect[1][8] = 0.4;
Rect[2][8] = -0.4;
Rect[0][9] = 1.0-12;
Rect[1][9] = -0.4;
Rect[2][9] = 0.4;
Rect[0][10] = 1.0-12;
Rect[1][10] = -0.4;
Rect[2][10] = 0.4;
Rect[0][11] = 1.0-12;
Rect[1][11] = 0.4;
Rect[2][11] = -0.4;
Rect[0][12] = -12;
Rect[1][12] = 0.0;
Rect[2][12] = 0.0;

num_areas = 1;

/* Set up top side */
i = 0;
Vertices[0].x = Rect[0][11];
Vertices[0].y = Rect[1][11];
Vertices[0].s = Rect[2][11];
Vertices[1].x = Rect[0][10];
Vertices[1].y = Rect[1][10];
Vertices[1].s = Rect[2][10];
Vertices[2].x = Rect[0][9];
Vertices[2].y = Rect[1][9];
Vertices[2].s = Rect[2][9];
Vertices[3].x = Rect[0][8];
Vertices[3].y = Rect[1][8];
Vertices[3].s = Rect[2][8];
Vertices[0].delta_x = 0.0;
Vertices[0].delta_y = 0.0;
Norms[0].delta_x = 1.0;
Norms[0].delta_y = 0.0;
Norms[1].delta_x = 1.0;
Norms[1].delta_y = 0.0;
Norms[2].delta_x = 1.0;
Norms[2].delta_y = 0.0;
Norms[3].delta_x = 1.0;
Norms[3].delta_y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.num_vertices = 4;
Facets.colr = FColor;
Fill area.set3.data(FACET COLOUR, PEDGE MORE, FVERT COORD NORMAL,
MODEL RGB, fFacets, num_areas, NULL, &VertexSet);
/* Bottom side */
i = 4;
Vertices[0].x = Rect[0][10];
Vertices[0].y = Rect[1][10];
Vertices[0].s = Rect[2][10];
Vertices[1].x = Rect[0][9];
Vertices[1].y = Rect[1][9];
Vertices[1].s = Rect[2][9];
Vertices[2].x = Rect[0][8];
Vertices[2].y = Rect[1][8];
Vertices[2].s = Rect[2][8];
Vertices[3].x = Rect[0][7];
Vertices[3].y = Rect[1][7];
Vertices[3].s = Rect[2][7];
Vertices[0].delta_x = 0.0;
Vertices[0].delta_y = 0.0;
Norms[0].delta_x = -1.0;
Norms[0].delta_y = 0.0;
Norms[1].delta_x = 0.0;
Norms[1].delta_y = 0.0;
Norms[2].delta_x = 0.0;
Norms[2].delta_y = 0.0;
Norms[3].delta_x = 0.0;
Norms[3].delta_y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.num_vertices = 4;
Facets.colr = FColor;
Fill area.set3.data(FACET COLOUR, PEDGE MORE, FVERT COORD NORMAL,
MODEL RGB, fFacets, num_areas, NULL, &VertexSet);
/* Left side */
i = 0;
Vertices[0].x = Rect[0][11];
Vertices[0].y = Rect[1][11];
Vertices[0].s = Rect[2][11];
Vertices[1].x = Rect[0][10];
Vertices[1].y = Rect[1][10];
Vertices[1].s = Rect[2][10];
Vertices[2].x = Rect[0][9];
Vertices[2].y = Rect[1][9];
Vertices[2].s = Rect[2][9];
Vertices[3].x = Rect[0][8];
Vertices[3].y = Rect[1][8];
Vertices[3].s = Rect[2][8];
Vertices[0].delta_x = 0.0;
Vertices[0].delta_y = 0.0;
Norms[0].delta_x = 1.0;
Norms[0].delta_y = 0.0;
Norms[1].delta_x = 1.0;
Norms[1].delta_y = 0.0;
Norms[2].delta_x = 1.0;
Norms[2].delta_y = 0.0;
Norms[3].delta_x = 1.0;
Norms[3].delta_y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.num_vertices = 4;
Facets.colr = FColor;
Fill area.set3.data(FACET COLOUR, PEDGE MORE, FVERT COORD NORMAL,
MODEL RGB, fFacets, num_areas, NULL, &VertexSet);

```



```

Norms[2].delta x = 0.0;
Norms[2].delta y = 1.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.0;
Norms[3].delta x = 1.0;
Norms[3].delta y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.vertex.data = VertexList;
VertexList.num.vertices = 4;
Facets.colr = FColor;
fill area set3 data(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, AFACETS, num areas, NULL, &VertexSet);

/* Top arrow side */
1 = 0;
Vertices[0].x = Best[0][1];
Vertices[0].y = Best[1][1];
Vertices[1].x = Best[2][1];
Vertices[1].y = Best[0][1];
Vertices[2].x = Best[1][1];
Vertices[2].y = Best[2][1];
Vertices[3].x = Best[0][1];
Vertices[3].y = Best[1][1];
Norms[0].delta x = -0.37;
Norms[0].delta y = 0.0;
Norms[1].delta x = 0.0;
Norms[1].delta y = 0.93;
Norms[2].delta x = -0.37;
Norms[2].delta y = 0.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.93;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.vertex.data = VertexList;
VertexList.num.vertices = 4;
Facets.colr = FColor;
fill area set3 data(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, AFACETS, num areas, NULL, &VertexSet);

/* Back arrow side */
1 = 9;
Vertices[0].x = Best[0][1];
Vertices[0].y = Best[1][1];
Vertices[1].x = Best[2][1];
Vertices[1].y = Best[0][1];
Vertices[2].x = Best[1][1];
Vertices[2].y = Best[2][1];
Vertices[3].x = Best[0][1];
Vertices[3].y = Best[1][1];
Norms[0].delta x = -0.37;
Norms[0].delta y = 0.0;
Norms[1].delta x = 0.0;
Norms[1].delta y = 0.93;
Norms[2].delta x = -0.37;
Norms[2].delta y = 0.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.93;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.vertex.data = VertexList;
VertexList.num.vertices = 4;
Facets.colr = FColor;
fill area set3 data(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL RGB, AFACETS, num areas, NULL, &VertexSet);

/* Bottom arrow side */
1 = 10;
Vertices[0].x = Best[0][1];

```

bond.c Tue Sep 20 12:10:46 1994 20

```

    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertextList.pnorms = pointnorms;
    VertextList.vertex data = VertextList;
    VertextList.num vertices = 4;
    Facets.colr = FColor;
    fill area set3 data(FPACKET COLOUR, PEDGE NONE, EVENT COORD NORMAL,
    MODEL ROB, aFacets, num areas, NULL, &VertextSet);

/* Front arrow side */
1 = 0;
    Vertices[0].x = Rect[0][1+2];
    Vertices[0].y = Rect[1][1+2];
    Vertices[0].z = Rect[2][1+2];
    Vertices[1].x = Rect[0][1];
    Vertices[1].y = Rect[1][1];
    Vertices[1].z = Rect[2][1];
    Vertices[2].x = Rect[0][1+4];
    Vertices[2].y = Rect[1][1+4];
    Vertices[2].z = Rect[2][1+4];
    Vertices[3].x = Rect[0][1+6];
    Vertices[3].y = Rect[1][1+6];
    Vertices[3].z = Rect[2][1+6];
    Norms[0].delta x = -0.37;
    Norms[0].delta y = -0.93;
    Norms[0].delta z = 0.0;
    Norms[1].delta x = -0.37;
    Norms[1].delta y = -0.93;
    Norms[1].delta z = 0.0;
    Norms[2].delta x = -0.37;
    Norms[2].delta y = -0.93;
    Norms[2].delta z = 0.0;
    Norms[3].delta x = -0.37;
    Norms[3].delta y = -0.93;
    Norms[3].delta z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertextList.pnorms = pointnorms;
    VertextList.vertex data = VertextList;
    VertextList.num vertices = 4;
    Facets.colr = FColor;
    fill area set3 data(FPACKET COLOUR, PEDGE NONE, EVENT COORD NORMAL,
    MODEL ROB, aFacets, num areas, NULL, &VertextSet);

/* arrow face up */
1 = 0;
    Vertices[0].x = Rect[0][1+9];
    Vertices[0].y = Rect[1][1+9];
    Vertices[0].z = Rect[2][1+9];
    Vertices[1].x = Rect[0][1+8];
    Vertices[1].y = Rect[1][1+8];
    Vertices[1].z = Rect[2][1+8];
    Vertices[2].x = Rect[0][1+7];
    Vertices[2].y = Rect[1][1+7];
    Vertices[2].z = Rect[2][1+7];
    Vertices[3].x = Rect[0][1+6];
    Vertices[3].y = Rect[1][1+6];
    Vertices[3].z = Rect[2][1+6];
    Norms[0].delta x = 1.0;
    Norms[0].delta y = 0.0;
    Norms[0].delta z = 0.0;
    Norms[1].delta x = 1.0;
    Norms[1].delta y = 0.0;
    Norms[1].delta z = 0.0;
    Norms[2].delta x = 1.0;
    Norms[2].delta y = 0.0;
    Norms[2].delta z = 0.0;
    Norms[3].delta x = 1.0;
    Norms[3].delta y = 0.0;
    Norms[3].delta z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertextList.pnorms = pointnorms;
    VertextList.vertex data = VertextList;
    VertextList.num vertices = 4;
    Facets.colr = FColor;
    fill area set3 data(FPACKET COLOUR, PEDGE NONE, EVENT COORD NORMAL,
    MODEL ROB, aFacets, num areas, NULL, &VertextSet);

/* arrow face down */
1 = 0;
    Vertices[0].x = Rect[0][1];
    Vertices[0].y = Rect[1][1];
    Vertices[0].z = Rect[2][1];
    Vertices[1].x = Rect[0][1+1];
    Vertices[1].y = Rect[1][1+1];
    Vertices[1].z = Rect[2][1+1];
    Vertices[2].x = Rect[0][1+6];
    Vertices[2].y = Rect[1][1+6];
    Vertices[2].z = Rect[2][1+6];
    Vertices[3].x = Rect[0][1+7];
    Vertices[3].y = Rect[1][1+7];
    Vertices[3].z = Rect[2][1+7];
    Norms[0].delta x = 1.0;
    Norms[0].delta y = 0.0;
    Norms[0].delta z = 0.0;
    Norms[1].delta x = 1.0;
    Norms[1].delta y = 0.0;
    Norms[1].delta z = 0.0;
    Norms[2].delta x = 1.0;
    Norms[2].delta y = 0.0;
    Norms[2].delta z = 0.0;
    Norms[3].delta x = 1.0;
    Norms[3].delta y = 0.0;
    Norms[3].delta z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norms[0];
    pointnorms[1].norm = Norms[1];
    pointnorms[2].norm = Norms[2];
    pointnorms[3].norm = Norms[3];
    VertextList.pnorms = pointnorms;
    VertextList.vertex data = VertextList;
    VertextList.num vertices = 4;
    Facets.colr = FColor;
    fill area set3 data(FPACKET COLOUR, PEDGE NONE, EVENT COORD NORMAL,
    MODEL ROB, aFacets, num areas, NULL, &VertextSet);

```


bond.c Tue Sep 20 12:10:46 1994 21

```

VertexList.pnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PColor;
fill area set3 data(PFACET COLOUR, PEDGE NORMS, EVERT COORD NORMAL,
PMODEL RGB, AFACETS, num areas, NULL, &VertexList);

/* arrow face front */
1 = 1;
Vertices[0].x = Rect[0][1+4];
Vertices[0].y = Rect[1][1+4];
Vertices[0].z = Rect[2][1+4];
Vertices[1].x = Rect[0][1];
Vertices[1].y = Rect[1][1];
Vertices[1].z = Rect[2][1];
Vertices[2].x = Rect[0][1+7];
Vertices[2].y = Rect[1][1+7];
Vertices[2].z = Rect[2][1+7];
Vertices[3].x = Rect[0][1+9];
Vertices[3].y = Rect[1][1+9];
Vertices[3].z = Rect[2][1+9];
Norms[0].delta x = 1.0;
Norms[0].delta y = 0.0;
Norms[0].delta z = 0.0;
Norms[1].delta x = 1.0;
Norms[1].delta y = 0.0;
Norms[1].delta z = 0.0;
Norms[2].delta x = 1.0;
Norms[2].delta y = 0.0;
Norms[2].delta z = 0.0;
Norms[3].delta x = 1.0;
Norms[3].delta y = 0.0;
Norms[3].delta z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PColor;
fill area set3 data(PFACET COLOUR, PEDGE NORMS, EVERT COORD NORMAL,
PMODEL RGB, AFACETS, num areas, NULL, &VertexList);

polose struct();
/***** Recreate Cyl w Color *****/
/***** Recreate Cyl w Color *****/

void Recreate Cyl w Color(Pint structno, real length, Poval PColor)
{
    Pfloat Cyl[3][51][51];
    Pfloat Cone[3][51][51];
    Pint Longitude, Height, I;
    Pint NumLong, NumHeight, ConeNumHeight;
    Pfloat Theta, R;
    Ppoint Vertices[6];
    Pvec3 Norms[6];
    Pfacet data3 Facets;
    Pfacet vdata list3 VertexList;
    Pfacet vdata arr3 VertexList;
    Ppointnorm pointnorms[6];
    Pint num areas;
    real r = 0.4;
    real l2, ro, tempo;
    extern Pint WorkstatID;
    static Pint Lights[3] = {1, 2, 3};
    Pint list LightsOn, LightsOff;

    popen struct(structno);
    pempty struct(structno);
    pset elem ptr(0);
    pset edit mode(PEDIT INSERT);

    /***** define the cylinder's data base *****/
    l2 = length / 2.0;
    ro = 2.0 * r;
    NumLong = 12;
    NumHeight = 6;
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Theta = 360.0 * Longitude / NumLong;
        for (Height = 0; Height < NumHeight; Height++) {
            Vertices[0].x = Rect[0][1+4];
            Vertices[0].y = Rect[1][1+4];
            Vertices[0].z = Rect[2][1+4];
            Vertices[1].x = Rect[0][1];
            Vertices[1].y = Rect[1][1];
            Vertices[1].z = Rect[2][1];
            Vertices[2].x = Rect[0][1+7];
            Vertices[2].y = Rect[1][1+7];
            Vertices[2].z = Rect[2][1+7];
            Vertices[3].x = Rect[0][1+9];
            Vertices[3].y = Rect[1][1+9];
            Vertices[3].z = Rect[2][1+9];
            Norms[0].delta x = 1.0;
            Norms[0].delta y = 0.0;
            Norms[0].delta z = 0.0;
            Norms[1].delta x = 1.0;
            Norms[1].delta y = 0.0;
            Norms[1].delta z = 0.0;
            Norms[2].delta x = 1.0;
            Norms[2].delta y = 0.0;
            Norms[2].delta z = 0.0;
            Norms[3].delta x = 1.0;
            Norms[3].delta y = 0.0;
            Norms[3].delta z = 0.0;
            pointnorms[0].point = Vertices[0];
            pointnorms[1].point = Vertices[1];
            pointnorms[2].point = Vertices[2];
            pointnorms[3].point = Vertices[3];
            pointnorms[0].norm = Norms[0];
            pointnorms[1].norm = Norms[1];
            pointnorms[2].norm = Norms[2];
            pointnorms[3].norm = Norms[3];
            VertexList.pnorms = pointnorms;
            VertexList.vertex data = VertexList;
            VertexList.num vertices = 4;
            Facets.colr = PColor;
            fill area set3 data(PFACET COLOUR, PEDGE NORMS, EVERT COORD NORMAL,
PMODEL RGB, AFACETS, num areas, NULL, &VertexList);
        }
    }
}

```

```

    M = Height / NumHeight * (length-1.0) - 12;
    Cyl[0][Longitude][Height] = M;
    Cyl[1][Longitude][Height] = r * cos(Theta deg);
    Cyl[2][Longitude][Height] = r * sin(Theta deg);
}

/***** create the cylinder out of unicolor patches *****/

post view ind(1);
post face onli mode(PCULL BACKFACE);
post int style(PFILL SOLID);
post red_ega(PREFL AND DIFF SPEC);
post int shad meth(PED COLOUR);
setReflectanceProperties(WorkMatID,
    1.0, 1.0, 1.0,
    1.0, 1.0, 1.0,
    20.0);
LightOn.num.ints = 3;
LightOn.ints = Lights;
LightOff.num.ints = 0;
post light_ara state(LightsOn, ALightsOff);

num_areas = 1;
for (Height = 0; Height < NumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cyl[0][Longitude][Height];
        Vertices[0].y = Cyl[1][Longitude][Height];
        Vertices[0].z = Cyl[2][Longitude][Height];
        Vertices[1].x = Cyl[0][Longitude+1][Height];
        Vertices[1].y = Cyl[1][Longitude+1][Height];
        Vertices[1].z = Cyl[2][Longitude+1][Height];
        Vertices[2].x = Cyl[0][Longitude+1][Height+1];
        Vertices[2].y = Cyl[1][Longitude+1][Height+1];
        Vertices[2].z = Cyl[2][Longitude+1][Height+1];
        Vertices[3].x = Cyl[0][Longitude][Height+1];
        Vertices[3].y = Cyl[1][Longitude][Height+1];
        Vertices[3].z = Cyl[2][Longitude][Height+1];
        Norms[0].delta_x = 0.0;
        Norms[0].delta_y = Cyl[1][Longitude][Height]/r;
        Norms[0].delta_z = Cyl[2][Longitude][Height]/r;
        Norms[1].delta_x = 0.0;
        Norms[1].delta_y = Cyl[1][Longitude+1][Height]/r;
        Norms[1].delta_z = Cyl[2][Longitude+1][Height]/r;
        Norms[2].delta_x = 0.0;
        Norms[2].delta_y = Cyl[1][Longitude+1][Height+1]/r;
        Norms[2].delta_z = Cyl[2][Longitude+1][Height+1]/r;
        Norms[3].delta_x = 0.0;
        Norms[3].delta_y = Cyl[1][Longitude][Height+1]/r;
        Norms[3].delta_z = Cyl[2][Longitude][Height+1]/r;
        pointnorms[0].point = Vertices[0];
        pointnorms[1].point = Vertices[1];
        pointnorms[2].point = Vertices[2];
        pointnorms[3].point = Vertices[3];
        VertextList.pnorms = pointnorms;
        VertextSet.vertex.data = VertextList;
        VertextSet.num.vertices = 4;
        Facets.colr = FColor;
        fill_ara.set3(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
            PRODEL, ROB, sFacets, num_areas, NULL, sVertextSet);
    }
}

)

/**** create the cone database *****/
ConeNumHeight = 4;
for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Theta = 360.0 * Longitude / NumLong;
    for (Height = 0; Height < ConeNumHeight; Height++) {
        M = Height / ConeNumHeight;
        Cone[0][Longitude][Height] = M + 12 - 1.0;
        Cone[1][Longitude][Height] =
            rc * (1.0-M) * cos(Theta deg);
        Cone[2][Longitude][Height] =
            rc * (1.0-M) * sin(Theta deg);
    }
}

/**** create the cone out of unicolor patches *****/
for (Height = 0; Height < ConeNumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cone[0][Longitude][Height];
        Vertices[0].y = Cone[1][Longitude][Height];
        Vertices[0].z = Cone[2][Longitude][Height];
        Vertices[1].x = Cone[0][Longitude+1][Height];
        Vertices[1].y = Cone[1][Longitude+1][Height];
        Vertices[1].z = Cone[2][Longitude+1][Height];
        Vertices[2].x = Cone[0][Longitude+1][Height+1];
        Vertices[2].y = Cone[1][Longitude+1][Height+1];
        Vertices[2].z = Cone[2][Longitude+1][Height+1];
        Vertices[3].x = Cone[0][Longitude][Height+1];
        Vertices[3].y = Cone[1][Longitude][Height+1];
        Vertices[3].z = Cone[2][Longitude][Height+1];
        Norms[0].delta_x =
            (Cone[1][Longitude][Height]) * (Cone[1][Longitude+1][Height]) +
            (Cone[2][Longitude][Height]) * (Cone[2][Longitude+1][Height]);
        temp = Norms[0].delta_x * Norms[0].delta_x + Norms[0].delta_z *
            Norms[0].delta_z;
        Norms[0].delta_x = Cone[1][Longitude][Height]/temp;
        Norms[0].delta_y = Cone[1][Longitude+1][Height]/temp;
        Norms[0].delta_z = Cone[2][Longitude][Height]/temp;
        Norms[1].delta_x =
            (Cone[1][Longitude+1][Height]) * (Cone[1][Longitude+1][Height]) +
            (Cone[2][Longitude+1][Height]) * (Cone[2][Longitude+1][Height]);
        temp = Norms[1].delta_x * Norms[1].delta_x + Norms[1].delta_z *
            Norms[1].delta_z;
        Norms[1].delta_x = Cone[1][Longitude+1][Height]/temp;
        Norms[1].delta_y = Cone[1][Longitude+1][Height+1]/temp;
        Norms[1].delta_z = Cone[2][Longitude+1][Height+1]/temp;
        Norms[2].delta_x =
            (Cone[1][Longitude+1][Height+1]) * (Cone[1][Longitude+1][Height+1]) +
            (Cone[2][Longitude+1][Height+1]) * (Cone[2][Longitude+1][Height+1]);
        temp = Norms[2].delta_x * Norms[2].delta_x + Norms[2].delta_z *
            Norms[2].delta_z;
        Norms[2].delta_x = Cone[1][Longitude+1][Height+1]/temp;
        Norms[2].delta_y = Cone[1][Longitude+1][Height+1]/temp;
        Norms[2].delta_z = Cone[2][Longitude+1][Height+1]/temp;
        Norms[3].delta_x =
            (Cone[1][Longitude][Height+1]) * (Cone[1][Longitude+1][Height+1]) +
            (Cone[2][Longitude][Height+1]) * (Cone[2][Longitude+1][Height+1]);
        temp = Norms[3].delta_x * Norms[3].delta_x + Norms[3].delta_z *
            Norms[3].delta_z;
        Norms[3].delta_x = Cone[1][Longitude][Height+1]/temp;
        Norms[3].delta_y = Cone[1][Longitude][Height+1]/temp;
        Norms[3].delta_z = Cone[2][Longitude][Height+1]/temp;
        pointnorms[0].point = Vertices[0];
        pointnorms[1].point = Vertices[1];
        pointnorms[2].point = Vertices[2];
        pointnorms[3].point = Vertices[3];
    }
}

```

```

pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
FaceSet.colr = Color;
fill area set3 data(PFACE COLOR, PFACE NORM, PFACE COORD NORMAL,
PFACE RGB, AFacets, num areas, NULL, &VertexSet);
)

```

... create the back surface to the cone ... /

```

for (Longitude = 0; Longitude < NumLong; Longitude++)
    Vertices[0].x = Coord[0][Longitude+1][0];
    Vertices[0].y = Coord[0][Longitude+1][0];
    Vertices[1].x = Coord[1][Longitude+1][0];
    Vertices[1].y = Coord[1][Longitude+1][0];
    Vertices[2].x = Coord[2][Longitude+1][0];
    Vertices[2].y = Coord[2][Longitude+1][0];
    Vertices[3].x = Cyl[0][Longitude][NumLat+1];
    Vertices[3].y = Cyl[0][Longitude][NumLat+1];
    Vertices[4].x = Cyl[1][Longitude+1][NumLat+1];
    Vertices[4].y = Cyl[1][Longitude+1][NumLat+1];
    Vertices[5].x = Cyl[2][Longitude+1][NumLat+1];
    Vertices[5].y = Cyl[2][Longitude+1][NumLat+1];
    Norms[0].delta_x = -1.0;
    Norms[0].delta_y = 0.0;
    Norms[1].delta_x = 0.0;
    Norms[1].delta_y = 0.0;
    Norms[2].delta_x = -1.0;
    Norms[2].delta_y = 0.0;
    Norms[3].delta_x = 0.0;
    Norms[3].delta_y = 0.0;
    Norms[4].delta_x = -1.0;
    Norms[4].delta_y = 0.0;
    Norms[5].delta_x = 0.0;
    Norms[5].delta_y = 0.0;
    Norms[6].delta_x = 0.0;
    Norms[6].delta_y = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[4].norm = Norms[0];
    pointnorms[5].norm = Norms[1];
    pointnorms[6].norm = Norms[2];
    pointnorms[7].norm = Norms[3];
    VerticesList.ptnorms = pointnorms;
    VerticesList.vertex_data = VerticesList;
    VerticesSet.num_vertices = 6;
    Facesets.colr = "Color";
    pfill area set3 data(PACCT COLOR, FUDGE MORE, FVERT
    FMODEL, ROB, sFacets, num_areas, NULL, sVerticesSet);

```

```

**** create the concrete end of the cylinder ****/

```

```
for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Vertices[0].x = Cyl[0][Longitude+1][0];
    Vertices[0].y = Cyl[1][Longitude+1][0];
    Vertices[0].z = Cyl[2][Longitude+1][0];
    Vertices[1].x = Cyl[0][Longitude][0];
    Vertices[1].y = Cyl[1][Longitude][0];
    Vertices[1].z = Cyl[2][Longitude][0];
```

```

Vertices[1].x = Cyl[2][Longitude][0];
Vertices[2].x = Cyl[0][Longitude][0];
Vertices[2].y = 0.0;
Vertices[3].x = 0.0;
Vertices[3].y = 0.0;
Vertices[3].z = Cyl[0][Longitude][0];
Vertices[3].y = 0.0;
Vertices[3].z = 0.0;
Norms[0].delta.x = -1.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = 0.0;
Norms[1].delta.x = -1.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = 0.0;
Norms[2].delta.x = -1.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = 0.0;
Norms[3].delta.x = -1.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex_data = VertexList;
VertexList.num_vertices = 4;
FaceList.colr = Colors;
fill_area.set3_data(SPACET.COLOUR, FUDGE.NORM, FVERT.COORD.NORMAL,
                    FMODE.RGB, fFaceList, num_areas, MWLE, aVertexList);
)
police_struct();
)

.....
..... Recreate Cyl Backward ....
.....

void Recreate_Cyl_Backward(Pint strucone, real length, Pconval PColor)
{
    Pfloat Cyl[3][51][51];
    Pfloat Cone[3][51][51];
    Pint Longitude, Height, I;
    Pint NumLong, NumHeight, ConeNumHeight;
    Pint Theta, N;
    Ppoint Vertices[6];
    Pvec3 Norms[6];
    Pfloat data1, FaceList;
    Pfloat vdata, list3 VertexList;
    Pfloat vdata arr3 VertexList;
    Pfloat3 pointnorms[6];
    Pint num_areas;
    real x = 0.4;
    real l2, ro, temp1;
    extern Pint WorstaID;
    static Pint Lights[3] = {1, 2, 3};
    Pint list LightsOn, LightsOff;

    popen_struct(strucone);
    pempty_struct(strucone);
    pset_alon_ptr(0);
}

```

```

post_edit mode(PEDIT_INSERT);

/***** Define the cylinder's data base *****/

l2 = length / 2.0;
r0 = 2.0 * r;
numlong = 12;
numheight = 6;
for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Theta = 360.0 * Longitude / NumLong;
    for (Height = 0; Height < NumHeight; Height++) {
        M = Height / NumHeight * (length-1.0) - 12 + 1.0;
        Cyl[0][Longitude][Height] = M;
        Cyl[1][Longitude][Height] = r * cos(Theta deg);
        Cyl[2][Longitude][Height] = r * sin(Theta deg);
    }
}

/***** create the cylinder out of unicolor patches *****/

post_view ind(1);
post_face onll mode(PCULL_BACKFACE);
post_int style(PSTYL_SOLID);
post_redl_aps(PREFL_APS DIFF SPEC);
post_int shad meth(PED_COLOR);
setReflectionProperties (REFRSTAD,
    1.0, 1.0, 1.0,
    1.0, 1.0, 1.0,
    20.0);
LightOn.num into = 3;
LightOn.into = lights;
LightOff.num into = 0;
post_light arc_state(lightOn, alightOff);

num areas = 1;
for (Height = 0; Height < NumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cyl[0][Longitude][Height];
        Vertices[0].y = Cyl[1][Longitude][Height];
        Vertices[0].z = Cyl[2][Longitude][Height];
        Vertices[1].x = Cyl[0][Longitude+1][Height];
        Vertices[1].y = Cyl[1][Longitude+1][Height];
        Vertices[1].z = Cyl[2][Longitude+1][Height];
        Vertices[2].x = Cyl[0][Longitude+1][Height+1];
        Vertices[2].y = Cyl[1][Longitude+1][Height+1];
        Vertices[2].z = Cyl[2][Longitude+1][Height+1];
        Vertices[3].x = Cyl[0][Longitude][Height+1];
        Vertices[3].y = Cyl[1][Longitude][Height+1];
        Vertices[3].z = Cyl[2][Longitude][Height+1];
        Norms[0].delta_x = 0.0;
        Norms[0].delta_y = Cyl[1][Longitude][Height]/r;
        Norms[0].delta_z = 0.0;
        Norms[1].delta_x = 0.0;
        Norms[1].delta_y = Cyl[1][Longitude+1][Height]/r;
        Norms[1].delta_z = Cyl[2][Longitude+1][Height]/r;
        Norms[2].delta_x = 0.0;
        Norms[2].delta_y = Cyl[1][Longitude+1][Height+1]/r;
        Norms[2].delta_z = Cyl[2][Longitude+1][Height+1]/r;
        Norms[3].delta_x = 0.0;
        Norms[3].delta_y = Cyl[1][Longitude][Height+1]/r;
        Norms[3].delta_z = Cyl[2][Longitude][Height+1]/r;
        pointnorms[0].point = Vertices[0];
        pointnorms[1].point = Vertices[1];
        pointnorms[2].point = Vertices[2];
        pointnorms[3].point = Vertices[3];
    }
}

pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex_data = VertexList;
VertexList.num vertices = 4;
FaceList.colr = FColor;
Fill area setj data(PFACET_COLOR, PEDGE_NONE, PVERT_COORD_NORMAL,
    PRODEL_RGB, sFaceList, num areas, NULL, sVertexList);
}

/**** create the cone database *****/

ConeNumHeight = 4;
for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Theta = 360.0 * Longitude / NumLong;
    for (Height = 0; Height < ConeNumHeight; Height++) {
        M = Height / ConeNumHeight;
        Cone[0][Longitude][Height] = M - 12;
        Cone[1][Longitude][Height] =
            r0 * M * cos(Theta deg);
        Cone[2][Longitude][Height] =
            r0 * M * sin(Theta deg);
    }
}

/**** create the cone out of unicolor patches *****/

for (Height = 0; Height < ConeNumHeight; Height++) {
    for (Longitude = 0; Longitude < NumLong; Longitude++) {
        Vertices[0].x = Cone[0][Longitude][Height];
        Vertices[0].y = Cone[1][Longitude][Height];
        Vertices[0].z = Cone[2][Longitude][Height];
        Vertices[1].x = Cone[0][Longitude+1][Height];
        Vertices[1].y = Cone[1][Longitude+1][Height];
        Vertices[1].z = Cone[2][Longitude+1][Height];
        Vertices[2].x = Cone[0][Longitude+1][Height+1];
        Vertices[2].y = Cone[1][Longitude+1][Height+1];
        Vertices[2].z = Cone[2][Longitude+1][Height+1];
        Vertices[3].x = Cone[0][Longitude][Height+1];
        Vertices[3].y = Cone[1][Longitude][Height+1];
        Vertices[3].z = Cone[2][Longitude][Height+1];
        Norms[0].delta_x =
            (Cone[1][Longitude][Height]) * (Cone[1][Longitude+1][Height]) *
            (Cone[2][Longitude][Height]) * (Cone[2][Longitude+1][Height]);
        Norms[0].delta_y = Norms[0].delta_x * Norms[0].delta_z / temp;
        Norms[0].delta_z = Cone[1][Longitude][Height] / temp;
        Norms[1].delta_x =
            (Cone[1][Longitude+1][Height]) * (Cone[1][Longitude+1][Height]) *
            (Cone[2][Longitude+1][Height]) * (Cone[2][Longitude+1][Height]);
        Norms[1].delta_y = Norms[1].delta_x * Norms[1].delta_z / temp;
        Norms[1].delta_z = Cone[1][Longitude+1][Height] / temp;
        Norms[2].delta_x =
            (Cone[1][Longitude+1][Height+1]) * (Cone[1][Longitude+1][Height+1]) *
            (Cone[2][Longitude+1][Height+1]) * (Cone[2][Longitude+1][Height+1]);
        Norms[2].delta_y = Norms[2].delta_x * Norms[2].delta_z / temp;
        Norms[2].delta_z = Cone[1][Longitude+1][Height+1] / temp;
        Norms[3].delta_x =
            (Cone[1][Longitude][Height+1]) * (Cone[1][Longitude][Height+1]) *
            (Cone[2][Longitude][Height+1]) * (Cone[2][Longitude][Height+1]);
        Norms[3].delta_y = Norms[3].delta_x * Norms[3].delta_z / temp;
        Norms[3].delta_z = Cone[1][Longitude][Height+1] / temp;
    }
}

```

bond.c Tue Sep 20 12:10:46 1994 25

```

Norma[3].delta x =
    (Cone[1][Longitude][Height+1])*(Cone[1][Longitude][Height+1]) +
    (Cone[2][Longitude][Height+1])*(Cone[2][Longitude][Height+1]);
temp = Norma[3].delta x * Norma[3].delta x / temp;
Norma[3].delta x = Norma[3].delta x / temp;
Norma[3].delta y = Cone[1][Longitude][Height+1] / temp;
Norma[3].delta z = Cone[2][Longitude][Height+1] / temp;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norma[0];
pointnorms[1].norm = Norma[1];
pointnorms[2].norm = Norma[2];
pointnorms[3].norm = Norma[3];
VertexList.pnorms = pointnorms;
VertexList.vertex_data = VertexList;
VertexList.num_vertices = 4;
Facets.colr = FColor;
fill area set3 data (PFACET COLOUR, FEDGE NONE, FVERT COORD NORMAL,
    PMODEL_RGB, sFacets, num_areas, NULL, sVertexSet);
}
/* create the back surface to the cone ****/

for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Vertices[0].x = Cone[0][Longitude][ConeNumHeight];
    Vertices[0].y = Cone[1][Longitude][ConeNumHeight];
    Vertices[0].z = Cone[2][Longitude][ConeNumHeight];
    Vertices[1].x = Cone[0][Longitude+1][ConeNumHeight];
    Vertices[1].y = Cone[1][Longitude+1][ConeNumHeight];
    Vertices[1].z = Cone[2][Longitude+1][ConeNumHeight];
    Vertices[2].x = Cone[0][Longitude+1][0];
    Vertices[2].y = Cone[1][Longitude+1][0];
    Vertices[2].z = Cone[2][Longitude+1][0];
    Vertices[3].x = Cone[0][Longitude][0];
    Vertices[3].y = Cone[1][Longitude][0];
    Vertices[3].z = Cone[2][Longitude][0];
    Norma[0].delta x = 1.0;
    Norma[0].delta y = 0.0;
    Norma[0].delta z = 0.0;
    Norma[1].delta x = 1.0;
    Norma[1].delta y = 0.0;
    Norma[1].delta z = 0.0;
    Norma[2].delta x = 1.0;
    Norma[2].delta y = 0.0;
    Norma[2].delta z = 0.0;
    Norma[3].delta x = 1.0;
    Norma[3].delta y = 0.0;
    Norma[3].delta z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norma[0];
    pointnorms[1].norm = Norma[1];
    pointnorms[2].norm = Norma[2];
    pointnorms[3].norm = Norma[3];
    VertexList.pnorms = pointnorms;
    VertexList.vertex_data = VertexList;
    VertexList.num_vertices = 4;
    Facets.colr = FColor;
    fill area set3 data (PFACET COLOUR, FEDGE NONE, FVERT COORD NORMAL,
        PMODEL_RGB, sFacets, num_areas, NULL, sVertexSet);
}

/* create the opposite end of the cylinder ****/
for (Longitude = 0; Longitude < NumLong; Longitude++) {
    Vertices[0].x = Cy[0][Longitude][NumHeight];
    Vertices[0].y = Cy[1][Longitude][NumHeight];
    Vertices[0].z = Cy[2][Longitude][NumHeight];
    Vertices[1].x = Cy[0][Longitude+1][NumHeight];
    Vertices[1].y = Cy[1][Longitude+1][NumHeight];
    Vertices[1].z = Cy[2][Longitude+1][NumHeight];
    Vertices[2].x = Cy[0][Longitude+1][0];
    Vertices[2].y = Cy[1][Longitude+1][0];
    Vertices[2].z = Cy[2][Longitude+1][0];
    Vertices[3].x = Cy[0][Longitude][0];
    Vertices[3].y = Cy[1][Longitude][0];
    Vertices[3].z = Cy[2][Longitude][0];
    Norma[0].delta x = 1.0;
    Norma[0].delta y = 0.0;
    Norma[0].delta z = 0.0;
    Norma[1].delta x = 1.0;
    Norma[1].delta y = 0.0;
    Norma[1].delta z = 0.0;
    Norma[2].delta x = 1.0;
    Norma[2].delta y = 0.0;
    Norma[2].delta z = 0.0;
    Norma[3].delta x = 1.0;
    Norma[3].delta y = 0.0;
    Norma[3].delta z = 0.0;
    pointnorms[0].point = Vertices[0];
    pointnorms[1].point = Vertices[1];
    pointnorms[2].point = Vertices[2];
    pointnorms[3].point = Vertices[3];
    pointnorms[0].norm = Norma[0];
    pointnorms[1].norm = Norma[1];
    pointnorms[2].norm = Norma[2];
    pointnorms[3].norm = Norma[3];
    VertexList.pnorms = pointnorms;
    VertexList.vertex_data = VertexList;
    VertexList.num_vertices = 4;
    Facets.colr = FColor;
    fill area set3 data (PFACET COLOUR, FEDGE NONE, FVERT COORD NORMAL,
        PMODEL_RGB, sFacets, num_areas, NULL, sVertexSet);
}

polose_struct();
}

/* create the connector ****/
void Create Connector (real length, int index, Pcolor PColor)
{
    Pfloat Rect[3][13];
    Pvec3 Norma[4];
    Point3 Vertices[4];
    Pfacet data3 Facets;
    Pfacet vdata list3 VertexSet;
    Pfacet FacetColorRed;
    Pint num_areas;
    Ppnormal pointnorms[4];
}

```



```

Norms[3].delta x = -1.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PCOLOR;
pfill area set3 data (PPACKET COLOR, PEDGE NORMS, PVERT COORD NORMAL,
PRODEL RGB, AFacets, num areas, NULL, &VertexSet);
/* Left side */
1 = 0;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[0][1];
Vertices[1].x = Rect[2][1];
Vertices[1].y = Rect[2][1];
Vertices[2].x = Rect[0][1+4];
Vertices[2].y = Rect[0][1+4];
Vertices[3].x = Rect[2][1+4];
Vertices[3].y = Rect[2][1+4];
Vertices[0].x = Rect[0][1+5];
Vertices[0].y = Rect[0][1+5];
Vertices[1].x = Rect[2][1+5];
Vertices[1].y = Rect[2][1+5];
Vertices[2].x = Rect[0][1+5];
Vertices[2].y = Rect[0][1+5];
Vertices[3].x = Rect[2][1+5];
Vertices[3].y = Rect[2][1+5];
Vertices[0].delta x = -1.0;
Vertices[0].delta y = 0.0;
Vertices[1].delta x = 0.0;
Vertices[1].delta y = 0.0;
Vertices[2].delta x = -1.0;
Vertices[2].delta y = 0.0;
Vertices[3].delta x = 0.0;
Vertices[3].delta y = 0.0;
Norms[0].delta x = 0.0;
Norms[0].delta y = 0.0;
Norms[1].delta x = 0.0;
Norms[1].delta y = 0.0;
Norms[2].delta x = -1.0;
Norms[2].delta y = 0.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PCOLOR;
pfill area set3 data (PPACKET COLOR, PEDGE NORMS, PVERT COORD NORMAL,
PRODEL RGB, AFacets, num areas, NULL, &VertexSet);
/* Right side */
1 = 2;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[1].x = Rect[2][1];
Vertices[1].y = Rect[2][1];
Vertices[2].x = Rect[0][1+4];
Vertices[2].y = Rect[0][1+4];
Vertices[3].x = Rect[2][1+4];
Vertices[3].y = Rect[2][1+4];
Vertices[0].x = Rect[0][1+5];
Vertices[0].y = Rect[0][1+5];
Vertices[1].x = Rect[2][1+5];
Vertices[1].y = Rect[2][1+5];
Vertices[2].x = Rect[0][1+5];
Vertices[2].y = Rect[0][1+5];
Vertices[3].x = Rect[2][1+5];
Vertices[3].y = Rect[2][1+5];
Vertices[0].delta x = -1.0;
Vertices[0].delta y = 0.0;
Vertices[1].delta x = 0.0;
Vertices[1].delta y = 0.0;
Vertices[2].delta x = -1.0;
Vertices[2].delta y = 0.0;
Vertices[3].delta x = 0.0;
Vertices[3].delta y = 0.0;
Norms[0].delta x = 0.0;
Norms[0].delta y = 0.0;
Norms[1].delta x = 0.0;
Norms[1].delta y = 0.0;
Norms[2].delta x = -1.0;
Norms[2].delta y = 0.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PCOLOR;
pfill area set3 data (PPACKET COLOR, PEDGE NORMS, PVERT COORD NORMAL,
PRODEL RGB, AFacets, num areas, NULL, &VertexSet);
/* Front side */
1 = 1;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[1].x = Rect[2][1];
Vertices[1].y = Rect[2][1];
Vertices[2].x = Rect[0][1+4];
Vertices[2].y = Rect[0][1+4];
Vertices[3].x = Rect[2][1+4];
Vertices[3].y = Rect[2][1+4];
Vertices[0].x = Rect[0][1+5];
Vertices[0].y = Rect[0][1+5];
Vertices[1].x = Rect[2][1+5];
Vertices[1].y = Rect[2][1+5];
Vertices[2].x = Rect[0][1+5];
Vertices[2].y = Rect[0][1+5];
Vertices[3].x = Rect[2][1+5];
Vertices[3].y = Rect[2][1+5];
Vertices[0].delta x = -1.0;
Vertices[0].delta y = 0.0;
Vertices[1].delta x = 0.0;
Vertices[1].delta y = 0.0;
Vertices[2].delta x = -1.0;
Vertices[2].delta y = 0.0;
Vertices[3].delta x = 0.0;
Vertices[3].delta y = 0.0;
Norms[0].delta x = 0.0;
Norms[0].delta y = 0.0;
Norms[1].delta x = 0.0;
Norms[1].delta y = 0.0;
Norms[2].delta x = -1.0;
Norms[2].delta y = 0.0;
Norms[3].delta x = 0.0;
Norms[3].delta y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexList.vertex data = VertexList;
VertexList.num vertices = 4;
Facets.colr = PCOLOR;
pfill area set3 data (PPACKET COLOR, PEDGE NORMS, PVERT COORD NORMAL,
PRODEL RGB, AFacets, num areas, NULL, &VertexSet);

```

```

bond.c          Tue Sep 20 12:10:46 1994      28

VertexSet.vertex data = VertexList;
VertexSet.num vertices = 4;
Facets.colr = PColor;
pfill area set3 data(IPFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL ROB, aFacets, num_areas, NULL, aVertexSet);
/* Back side */
i = 0;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[0].z = Rect[2][1];
Vertices[1].x = Rect[0][1+1];
Vertices[1].y = Rect[1][1+1];
Vertices[1].z = Rect[2][1+1];
Vertices[2].x = Rect[0][1+2];
Vertices[2].y = Rect[1][1+2];
Vertices[2].z = Rect[2][1+2];
Vertices[3].x = Rect[0][1+3];
Vertices[3].y = Rect[1][1+3];
Vertices[3].z = Rect[2][1+3];
Vertices[0].norm = Norms[0];
Vertices[1].norm = Norms[1];
Vertices[2].norm = Norms[2];
Vertices[3].norm = Norms[3];
VertexList.pnorms = Pointnorms;
VertexSet.vertex data = VertexList;
VertexSet.num vertices = 4;
Facets.colr = PColor;
pfill area set3 data(IPFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL ROB, aFacets, num_areas, NULL, aVertexSet);
/* Bottom arrow side */
i = 10;
Vertices[0].x = Rect[0][1];
Vertices[0].y = Rect[1][1];
Vertices[0].z = Rect[2][1];
Vertices[1].x = Rect[0][1+1];
Vertices[1].y = Rect[1][1+1];
Vertices[1].z = Rect[2][1+1];
Vertices[2].x = Rect[0][1+2];
Vertices[2].y = Rect[1][1+2];
Vertices[2].z = Rect[2][1+2];
Vertices[3].x = Rect[0][1+3];
Vertices[3].y = Rect[1][1+3];
Vertices[3].z = Rect[2][1+3];
Vertices[0].norm = Norms[0];
Vertices[1].norm = Norms[1];
Vertices[2].norm = Norms[2];
Vertices[3].norm = Norms[3];
VertexList.pnorms = Pointnorms;
VertexSet.vertex data = VertexList;
VertexSet.num vertices = 4;
Facets.colr = PColor;
pfill area set3 data(IPFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL ROB, aFacets, num_areas, NULL, aVertexSet);
/* Back arrow side */
i = 9;
VertexSet.vertex data = VertexList;
VertexSet.num vertices = 4;
Facets.colr = PColor;
pfill area set3 data(IPFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL ROB, aFacets, num_areas, NULL, aVertexSet);
/* Top arrow side */
i = 8;
Vertices[0].x = Rect[0][1+1];
Vertices[0].y = Rect[1][1+1];
Vertices[0].z = Rect[2][1+1];
Vertices[1].x = Rect[0][1+2];
Vertices[1].y = Rect[1][1+2];
Vertices[1].z = Rect[2][1+2];
Vertices[2].x = Rect[0][1+3];
Vertices[2].y = Rect[1][1+3];
Vertices[2].z = Rect[2][1+3];
Vertices[3].x = Rect[0][1+4];
Vertices[3].y = Rect[1][1+4];
Vertices[3].z = Rect[2][1+4];
Vertices[0].norm = Norms[0];
Vertices[1].norm = Norms[1];
Vertices[2].norm = Norms[2];
Vertices[3].norm = Norms[3];
VertexList.pnorms = Pointnorms;
VertexSet.vertex data = VertexList;
VertexSet.num vertices = 4;
Facets.colr = PColor;
pfill area set3 data(IPFACCT COLOUR, PEDGE NONE, PVERT COORD NORMAL,
MODEL ROB, aFacets, num_areas, NULL, aVertexSet);

```



```

Vertices[0].x = Root[0][1+2];
Vertices[0].y = Root[1][1+2];
Vertices[0].z = Root[2][1+2];
Vertices[1].x = Root[0][1];
Vertices[1].y = Root[1][1];
Vertices[1].z = Root[2][1];
Vertices[2].x = Root[0][1+3];
Vertices[2].y = Root[1][1+3];
Vertices[2].z = Root[2][1+3];
Vertices[3].x = Root[0][1+3];
Vertices[3].y = Root[1][1+3];
Vertices[3].z = Root[2][1+3];
Norms[0].delta_x = 0.37;
Norms[0].delta_y = 0.93;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = 0.37;
Norms[1].delta_y = 0.93;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = 0.37;
Norms[2].delta_y = 0.93;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = 0.37;
Norms[3].delta_y = 0.93;
Norms[3].delta_z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pointnorms = pointnorms;
VertexSet.vertex_data = VertexList;
VertexSet.num_vertices = 4;
Facets.colr = FColor;
fill_area.set3_data(PFACCT.COLOUR, FIDES.NORM, FVERT.COORD.NORMAL,
MODEL_RGB, aFacets, num_areas, NULL, aVertexSet);

/* arrow face up */
1 = 2;
Vertices[0].x = Root[0][1+6];
Vertices[0].y = Root[1][1+6];
Vertices[0].z = Root[2][1+6];
Vertices[1].x = Root[0][1+7];
Vertices[1].y = Root[1][1+7];
Vertices[1].z = Root[2][1+7];
Vertices[2].x = Root[0][1+1];
Vertices[2].y = Root[1][1+1];
Vertices[2].z = Root[2][1+1];
Vertices[3].x = Root[0][1];
Vertices[3].y = Root[1][1];
Vertices[3].z = Root[2][1];
Norms[0].delta_x = -1.0;
Norms[0].delta_y = 0.0;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = -1.0;
Norms[1].delta_y = 0.0;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = -1.0;
Norms[2].delta_y = 0.0;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = -1.0;
Norms[3].delta_y = 0.0;
Norms[3].delta_z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.pointnorms = pointnorms;
VertexSet.vertex_data = VertexList;
VertexSet.num_vertices = 4;
Facets.colr = FColor;
fill_area.set3_data(PFACCT.COLOUR, FIDES.NORM, FVERT.COORD.NORMAL,
MODEL_RGB, aFacets, num_areas, NULL, aVertexSet);

/* arrow face down */
1 = 6;
Vertices[0].x = Root[0][1];
Vertices[0].y = Root[1][1];
Vertices[0].z = Root[2][1];
Vertices[1].x = Root[0][1+1];
Vertices[1].y = Root[1][1+1];
Vertices[1].z = Root[2][1+1];
Vertices[2].x = Root[0][1+5];
Vertices[2].y = Root[1][1+5];
Vertices[2].z = Root[2][1+5];

```

bond.c Tue Sep 20 12:10:46 1994 30

```

Vertices[2].y = Rect[1][i+5];
Vertices[2].z = Rect[2][i+5];
Vertices[3].x = Rect[0][i+4];
Vertices[3].y = Rect[1][i+4];
Vertices[3].z = Rect[2][i+4];
Norms[0].delta.x = -1.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = 0.0;
Norms[1].delta.x = -1.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = 0.0;
Norms[2].delta.x = -1.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = 0.0;
Norms[3].delta.x = -1.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex.data = VertexList;
VertexSet.num.vertices = 4;
Faces.colr = FColor;
fill area set3 data(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
PRODEL RGB, aFacets, num_areas, NULL, aVertexSet);

/* arrow face back */
i = 3;
Vertices[0].x = Rect[0][i];
Vertices[0].y = Rect[1][i];
Vertices[0].z = Rect[2][i];
Vertices[1].x = Rect[0][i+6];
Vertices[1].y = Rect[1][i+6];
Vertices[1].z = Rect[2][i+6];
Vertices[2].x = Rect[0][i+8];
Vertices[2].y = Rect[1][i+8];
Vertices[2].z = Rect[2][i+8];
Vertices[3].x = Rect[0][i+4];
Vertices[3].y = Rect[1][i+4];
Vertices[3].z = Rect[2][i+4];
Norms[0].delta.x = -1.0;
Norms[0].delta.y = 0.0;
Norms[0].delta.z = 0.0;
Norms[1].delta.x = -1.0;
Norms[1].delta.y = 0.0;
Norms[1].delta.z = 0.0;
Norms[2].delta.x = -1.0;
Norms[2].delta.y = 0.0;
Norms[2].delta.z = 0.0;
Norms[3].delta.x = -1.0;
Norms[3].delta.y = 0.0;
Norms[3].delta.z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex.data = VertexList;
VertexSet.num.vertices = 4;
Faces.colr = FColor;
fill area set3 data(PFACET COLOUR, PEDGE NONE, PVERT COORD NORMAL,
PRODEL RGB, aFacets, num_areas, NULL, aVertexSet);

polose struct();
}

/*****
/**** Draw Axes ****
/****
void Draw Axes()
{
    Point list3 xaxedata, yaxedata, zaxedata;
    Point3 xpoints[2], ypoints[2], zpoints[2];
    Pvec3 offsets, offsets, offsets;
    xpoints[0].x = -40.0, xpoints[0].y = 0.0, xpoints[0].z = 0.0;
    xpoints[1].x = 40.0, xpoints[1].y = 0.0, xpoints[1].z = 0.0;
}

```

bond.c Tue Sep 20 12:10:46 1994 31

```

xazedata.num_points = 2;
xazedata.points = xpoints;
ypoints[0].x = 0.0, ypoints[0].y = -40.0, ypoints[0].z = 0.0;
ypoints[1].x = 0.0, ypoints[1].y = 40.0, ypoints[1].z = 0.0;
yazedata.num_points = 2;
yazedata.points = ypoints;
ypoints[0].x = 0.0, ypoints[0].y = 0.0, ypoints[0].z = -40.0;
ypoints[1].x = 0.0, ypoints[1].y = 0.0, ypoints[1].z = 40.0;
xazedata.num_points = 2;
xazedata.points = xpoints;
ypolyline3(xazedata);
ypolyline3(yazedata);
ypolyline3(xazedata);

offsetx.delta.x = 0.02, offsetx.delta.y = 0.00, offsetx.delta.z = 0.00;
offsety.delta.x = 0.00, offsety.delta.y = 0.02, offsety.delta.z = 0.00;
offsetz.delta.x = 0.00, offsetz.delta.y = 0.00, offsetz.delta.z = 0.02;
panno_text_roll(xpoints[1], offsetx, "X");
panno_text_roll(ypoints[1], offsety, "Y");
panno_text_roll(zpoints[1], offsetz, "Z");
}

/*****
/**** Main Cancel ****
/****
*****/

void Main_Cancel(Widget w)
{
    exit();
}

/*****
/**** PhisLoad File ****
/****
*****/

void PhisLoad_File(Widget w)
{
    extern Widget toplevel;
    extern Widget Selectpball;
    xgetfile(".amp", "Load File", (void (*)(void *)) (Phisload it), NULL);
}

/*****
/**** PhisLoad Data ****
/****
*****/

void PhisLoad_Data(Widget w)
{
    extern Widget toplevel;
    extern Widget Selectpball;
    char m2ofile[255];
    char m2ofile[255];
    int result;
    xgetfile(".s2p", "Load Data File", (void (*)(void *)) (Pload equations),
    NULL);
    return;
}

/*****
/**** Flood equations ****
*****/

/*****
/**** Flood equations (char *fname)
*****/
void Flood_equations (char *fname)
{
    char newfname[255];
    int okay;
    int problem = 0;
    int result = 0;
    FILE *fp;
    int i, ondx;
    Argv argv[1];
    char Temp[13];
    /* Read equations data */
    /* fp = fopen(fname, "r");
    if (!fp) Problem_Pop ("Error opening Results file!");
    OutMatchLine(fp, NULL, line, sizeof(line), SKIPBLANKS, PAD);
    if (read Outputs(fp) == ERROR) problem = 1;
    if (read ParamName(fp) == ERROR) problem = 1;
    if (read ParamData(fp) == ERROR) problem = 1;
    if (read InputMdx(fp) == ERROR) problem = 1;
    if (read InputOutputPtrs(fp) == ERROR) problem = 1;
    if (read Param(fp) == ERROR) problem = 1;
    if (read PttParam(fp) == ERROR) problem = 1;
    if (problem) {
        Problem_Pop ("Error reading equations!\n");
    }
    else {
        sorted (okay);
        result = read postproc file (fname);
        if (result == 0) {
            sprintf(Temp, "%5E", actib4_1.tin);
            formatParam(Temp);
            i = 0;
            XtSetArg(args[1], XtMetricing, Temp); ++i;
            XtSetValues(MainDialogInitValue, args, i);
            sprintf(Temp, "%5E", actib4_1.tfin);
            formatParam(Temp);
            i = 0;
            XtSetArg(args[1], XtMetricing, Temp); ++i;
            XtSetValues(MainDialogFinValue, args, i);
            Problem_Pop ("Data successfully loaded");
        }
        else {
            Problem_Pop ("Error reading results!");
        }
    }
}

/*****
/**** init view mapping ****
*****/
void init view_mapping ()
{
    map.proj.type = PTYPE PERSPECT;
    map.proj.ref.point.z = 6.0 * WC MAX;
}

```

bond.c Tue Sep 20 12:10:46 1994 32

```

map.win.x.min = WC_MIN; map.win.x.max = WC_MAX;
map.win.y.min = WC_MIN; map.win.y.max = WC_MAX;
map.front.plane = 2.0 * WC_MIN;
map.back.plane = 2.0 * WC_MAX;
map.proj.plane = 0.4 * map.proj.ref_point.x;
map.proj.ref_point.x = (map.win.x.min + map.win.x.max) / 2.0;
map.proj.ref_point.y = (map.win.y.min + map.win.y.max) / 2.0;
map.proj.vp.x.min = 0.0; map.proj.vp.x.max = 1.0;
map.proj.vp.y.min = 0.0; map.proj.vp.y.max = 1.0;
map.proj.vp.z.min = 0.0; map.proj.vp.z.max = 1.0;
return;
}

/*****
/eval_view_rep *****/
/****
/****
/****
void
eval_view_rep( rep )
{
    Pview_rep3
    {
        Pint
        err;

        peval_view_map_matrix3( &map, &err, rep->map_matrix );
        if ( err )
            fprintf( stderr, "Error from eval mapping %d\n", err );
    }

    peval_view_ori_matrix3( &vp, &vpa, &vup, &err,
        rep->ori_matrix );
    if ( err )
        fprintf( stderr, "Error from eval orientation %d\n", err );
}

/****
/**** set view rep ****/
/****
/****
void
set_view_rep()
{
    eval_view_rep( &rep );
    rep.clip_limit = map.proj.vp;
    rep.xy_clip = rep.back_clip = rep.front_clip = FIND_NO_CLIP;
    pset_view_rep3( 1, 1, &rep );
}

/****
/**** init input ****/
/****
/****
void
init_input()
{
    Pobjec_data   cvec;
    Pval_data     vvec;
    Pint          ea;
    char          label[20], format[20];
    char          null_string = '\0';

    cvec.pets.pet.r3.num_strings = num_strings;
    cvec.pets.pet.r3.strings = strings;
    ea.x.min = 0.0; ea.x.max = 597.0; ea.y.min = 0.0; ea.y.max = 597.0;
    pint_choices(1, 3, PIN_STATUS_OF, 1, 1, &ea, &vvec);
}

```

```

vvec.low = -1.0;
vvec.high = 1.0;
vvec.pets.pet.ul.label = label;
vvec.pets.pet.ul.format = format;
vvec.pets.pet.ul.low_label = null_string;
vvec.pets.pet.ul.high_label = null_string;
strcpy( format, "%06.2f ");
pinit_val( 1, 1, vpa.delta.x );
pinit_val( 1, 1, vpa.delta.y );
pinit_val( 1, 2, vpa.delta.x );
pinit_val( 1, 2, vpa.delta.y );
pinit_val( 1, 3, vpa.delta.x );
pinit_val( 1, 3, vpa.delta.y );
vvec.low = 4.0 * (-map.proj.ref_point.x);
vvec.high = 4.0 * map.proj.ref_point.x;
strcpy( label, "view plane ");
pinit_val( 1, 4, map.view_plane, -1, &ea, &vvec );

pset_choice_mode( 1, 3, POP_EVENT, PSWITCH_ECHO );
pset_val_mode( 1, 1, POP_EVENT, PSWITCH_ECHO );
pset_val_mode( 1, 2, POP_EVENT, PSWITCH_ECHO );
pset_val_mode( 1, 3, POP_EVENT, PSWITCH_ECHO );
pset_val_mode( 1, 4, POP_EVENT, PSWITCH_ECHO );

/****
/**** valuator event ****/
/****
/****
void
valuator_event( val, in_num )
{
    Pfloat   val;
    Pint     in_num;

    switch ( in_num )
    {
        case 1: vpa.delta.x = val; break;
        case 2: vpa.delta.y = val; break;
        case 3: vpa.delta.x = val; break;
        case 4: map.view_plane = val; break;
        default: /* ? */
            fprintf( stderr, "Unknown valuator device %d\n", in_num );
            return;
    }
    set_view_rep();
}

/****
/**** choice event ****/
/****
/****
int
choice_event( ch )
{
    Pint     ch;
    int      status = 0;
    Pint     int_style;
    extern Pint ConfLabels;
    extern Pint ModLabels;
    extern Pint WorkstnId;
    extern int Cur Stage;

    switch ( ch )
    {

```

```

case 1:
    set view_rep();
    break;
case 2:
    status = QUIT;
    break;
case 3:
    pmapset_struct(Workstaid, ConLabels);
    set view_rep();
    break;
case 4:
    pmapset_struct(Workstaid, ModLabels);
    set view_rep();
    break;
case 5:
    pmapset_struct(Workstaid, ConLabels, 1.0);
    set view_rep();
    break;
case 6:
    pmapset_struct(Workstaid, ModLabels, 1.0);
    set view_rep();
    break;
case 7:
    start animate();
    break;
case 8:
    step forward();
    set view_rep();
    break;
case 9:
    step backward();
    set view_rep();
    break;
case 10:
    goto time();
    set view_rep();
    break;
    )
    return status;
}

/*----- check input -----*/
/*----- check input -----*/
/*----- check input -----*/

int
check_input()
{
    int     status = 0;
    Pint    choice;
    Pint    val;
    Pint    pfloat;
    Pint    class;
    Pint    class;
    Pint    class;
    Pint    status;
    PWS_at  ws_state;

    do {
        ping_ws_at( ws_state );
        if ( ws_state == PWS_ST_WBCL )
            status = QUIT;
        else {
            pmapset_struct( 0.02, &wsat, &class, &sin_num );
            switch ( class ) {

```


bond.c Tue Sep 20 12:10:46 1994 35

```

Norms[3].delta.x = 1.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
Faceets.colr = PCOLOR;
pfill area set3 data(PACSET COLOR, PEDGE NORM, EVERY COORD NORMAL,
FMODEL RGB, aFaceets, num_areas, NULL, aVertexSet);
/* Bottom side */
i = 4;
Vertices[0].x = Square[0][4*3];
Vertices[0].y = Square[0][4*3];
Vertices[1].x = Square[1][4*3];
Vertices[1].y = Square[1][4*3];
Vertices[2].x = Square[2][4*3];
Vertices[2].y = Square[2][4*3];
Vertices[3].x = Square[3][4*3];
Vertices[3].y = Square[3][4*3];
Vertices[0].delta.x = -1.0;
Vertices[0].delta.y = 0.0;
Vertices[1].delta.x = -1.0;
Vertices[1].delta.y = 0.0;
Vertices[2].delta.x = -1.0;
Vertices[2].delta.y = 0.0;
Vertices[3].delta.x = -1.0;
Vertices[3].delta.y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
Faceets.colr = PCOLOR;
pfill area set3 data(PACSET COLOR, PEDGE NORM, EVERY COORD NORMAL,
FMODEL RGB, aFaceets, num_areas, NULL, aVertexSet);
/* Right side */
i = 2;
Vertices[0].x = Square[0][1];
Vertices[0].y = Square[0][1];
Vertices[1].x = Square[1][1];
Vertices[1].y = Square[1][1];
Vertices[2].x = Square[2][1];
Vertices[2].y = Square[2][1];
Vertices[3].x = Square[3][1];
Vertices[3].y = Square[3][1];
Vertices[0].delta.x = 1.0;
Vertices[0].delta.y = 0.0;
Vertices[1].delta.x = 1.0;
Vertices[1].delta.y = 0.0;
Vertices[2].delta.x = 1.0;
Vertices[2].delta.y = 0.0;
Vertices[3].delta.x = 1.0;
Vertices[3].delta.y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
Faceets.colr = PCOLOR;
pfill area set3 data(PACSET COLOR, PEDGE NORM, EVERY COORD NORMAL,
FMODEL RGB, aFaceets, num_areas, NULL, aVertexSet);
/* Left side */
i = 0;
Vertices[0].x = Square[0][4];
Vertices[0].y = Square[0][4];
Vertices[1].x = Square[1][4];
Vertices[1].y = Square[1][4];
Vertices[2].x = Square[2][4];
Vertices[2].y = Square[2][4];
Vertices[3].x = Square[3][4];
Vertices[3].y = Square[3][4];
Vertices[0].delta.x = 1.0;
Vertices[0].delta.y = 0.0;
Vertices[1].delta.x = 1.0;
Vertices[1].delta.y = 0.0;
Vertices[2].delta.x = 1.0;
Vertices[2].delta.y = 0.0;
Vertices[3].delta.x = 1.0;
Vertices[3].delta.y = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
Faceets.colr = PCOLOR;
pfill area set3 data(PACSET COLOR, PEDGE NORM, EVERY COORD NORMAL,
FMODEL RGB, aFaceets, num_areas, NULL, aVertexSet);

```

```

VertexSet.vertex_data = VertexList;
VertexSet.num_vertices = 6;
Facets.colr = FColor;
pfill_area.set3_data(SPACET COLOUR, FIDGE NONE, FVERT COORD NORMAL,
                    FMODEL RGB, fFacets, num_areas, NULL, &VertexSet);

/* Front side */
1 = 1;
Vertices[0].x = Square[0][1];
Vertices[0].y = Square[1][1];
Vertices[0].z = Square[2][1];
Vertices[1].x = Square[0][1+4];
Vertices[1].y = Square[1][1+4];
Vertices[1].z = Square[2][1+4];
Vertices[2].x = Square[0][1+5];
Vertices[2].y = Square[1][1+5];
Vertices[2].z = Square[2][1+5];
Vertices[3].x = Square[0][1+1];
Vertices[3].y = Square[1][1+1];
Vertices[3].z = Square[2][1+1];
Norms[0].delta_x = 0.0;
Norms[0].delta_y = -1.0;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = 0.0;
Norms[1].delta_y = -1.0;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = 0.0;
Norms[2].delta_y = -1.0;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = 0.0;
Norms[3].delta_y = -1.0;
Norms[3].delta_z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
VertexSet.num_vertices = 6;
Facets.colr = FColor;
pfill_area.set3_data(SPACET COLOUR, FIDGE NONE, FVERT COORD NORMAL,
                    FMODEL RGB, fFacets, num_areas, NULL, &VertexSet);

/* Back side */
1 = 0;
Vertices[0].x = Square[0][1];
Vertices[0].y = Square[1][1];
Vertices[0].z = Square[2][1];
Vertices[1].x = Square[0][1+3];
Vertices[1].y = Square[1][1+3];
Vertices[1].z = Square[2][1+3];
Vertices[2].x = Square[0][1+7];
Vertices[2].y = Square[1][1+7];
Vertices[2].z = Square[2][1+7];
Vertices[3].x = Square[0][1+4];
Vertices[3].y = Square[1][1+4];
Vertices[3].z = Square[2][1+4];
Norms[0].delta_x = 0.0;
Norms[0].delta_y = 1.0;
Norms[0].delta_z = 0.0;
Norms[1].delta_x = 0.0;
Norms[1].delta_y = 1.0;
Norms[1].delta_z = 0.0;
Norms[2].delta_x = 0.0;
Norms[2].delta_y = 1.0;
Norms[2].delta_z = 0.0;

```

```

Norms[2].delta_x = 0.0;
Norms[2].delta_y = 1.0;
Norms[2].delta_z = 0.0;
Norms[3].delta_x = 0.0;
Norms[3].delta_y = 1.0;
Norms[3].delta_z = 0.0;
pointnorms[0].point = Vertices[0];
pointnorms[1].point = Vertices[1];
pointnorms[2].point = Vertices[2];
pointnorms[3].point = Vertices[3];
pointnorms[0].norm = Norms[0];
pointnorms[1].norm = Norms[1];
pointnorms[2].norm = Norms[2];
pointnorms[3].norm = Norms[3];
VertexList.ptnorms = pointnorms;
VertexSet.vertex_data = VertexList;
VertexSet.num_vertices = 4;
Facets.colr = FColor;
pfill_area.set3_data(SPACET COLOUR, FIDGE NONE, FVERT COORD NORMAL,
                    FMODEL RGB, fFacets, num_areas, NULL, &VertexSet);

)

/***** SetReflectedanceProperties *****/
/***** SetReflectedanceProperties *****/

SetReflectedanceProperties(WorkstnID, AmbientRef, DiffuseRef, SpecRef,
                             SpecR, SpecG, SpecB, SpecExp)
/* Workstation ID */
/* Ambient Refl. coefficient (0.0->1.0) */
/* Diffuse Refl. coefficient (0.0->1.0) */
/* Specular Refl. coefficient (0.0->1.0) */
/* Specular reflection colour */
/* Specular exponent ("shininess") */

{
    Prefl_props ReflProp;

    ReflProp.ambient_coef = AmbientRef;
    ReflProp.diffuse_coef = DiffuseRef;
    ReflProp.specular_coef = SpecRef;
    ReflProp.specular_coef.type = FMODEL RGB;
    ReflProp.specular_coef.val.general.x = SpecR;
    ReflProp.specular_coef.val.general.y = SpecG;
    ReflProp.specular_coef.val.general.z = SpecB;
    ReflProp.specular_exp = SpecExp;
    post_refl_props(&ReflProp);
}

/***** DefineLightSources *****/
/***** DefineLightSources *****/

DefineLightSources(WorkstnID, 1, /* light source 1 */
                   FLIGHT AMBIENT, /* colour (dark gray) */
                   0.4, 0.4, 0.4, /* position (unused) */
                   0.0, 0.0, 0.0, /* direction (unused) */
                   0.0, 0.0, 0.0, /* conc/spread/atten (unused) */
                   0.0, 0.0, 0.0, 0.0); /* light source 2 */
DefineLightSources(WorkstnID, 2,
                   FLIGHT DIRECTIONAL, /* directional */
                   0.6, 0.6, 0.6, /* colour (dim white) */

```



```

bond.c      Tue Sep 20 12:10:46 1994      37

    0.0, 0.0, 0.0,
    10.0, 10.0, -10.0,
    0.0, 0.0, 0.0, 0.0);
DefineLightSource(WorkstatID, 3,
    PLIGHT DIRECTIONAL,
    0.7, 0.7, 0.7,
    0.0, 0.0, 0.0,
    -10.0, -10.0, -2.0,
    0.0, 0.0, 0.0, 0.0);
)
//*****
//** DefineLightSource ***//
//*****

DefineLightSource(WorkstatID, LightMo, LightType, R, G, B, X, Y, Z, dX, dY, dZ,
    Exponent, Spread, Att1, Att2)
{
    Plint WorkstatID;
    Plint LightMo, LightType;
    Plloat R, G, B;
    Plloat X, Y, Z;
    Plloat dX, dY, dZ;
    Plloat Exponent;
    Plloat Spread;
    Plloat Att1, Att2;
    {
        Plight_src_bundle LightSource; /* light source representation */
        LightSource.type = LightType;
        switch (LightType) {
            case PLIGHT AMBIENT:
                SetOCOL(LightSource.rec.ambient.colr, R, G, B);
                break;
            case PLIGHT DIRECTIONAL:
                SetOCOL(LightSource.rec.directional.colr, R, G, B);
                LightSource.rec.directional.dir.delta_x = dX;
                LightSource.rec.directional.dir.delta_y = dY;
                LightSource.rec.directional.dir.delta_z = dZ;
                break;
            case PLIGHT POSITIONAL:
                SetOCOL(LightSource.rec.positional.colr, R, G, B);
                LightSource.rec.positional.pos.x = X;
                LightSource.rec.positional.pos.y = Y;
                LightSource.rec.positional.pos.z = Z;
                LightSource.rec.positional.coef[0] = Att1;
                LightSource.rec.positional.coef[1] = Att2;
                break;
            case PLIGHT SPOT:
                SetOCOL(LightSource.rec.spot.colr, R, G, B);
                LightSource.rec.spot.pos.x = X;
                LightSource.rec.spot.pos.y = Y;
                LightSource.rec.spot.pos.z = Z;
                LightSource.rec.spot.dir.delta_x = dX;
                LightSource.rec.spot.dir.delta_y = dY;
                LightSource.rec.spot.dir.delta_z = dZ;
                LightSource.rec.spot.exp = Exponent;
                LightSource.rec.spot.coef[0] = Att1;
                LightSource.rec.spot.coef[1] = Att2;
                LightSource.rec.spot.angle = Spread;
                break;
        }
        post_light_src_rep(WorkstatID, LightMo, LightSource);
    }
}
//*****

```

```

//** Phigload it ***//
//*****
void Phigload_it(char *fname)
{
    int ok;
    ModeType *traverseModes;
    int nx, ny, nz;
    char *name;
    FILE *fp;

    fp = fopen("modedata", "w");

    strcpy(FileName, fname);

    printf("%s\n", FileName);

    ok = ReadFile(FileName);

    if (ok) printf("okay\n");

    traverseModes = FirstMode->Next;
    while (traverseModes) {
        name = traverseModes->name;
        nx = traverseModes->x;
        ny = traverseModes->y;
        nz = traverseModes->z;
        traverseModes = traverseModes->Next;
    }

    return;
}

//*****
//** Postproc Make Scale Buttons ***//
//*****

void Postproc_Make_Scale_Buttons()
{
    int i;
    Arg args[10];
    Widget ShButton[3];
    extern Widget ChoicePopupShell1, topLevel;
    extern Widget ChoiceBox1;
    extern int Scale_Var;

    static char *translations = "Override\n\
    <BtnUp> MenuPopdown(Scaling)";
    static char *trans2 = "Override\n\
    <BtnUp> set() notify() unset()";
    ItTranslations changed_translations;

    //** Create the Popup Shell ***//

    ChoicePopupShell1 = ItCreatePopupShell("Scaling",
        transientShellWidgetClass,
        topLevel, NULL, 0);

    changed_translations = ItParseTranslationTable(translations);
    ItOverrideTranslations(ChoicePopupShell1, changed_translations);

    //** Create the menu box ***//

```


bond.c Tue Sep 20 12:10:46 1994 39

```

Scale_Var = variable;

XtSetArg(args[0], XtN"label", heading(variable));
XtSetValues(MainDialogScale, args, 1);

Our_Stage = init_timestage;

return;
)

/*****
/**** Postproc_Make_Delta_Buttons() ****/
/*****

void Postproc_Make_Delta_Buttons()
{
    int i;
    Arg args[10];
    Widget Button[0];
    extern Widget ChoicePopupMenu, toplabel;
    extern Widget ChoiceBox;
    extern int Our_Var;

    static char *translations = "Override\n\
    <Alt>: MenuPopDown(Variable)";
    static char *trans2 = "Override\n\
    <Alt>: set() notify() unset()";
    XtTranslations changed_translations;

    /**** Create the Popup Shell ****/

    ChoicePopupMenu = XtCreatePopupMenu("Variable", transientShellWidgetClass,
    toplabel, NULL, 0);

    changed_translations = XtParseTranslations(changed_translations);
    XtOverrideTranslations(ChoicePopupMenu, changed_translations);

    /**** Create the menu box ****/

    ChoiceBox = XtCreateManagedWidget("ChoiceBox", formWidgetClass,
    ChoicePopupMenu, NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 1); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetValues(ChoiceBox, args, 1);

    /**** Create the buttons ****/

    Our_Var = NONE;

    Button[0] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Energy"); i++;
    XtSetValues(Button[0], args, 1);
    XtSetCallBack(Button[0], XtNcallback, Change_Var, (int *) POWER);

    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Momentum"); i++;
    XtSetValues(Button[1], args, 1);
    XtSetCallBack(Button[1], XtNcallback, Change_Var, (int *) MOMENTUM);

    Button[2] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Displacement"); i++;
    XtSetValues(Button[2], args, 1);
    XtSetCallBack(Button[2], XtNcallback, Change_Var, (int *) DISPLACEMENT);

    Button[3] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Flow"); i++;
    XtSetValues(Button[3], args, 1);
    XtSetCallBack(Button[3], XtNcallback, Change_Var, (int *) FLOW);

    Button[4] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Effort"); i++;
    XtSetValues(Button[4], args, 1);
    XtSetCallBack(Button[4], XtNcallback, Change_Var, (int *) EFFORT);

    Button[5] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Energy"); i++;
    XtSetValues(Button[5], args, 1);
    XtSetCallBack(Button[5], XtNcallback, Change_Var, (int *) ENERGY);

    Button[6] = XtCreateManagedWidget("button", commandWidgetClass,
    ChoiceBox,
    NULL, 0);

    i = 0;
    XtSetArg(args[1], XtNborderWidth, 0); i++;
    XtSetArg(args[1], XtNborderStyle, NULL); i++;
    XtSetArg(args[1], XtNresizable, FALSE); i++;
    XtSetArg(args[1], XtNlabel, "Power"); i++;
    XtSetValues(Button[6], args, 1);
    XtSetCallBack(Button[6], XtNcallback, Change_Var, (int *) POWER);

```

```

bond.c      Tue Sep 20 12:10:46 1994      40

    XSetArg(args[1], XtBorderWidth, Pbutton[5]); i++;
    XSetArg(args[1], XtBorderWidth, MWLd); i++;
    XSetArg(args[1], XtLabel, " signal "); i++;
    XSetArg(args[1], XtResourceable, FALSE); i++;
    XtAddValues(Pbutton[6], args, 1);
    XtAddCallback(Pbutton[6], XtCallback, Change_Var, (int *) SIGNAL);

    Pbutton[7] = XtCreateManagedWidget("button", commandWidgetClass,
        CholesterolBox,
        MWLd, 0);

    i = 0;
    XSetArg(args[1], XtBorderWidth, 0); i++;
    XSetArg(args[1], XtBorderWidth, Pbutton[6]); i++;
    XSetArg(args[1], XtBorderWidth, MWLd); i++;
    XSetArg(args[1], XtLabel, " None "); i++;
    XSetArg(args[1], XtResourceable, FALSE); i++;
    XtAddValues(Pbutton[7], args, 1);
    XtAddCallback(Pbutton[7], XtCallback, Change_Var, (int *) NONE);

    changed_translations = XtParseTranslationsTable(trans2);
    for (i = 0; i < 6; i++)
        XtOverrideTranslations(Pbutton[i], changed_translations);

    return;
}

/*.....*/
/*... Change_Var ....*/
/*.....*/

void Change_Var(w, client_data, call_data)
Widget w;
caddr_t call_data, client_data;
{
    int variable = (int) client_data;
    Change_Var2(variable);
    return;
}

/* FUNCTION Change_Var2
 *
 * Purpose
 *   Change the heading in the menu box - called by ChangeVar
 *
 * Inputs
 *   int variable - the value of the new variable
 *
 * Outputs
 *   None
 *
 * Programming History
 *   1993 Sept, Gayle E. Ermar, Original function written
 *
 */

void Change_Var2(int variable)
{
    extern int Cur_Var;
    extern Widget MainDialogVariable;
    extern int Cur_Stage;
    extern int init_timestep;
    char *heading[] = {
        "Power",
        "Effort",
        "Flow"
    };
}

```

bond.c Tue Sep 20 12:10:46 1994 41

```

char *ptr;
char Temp[13];
int i, j, tok, stage;
extern int num_stages;
real value;
extern real ppdata[200][300][2];
FILE *fp;
extern int Cur_Stage;
extern int new_num_stages, init_timestep, final_timestep;
extern real pppmDialogFinalTimeVal, pppmDialogFinalTimeVal;

fp = fopen("data1", "w");

/*** Calculate actual time range ****/
new_num_stages = (pppmDialogFinalTimeVal -
pppmDialogInitTimeVal)/setbl_1.dtsr + 1;
init_timestep = (pppmDialogInitTimeVal -
setbl_1.tin)/setbl_1.dtsr;
final_timestep = num_stages - (setbl_1.tfin -
pppmDialogFinalTimeVal)/setbl_1.dtsr;

Cur_Stage = init_timestep;

/*** First store Efforts in array in order of bonds ****/
i = 0;
traverseCons = First_Connector->Next;
while (traverseCons) {
    for (ondx = 0; ondx < seqlb2_1.novv; ++ondx) {
        ptr = traverseCons->name;
        tempstr[0] = '\0';
        tempstr[1] = '\0';
        strcat(tempstr, ptr);
        ptr = seqlb1_1.ovnl + (12 * ondx);
        sprintf(Temp, "%s", mchare(ptr, 12), ptr);
        tok = strump(tempstr, Temp);
        if (tok == 0) {
            for (stage = init_timestep; stage <= final_timestep; ++stage) {
                Get_Value(&pos0bk_1.relt[ondx*MAXRE], stage, &value);
                ppdata[i][stage][0] = value;
            }
        }
        traverseCons = traverseCons->Next; ++i;
    }
    /*** Next store Flows in array in order of bonds ****/
    traverseCons = First_Connector->Next;
    i = 0;
    while (traverseCons) {
        for (ondx = 0; ondx < seqlb2_1.novv; ++ondx) {
            ptr = traverseCons->name;
            tempstr[0] = '\0';
            tempstr[1] = '\0';
            strcat(tempstr, ptr);
            ptr = seqlb1_1.ovnl + (12 * ondx);
            sprintf(Temp, "%s", mchare(ptr, 12), ptr);
            tok = strump(tempstr, Temp);
            if (tok == 0) {
                for (stage = init_timestep; stage <= final_timestep; ++stage) {
                    Get_Value(&pos0bk_1.relt[ondx*MAXRE], stage, &value);
                    ppdata[i][stage][1] = value;
                }
            }
            traverseCons = traverseCons->Next; ++i;
        }
        /*** find ranges and max/mins **/
        ranges = 0.0;
        minranges = 9999999999.0;
        num_cons = (seqlb2_1.novv - seqlb2_1.mxi) / 2;
        for (stage = 0; stage < new_num_stages; stage++) {
            for (index = 0; index < num_cons; index++) {
                datas = &data[index][stage];
                if (datas < 0.0) datas = -datas;
                if (datas > ranges) ranges = datas;
            }
        }
    }
}

traverseCons = traverseCons->Next; ++i;
}

/*** find ranges and max/mins **/
ranges = 0.0;
minranges = 9999999999.0;
num_cons = (seqlb2_1.novv - seqlb2_1.mxi) / 2;
for (stage = 0; stage < new_num_stages; stage++) {
    for (index = 0; index < num_cons; index++) {
        datas = &data[index][stage];
        if (datas < 0.0) datas = -datas;
        if (datas > ranges) ranges = datas;
    }
}

```

```

    }
    range = ranges;
    for (stage = 0; stage < new_num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            data = pdata[index][stage];
            if (data < 0.0) minrange = minrange;
            if (data < 0.0) data = -data;
            if (data < minrange && data != 0.0) minrange = data;
        }
        minrange = minrange;
        max = 99999999.0;
        min = 999999999.0;
        for (stage = 0; stage < num_stages; stage++) {
            for (index = 0; index < num_conns; index++) {
                data = pdata[index][stage];
                if (data > max) max = data;
                if (data < min) min = data;
            }
        }
        max = max; min = min;
    }
}

/* Find Data_Ranges */
void Find_Data_Ranges()
{
    extern real pdata[200][300][2];
    extern int num_stages, new_num_stages;
    int stage, index, num_conns;
    real data, dataf, datap, max, minf, minp, minr, minf, minp;
    extern real prange, orange, frange;
    extern real pairng, emingr, emingp;

    max = maxf = maxp = 0.0;
    min = minf = minp = 0.0;
    num_conns = (scab2_1.nrvv - scab2_1.nxi) / 2;

    /* find maximum magnitude */
    for (stage = 0; stage < new_num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            data = pdata[index][stage][0];
            dataf = pdata[index][stage][1];
            datap = data + dataf;
            if (data < 0.0) data = -data;
            if (dataf < 0.0) dataf = -dataf;
            if (datap > max) max = data;
            if (dataf > maxf) maxf = dataf;
            if (datap > maxp) maxp = datap;
        }
    }
    prange = maxp;
    orange = max;
    frange = maxf;

    /* find minimum magnitude */
    for (stage = 0; stage < new_num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            data = pdata[index][stage][0];
            dataf = pdata[index][stage][1];
            datap = data + dataf;
            if (data < 0.0) minrange = minrange;
            if (dataf < 0.0) minf = minf;
            if (datap < 0.0) minp = minp;
            if (data < 0.0) data = -data;
            if (dataf < 0.0) dataf = -dataf;
            if (datap < minrange && data != 0.0) minrange = data;
            if (dataf < minf && dataf != 0.0) minf = dataf;
            if (datap < minp && datap != 0.0) minp = datap;
        }
    }
    pairng = minp;
    emingr = min;
    emingp = minf;
}

/* Integrate Data */
void Integrate_Data(Widget w)
{
    ModeType *traverseModes;
    ConnectorType *traverseCons;
    real tinit, tfinal, dtat, sum;
    int i, j, stage;
    extern int num_stages, new_num_stages;
    extern real pdata[200][300][2];
    extern real ppdata[200][300][3];
    FILE *fp;
    char *tmp;

    tinit = ppmindialogInitTimeVal;
    tfinal = ppmindialogFinalTimeVal;
    dtat = scib4_1.dtcstr;

    /* Calculate momentums from efforts */
    /* First stage */
    traverseCons = First_Connector->Next;
    index = 0;
    while (traverseCons) {
        ppdata[index][0][0] = 0.0;
        traverseCons = traverseCons->Next; ++index;
    }
    /* Second stage */
    sum = 0;
    traverseCons = First_Connector->Next;
    index = 0;
    while (traverseCons) {
        sum = ppdata[index][0][0] + ppdata[index][1][0];
    }
}

```

bond.c Tue Sep 20 12:10:46 1994 43

```

ppdata[index][1][0] = sum * deltat / 2.0;
traverseCons = traverseCons->Next; ++index;
}

/*** Rest of stages ****/
for (stage = 2; stage < new_num_stages; stage++) {
    sum = 0;
    traverseCons = First_Connector->Next;
    index = 0;
    while (traverseCons) {
        sum = 0;
        sum = ppdata[index][0][0] + ppdata[index][stage][0];
        for (j = 1; j < stage; j++) {
            sum = sum + 2.0 * ppdata[index][j][0];
        }
        ppdata[index][stage][0] = (((real) stage * deltat)
            / (2.0 * (real) (stage))) * sum;
        traverseCons = traverseCons->Next; ++index;
    }

    /*** Calculate displacements from flows ****/

    /*** First stage ****/
    traverseCons = First_Connector->Next;
    index = 0;
    while (traverseCons) {
        sum = 0;
        sum = ppdata[index][0][0] + ppdata[index][0][1]
            + ppdata[index][1][0] + ppdata[index][1][1];
        ppdata[index][1][2] = sum * deltat / 2.0;
        traverseCons = traverseCons->Next; ++index;
    }

    /*** Rest of stages ****/
    for (stage = 2; stage < new_num_stages; stage++) {
        sum = 0;
        traverseCons = First_Connector->Next;
        index = 0;
        while (traverseCons) {
            sum = 0;
            sum = ppdata[index][0][0] + ppdata[index][0][1]
                + ppdata[index][stage][0] + ppdata[index][stage][1];
            for (j = 1; j < stage; j++) {
                sum = sum + 2.0 * ppdata[index][j][0] + ppdata[index][j][1];
            }
            ppdata[index][stage][2] = (((real) stage * deltat)
                / (2.0 * (real) (stage))) * sum;
            traverseCons = traverseCons->Next; ++index;
        }

        traverseCons = First_Connector->Next;
        index = 0;
        while (traverseCons) {
            temp = traverseCons->name;
            traverseCons = traverseCons->Next;
            index++;
        }
        Int_Ranges();
    }
}

/* FUNCTION Int_Ranges

```

bond.c Tue Sep 20 12:10:46 1994 44

```

    *
    */
void Init Ranges()
{
    extern real pndint[200][300][3];
    extern int num_stages, new num_stages;
    int stage, index, num_conns;
    real dataq, dataq, datan, maxq, maxp, minq, minp, minn;
    extern real parange, qrange, nrange;
    extern real pmining, qmining, nmining;

    maxp = maxq = maxn = 0.0;
    minp = minq = minn = 999999999.0;

    num_conns = (seqlb2 1.novv - solab2 1.nxi) / 2;

    /* find maximum magnitudes */
    for (stage = 0; stage < new num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            dataq = pndint[index][stage][0];
            datan = pndint[index][stage][1];
            dataq = pndint[index][stage][2];
            if (dataq < 0.0) dataq = -dataq;
            if (datan < 0.0) datan = -datan;
            if (dataq > 0.0) dataq = dataq;
            if (datan > 0.0) datan = datan;
        }
    }
    nrange = maxn;
    qrange = maxq;
    parange = maxp;

    /* find minimum magnitudes */
    for (stage = 0; stage < new num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            dataq = pndint[index][stage][0];
            datan = pndint[index][stage][1];
            dataq = pndint[index][stage][2];
            if (dataq < 0.0) dataq = -dataq;
            if (datan < 0.0) datan = -datan;
            if (dataq > 0.0) dataq = dataq;
            if (datan > 0.0) datan = datan;
        }
    }
    mining = minn;
    qmining = minq;
    pmining = minp;

    /* Find Makes_Mins */
    void Find_Makes_Mins(Widget w)
    {
        extern real pndint[200][300][3];
        extern real pndata[200][300][2];
        extern int num_stages;
        int stage, index, num_conns;

```

```

    real dataq, dataq, datan, dataq, dataq, dataq;
    real maxp, maxq, maxn, maxp, maxq, maxn;
    real minp, minq, minn, minp, minq, minn;
    extern real pmax, qmax, nmax, pmax, qmax, nmax;
    extern real pmin, qmin, nmin, pmin, qmin, nmin;
    FILE *fp;

    fp = fopen("mimmaxes", "w");

    maxp = maxq = maxn = maxn = maxn = maxp = maxp = maxp = -999999999.0;
    minp = minq = minn = minn = minn = minp = minp = minp = 999999999.0;

    num_conns = (seqlb2 1.novv - solab2 1.nxi) / 2;

    for (stage = 0; stage < num_stages; stage++) {
        for (index = 0; index < num_conns; index++) {
            dataq = pndint[index][stage][0];
            datan = pndint[index][stage][1];
            dataq = pndint[index][stage][2];
            dataq = pndata[index][stage][0];
            dataq = pndata[index][stage][1];
            dataq = dataq + dataq;
            if (dataq > maxp) maxp = dataq;
            if (dataq > maxq) maxq = dataq;
            if (dataq > maxn) maxn = dataq;
            if (dataq > maxp) maxp = dataq;
            if (dataq > maxq) maxq = dataq;
            if (dataq > maxn) maxn = dataq;
            if (dataq < minp) minp = dataq;
            if (dataq < minq) minq = dataq;
            if (dataq < minn) minn = dataq;
            if (dataq < minp) minp = dataq;
            if (dataq < minq) minq = dataq;
            if (dataq < minn) minn = dataq;
        }
    }
    nmax = maxn; minn = minn;
    qmax = maxq; qmin = minq;
    pmax = maxp; pmin = minp;
    maxp = maxp; pmin = minp;
    qmax = maxq; qmin = minq;
    fmax = maxf; fmin = minf;

    /* Get Value */
    void Get_Value(real data[], int stage, real *value)
    {
        real datavalue;

        datavalue = data[stage];

        *value = datavalue;
    }

    /* Main Dialog Check */
    int Main Dialog Check()
    {
        int

```


bond.c Tue Sep 20 12:10:46 1994 45

```

i, j;
Argv[5];
char
extern Widget MainDialogInitTimeValue, MainDialogFinalTimeValue;
/*Temp, Message[100];
extern Widget MainDialogInitTimeValue, MainDialogFinalTimeValue;

/* Get the initial time they entered and check it */
i = 0;
XtSetArg(args[1], XtMString, &Temp); ++i;
XtGetValues(MainDialogInitTimeValue, args, 1);

if (!asciitof(Temp, AppMainDialogInitTimeVal)) {
    Problem Pop("Initial display time is not a valid number");
    return 0;
}
if (ppMainDialogInitTimeVal < actib4.i.tfin) {
    Problem Pop("Initial display time must be greater than or equal to -
    'initial integration time'");
}

i = 0;
sprintf(Temp, "%.5f", actib4.i.tfin);
formatParam(Temp);
XtSetArg(args[1], XtMString, Temp); ++i;
XtGetValues(MainDialogInitTimeValue, args, 1);
return 0;
}

if (ppMainDialogFinalTimeVal >= actib4.i.tfin) {
    Problem Pop("Initial display time must be less than -
    'the final integration time'");
}

i = 0;
sprintf(Temp, "%.5f", actib4.i.tfin);
formatParam(Temp);
XtSetArg(args[1], XtMString, Temp); ++i;
XtGetValues(MainDialogFinalTimeValue, args, 1);
return 0;
}

/* Get the final time they entered */
i = 0;
XtSetArg(args[1], XtMString, &Temp); ++i;
XtGetValues(MainDialogFinalTimeValue, args, 1);
if (!asciitof(Temp, AppMainDialogFinalTimeVal)) {
    Problem Pop("Final time is not a valid number");
    return 0;
}
if (ppMainDialogFinalTimeVal > actib4.i.tfin) {
    Problem Pop("Final display time must be less than or equal to -
    'final integration time'");
}

i = 0;
sprintf(Temp, "%.5f", actib4.i.tfin);
formatParam(Temp);
XtSetArg(args[1], XtMString, Temp); ++i;
XtGetValues(MainDialogFinalTimeValue, args, 1);
return 0;
}

if (ppMainDialogFinalTimeVal <= ppMainDialogInitTimeVal) {
    Problem Pop("Final display time must be greater than -
    'initial display time'");
}

i = 0;
sprintf(Temp, "%.5f", actib4.i.tfin);
formatParam(Temp);
XtSetArg(args[1], XtMString, Temp); ++i;
XtGetValues(MainDialogFinalTimeValue, args, 1);
return 0;
}

formatParam(Temp);
XtSetArg(args[1], XtMString, Temp); ++i;
XtGetValues(MainDialogFinalTimeValue, args, 1);
return 0;
}

return 1;
}

/*.....*/
/*... step forward ....*/
/*.....*/

void step_forward()
{
    extern int Cur Stage;
    extern int timestep;
    extern int final timestep;
    int flag;

    flag = Postproc Check Log();
    if (flag < 0) return;

    timestep = Cur Stage;
    if (timestep >= final timestep) {
    }
    else {
        Phigs get colors();
        /* get values and edit connector structure */
        Cur Stage = Cur Stage + 1;
    }
}

/*.....*/
/*... step backward ....*/
/*.....*/

void step_backward()
{
    extern int Cur Stage;
    extern int timestep;
    extern int final timestep;
    int flag;

    flag = Postproc Check Log();
    if (flag < 0) return;

    timestep = Cur Stage - 2;
    if (timestep >= final timestep) {
    }
    else {
        /* get values and edit connector structure */
        Phigs get colors();
        Cur Stage = Cur Stage - 1;
    }
}

/*.....*/
/*... goto time ....*/

```

bond.c Tue Sep 20 12:10:46 1994 46

```

/*****
void goto_time()
{
    extern int Cur Stage;
    extern int timestage;
    extern int init_timestage;
    extern int final_timestage;
    real new_time;
    int new_stage;
    char *time;
    int flag;
    extern FILE *fp1;

    fp1 = fopen("Thrdpwr", "w");
    flag = Postproc Check Log();
    if (flag < 0) return;

    /*** Get time value to go to ***/
    OutDialog("Enter time to Display: ", atime);
    new_time = atof(time);

    if (new_time < ppMainDialogInitTimeVal ||
        new_time > ppMainDialogFinalTimeVal) {
    }
    else {
        new_stage = (new_time - actib4.l.tin) / actib4.l.dstr;
        timestage = new_stage;
        Phigs_get_colors();
        Phigs_get_colors();
        Cur_Stage = timestage + 1;
    }
}

/***** start animate *****/
void start_animate()
{
    extern int timestage;
    extern int new_num_stages, init_timestage, final_timestage;
    extern int Scale Var;
    extern int Cur Var;
    extern int Cur Stage;
    extern int WorkethID;
    int delay;
    real end_time;
    int end_stage;
    int i, length;
    Arg args(5);
    char *time;
    Display "diop;
    char *end string;
    extern real ppMainDialogFinalTimeVal, ppMainDialogInitTimeVal;
    XtEvent event;
    int flag;

    /*** check for log of negative problem ***/
    flag = Postproc Check Log();

```

```

bond.c      Tue Sep 20 12:10:46 1994      47

case MEMORY: min = pmin; break;
case MEMORY: min = min; break;
}
if (min < 0 || min <= 0) {
    Problem Pop("Cannot use log scaling with negative values");
    return -1;
}
}
else return 0;
}

/***** Phigs get colors *****/
/***** *****/

void Phigs get_colors()
{
    extern int Cur Var;
    extern real orange, frange, prange, grange, parange, nrange;
    extern real ppdata[200][300][2];
    extern real ppdata[200][300][3];
    real value, evalue, fvalue;
    NodeType *traverseCons;
    ConnectorType *traverseCons;
    NodeType *fromNode, *toNode;
    int i, index;
    Pint struct ind;
    extern int num_stages;
    extern int timestage;
    int switch_arrow;
    real time;
    char *ptring;
    FILE *fp;
    Poval bond_color;
    real conx1, cony1, conx2, cony2, conx1, conx2;
    real length, length;
    extern real scale_factor;
    int type;
    Poval FacetColorRed;
    static Prgb Red;
    extern FILE *fp1;

    Red.red = 0.7, Red.green = 0.6, Red.blue = 0.6;
    FacetColorRed.ind = 1;
    FacetColorRed.direct.rgb = Red;

    /* go through bonds and get color */

    traverseCons = First_Connector->Next;
    index = 0;
    struct_ind = 10;
    while (traverseCons) {
        /* determine if signal */

        Type = traverseCons->type;
        if (Type == SIGNAL) {
            switch_arrow = 0;
            if (Cur Var == SIGNAL) {
                value = ppdata[index][timestage];
                value = value / orange;
                if (value < 0) {
                    switch_arrow = 1;
                    value = -value;
                }
            }
            else switch_arrow = 0;
            break;
        }
        case MEMORY:
            value = ppdata[index][timestage][0];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case DISPLACEMENT:
            value = ppdata[index][timestage][1];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case MOMENTUM:
            value = ppdata[index][timestage][0];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case POWER:
            value = ppdata[index][timestage][0];
            value = ppdata[index][timestage][1];
            fvalue = ppdata[index][timestage][1];
            value = value * fvalue;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case FLOW:
            value = ppdata[index][timestage][1];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
    }
    switch (Cur Var) {
        case EFFORT:
            value = ppdata[index][timestage][0];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case FLOW:
            value = ppdata[index][timestage][1];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case POWER:
            value = ppdata[index][timestage][0];
            value = ppdata[index][timestage][1];
            fvalue = ppdata[index][timestage][1];
            value = value * fvalue;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
        case FLOW:
            value = ppdata[index][timestage][1];
            value = value / orange;
            if (value < 0) {
                switch_arrow = 1;
                value = -value;
            }
            else switch_arrow = 0;
            break;
    }
}

```

```

    )
    else switch arrow = 0;
    break;
))

/* determine color depending on scaling type **/
switch (Scale_Var) {
case LIN:
    if (Type == SIGNAL && Cur_Var != SIGNAL ||
        Type != SIGNAL && Cur_Var == SIGNAL) {
        bond_color = PowerGray;
    }
    else {
        bond_color = lin_color_value(value);
    }
    break;
case LOG:
    if (Type == SIGNAL && Cur_Var != SIGNAL ||
        Type != SIGNAL && Cur_Var == SIGNAL) {
        bond_color = PowerGray;
    }
    else {
        bond_color = log_color_value(value);
    }
    break;
case ASLOG:
    if (Type == SIGNAL && Cur_Var != SIGNAL ||
        Type != SIGNAL && Cur_Var == SIGNAL) {
        bond_color = PowerGray;
    }
    else {
        bond_color = abelog_color_value(value);
    }
    break;
}

/* calculate the bond length */
fromNode = traverseCons->from;
toNode = traverseCons->to;
conx1 = (real) fromNode->x;
cony1 = (real) fromNode->y;
conx2 = (real) fromNode->z;
cony2 = (real) toNode->x;
conz2 = (real) toNode->z;

lengthraw = sqrt((conx1-conx2)*(conx1-conx2) +
                  (cony1-cony2)*(cony1-cony2) + (conz1-conz2)*(conz1-conz2));
length = (lengthraw / scale_factor) - 3.0;

/* modify structure (color and direction) */
switch (Type) {
case SIGNAL:
    if (switch_arrow == 0) {
        Recreate Connector(struct_ind, length, bond_color);
    }
    else {
        Recreate Connector_Backward(struct_ind, length, bond_color);
    }
    break;
default:
    if (switch_arrow == 0) {
        Recreate Cyl w Color(struct_ind, length, bond_color);
    }
    else {
        Recreate Cyl Backward(struct_ind, length, bond_color);
    }
    break;
}

/* increment the indices and move to next connector **/
++index;
++struct ind;
traverseCons = traverseCons->Next;
}
change time();
}

)

/***** lin color value *****/
/***** log color value *****/
/***** abelog color value *****/

Poval lin_color_value(double value)
{
    extern Poval PCViolet, PCBlue, PCLtBlue, PCGreen, PCYellowGn,
    PCYellow, PCGold, PCOrange, PCPink, PCRed;
    Poval ColorValue;
    extern FILE *fp1;

    if (value < 0.1)
        ColorValue = PCViolet;
    else if (value < 0.2) ColorValue = PCBlue;
    else if (value < 0.3) ColorValue = PCLtBlue;
    else if (value < 0.4) ColorValue = PCGreen;
    else if (value < 0.5) ColorValue = PCYellowGn;
    else if (value < 0.6) ColorValue = PCYellow;
    else if (value < 0.7) ColorValue = PCGold;
    else if (value < 0.8) ColorValue = PCOrange;
    else if (value < 0.9) ColorValue = PCPink;
    else ColorValue = PCRed;

    return ColorValue;
}

/***** log_color_value(real value) *****/
/***** log_color_value *****/
/***** abelog_color_value *****/

Poval log_color_value(real value)
{
    extern Poval PCViolet, PCBlue, PCLtBlue, PCGreen, PCYellowGn,
    PCYellow, PCGold, PCOrange, PCPink, PCRed;

    int actcolor;
    double max, min, delta;
    extern int Cur_Var;
    extern real emax, emin, fmax, fmin, pmax, pmin, qmax, qmin, pmaxx,
    pminx, nmaxx, nminx;

    switch (Cur_Var) {
        case EFFORT: max = emax, min = emin; break;
    }
}

```

bond.c Tue Sep 20 12:10:46 1994 49

```

    case FLOW: max = fmax; min = fmin; break;
    case POWER: max = pmax; min = pmin; break;
    case DISPLACEMENT: max = qmax; min = qmin; break;
    case MOMENTUM: max = pmx; min = pmi; break;
    case ENERGY: max = emx; min = emi; break;
  )

  delta = (log10(max) - log10(min)) / 10;

  if (log10(value) < delta * 1 + log10(min))
    return PCViolet;
  else if (log10(value) < delta * 2 + log10(min))
    return FCBue;
  else if (log10(value) < delta * 3 + log10(min))
    return FCLtBlue;
  else if (log10(value) < delta * 4 + log10(min))
    return FCGreen;
  else if (log10(value) < delta * 5 + log10(min))
    return FCTellowGn;
  else if (log10(value) < delta * 6 + log10(min))
    return FCTellow;
  else if (log10(value) < delta * 7 + log10(min))
    return FCGold;
  else if (log10(value) < delta * 8 + log10(min))
    return FCOrange;
  else if (log10(value) < delta * 9 + log10(min))
    return FCPink;
  else
    return FCRed;
}

/***** set colors *****/
/***** void set colors() *****/

void set colors()
{
  extern Pcolor PCViolet, FCBue, FCLtBlue, FCGreen, FCTellowGn,
  FCTellow, FCGold, FCOrange, FCPink, FCRed;
  static Prgb Violet, Blue, LtBlue, Green, YellowGn,
  Yellow, Gold, Orange, Pink, Red;
  extern Pcolor PowerGray, PowerGreen, PowerBlue, PowerLtBlue,
  PowerPurple, PowerYellow;
  static Prgb PGray, PGreen, PBlue, PCLtBlue, PPurple, PFellow;

  Violet.red = 0.6, Violet.green = 0.0, Violet.blue = 0.7;
  Blue.red = 0.2, Blue.green = 0.0, Blue.blue = 0.9;
  LtBlue.red = 0.0, LtBlue.green = 0.6, LtBlue.blue = 0.7;
  Green.red = 0.0, Green.green = 0.6, Green.blue = 0.3;
  YellowGn.red = 0.0, YellowGn.green = 0.9, YellowGn.blue = 0.1;
  Yellow.red = 0.8, Yellow.green = 0.8, Yellow.blue = 0.2;
  Gold.red = 0.8, Gold.green = 0.6, Gold.blue = 0.1;
  Orange.red = 0.8, Orange.green = 0.4, Orange.blue = 0.0;
  Pink.red = 0.8, Pink.green = 0.1, Pink.blue = 0.2;
  Red.red = 1.0, Red.green = 0.1, Red.blue = 0.55;

  PGray.red = 0.9, PGray.green = 0.9, PGray.blue = 0.9;
  PGreen.red = 0.5, PGreen.green = 1.0, PGreen.blue = 0.5;
  PBlue.red = 0.2, PBlue.green = 0.2, PBlue.blue = 0.7;
  PCLtBlue.red = 0.5, PCLtBlue.green = 0.6, PCLtBlue.blue = 1.0;
  PPurple.red = 0.9, PPurple.green = 0.5, PPurple.blue = 0.8;
  PFellow.red = 0.9, PFellow.green = 0.8, PFellow.blue = 0.2;

  PCViolet.ind = 1, PCViolet.direct.rgb = Violet;
  FCBue.ind = 1, FCBue.direct.rgb = Blue;
  FCLtBlue.ind = 1, FCLtBlue.direct.rgb = LtBlue;
  FCGreen.ind = 1, FCGreen.direct.rgb = Green;
  FCTellowGn.ind = 1, FCTellowGn.direct.rgb = YellowGn;
  FCTellow.ind = 1, FCTellow.direct.rgb = Yellow;
  FCGold.ind = 1, FCGold.direct.rgb = Gold;
  FCOrange.ind = 1, FCOrange.direct.rgb = Orange;
  FCPink.ind = 1, FCPink.direct.rgb = Pink;
  FCRed.ind = 1, FCRed.direct.rgb = Red;

  PowerGray.ind = 1, PowerGray.direct.rgb = PGray;
  PowerGreen.ind = 1, PowerGreen.direct.rgb = PGreen;
  PowerBlue.ind = 1, PowerBlue.direct.rgb = PBlue;
  PowerLtBlue.ind = 1, PowerLtBlue.direct.rgb = PCLtBlue;
}

```

```
bond.c      Tue Sep 20 12:10:46 1994
    PowerPurple.ind = 1, PowerPurple.direct.rgb = PPurple;
    PowerYellow.ind = 1, PowerYellow.direct.rgb = PYellow;
)
/* End of export3d.c */
```

Appendix C

ENPORT Model Files

The following pages contain the ENPORT model files that were generated to model and simulate each of the examples described in chapters 5 and 6. Stored in these files are all of the positions of nodes and connectors in the system graph along with the constitutive equation functions and parameters for each node. The last file in the series is a modified 3-D file which contains z locations for nodes, as well as x and y values.

Carvps.asp Thu Aug 11 09:59:09 1994 1

READING

FILE MODEL
Carvps.asp 09:59:01 08/11/94 IMPORT/we 1.0-beta

SYSTEM GRAPH DESCRIPTION

MODE	TYPE	YLOC	YLOC	ACT	MACRO
136	MIOW	-397.	1293.	T	
1D	MIOW	49.	995.	T	
1M6	MIOW	49.	1293.	T	
826	MIOW	-845.	1293.	T	
RD	MIOW	497.	1144.	T	
CK	MIOW	497.	995.	T	
TP7	MIOW	796.	845.	T	
TP9	MIOW	-99.	696.	T	
TP10	MIOW	-397.	995.	T	
874	MIOW	1095.	-1095.	T	
873	MIOW	347.	-1095.	T	
872	MIOW	-99.	-1095.	T	
871	MIOW	-995.	-1095.	T	
CM4	MIOW	1393.	-1095.	T	
RM4	MIOW	1692.	-945.	T	
1M4	MIOW	1393.	-945.	T	
CM3	MIOW	497.	-1095.	T	
RM3	MIOW	945.	-945.	T	
1M3	MIOW	497.	-945.	T	
CM2	MIOW	-547.	-1095.	T	
RM2	MIOW	-845.	-945.	T	
1M2	MIOW	-397.	-945.	T	
CM1	MIOW	-1144.	-1095.	T	
RM1	MIOW	-1592.	-945.	T	
1M1	MIOW	-1144.	-945.	T	
01	MIOW	1095.	-796.	T	
02	MIOW	347.	-796.	T	
03	MIOW	-99.	-796.	T	
04	MIOW	-995.	-796.	T	
82M6	MIOW	1542.	-796.	T	
1M6	MIOW	1542.	-697.	T	
82M3	MIOW	796.	-796.	T	
1M3	MIOW	796.	-697.	T	
CM2	MIOW	-547.	-796.	T	
RM2	MIOW	-1442.	-697.	T	
1M2	MIOW	-1442.	-697.	T	
CM1	MIOW	1095.	-697.	T	
RM1	MIOW	347.	-697.	T	
1M1	MIOW	-99.	-697.	T	
013	MIOW	1542.	-49.	T	
013	MIOW	1095.	-347.	T	
CM3	MIOW	796.	-347.	T	
RM3	MIOW	796.	-99.	T	
115	MIOW	796.	-49.	T	
015	MIOW	347.	-347.	T	
CM2	MIOW	-547.	-347.	T	
RM2	MIOW	-547.	-99.	T	
111	MIOW	-547.	-49.	T	
011	MIOW	-99.	-247.	T	
CM1	MIOW	-1442.	-347.	T	
RM1	MIOW	-1442.	-99.	T	
117	MIOW	-1442.	-49.	T	

CONNECTOR TYPE FROM TO VERTICES

09	BOV	126	IM6
90	BOV	826	126
00	BOV	126	0710
91	BOV	0710	1M
07	BOV	1D	RD
06	BOV	1D	CK
05	BOV	0710	1D
04	BOV	TP7	0710
03	BOV	1Q	TP7
02	BOV	TP9	0710
01	BOV	1P	TP9
401M	BOV	874	04
301M	BOV	873	03
201M	BOV	872	02
101M	BOV	871	01
12	BOV	1M6	CM4
11	BOV	1M4	RM4
10	BOV	04	1M4
9	BOV	1M3	CM3
8	BOV	1M3	RM3
7	BOV	03	1M3
6	BOV	1M2	CM2
5	BOV	1M2	RM2
4	BOV	02	1M2
3	BOV	1M1	CM1
2	BOV	1M1	RM1
1	BOV	01	1M1
31	BOV	04	1212
21	BOV	03	1214
20	BOV	02	1210
13	BOV	01	1216
39	BOV	SEW4	1212
38	BOV	1212	IM4
37	BOV	SEW3	1214
36	BOV	1214	IM3
35	BOV	SEW2	1210
34	BOV	1210	IM2
33	BOV	SEW1	1216

Carvps.end

Thu Aug 11 09:59:09 1994

3

```

Mode: RM3
Equation:
Y list
E6
Connectors: 8
Y = GAIN (X, P)
X list
P8
Parameters
+1.00000E+02

Mode: CM2
Equation:
Y list
E6
Connectors: 6
Y = 225065 (X, P)
X list
Q6
Parameters
+1.92000E+09

Mode: RM2
Equation:
Y list
E5
Connectors: 5
Y = GAIN (X, P)
X list
P5
Parameters
+1.00000E+02

Mode: CM1
Equation:
Y list
E3
Connectors: 3
Y = 225065 (X, P)
X list
Q3
Parameters
+1.92000E+09

Mode: RM1
Equation:
Y list
E2
Connectors: 2
Y = GAIN (X, P)
X list
P2
Parameters
+1.00000E+02

Mode: SEM4
Equation:
Y list
E39
Connectors: 39
Y = COM (X, P)
X list
P39
Parameters
+4.41000E+02

Mode: IM4
Equation:
Y list
P38
Connectors: 38
Y = ATT (X, P)
X list
P38
Parameters
+4.50000E+01

Mode: SEM3
Equation:
Y list
E37
Connectors: 37
Y = COM (X, P)
X list
P37
Parameters
+4.41000E+02

Mode: IM3
Equation:
Y list
P36
Connectors: 36
Y = ATT (X, P)
X list
P36
Parameters
+4.50000E+01

Mode: SEM2
Equation:
Y list
E35
Connectors: 35
Y = COM (X, P)
X list
P35
Parameters
+4.41000E+02

Mode: IM2
Equation:
Y list
P34
Connectors: 34
Y = ATT (X, P)
X list
P34
Parameters
+4.50000E+01

Mode: SEM1
Equation:
Y list
E33
Connectors: 33
Y = COM (X, P)
X list
P33
Parameters
+4.41000E+02

Mode: IM1
Equation:
Y list
P32
Connectors: 32
Y = ATT (X, P)
X list
P32
Parameters
+4.50000E+01

Mode: CS4
Equation:
Y list
E51
Connectors: 51
Y = GAIN (X, P)
X list
Q51
Parameters
+5.52320E+04

Mode: RS6
Equation:
Y list
E50
Connectors: 50
Y = GAIN (X, P)
X list
P50
Parameters
+5.00000E+03

Mode: CS3
Equation:
Y list
E48
Connectors: 48
Y = GAIN (X, P)
X list
Q48
Parameters
+5.52320E+04

Mode: RS3
Equation:
Y list
E47
Connectors: 47
Y = GAIN (X, P)
X list
P47
Parameters
+5.00000E+03

Mode: CS2
Equation:
Y list
E45
Connectors: 45
Y = GAIN (X, P)
X list
Q45
Parameters
+5.52320E+04

Mode: RS2
Equation:
Y list
E44
Connectors: 44
Y = GAIN (X, P)
X list
P44
Parameters
+5.00000E+03

Mode: CS1
Equation:
Y list
E42
Connectors: 42
Y = GAIN (X, P)
X list
Q42
Parameters
+5.52320E+04

Mode: RS1
Equation:
Y list
E41
Connectors: 41
Y = GAIN (X, P)
X list
P41
Parameters
+5.00000E+03

Mode: TP15Y
Equation:
Y list
E75
Connectors: 75
Y = GAIN (X, P)
X list
P75
Parameters
+1.25000E+00

Equation:
Y list
E74
Connectors: 74
Y = GAIN (X, P)
X list
P74
Parameters
+1.25000E+00

Mode: TP13
Equation:
Y list
E77
Connectors: 78
Y = GAIN (X, P)
X list
P77
Parameters
+7.50000E-01

Equation:
Y list
E76
Connectors: 77
Y = GAIN (X, P)
X list
P76
Parameters
+7.50000E-01

Mode: TP15
Equation:
Y list
E73
Connectors: 73
Y = GAIN (X, P)
X list
P73
Parameters
+7.50000E-01

Equation:
Y list
E72
Connectors: 72
Y = GAIN (X, P)
X list
P72
Parameters
+7.50000E-01

```

Carwps.end Thu Aug 11 09:59:09 1994

Mode: T911Y Connectors: 67.66
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E66 E67 +1.25000E+00
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E67 E68 +1.25000E+00

Mode: T911Y Connectors: 80.79
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E69 E70 +1.25000E+00
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E70 E71 +1.25000E+00

Mode: T911Y Connectors: 71.78
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E71 E72 +1.25000E+00
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E72 E73 +1.25000E+00

Mode: T911 Connectors: 69.68
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E68 E69 +7.50000E-01
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E69 E70 +7.50000E-01

Mode: T917 Connectors: 65.64
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E65 E66 +7.50000E-01
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E66 E67 +7.50000E-01

Mode: IQ Connectors: 59
 Equation: Y = ATT (X, P) 1 1 1
 Y list X list Parameters
 E67 E68 +3.68000E+02

Mode: IP Connectors: 58
 Equation: Y = ATT (X, P) 1 1 1
 Y list X list Parameters
 E68 E69 +8.21000E+02

Mode: IW Connectors: 57
 Equation: Y = ATT (X, P) 1 1 1
 Y list X list Parameters
 E69 E70 +1.36000E+03

Mode: IS Connectors: 56
 Equation: Y = COM (X, P) 1 0 1
 Y list X list Parameters
 E70 E71 +1.33620E+04

END-FILE

```

chvlyv.enp      Wed Jul 6 13:58:53 1994      1
READING
FILE MODEL
chvlyv.enp      13:56:27 07/06/94 EXPORT/wa 1.0-beta

SYSTEM GRAPH DESCRIPTION
MODE TYPE YLOC ACT MACRO
Rz22 MENV 568. -103. T
Rz21 MENV -129. -25. T
Rz21 MENV -161. 129. T
Rz21 MENV -51. -258. T
Ocomp MENV 51. -103. T
SESP MENV 103. -413. T
Rz22 MENV 568. -258. T
OBP22 MENV 413. -413. T
SEAR MENV 103. -103. T
Rz22 MENV 413. -103. T
TF22 MENV 413. 51. T
Rz22 MENV 103. 206. T
Cep22 MENV 258. 361. T
Cet22 MENV 491. 361. T
R22 MENV 568. 206. T
Rz22 MENV 413. 206. T
SEAR MENV -594. -180. T
Rz21 MENV -361. -180. T
Rz21 MENV -361. -25. T
Rz1 MENV -129. 284. T
Cep21 MENV -206. 439. T
Cet21 MENV -516. 439. T
R21 MENV -594. 284. T
Rz21 MENV -361. 284. T

CONNECTOR TYPE FROM TO VERTICES
221Rz MENV 1Rz22 Rz22
210Rz MENV 0Rz21 Rz21
AR21R2 MENV 1Rz22 SEAR
22BP01 MENV 0BP22 1BP22
21SE1 MENV SE21 1X21
210SE MENV 0R21 SE21
OComp MENV OComp OComp
01BP MENV OComp 1Rz22
10BP MENV 1BP21 OComp
221R MENV 1X22 R22
221Cet MENV 1X22 Cet22
221Cep MENV 1X22 Cep22
221I MENV 1X22 I22
22TF1 MENV TF22 1X22
22Rz10 MENV 1Rz22 0BP22
221TF MENV 1BP22 TF22
BP21R2 MENV SE2P 0BP22
211R MENV 1X21 R21
211Cet MENV 1X21 Cet21
211Cep MENV 1X21 Cep21
211I MENV 1X21 I21
2110 MENV 1BP21 0R21
Rz12 MENV SEAR 1BP21

MODE EQUATIONS
Named parameters: 0
Global variables: 0

```

chv1v.enp Wed Jul 6 13:58:53 1994

```

Mode: R22
Equation: Y = QAIN ( X, P )      1 1 1
Y list   X list   Parameters
R221R    P221R    +1.00000E+01

Mode: R22R
Equation: Y = CCM ( X, P )      1 0 1
Y list   X list   Parameters
R221R    P221R    +4.80000E+05

Mode: R21
Equation: Y = QAIN ( X, P )      1 1 1
Y list   X list   Parameters
R211R    P211R    +1.00000E+01

Mode: Cst21
Equation: Y = RESU07 ( X, P )    1 1 1
Y list   X list   Parameters
R211Cst  Q211Cst +1.00000E+04

Mode: Cst21
Equation: Y = LHM ( X, P )      1 1 2
Y list   X list   Parameters
R211Cst  Q211Cst +1.00000E-01
          P211Cst +4.00000E+03

Mode: I21
Equation: Y = AFT ( X, P )      1 1 1
Y list   X list   Parameters
R211I    P211I    +2.00000E-03

```

END-FILE

mainch.enp Tue Aug 2 13:56:42 1994 1

READING

FILE MODEL 11:23:17 08/02/94 EXPORT/VS 1.0-beta

maincom.enp

SYSTEM GRAPH DESCRIPTION

MODE	TYPE	ELOC	YLOC	ACT	MACRO
1r5	MLNV	-598.	-260.	T	
oep1	MCNV	-754.	-416.	T	
oetr	MCNV	-520.	-572.	T	
lrel	MLNV	-754.	-572.	T	
sear1	MCNV	-832.	-694.	T	
lrel	MLNV	-598.	-494.	T	
rmar	MLNV	-364.	-494.	T	
lea	MLNV	-285.	-337.	T	
rmar	MLNV	-364.	-103.	T	
lea	MLNV	-520.	-103.	T	
r5at	MLNV	-598.	-260.	T	
oat	MCNV	-598.	-337.	T	
sear2	MCNV	-832.	-337.	T	
lbp2	MLNV	494.	364.	T	
lbp2	MLNV	337.	364.	T	
rx22	MLNV	182.	598.	T	
lcr2	MLNV	182.	442.	T	
oep2	MCNV	25.	520.	T	
lbp1	MLNV	25.	364.	T	
lbp	MLNV	-130.	442.	T	
rx21	MLNV	-520.	364.	T	
o21	MCNV	-285.	364.	T	
se31	MCNV	-285.	520.	T	
rx1	MLNV	-520.	520.	T	
121	MLNV	-207.	520.	T	
oat21	MCNV	-598.	598.	T	
oep21	MCNV	-51.	598.	T	
1x21	MLNV	-285.	598.	T	
tx22	MLNV	494.	520.	T	
122	MLNV	337.	520.	T	
oat22	MCNV	728.	598.	T	
oep22	MCNV	337.	598.	T	
1x22	MLNV	494.	598.	T	
05	MCNV	572.	51.	T	
0o5	MCNV	572.	130.	T	
obo	MCNV	-832.	207.	T	
obo	MCNV	-598.	207.	T	
rx5	MLNV	-364.	130.	T	
1rx5	MLNV	-520.	130.	T	
1x5	MLNV	572.	207.	T	
oep5	MCNV	728.	130.	T	
rx5	MLNV	728.	285.	T	
15	MLNV	728.	207.	T	
oap5	MCNV	416.	207.	T	
oat5	MCNV	494.	285.	T	
rx2	MLNV	728.	-25.	T	
rx2	MLNV	337.	-494.	T	
rx6	MLNV	240.	-416.	T	
rx6	MLNV	-207.	-494.	T	
16	MLNV	103.	-572.	T	
oat6	MCNV	103.	-572.	T	
oep6	MCNV	103.	-494.	T	
1x6	MLNV	-51.	-572.	T	
seer	MLNV	-51.	-494.	T	
rx6	MLNV	-207.	-416.	T	
0er	MLNV	-51.	-416.	T	
ler	MLNV	-51.	-260.	T	
oer	MLNV	182.	-337.	T	
oer	MLNV	103.	-416.	T	
rk	MLNV	-51.	-25.	T	
lck	MLNV	-207.	51.	T	
lck2	MLNV	-51.	285.	T	
rx	MLNV	-51.	-103.	T	
lrx	MLNV	103.	-103.	T	
cd	MLNV	182.	51.	T	
0	MLNV	25.	51.	T	
rk2	MLNV	103.	285.	T	
oer	MLNV	-832.	51.	T	
oer1	MLNV	-520.	51.	T	
oer	MLNV	-832.	-182.	T	
oer	MLNV	-285.	-182.	T	
sebp	MLNV	-832.	285.	T	
lbp1	MLNV	-598.	285.	T	
oer	MLNV	240.	-260.	T	
oep	MLNV	240.	207.	T	
oat3	MCNV	884.	-416.	T	
oep3	MCNV	884.	-337.	T	
rx3	MLNV	884.	-494.	T	
13	MLNV	728.	-694.	T	
12	MLNV	572.	-694.	T	
rx2	MLNV	416.	-616.	T	
oep2	MCNV	494.	-694.	T	
oat2	MCNV	337.	-337.	T	
1x3	MLNV	728.	-337.	T	
lck2	MLNV	572.	-337.	T	
oep3	MCNV	884.	-260.	T	
ops	MCNV	337.	-260.	T	
1	MLNV	416.	51.	T	
oat	MCNV	728.	-103.	T	
oep	MCNV	728.	51.	T	
o23	MCNV	728.	-260.	T	
0c	MLNV	572.	-260.	T	
1x	MLNV	572.	-25.	T	
tx	MLNV	416.	-25.	T	
lck	MLNV	240.	-25.	T	

CONNECTOR TYPE FROM TO VERTICES

100	MLNV	0bc	1r5
101	MLNV	1r5	0at
102	MLNV	1r5	r5at
oer	MLNV	lrel	rmar
sear	MLNV	lrel	rmar
113	MLNV	lrel	lrel
105	MLNV	lrel	oep2
104	MLNV	lrel	oetr
106	MLNV	sear1	lrel
112	MLNV	lea	rmar
111	MLNV	lea	0at
110	MLNV	0er	lea
109	MLNV	lea	rmar
108	MLNV	lea	0at
107	MLNV	0er1	lea
103	MLNV	0at	sear2
07	MLNV	lbp2	lbp2
86	MLNV	lbp2	lck2
92	MLNV	lbp2	0er1
91	MLNV	lbp2	lbp
90	MLNV	0er	lbp
89	MLNV	ler2	rx22

mainch.enp Tue Aug 2 13:56:42 1994

```

Equation:
Y list
E103
Parameters
+1.00000E+01

Mode: rz22
Connectors: 89
Equation:
Y = ZSU07 ( X, P )
Y list
E104
Parameters
+5.00000E-03

Mode: czp
Connectors: 79
Equation:
Y = QAIN ( X, P )
Y list
E105
Parameters
+1.00000E+10

Mode: rz21
Connectors: 76
Equation:
Y = ZSU07 ( X, P )
Y list
E106
Parameters
+5.00000E-03

Mode: sz21
Connectors: 71 se7
Equation:
Y = QAIN ( X, P )
Y list
E107
Parameters
+1.00000E-04

Mode: rz1
Connectors: 72
Equation:
Y = QAIN ( X, P )
Y list
E108
Parameters
+1.00000E+01

Mode: iz1
Connectors: 73
Equation:
Y = ATT ( X, P )
Y list
E109
Parameters
+2.00000E-03

Mode: sz21
Connectors: 74
Equation:
Y = ZSU07 ( X, P )
Y list
E110
Parameters
+1.00000E+04

Mode: czp21
Connectors: 75
Equation:
Y = LIM ( X, P )
Y list
E111
Parameters
+1.00000E-01
+4.00000E+03

Mode: tz22
Connectors: 86 81
Equation:
Y = QAIN ( X, P )
Y list
E112
Parameters
+1.00000E-04

Equation:
Y = QAIN ( X, P )
Y list
E113
Parameters
+1.00000E-04

Mode: rz2
Connectors: 83
Equation:
Y = QAIN ( X, P )
Y list
E114
Parameters
+1.00000E+01

Mode: iz2
Connectors: 82
Equation:
Y = ATT ( X, P )
Y list
E115
Parameters
+5.00000E-03

Mode: cz22
Connectors: 84
Equation:
Y = ZSU07 ( X, P )
Y list
E116
Parameters
+1.00000E+04

Mode: czp22
Connectors: 85
Equation:
Y = LIM ( X, P )
Y list
E117
Parameters
+1.00000E-01
+4.00000E+03

Mode: czb
Connectors: 61
Equation:
Y = QAIN ( X, P )
Y list
E118
Parameters
+1.00000E+08

Mode: rz5
Connectors: 59
Equation:
Y = ORIPICE ( X, P )
Y list
E119
Parameters
+5.00000E-03
+5.00000E-01

Mode: czp5
Connectors: 62
Equation:
Y = ZSU07 ( X, P )
Y list
E120
Parameters
+1.00000E-06
+1.00000E+04

Mode: rz5
Connectors: 57
Equation:
Y = QAIN ( X, P )
Y list
E121
Parameters
+1.00000E+01

Mode: iz5
Connectors: 56
Equation:
Y = ATT ( X, P )
Y list
E122
Parameters
+2.00000E-03

Mode: czp5
Connectors: 55
Equation:
Y = LIM ( X, P )
Y list
E123
Parameters
+1.00000E-01
+4.00000E+03

Mode: cz25
Connectors: 54
Equation:
Y = ZSU07 ( X, P )
Y list
E124
Parameters
+1.00000E+04

Mode: rz
Connectors: 13a
Equation:
Y = QAIN ( X, P )
Y list
E125
Parameters
+1.00000E+01

Mode: rz2
Connectors: 16
Equation:
Y = ORIPICE ( X, P )
Y list
E126
Parameters
+5.00000E-03
+5.00000E-01

Mode: rz4
Connectors: 46
Equation:
Y = QAIN ( X, P )
Y list
E127
Parameters
+1.00000E+01

Mode: cz22
Connectors: 47

```


mainch.enp Tue Aug 2 13:56:42 1994 4

```

Equation: Y = ATT ( X, P )      1 1 1
Y list   X list      Parameters
P47      P47      +2.00000E-03

Mode: otd4
Equation: Y = E2SU07 ( X, P )    1 1 1
Y list   X list      Parameters
E45      Q45      +1.00000E+04

Mode: csp4
Equation: Y = LIM ( X, P )      1 1 2
Y list   X list      Parameters
E46      Q46      +1.00000E-01
           +4.00000E+03

Mode: sear
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E43      sed      +1.00000E-04

Mode: rtd
Equation: Y = E2SU077 ( X, P )   1 2 1
Y list   X list      Parameters
P42      Q45      +5.00000E-03
           E42

Mode: ocar
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E49      Q49      +1.00000E+10

Mode: rob
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
P13      E13      +1.00000E-08

Mode: rx
Equation: Y = ORIFICH ( X, P )   1 2 2
Y list   X list      Parameters
P33      E13      +5.00000E-03
           Q20      +5.00000E-01

Mode: cd
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E40      Q40      +1.00000E+10

Mode: rob2
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
P16      E16      +1.00000E-08

Mode: car
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E4       Q4       +1.92000E+07

Mode: car
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E2       Q2       +1.92000E+07

Mode: sebp
Equation: Y = E2SU57 ( X, P )    1 1 1
Y list   X list      Parameters

```

```

E1      time      +4.80000E+05

Mode: otd3
Equation: Y = E2SU07 ( X, P )    1 1 1
Y list   X list      Parameters
E29      Q29      +1.00000E+04

Mode: csp3
Equation: Y = LIM ( X, P )      1 1 2
Y list   X list      Parameters
E30      Q30      +1.00000E-01
           +4.00000E+03

Mode: r3
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E31      P31      +1.00000E+01

Mode: i3
Equation: Y = ATT ( X, P )      1 1 1
Y list   X list      Parameters
P32      P32      +2.00000E-03

Mode: i2
Equation: Y = ATT ( X, P )      1 1 1
Y list   X list      Parameters
P22      P22      +2.00000E-03

Mode: r2
Equation: Y = QAIM ( X, P )      1 1 1
Y list   X list      Parameters
E21      P21      +1.00000E+01

Mode: csp2
Equation: Y = LIM ( X, P )      1 1 2
Y list   X list      Parameters
E20      Q20      +1.00000E-01
           +4.00000E+03

Mode: otd2
Equation: Y = E2SU07 ( X, P )    1 1 1
Y list   X list      Parameters
E19      Q19      +1.00000E+04

Mode: cpa3
Equation: Y = E2SU67 ( X, P )    1 1 2
Y list   X list      Parameters
E27      Q27      +1.00000E-06
           +1.00000E+04

Mode: cpa
Equation: Y = E2SU67 ( X, P )    1 1 2
Y list   X list      Parameters
E24      Q24      +1.00000E-06
           +1.00000E+04

Mode: i
Equation: Y = ATT ( X, P )      1 1 1
Y list   X list      Parameters
P12      P12      +4.00000E-03

Mode: cat
Equation: Y = BACFLASH ( X, P )  1 1 2
Y list   X list      Parameters
E11      Q11      +1.00000E-01

```

5

mainch.enp Tue Aug 2 13:56:42 1994

+1.00000E+05

Mode: cap Connectors: 10
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E10 Q10 +4.00000E+04

Mode: tf Connectors: 9 8
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 E9 E8 +1.00000E-03
 Equation: Y = GAIN (X, P) 1 1 1
 Y list X list Parameters
 F9 F8 +1.00000E-03

END-FILE

mode.enp Tue Jul 26 13:54:24 1994 1
 13:53:52 07/26/94 EXPORT/MS 1.0-beta

READING

FILE MODEL
 mode.enp

SYSTEM GRAPH DESCRIPTION

MODE	TYPE	ELOC	YLOC	ACT	MACHO
C3	MCNV	-175.	-12.	T	
C2	MCNV	-175.	62.	T	
C1	MCNV	-175.	137.	T	
I3	MLNV	50.	25.	T	
I2	MLNV	50.	100.	T	
I1	MLNV	50.	175.	T	
IF	MLNV	-62.	-87.	T	
O3	MLNV	-62.	-50.	T	
O2	MLNV	-62.	-12.	T	
O1	MLNV	-62.	25.	T	
I3	MLNV	-62.	62.	T	
I2	MLNV	-62.	100.	T	
O1	MLNV	-62.	137.	T	
I3	MLNV	-62.	175.	T	

CONNECTOR TYPE FROM TO VERTICES

CONNECTOR	TYPE	FROM	TO	VERTICES
13	MCNV	01	131	
12	MCNV	132	01	
11	MCNV	02	132	
10	MCNV	133	02	
9	MCNV	03	133	
8	MCNV	134	03	
7	MCNV	04	134	
6	MCNV	05	134	
5	MCNV	02	C2	
4	MCNV	01	C1	
3	MCNV	133	I3	
2	MCNV	132	I2	
1	MCNV	131	I1	

MODE EQUATIONS

Named parameters: 0
 Global variables: 0
 Number of equations: 7

Mode: C3 Connectors: 6
 Equation: $Y = \text{GAIN} (X, P)$ 1 1 1
 Y list X list Parameters
 E6 Q6 +2.00000E+00

Mode: C2 Connectors: 5
 Equation: $Y = \text{GAIN} (X, P)$ 1 1 1
 Y list X list Parameters
 E5 Q5 +1.00000E+00

Mode: C1 Connectors: 4
 Equation: $Y = \text{GAIN} (X, P)$ 1 1 1
 Y list X list Parameters
 E4 Q4 +1.00000E+00

Mode: I3 Connectors: 3
 Equation: $Y = \text{ATT} (X, P)$ 1 1 1

END-FILE

Y list X list Parameters
 P3 P3 +2.00000E+00

Mode: I2 Connectors: 2
 Equation: $Y = \text{ATT} (X, P)$ 1 1 1
 Y list X list Parameters
 P2 P2 +2.00000E+00

Mode: I1 Connectors: 1
 Equation: $Y = \text{ATT} (X, P)$ 1 1 1
 Y list X list Parameters
 P1 P1 +1.00000E+00

Mode: SF Connectors: 7
 Equation: $Y = \text{CON} (X, P)$ 1 0 1
 Y list X list Parameters
 P7 P7 +0.00000E+00

Carvp3d.enp

Tue Aug 16 10:08:22 1994

2

```

32 DOV 1216 IW1
402M DOV 1212 013
302M DOV 1214 015
202M DOV 1210 011
102M DOV 1216 017
51 DOV 113 C84
50 DOV 113 R84
49 DOV 013 113
48 DOV 115 R83
47 DOV 115 R83
46 DOV 015 115
45 DOV 111 C82
44 DOV 111 R82
43 DOV 011 111
42 DOV 117 C81
41 DOV 117 R81
40 DOV 017 117
404M DOV 013 1213
304M DOV 015 1215
204M DOV 011 1211
104M DOV 017 1217
55 DOV 1213 0213
54 DOV 1215 0215
53 DOV 1211 0211
52 DOV 1217 0217
80 DOV T213Y 1Q
79 DOV 0213 T213Y
78 DOV T213 0213
77 DOV 1P T213
75 DOV T215Y 1Q
74 DOV 0215 T215Y
73 DOV T215 1P
72 DOV 0215 T215
71 DOV T211Y 0211
70 DOV 1Q T211Y
69 DOV T211 0211
68 DOV 1P T211
67 DOV T217Y 0217
66 DOV 1Q T217Y
65 DOV T217 1P
64 DOV 0217 T217
63 DOV 0213 1W
62 DOV 0215 1W
61 DOV 0211 1W
60 DOV 0217 1W
59 DOV 1Q 1W
58 DOV 1P 1W
57 DOV 1W 1W
56 DOV 5R 1W

MODE EQUATIONS
Named_parameters: 0
Global_variables: 0
Number of equations: 56

Mode: IM6 Connectors: 89
Equation: Y = ATT ( X, P )
Y list
#89
Parameters
+7.00000E+01

Mode: SM6 Connectors: 90
Equation: Y = COM ( X, P )
1 0 1

```

```

Y list X list Parameters
#90 +6.87000E+02

Mode: RD Connectors: 87
Equation: Y = QAIM ( X, P )
Y list
#87
Parameters
+1.00000E+03

Mode: CK Connectors: 86
Equation: Y = QAIM ( X, P )
Y list
#86
Parameters
+2.70000E+04

Mode: T77 Connectors: 84 83
Equation: Y = QAIM ( X, P )
Y list
#83
Parameters
+7.60000E-01
Equation: Y = QAIM ( X, P )
Y list
#84
Parameters
+7.60000E-01

Mode: T78 Connectors: 82 81
Equation: Y = QAIM ( X, P )
Y list
#81
Parameters
+5.10000E-01
Equation: Y = QAIM ( X, P )
Y list
#82
Parameters
+5.10000E-01

Mode: S74 Connectors: 60LM
Equation: Y = COM ( X, P )
Y list
#60LM
Parameters
+0.00000E+00

Mode: S73 Connectors: 30LM
Equation: Y = S2SU66 ( X, P )
Y list
#30LM
Parameters
1 1 2
+2.90000E-01
+1.34200E+01

Mode: S72 Connectors: 20LM
Equation: Y = COM ( X, P )
Y list
#20LM
Parameters
1 0 1
+0.00000E+00

Mode: S71 Connectors: 10LM
Equation: Y = S2SU66 ( X, P )
Y list
#10LM
Parameters
1 1 2
+1.00000E-01
+1.34200E+01

Mode: CW4 Connectors: 12
Equation: Y = S2SU65 ( X, P )
Y list
#12
Parameters
1 1 1
+1.92000E+09

Mode: RW4 Connectors: 11
Equation: Y = QAIM ( X, P )
Y list
#11
Parameters
1 1 1
+1.00000E+02

Mode: CW3 Connectors: 9
Equation: Y = S2SU65 ( X, P )
Y list
#9
Parameters
1 1 1
+1.92000E+09

```

Carvp3d.enp Tue Aug 16 10:08:22 1994 3

```

Mode: MW1
Equation: Y = GAIN (X, P)
Y_list
E6
Connectors: 8
X_list
P8
Parameters
+1.00000E+02
1 1 1

Mode: MW2
Equation: Y = SIGU65 (X, P)
Y_list
E6
Connectors: 6
X_list
P6
Parameters
+1.22000E+09
1 1 1

Mode: MW3
Equation: Y = GAIN (X, P)
Y_list
E5
Connectors: 5
X_list
P5
Parameters
+1.00000E+02
1 1 1

Mode: CW1
Equation: Y = SIGU65 (X, P)
Y_list
E3
Connectors: 3
X_list
P3
Parameters
+1.92000E+09
1 1 1

Mode: MW1
Equation: Y = GAIN (X, P)
Y_list
E2
Connectors: 2
X_list
P2
Parameters
+1.00000E+02
1 1 1

Mode: MW4
Equation: Y = COM (X, P)
Y_list
E39
Connectors: 39
X_list
P39
Parameters
+4.41000E+02
1 0 1

Mode: MW4
Equation: Y = ATT (X, P)
Y_list
P38
Connectors: 38
X_list
P38
Parameters
+4.50000E+01
1 1 1

Mode: MW3
Equation: Y = COM (X, P)
Y_list
E37
Connectors: 37
X_list
P37
Parameters
+4.41000E+02
1 0 1

Mode: MW3
Equation: Y = ATT (X, P)
Y_list
P36
Connectors: 36
X_list
P36
Parameters
+4.50000E+01
1 1 1

Mode: MW2
Equation: Y = COM (X, P)
Y_list
E35
Connectors: 35
X_list
P35
Parameters
+4.41000E+02
1 0 1

Mode: MW2
Equation: Y = ATT (X, P)
Y_list
P34
Connectors: 34
X_list
P34
Parameters
+4.50000E+01
1 1 1

Mode: MW1
Equation: Y = COM (X, P)
Y_list
E33
Connectors: 33
X_list
P33
Parameters
+4.41000E+02
1 0 1

Mode: MW1
Equation: Y = ATT (X, P)
Y_list
P32
Connectors: 32
X_list
P32
Parameters
+4.50000E+01
1 1 1

Mode: CW4
Equation: Y = GAIN (X, P)
Y_list
Q51
Connectors: 51
X_list
P51
Parameters
+5.52320E+04
1 1 1

Mode: MW4
Equation: Y = GAIN (X, P)
Y_list
P50
Connectors: 50
X_list
P50
Parameters
+5.00000E+03
1 1 1

Mode: CW3
Equation: Y = GAIN (X, P)
Y_list
Q48
Connectors: 48
X_list
P48
Parameters
+5.52320E+04
1 1 1

Mode: MW3
Equation: Y = GAIN (X, P)
Y_list
P47
Connectors: 47
X_list
P47
Parameters
+5.00000E+03
1 1 1

Mode: CW2
Equation: Y = GAIN (X, P)
Y_list
Q45
Connectors: 45
X_list
P45
Parameters
+5.52320E+04
1 1 1

Mode: MW2
Equation: Y = GAIN (X, P)
Y_list
P44
Connectors: 44
X_list
P44
Parameters
+5.00000E+03
1 1 1

Mode: CW1
Equation: Y = GAIN (X, P)
Y_list
Q42
Connectors: 42
X_list
P42
Parameters
+5.52320E+04
1 1 1

Mode: MW1
Equation: Y = GAIN (X, P)
Y_list
P41
Connectors: 41
X_list
P41
Parameters
+5.00000E+03
1 1 1

Mode: MW15Y
Equation: Y = GAIN (X, P)
Y_list
E75
Connectors: 75
X_list
P75
Parameters
+1.35000E+00
1 1 1

Equation: Y = GAIN (X, P)
Y_list
P74
Connectors: 74
X_list
P74
Parameters
+1.35000E+00
1 1 1

Mode: MW13
Equation: Y = GAIN (X, P)
Y_list
E77
Connectors: 76
X_list
P76
Parameters
+7.50000E-01
1 1 1

Equation: Y = GAIN (X, P)
Y_list
P76
Connectors: 76
X_list
P76
Parameters
+7.50000E-01
1 1 1

Mode: MW15
Equation: Y = GAIN (X, P)
Y_list
E73
Connectors: 73
X_list
P73
Parameters
+7.50000E-01
1 1 1

Equation: Y = GAIN (X, P)
Y_list
P72
Connectors: 72
X_list
P72
Parameters
+7.50000E-01
1 1 1

```

carvp3d.enp Tue Aug 16 10:08:22 1994 4

Mode: TP17Y Connectors: 67 66
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 E66 E67 +1.35000E+00
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 P67 P66 +1.35000E+00

Mode: TP13Y Connectors: 80 79
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 E80 E79 +1.35000E+00
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 P79 P80 +1.35000E+00

Mode: TP11Y Connectors: 71 70
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 E70 E71 +1.35000E+00
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 P71 P70 +1.35000E+00

Mode: TP211 Connectors: 69 68
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 E69 E68 +7.50000E-01
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 P69 P68 +7.50000E-01

Mode: TP217 Connectors: 65 64
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 E65 E64 +7.50000E-01
 Equation: $Y = GAIN (X, P)$ 1 1 1
 Y_list X_list Parameters
 P65 P64 +7.50000E-01

Mode: IQ Connectors: 59
 Equation: $Y = ATT (X, P)$ 1 1 1
 Y_list X_list Parameters
 P59 P58 +3.60000E+02

Mode: IP Connectors: 58
 Equation: $Y = ATT (X, P)$ 1 1 1
 Y_list X_list Parameters
 P58 P57 +6.31000E+02

Mode: IV Connectors: 57
 Equation: $Y = ATT (X, P)$ 1 1 1
 Y_list X_list Parameters
 P57 P56 +1.36000E+03

Mode: ME Connectors: 56
 Equation: $Y = COM (X, P)$ 1 0 1
 Y_list X_list Parameters
 E56 +1.33420E+04

END-FILE

REFERENCES

REFERENCES

- ADAMS - Automatic Dynamic Analysis of Mechanical Systems - User's Manual.** Ann Arbor, Mi: Mechanical Dynamics, Inc., 1989.
- ANSYS - Engineering Analysis System - User's Manual.** Houston, PA: Swanson Analysis Systems, Inc., 1993.
- Arora, J. S. *Introduction to Optimum Design.* MacGraw-Hill, Inc.: New York, 1989, ch. 1.
- Arnold, D. B., and D. A. Duce. *ISO Standards for Computer Graphics - The First Generation.* Butterworth & Co. Ltd.: London, 1990.
- Becker, R. A., and W. S. Cleveland. "Take a Broader View of Scientific Visualization." *Pixel*, July/August 1991, pp. 42-3.
- Beshers, C., and S. Feiner. "Automated Design of Virtual Worlds for Visualizing Multivariate Relations." *IEEE Visualization '92.* Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 283 - 290.
- Breedveld, P. C. "Multibond Graph Elements in Physical Systems Theory." *Journal of the Franklin Institute.* Vol. 319, No. 1/2, January/February, 1985, pp. 1-36.
- Brittain, D. "Getting a View on Large Data Sets," *Mechanical Engineering*, November, 1990, p. 8-9.
- Broenink, J. F. *Computer-Aided Physical Systems Modeling and Simulation: A Bond-Graph Approach.* Ph. D. Thesis. Enschede, Netherlands: University of Twente, 1990.
- Chernoff, H. "Using Faces to Represent Points in K-Dimensional Space Graphically." *Journal of the American Statistical Association.* Vol, 68, 1973, pp. 361- 368.
- Cox, D. J. "The Art of Scientific Visualization." *Academic Computing.* March, 1990, pp. 20ff.
- Ferguson, E. S. *Engineering and the Mind's Eye.* MIT Press: Cambridge, Massachusetts, 1992, ch. 2.
- Fuchs, H., M. Levoy, and S. M. Pizer. "Interactive Visualization of 3D Medical Data" *Computer*, Vol. 22, No. 8, August 1989, pp. 46-51.

- Gadh, R. and F. B. Prinz. "Abstraction of Manufacturing Features from Design." *Issues in Design/Manufacture Integration - 1991* (DE.Vol.39). New York, NY: ASME, 1991.
- Giles, M. and R. Haimes. "Advanced Interactive Visualization for CFD." *Computing Systems in Engineering*. Vol. 1, No. 1, 1990, pp. 51-62.
- Haber, R. B. "Visualization Techniques for Engineering Mechanics." *Computing Systems in Engineering*. Vol. 1, No. 1, 1990, pp. 37-50.
- Haber, R. N. and M. Hershenson. *The Psychology of Visual Perception, 2nd Edition*. New York: Holt, Rinegart and Winston, 1980.
- Haber, R. N. and L. Wilkinson. "Perceptual Components of Computer Displays," *IEEE Computer Graphics and Applications*. May, 1982, pp. 23-35.
- Hays, N. ed. "Visualization 91 Opens Eyes and Minds." *IEEE Computer Graphics and Applications*. Vol. 12, 1992, pp. 15-16.
- IEEE Visualization '92*. Los Alamitos, CA: IEEE Computer Society Press, 1992, pp. 283 - 290.
- International Standard for Information Processing Systems - Computer Graphics - Graphical Kernel System (GKS) function description*, ISO 7942, ISO Central Secretariat: Geneva, Switzerland, 1985.
- International Standard for Information Processing Systems - Computer Graphics - Graphical Kernel System for Three Dimensions (GKS-3D) functional description*, ISO 8805. ISO Central Secretariat: Geneva, Switzerland, 1988.
- International Standard for Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS)*. American National Standard ISO/IEC 9592-1, X3.144, 1988.
- International Standard for Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS), Part 4 Plus Lumiere und Surfaces, PHIGS PLUS*. ISO/IEC 9592-4, 1992(E).
- Karnopp, D. C., D. L. Margolis, and R. C. Rosenberg. *System Dynamics: A Unified Approach*. John Wiley & Sons, Inc.: New York, 1990.
- Kasper, J. E., and D. Arns. *Graphics Programming with PHIGS and PHIGS PLUS*. Addison-Wesley Publishing Company, Inc.: Reading, Massachusetts, 1993.
- MATRIXx User's Guide*. Palo Alto, CA: Integrated Systems, Inc., 1990.
- McCormick, B. H., T. A. DeFanti, and M. D. Brown (eds.). "Visualization in Scientific Computing." *Computer Graphics*. Vol. 21, July, 1987.
- Medland, A. J. *The Computer-Based Design Process*. Springer-Verlag: New York, 1986, ch. 1.

- Mihalisin, T., J. Timlin, and J. Schwegler. "Visualizing Multivariate Functions, Data, and Distributions." *IEEE Computer Graphics and Applications*. Vol. 11, No. 3, May, 1991, pp. 28 - 36.
- Moffat, A. S. "Molecular Dynamics and the Modelers' Art," *MOSAIC*, Volume 22, No. 4, Winter, 1991, pp. 2-11.
- New York Air Brake. *DB-60 Control Valve Instruction Manual MU-21*. New York Air Brake: Watertown, New York, December , 1991.
- Nielsen, G. M. "Visualizing Positron Emission Tomography (PET)." *IEEE Computer Graphics and Applications*, January, 1993.
- Nielsen, G. M., and B. Shriver (eds.). *Visualization in Scientific Computing*. IEEE Computer Society Press: Los Alamitos, CA, 1990.
- Nowacki, H. "Visualization for Engineering Design." *Scientific Visualization of Physical Phenomena*. Tokyo: Springer-Verlag, 1991, pp. 61-83.
- Nye, A. *Xlib Programming Manual, Volume 1*. Sebastapol, CA: O'Reilly & Associates, Inc., 1990.
- Nye, A. (ed.) *Xlib Reference Manual, Volume 2*. Sebastapol, CA: O'Reilly & Associates, Inc., 1990.
- Nye, A. and T. O'Reilly. *X Toolkit Intrinsics Programming Manual, Volume 4*. Sebastapol, CA: O'Reilly & Associates, Inc., 1990.
- O'Reilly, T. (ed.) *X Toolkit Intrinsics Reference Manual, Volume 5*. Sebastapol, CA: O'Reilly & Associates, Inc., 1990.
- Peskin, R. L., S. S. Walther, A. M. Froncioni, and T. I. Boubez. "Interactive Quantitative Visualization." *IBM Journal of Research and Development*. Vol. 35, No. 1/2, January/March 1991, pp. 205-226.
- PSPICE User Manual*. Irvine, CA: MicroSim Corporation, 1989.
- Puttre, M. "Virtual Prototypes Move Alongside Their Physical Counterparts." *Mechanical Engineering*. August, 1992, pp. 59-61.
- Rosenberg, R. C. *The ENPORT-7 Reference Manual*. Rosencode Associates: Lansing, MI, 1993.
- Rosenberg, R. C. and D. C. Karnopp. *Introduction to Physical System Dynamics*. New York, NY: McGraw-Hill, 1983.
- Rosenberg, R. C. and T. Zhou. "Power-Based Model Insight." Presented at the Winter Annual Meeting of ASME, Chicago, IL, Nov. 27 - Dec. 2, 1988.
- Rosenblum and Brown. "Guest Editor's Introduction: Visualization." *IEEE Computer Graphics and Applications*. Vol. 12, No. 4, July, 1992, p. 18.
- Schmalzel, J. L.. "Software Reviews - Interactive Visualization", *IEEE Spectrum*, February, 1992, p. 6.

- Schneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Reading, MA: Addison-Wesley, 1987.
- Senay, H. and E Ignatius. "Compositional Analysis and SYnthesis of Scientific Data Visualization Techniques." *Scientific Visualization of Physical Phenomena*. Tokyo: Springer-Verlag, 1991, pp. 269 - 281.
- Shearing, J. "Put Some Fun into Piping Design." *Chemical Engineering*. Vol. 91, Sept. 1992, pp. 191-2.
- Shoup, T. E., L. S. Fletcher and E. V. Mochel. *Introduction to Engineering Design with Graphics and Design Projects*. Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1981, ch.1.
- Springmeyer, R. M. M. Blattner, and N. L. Max. "A Characterization of the Scientific Data Analysis Process." *IEEE Visualization '93*. Los Alamitos, CA: IEEE Computer Society Press, 1992, pp.
- Standing, L, J. Conezio, and R. N. Haber. "Perception and Memory for Pictures: Single-Trial Learning of 2500 Visual Stimuli." *Psychonomic Science*. Vol. 19, 1970, pp. 73-74.
- Suckling, C. J., et al. *Chemistry Through Models*. Cambridge University Press: Cambridge, MA, 1978.
- Suh, Nam P. *The Principles of Design*. Oxford University Press, Inc.: New York, 1990.
- Thalmann, D. (ed.). *Scientific Visualization and Graphics Simulation*. John Wiley and Sons: New York, 1990.
- Tufte, E. R. *Envisioning Information*. Princeton University Graphics Press: Cheshire, CT, 1990.
- Tufte, E. R. *The Visual Display of Qualitative Information*. Princeton University Graphics Press: Princeton, NJ, 1983.
- Upson, "The Application Visualization System: A Computational Environment for Scientific Visualization." *IEEE Computer Graphics and Applications*. Vol. 9, No. 4, 1989, pp. 30 - 42.
- Wolfram, S. *Mathematica: A System for Doing Mathematics by Computer, 2nd Ed*. Redwood City, CA: Addison-Wesley, 1991.
- Zhou, T. *Improving Design Assessment and Simulation of Large-Scale Dynamic Systems*. Ph. D. Thesis. Michigan State University: East Lansing, MI, 1988.

MICHIGAN STATE UNIV. LIB



312930104605