AN INTERACTIVE PHIGS+ MODEL RENDERING SYSTEM APPLIED TO POSTPROCESSING OF SPATIAL MECHANISMS

by

David E. Montgomery

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Mechanical Engineering

APPROVED:

Dr. A. Myklebust, Chairman

Dr. C. F. Reinholtz

Dr. M. P. Deisenroth

April, 1990

Blacksburg, Virginia

AN INTERACTIVE PHIGS+ MODEL RENDERING SYSTEM APPLIED TO POSTPROCESSING OF SPATIAL MECHANISMS

by

David E. Montgomery

Committee Chairman: Dr. A. Myklebust
Mechanical Engineering

(ABSTRACT)

This thesis presents the concept, development, and use of PriSM (Postprocessing of Spatial Mechanisms), an interactive 3-D graphical postprocessor for spatial mechanism synthesis and design programs. This device-independent system provides visualization, modeling, and animation of spatial mechanisms.

New ideas and methods are described to simplify the interactive specification of scene rendering and color parameters using the international ISO standard for 3-D graphics, PHIGS (Programmer's Hierarchical Interactive Graphic System) and its proposed extensions, PHIGS+. Perception and evaluation of spatial mechanism designs are significantly improved by the use of PHIGS+ functionality to produce animated models that are shaded, lighted, and depth cued.

Examples are presented for the rendering and animation of spatial mechanisms on a Raster Technologies (*Alliant)* GX4000 workstation with a hardware based PHIGS+ graphics subsystem, UNIX, NeWS, and C. In addition to color photographs and grayscale bitmaps of the PriSM implementation, the program structure and source code listing are fully documented.

Acknowledgements

I would like to acknowledge the guidance and inspiration of Dr. A. Myklebust that began even before my pursuit of an M.S. degree. His enthusiasm motivated me to enter the mechanical engineering graduate program at VPI&SU.

I am fortunate to have had the opportunity to work with Mitchel J. Keil. My program on the Postprocessing of Spatial Mechanisms serves to display the results of his doctoral research on *Automatic Generation of Interference Free Geometric Models of Spatial Mechanisms.* Examples of his work are used to document this thesis.

I am thankful for the assistance of Dr. W. R. Winfrey who wrote the programs to format and manipulate the grayscale bitmaps presented in this thesis.

Finally, I am grateful for the patience and support of my loving wife, Karen, during this endeavor.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

An important aspect of the design of closed loop spatial mechanisms, as well as open loop robotic devices, is the ability to quickly and easily visualize proposed configurations. A successful spatial mechanism or robot design environment should include a single point of contact: a workstation that allows interactive graphical input (preprocessing) of mechanisms, synthesis, design, analysis, interference detection and elimination, and postprocessing using animated 3-D models. A significant feature provided by the workstation is rendering of the models. Realistic computer graphics scene generation with shading, lighting, depth cueing, and animation of spatial mechanisms has been possible only recently with available commercial products. Even so, rendering is ordinarily a difficult trial and error process of setting parameters and waiting for a single scene to be generated.

Although spatial mechanisms do not have the flexibility of programmable robots, they are less expensive, more energy efficient, and more reliable. Closed loop mechanisms can operate at higher speeds than open loop robotic devices. However, one drawback of spatial mechanisms is that they cannot be modified

without extensive disassembly, rework, and reassembly. A robot can be reprogrammed to follow a new path and perform new functions. Mechanisms must be correctly synthesized, designed, and analyzed before being manufactured; otherwise, remanufacturing costs will be overwhelming.

A successful computer-aided mechanism design cycle should include realistic modeling and animation of the desired mechanism. The lack of this type of easy visualization during the input and output stages of spatial mechanism design has hindered the application of spatial mechanisms. Computer simulations are important since the building of physical prototypes is not only costly and time consuming, but interrupts the iterative design cycle. Similar visualization of animated robot models is important in verifying the results of robot path planning tasks.

This thesis describes a methodology for visualizing spatial mechanisms based on the international ISO graphics standard for 3-D graphics, PHIGS (Programmer's Hierarchical Interactive Graphics System) and its proposed extensions, PHIGS+. Application of the methodology is illustrated by a program called PriSM (Postprocessing of Spatial Mechanisms) for interactive scene rendering and animation of spatial mechanism models. PriSM is independent of synthesis code for generating spatial mechanisms. It is highly interactive and requires no keyboard input. PriSM has been written with the PHIGS device independent graphics standard for portability. Graphics portability often implies reduced functionality, but this is not the case with PHIGS. In practice, the use of a graphics standard also reduces development costs and speeds application deployment.

Spatial mechanism components modeled by PriSM include revolute, prismatic, cylindric, and spheric internal and external joint pairs, multi-position connecting elements, and pillow block ground elements. Screw and planar joints can be simulated by using cylindric and revolute joints, because PriSM does not enforce kinematic constraints of the joint types.

The parallels between the PHIGS hierarchical structure and spatial mechanism modeling are described from the lowest level PHIGS graphics primitives, through mechanism joint and link construction, to the mechanism itself. The suitability of PHIGS+ for the modeling and simulation of open loop robotic mechanisms is also discussed.

Perception and evaluation of spatial mechanism designs are significantly improved with animated models that are shaded, lighted, and depth cued. New ideas and methods are presented to simplify the interactive specification of scene rendering and color parameters. In the past, cumbersome keyboard data entry specifications and slow image display have prevented graphics systems from being truly interactive.

Examples for rendering and animation of spatial mechanisms are shown on a Raster Technologies (*Alliant)* GX4000 workstation. The GX4000 is a Sun 4/260 workstation with a hardware-based PHIGS+ graphics subsystem, the UNIX operating system, Sun Microsystem's NeWS windowing system, and the C programming language. This thesis includes information on how to use PriSM and a complete source code listing to aid in future extensions to this work.

Chapter 2

Literature Review

Computer aided design of planar and spatial mechanisms has been evolving since the advent of the digital computer. Most of the early work concentrated on computational problems and did not have any graphical input or output. In 1963, Sutherland [1] introduced the Sketchpad system. It was the first interactive CRT-based 2-D graphics diagramming tool. One of Sutherland's examples was drawing and moving a planar four bar linkage. He stated that "...the most interesting application of Sketchpad so far has been drawing and moving linkages." Sutherland's examples were purely graphical models created without any computerized kinematic synthesis or analysis of the mechanisms. Sketchpad was extended by Johnson [2] into a 3-D wireframe version called Sketchpad III, but no examples of spatial mechanisms were documented.

Contributions to the design, analysis, and synthesis of planar and spatial mechanisms have been wide and varied. Initial mechanism design software produced only numeric output, but graphics output was added as computer graphics hardware became available. In 1972, Sheth and Uicker [3] presented

some of the earliest general mechanism design and analysis software called IMP (Integrated Mechanisms Program). IMP could generate stick figure graphics of mechanisms and linkage systems. IMP accommodated spatial and multiple loop mechanisms and performed kinematic, static, and dynamic analysis of arbitrary mechanisms. Even with its graphics output, IMP relied primarily on other software for postprocessing the standard analysis results.

The 1975 synthesis work of Myklebust and Tesar [4] resulted in MECSYN. Work by Sivertsen (1976), as reported by Sivertsen and Myklebust [5], enhanced MECSYN with an interactive computer graphics interface. MECSYN was an algebraic 2-D mechanism synthesis system that used the IMP flexible ring data structure. The display was based on the Tektronix PLOT 10 graphics subroutines. The mechanism synthesis language of MECSYN was modeled after IMP. It performed four and five multiply separated position path, motion, and function generation algebraic synthesis of four, five, and six link planar mechanisms.

Rubel and Kaufman's 1977 KINSYN III system [6] was a device dependent interactive graphics program for planar mechanism design which performed closed form analysis during synthesis. 2-D stick figures were used for animation, but 2-D schematics of actual links could be substituted. Analysis information included the Grashof type, transmission angle, change points, and need for reassembly.

Also in 1977, Erdman and Gustafson [7] developed an interactive mechanism synthesis system called LINCAGES (Linkage INteractive Computer Analysis and Graphically Enhanced Synthesis package) that produced 2-D graphics line drawings. LINCAGES was a computer graphics program for planar mechanism synthesis and analysis for a combination of path, motion, and function generation.

It supported three to five finitely spaced precision point synthesis based on the Burmester Theory and the complex number method. LINCAGES was originally interfaced via teletypes and Tektronix 4010 storage tube displays.

Again in 1977, Paul [8] introduced a dynamic analysis program called DYMAC (DYnamics of MAChinery). It analyzed multi-degree of freedom planar linkages under the influence of forces and moments as a function of velocity and displacement. Motion of any part of the system could be constrained to a specified path (to simulate a bumpy road, for example). DYMAC calculated the displacements, velocities, and accelerations of open and closed looped linkages; however, the only graphics output was plotted computational data.

Chuang, Strong, and Waldron [9] introduced RECSYN (RECtified SYNthesis) in 1981. RECSYN went beyond the interactive linkage synthesis of KINSYN and LINCAGES by adding a solution rectification technique to eliminate spurious and undesirable synthesis solutions. Synthesis rectification increased the efficiency of interactive location of good solutions. RECSYN was originally written for Tektronix 4014 storage tube terminals.

Chace [10] reviewed the dynamic analysis software DRAM (Dynamic Response of Articulated Machinery) and ADAMS (Automated Dynamic Analysis of Mechanical Systems). DRAM, which originated at the University of Michigan as DAMN (Dynamic Analysis of Mechanical Networks), was the first generalized program for analyzing time response of large displacement behavior of multi-freedom, constrained mechanical systems. It was a 2-D system with command language input and graphical output. ADAMS was a 3-D large-displacement dynamic analysis program similar to DRAM that also supplied graphical output. It was

useful for designing and analyzing light and heavy machinery, vehicle design, and accident reconstruction.

The 1983 dissertation of Reinholtz [11] contained an excellent review and history of spatial mechanism science, design, synthesis, analysis and optimization research. His research dealt with the optimization of dyad-based spatial mechanisms. The optimization process begins with a conceptual design that is developed into a system model for analysis. Optimization requires establishing a scalar objective function containing the weighted design variables to measure the system effectiveness during optimization. The selection of the weighted design variables can itself be subjective. In his conclusion, Reinholtz expressed concern about the workspace requirements of spatial mechanisms. He believed that it would be "a much greater problem for spatial mechanism design than in planar mechanism design since the motions are usually more difficult to visualize and to represent on graphic computer displays."

In 1983, Tilove [12] demonstrated the capability of modeling mechanisms in three dimensions by extending the GMSolid system to create link models. The user could manually create monochrome link models and manually simulate their motion. This work was not integrated with any analysis or synthesis code.

Coats and Cipra [13] presented a pre- and postprocessor for IMP in 1983. The spatial mechanism representations were simple wireframes. They used GRAFIC to graphically display the monochrome vector/schematic models.

In 1984, Keil and Myklebust [14,15] completed a postprocessor for planar mechanisms called ANIMEC. ANIMEC automatically generated and animated

3-D wireframe models of planar linkages. ANIMEC used MOVIE.BYU to display planar faceted models with hidden line elimination.

In 1985, Myklebust, Keil, and Reinholtz [16] introduced MECSYN-IMP-ANIMEC as a foundation for a comprehensive mechanism design system. It used MECSYN for synthesis, IMP for analysis, and ANIMEC for automatic geometric modeling, assembly, and animation of planar mechanisms.

An example of mechanism synthesis was presented by Myklebust, Reinholtz, Francis, and Keil [17] for design of a radar guidance mechanism. The mechanism was a planar four-bar linkage with four revolute joints for positioning an in-wing radar on small single-engine aircraft. MECSYN performed the synthesis calculations and working solutions were modeled with ANIMEC.

Early work for collision avoidance within planar mechanisms was pursued by Keil, Myklebust, and Reinholtz [18] in 1985. Their algorithm for predicting link interferences and eliminating them by reshaping the links was a first step toward collision avoidance within spatial mechanisms. The authors considered not only link and joint interference detection but restructuring of the mechanism to avoid interference. The concepts were demonstrated using ANIMEC.

Following its inception, the LINCAGES-4 system [19,20] was enhanced by moving it to an Apollo engineering workstation. High-speed color raster graphics coupled with "point-and-click" mouse interaction, multiple windows, and animation were a few of the benefits. Routines developed to use these new capabilities were collected into functional libraries, maintaining independence from device specific routines. However, a standard graphics package was not used.

In 1986 Pennington [21] completed GENMOD, a collection of subroutines which allowed automatic geometric modeling of spatial links and linkages. These subroutines were an interface to CADAM, and they produced both wireframe and surface models.

Some early progress on modeling spatial mechanisms to avoid link interference was reported by Pennington, Keil, and Myklebust [22] in 1987. This work was an application of the research activities for GENMOD [21]. Keil and Myklebust [23] presented an improved method of interference detection in the following year. This was a precursor to Keil's [24] doctoral research on collision detection and elimination in spatial mechanisms.

In 1987, Thatch [25] and Thatch and Myklebust [26] addressed the need for interactive graphical input to spatial mechanism analysis and synthesis codes with a PHIGS based program called MECHIN. MECHIN used the IBM version of PHIGS (graPHIGS) to manage all of the interactive graphics. Thatch demonstrated the feasibility of developing graphically device independent software for preprocessing of spatial mechanism design. He emphasized techniques in screen layout and the user interface for 3-D data entry and visualization. The graPHIGS library limited the program to 3-D stick-figure joints and links. MECHIN eliminated the need to write multiple preprocessing systems for different mechanism design programs. Although independent of specific analysis and synthesis code, MECHIN supported mechanism analysis using IMP. In his conclusion Thatch stated that "work must also be done to tie MECHIN in with complete postprocessing software, including solid model generation and rendering along with mechanism animation, to complete the mechanism design cycle."

In 1987, Thompson [27] completed a computer aided design and synthesis system for the RSCR spatial mechanism. Analytical capabilities for position, velocity, and acceleration of RSCR designs were demonstrated in his Master's thesis. Synthesis for rigid body guidance was shown and Thompson indicated that similar techniques could be applied to synthesis for function generation and path generation. Thompson's software was integrated with MECHIN for preprocessing and GENMOD for postprocessing

Graphical modeling of open loop robotic devices has received similar attention. Crane and Duffy [28] demonstrated interactive computer graphics for planning and evaluating robot tasks. Their system consisted of off-line previewing of solid, colored models with real-time animation on a super-graphics workstation. Following the off-line preview, the path data was sent to the robot controller and executed. The purpose of off-line robot path simulation is similar to mechanism animation, namely, to simplify the user's task by eliminating the need to enter and interpret numerical data.

Choi, Crane, and Matthew [29] extended the previous user interface by adding a six degree-of-freedom joystick controller. This work included a collision avoidance algorithm that prevented the operator from moving the simulated robot into previously specified obstacles. Path editing features were included to refine the robot motions generated from the joystick input.

In addition to off-line planning, real-time telepresence control was demonstrated by Crane and Duffy [30]. It permitted the operator to control the motion of the remote robot by using a six-degree-of-freedom joystick to orient the end effector and computer graphics to view the results of the unseen robot.

Kahloun, Malowany, and Angeles [31] applied solid modeling to 3-D animation of robots. Due to the computational intensity of solid modeling algorithms, real-time animation could only be achieved via play-back simulation of stored graphics frames. A wireframe display was used to maximize display speed.

At this time, the logical next progression in visualizing spatial mechanism designs is an interactive, realistic surface or solid modeler. A surface modeler with lighting, shading, and depth cueing could generate the desired interactive modeling environment. PriSM was designed to achieve this goal.

Chapter 3

Objectives

This thesis intends to show how parameter specifications for interactive model rendering can be simplified so that generating scenes of complex spatial mechanisms is easy and intuitive. The user should have control over all variables that define a scene, and the modification of any variable should produce immediate results. The following parameter lists were identified as necessary for visualizing design models with lighting, shading, and depth cueing. Foley and van Dam [32] and Rogers [33] indicate that these are common variables for graphics rendering, and they are all included in the proposed PHIGS+ standard [34].

Model Complexity

2 to 16 facets per 90 degree arc

Display Type

Grayscale double buffered *or* true color

Shading Type

Wireframe *or* Flat *or* Smooth (Gouraud)

Multi-view Display

Single view *or* 4 views

Model Orientation in World Coordinates

1. Rotation (XYZ)
2. Position (XYZ)
3. Scale (XYZ)

Model Surface Properties

1. Ambient coefficient
2. Diffuse coefficient
3. Specular coefficient
4. Specular exponent
5. Specular color (RGB)

Model View Orientation

1. View reference point (XYZ)
2. View plane normal (XYZ)
3. View up vector (XYZ)

Model View Mapping

1. View window (zoom factor)
2. Projection reference point (XYZ)
3. Front plane distance
4. View plane distance
5. Back plane distance
6. Projection type (perspective or parallel)

Model Depth Cue Parameters

1. Mode (suppressed *or* allowed)
2. Front reference plane
3. Back reference plane
4. Front scale value
5. Back scale value
6. Depth cue color (RGB)

Model Component Color

1. Red, Green, Blue *or*
2. Hue, Saturation, Value *or*
3. CIE x, CIE y, luminance

Background Color

1. Red, Green, Blue *or*
2. Hue, Saturation, Value *or*
3. CIE x, CIE y, luminance

Light Source Color

1. Red, Green, Blue *or*
2. Hue, Saturation, Value *or*
3. CIE x, CIE y, luminance

Infinite Light Source Orientation

1. Position (XYZ)
2. Direction (XYZ)

Point Light Source Orientation

Position (XYZ)

Spot Light Source Orientation

1. Position (XYZ)
2. Direction (XYZ)
3. Concentration exponent
4. Spread angle

Chapter 4

Computer Graphics Standards

Portability of computational mechanism design software has been achieved for many years with the use of FORTRAN and other standardized languages. However, most of the graphical presentation of mechanism modeling has been implemented with device dependent graphics. This severely limits the portability of the graphics display software. In a climate of ever changing computer graphics hardware, this is a liability to the long-term development of interactive computer graphics mechanism design software. Fortunately, international computer graphics standards began to emerge in the mid-1980's.

Graphics standards provide shorter program development cycles and lower cost of resources as more people use standardized facilities. Device independent graphics allow programmers to concentrate on the application instead of the target hardware. Standards help improve quality while reducing programmer training costs. For graphics hardware manufacturers, graphics standards allow access to large markets. Standards also protect the software investment of end-users.

Yet, the use of graphics standards can restrict innovation. The lengthy approval cycles for standards (commonly due to constant changes in functionality) often causes them to lag behind current hardware technology. The standards may include unneeded features or may not have all the features required for a project. Depending on the implementation, extra overhead in the graphics pipeline can slow down highly interactive applications. One method to circumvent these limitations is to extend the graphics standards to suit the application environment.

Early de facto graphics standards were developed by companies such as California Computer Products, Inc. (CALCOMP) and Tektronics, Inc. (PLOT–10). GINO (Graphical Input and Output) was an early device independent graphics subroutine package developed at the University of Cambridge. Other de facto graphics standards include GDDM (IBM's Graphical Data Display Manager), Postscript (Adobe's dynamic documentation page description language), and ReGIS (Digital Equipment Corporation's graphics instruction set). [35]

More than a decade elapsed after Sutherland's introduction of Sketchpad before the need for international graphics standards gained attention. The first steps were taken at the 1976 International Federation for Information Processing Working Group 5.2 Workshop on Methodology in Computer Graphics, also known as Seillac I. This meeting was the catalyst for the original international graphics standards movement. Underlying concepts developed at Seillac I eventually lead to the ANSI (American National Standards Institute) Core Graphics system and GKS (Graphics Kernel System). Seillac II followed in 1979 with added emphasis on interactive computer graphics. [36]

ACM SIGGRAPH originated the Core Graphics System when the Graphics Standards Planning Committee was created. First published in 1977, Core eventually became an ANSI standard, but it was overshadowed by GKS in the international graphics standards arena.

Process for Computer Graphics Standardization

The process for generating United States and international computer graphics standards takes many years. ANSI is a US coordinating organization that does not develop standards, but provides guidelines and procedures for its committees to attain consensus standards. The ANSI X3 committee is the standards development committee for information processing. Within X3, H3 is the technical committee for all computer graphics standards. [37]

ISO (International Organization for Standardization) is the international governing body for standards. ISO comprises over 1900 technical committees. The international counterpart to ANSI X3 is TC97 (Technical Committee 97). TC97 has a subcommittee, SC21, called the Open Systems Interconnection subcommittee, which in turn has WG2, a working group for computer graphics. Hence ANSI X3H3 corresponds to ISO TC97/SC21/WG2. ANSI is the official US member of ISO and is represented on its council, executive committee, and technical board, as well as on technical committees, subcommittees, and working groups. Figure 1 contains an approximate time line of the graphics standardization process. [37]

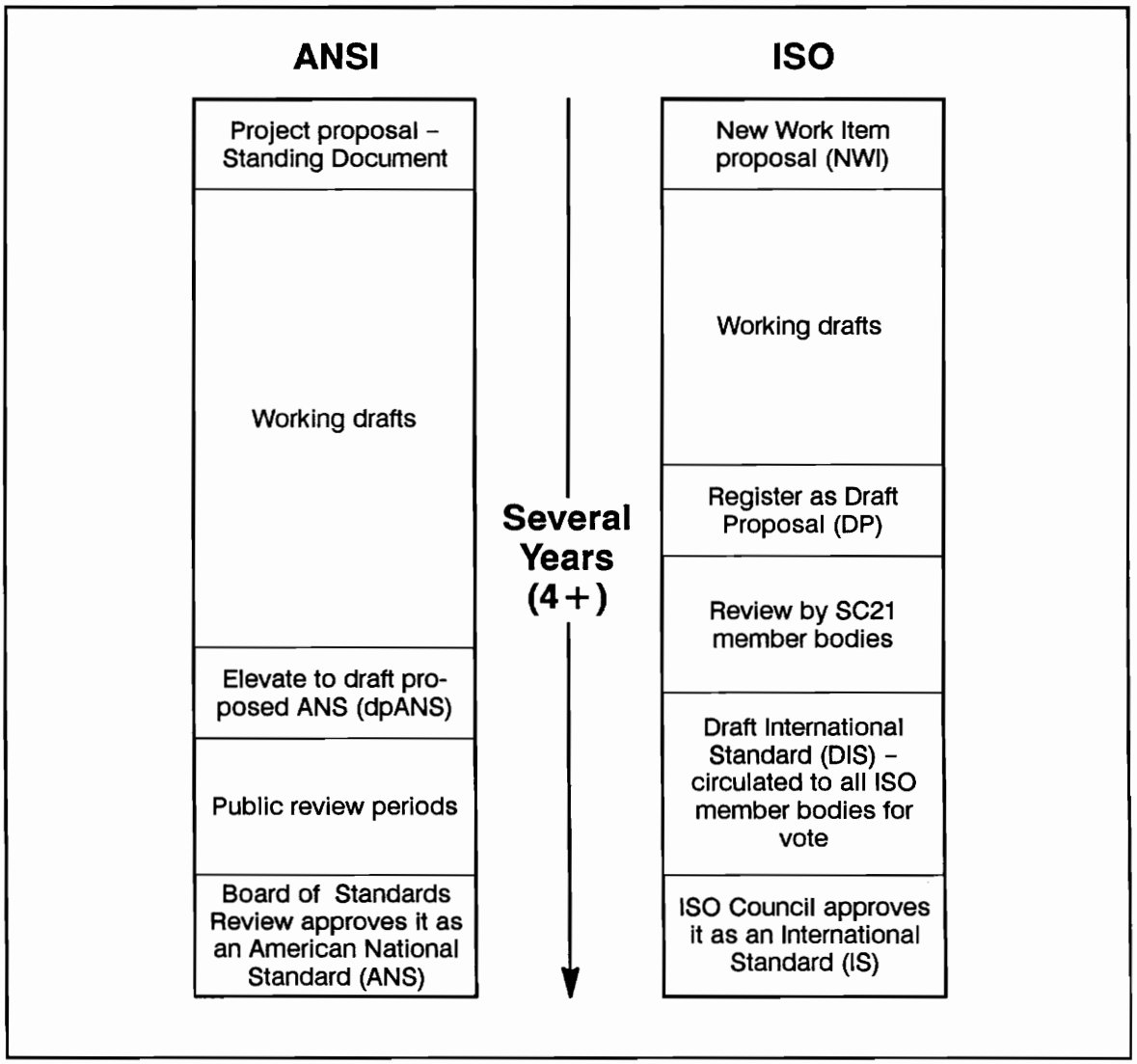

Figure 1. Computer Graphics Standardization Process

Reference Model

The Computer Graphics Reference Model is a conceptual model for unifying the computer graphics standards. An ad hoc group from the ISO TC97/SC21/WG2 working group formed in July 1985 to begin developing a computer graphics reference model. The purpose of a reference model is to

provide a framework for standardization and maintain consistency. A reference model is an abstract description of a particular area (computer graphics) that describes the relationship of standards required for that area (computer graphics standards and other information processing standards). The reference model for computer graphics standards should help accelerate the future creation and use of computer graphics standards [38]. A Special Working Group of Future Planning in the ISO TC97/SC21/WG2 has been working on recommendations for: a unified model for relating computer graphics standards, identifying new areas that need computer graphics standards, relating computer graphics standards with other standards, and evaluating the need to extend or create new computer graphics standards [39]. At present, there are two reference models. The first is an external model that relates computer graphics standards to the outside world and to each other. The second is a model to explain and define characteristics for computer graphics standard architectures [40].

Table 1 is a general comparison of six current or pending international computer graphics standards. There is considerable overlap in the basic features of computer graphics standards, but each standard addresses unique requirements for different graphics applications. Following are brief descriptions of these six graphics standards.

CGM

CGM (Computer Graphics Metafile), formerly VDM (Virtual Device Metafile), is an approved 1986 ANSI standard and 1987 ISO standard. CGM is not an application programming standard, but a specification for system designs and

Table 1. Computer Graphics Standards Comparison

FEATURE	CGM	CGI	GKS	GKS–3D	PHIGS	PHIGS+
Basic Output	x	x	x	x	x	x
Additional Primitives	x	x				
Input			x	x	x	x
Workstation Model		(x)	x	x	x	x
Bit Block Transfers		x				
Segments		x	x	x	(x)	(x)
Hierarchical Database					x	x
Modeling Data Editing					x	x
3-D				x	x	x
HLHSR				x	(x)	x
Lighting						x
Shading						x
Advanced Geometry						x

(x) items are common additions to the standards

implementations. It specifies a file format suitable for capture, storage, transfer, and retrieval of device independent pictorial/image data. A metafile is a graphical database; CGM is a static picture capture metafile. It standardizes the semantics and syntax of a set of elements, not the programs and techniques that generate and interpret the metafiles. [41]

CGM currently specifies only 2-D metafile data. However, 3-D extensions to this standard are in the review process and should result in CGM-3D.

CGI

CGI (Computer Graphics virtual device Interface) is a low level 2-D device independent interface standard specifying functionality on the level of bitmaps and

pixel arrays for raster operations (bit-block transfers). CGI standardization is progressing very slowly. It was formerly the ANSI VDI (Virtual Device Interface) and, prior to that, DI/DD (Device-Independent/Device-Dependent). CGI is intended to support GKS and CGM by defining methods for accepting input and generating, storing, and manipulating graphical data. CGI describes an abstract graphical device that could be a combination of hardware, firmware, and software. It details the interface between device-independent graphics software and device-dependent graphics device drivers. It is expected that most implementations will contain only a subset of CGI functions [42]. The increasing complexity of CGI is slowing its progress, and its role as a device driver is weakened by its potential to be an application interface [43].

GKS

GKS was the first approved international computer graphics standard. This standard for 2-D computer graphics was adopted by ANSI and ISO in 1985. GKS is a programmer level interface that originated in West Germany with DIN, the German Standardization Institute. GKS manages graphics output with segments; all graphics within a segment are on one level. One implication of segmentation is that output primitives and attributes are immediately posted to (displayed on) the screen. This was intended to prevent continual redraws, but this hinders GKS from supporting dynamic graphics. GKS is efficient for "dumb" graphics devices but inefficient for powerful ones. GKS also supports input through logical input devices that map to common physical devices.

GKS defines the application program interface for device independent graphics, but its graphics capabilities are not extensive. Due to these and other restrictions, GKS is commonly enhanced by computer graphics vendors. [36]

The 1985 GKS standard is being revised. The new standard may be approved in 1991 with the name GKS-91.

GKS-3D

GKS-3D is an emerging graphics standard that contains strict 3-D extensions to GKS. It does not support curves or curved surfaces. Some of the major extensions to GKS include: 3-D primitives, 3-D input, HLHSR (Hidden Line/ Hidden Surface Elimination) capability, and 3-D transformations and viewing operations via a single pipeline that integrates 2-D and 3-D functionality. All GKS-3D functions are 3-D; primitives in segments are stored in 3-D even if specified in 2-D. The default values of GKS-3D were chosen so GKS programs can operate in a GKS-3D environment. GKS and GKS-3D functions may be freely mixed; the Z-components of 2-D primitives are simply set to zero. [44]

Where possible, similar parts of GKS-3D and PHIGS are compatible. GKS-3D and PHIGS viewing operations are practically identical; this has created the greatest difficulty for the standards committees in defining GKS-3D. [44]

PHIGS

PHIGS is an interactive 3-D computer graphics standard. PHIGS was originally proposed in 1980 and it became an ANSI standard (X3.144) in June 1988 and an ISO standard (IS 9592) in October 1988. PHIGS provides dynamic,

highly interactive graphics during definition, display, and manipulation of hierarchical graphics structures. PHIGS assumes that the display can be quickly redrawn. The PHIGS logical input devices and modes are virtually the same as GKS.

Williams [45] asserted that there is very little support in the United States for GKS-3D, but there is a significant market appeal in the US for PHIGS. However, one criticism of PHIGS is that since some databases are tightly coupled to their applications, two data structures may have to be produced, one for the application and one for the graphics. Williams also believes that, as machine architectures move closer to the tightly coupled workstation model, the need for the built-in data structures of PHIGS may diminish.

PHIGS+

Currently a set of proposed extensions to PHIGS known as PHIGS+ [34] is being actively pursued on the international level. PHIGS+ extensions include: additional curves, shaded surfaces, lighting, direct color specification, depth cueing, transparency, but not shadows and reflections among primitives.

The PHIGS+ committee began in 1986 as an ad hoc committee striving to extend the ANSI PHIGS draft proposal. This group of industrial and university contributors was interested in utilizing advanced rendering features of the latest engineering workstations. Formal extensions to the PHIGS standard could reach the balloting stage in 1990 and be in the standard by 1991. These extensions will include some of what is known as PHIGS+, since advanced rendering is the primary focus of these new features. [46]

Williams [47] outlined some computer architectures for the real time rendering and manipulation of realistic 3-D graphics images. He discussed the limitations of extending PHIGS+ for some advanced rendering techniques (reflections, translucency, and texturing) and volume modeling. An even newer standard will need to be developed to address the combination of advanced modeling, animation, and rendering for higher level primitives than in PHIGS+. This newer standard might include solid primitives and procedurally defined objects.

PHIGS+ does not address windowing and bitmap graphics. PEX [48] (PHIGS, PHIGS+, and Extensions to X) is a new extension of the X window system that efficiently supports PHIGS/PHIGS+ and other 3-D graphics standards. It allows each window on the display to act as a virtual 3-D graphics workstation. PEX essentially provides all the capabilities of PHIGS with modifications needed to fit PHIGS into the X windows server/client model. This method is basically how RNeWS (the Raster Technologies version of NeWS) worked in conjunction with PHIGS+ on the GX4000 workstation (Appendix A contains a condensed description of the GX4000 architecture). The X windows environment concentrates on windowing, input, and simple graphics in a network-transparent method. X windows version 11 (X11) is a de facto standard that is rapidly progressing through the standardization process.

Chapter 5

PHIGS/PHIGS+ Overview

Judicious use of the PHIGS/PHIGS+ graphics standard was vital to the success of this thesis. Since every major feature of the available PHIGS+ library was utilized in PriSM, understanding the functionality of the PHIGS standard and the PHIGS+ extensions is essential to understanding how PriSM was conceived.

PHIGS addresses many shortcomings of other graphics standards. These shortcomings include the poor use of advanced hardware, slowness, inefficient data storage, and poor data management. PHIGS does not simply provide 3-D graphics for applications. PHIGS can model actual or hypothetical systems. It is a tool kit for graphical model building and manipulation. PHIGS provides a highly interactive environment by supporting 3-D hierarchical graphics data structures. Hierarchical data structures are well suited to animation and representation of models with multiple components.

PHIGS and GKS/GKS-3D both comprise device independent graphics, basic primitives and their attributes, logical input devices, error processing, view management, and the logical workstation concept. PHIGS also contains dynamic

hierarchical structure definition and flexible editing, traversal-time attribute binding, and a class set mechanism for detectability, highlighting, and visibility control. PHIGS concentrates on the graphics model, GKS on the flat graphics image. A PHIGS structure can be modified and updated at any time. After a GKS segment is defined, its contents cannot be changed; only attributes affecting the entire segment can be modified. PHIGS binds primitive attributes at run-time, not when they are created as with GKS. PHIGS offers multiple, cumulative modeling transformations, while GKS only provides limited model transformations.

The following sections in this chapter describe various components of the PHIGS/PHIGS+ graphics subsystems shown in Figure 2. Additional information on PHIGS/PHIGS+ is covered by Plaehn [49], Shuey, Bailey, and Morrissey [50],

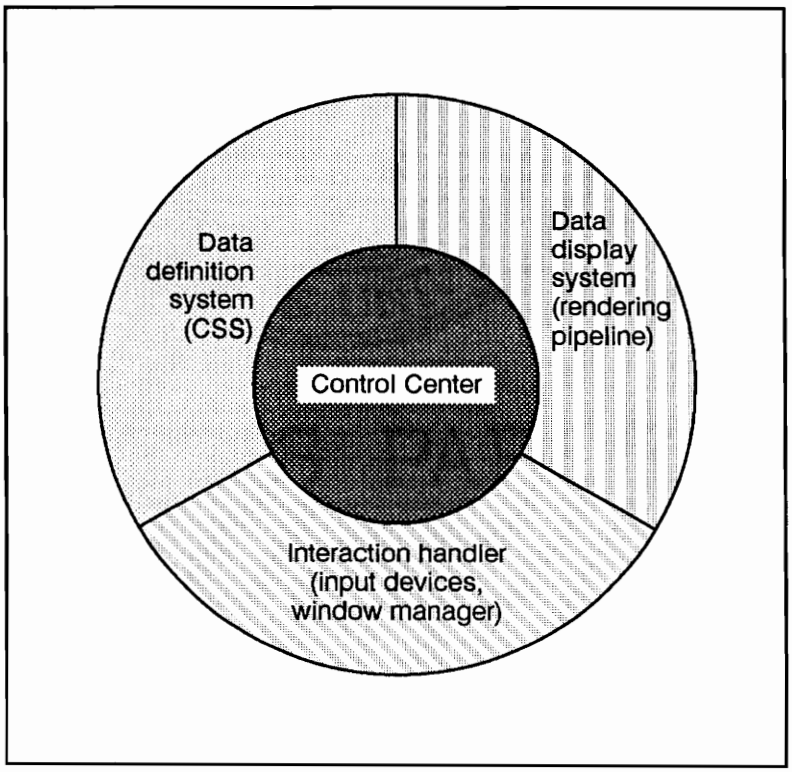

Figure 2. Major Subsystems of PHIGS

and the Visualization Series Programming Guide [51]. Foley and van Dam [32] describe many of the fundamental computer graphics concepts embraced by PHIGS/PHIGS+. They explain hierarchical graphics data structures, a Z-buffer algorithm, shading models for ambient light, diffuse reflection, Phong shading, and Gouraud shading. Foley and van Dam also describe the need and theory for the transformation pipeline in device independent graphics.

Logical Workstations

The PHIGS standard defines the concept of logical (abstract) workstations. Workstations provide device independence while allowing the effective use of device specific capabilities. Each logical workstation can have its own primitive representations, interior, edge, and color representations, workstation window and viewport, etc. The five logical workstations types are input, output, input/output, metafile input, and metafile output. The workstation description table has an entry for each available workstation. Information specified in this table includes a list of available input devices, a table of predefined view indices, and a list of available color models.

Centralized Structure Store

Elements, structures, hierarchical structure networks, and application data are stored in the PHIGS Central Structure Store. Structure elements include output primitives, attributes, viewing data, modeling transformations, labels, name set identifiers, application-specific data, and structure references.

Structure Hierarchy and Editing

The graphics entities of PHIGS are defined, edited, manipulated, and displayed using dynamic hierarchical structure networks. Figure 3 shows a

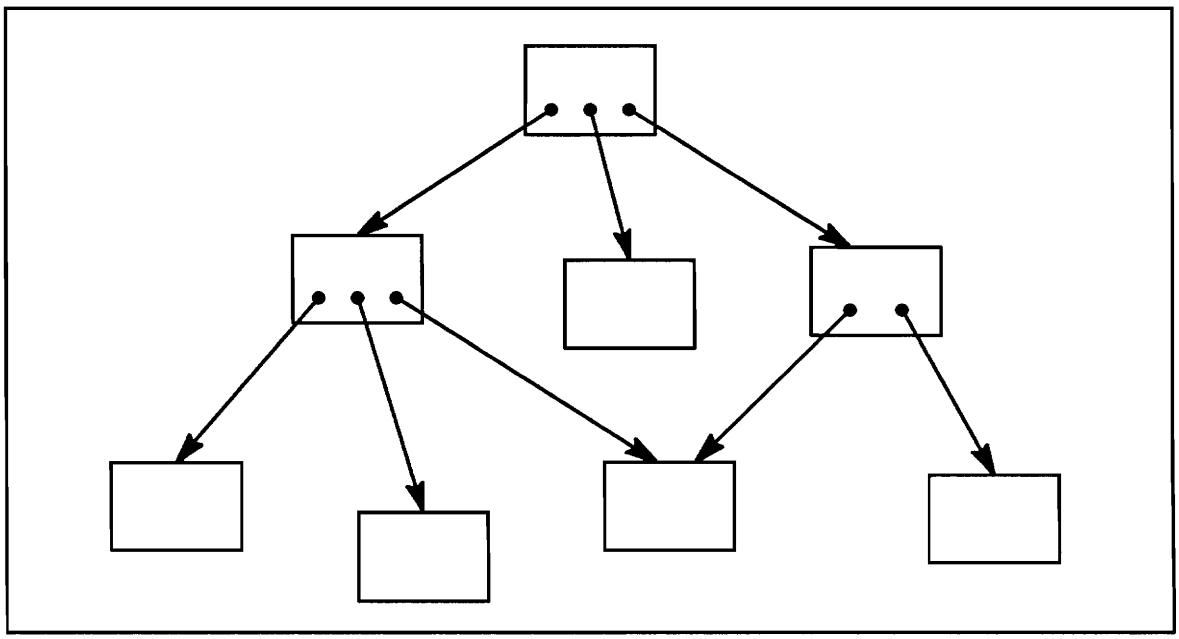

Figure 3. PHIGS Hierarchical Structure Network

conceptual structure hierarchy. Nodes represent individual PHIGS structures. Structures and sub-trees of structure networks can be referenced repeatedly. Elements within individual structures may be added, replaced, copied, and deleted. The structure network hierarchy can be modified by change and delete functions.

Posting structures to workstations associates the structure graphics with a PHIGS workstation. Only posted structures are traversed for display, other structures can be created for later use. Structures are usually displayed by erasing the workstation display and redrawing all posted structures. Figure 4 briefly outlines the PHIGS activity for displaying a structure.

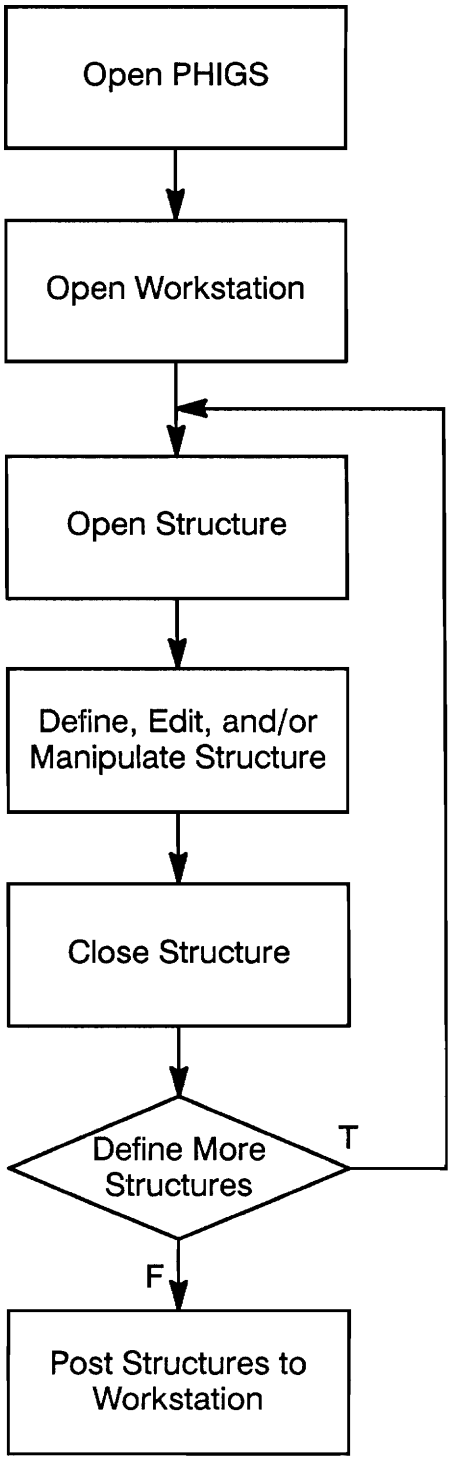

Figure 4. General Structure Definition Cycle

Traversal-time binding of PHIGS attributes is vital to the dynamic interaction between the user and the graphics system. The traversal of structures for display at run time means that the current attributes and view settings determine the display of primitives. The actions of the PHIGS *execute structure* function are:

1. suspend traversal of current structure
2. save current attributes
3. traverse the executed structure
4. restore attributes that were saved earlier
5. resume traversal of parent structure

In other words, child structures inherit attributes from their parents at the time of execution, but the parents are not affected by changes made in the child structure.

Primitives and Attributes

PHIGS primitives include 2-D and 3-D fill areas, polylines, text, cell arrays, and polymarkers. PHIGS+ primitives include "with data" polylines and fill areas, non-uniform B-spline curves and surfaces, parametric polynomial curves and surfaces, polyhedra, quadrilateral meshes, and triangular strips.

Line type, line width, color, and fill patterns are some of the common PHIGS attributes of graphical entities. Text attributes include font type, spacing, scaling, rotation, and alignment. Direct color, surface properties, transparency, and edge color are additional PHIGS+ attributes that may be assigned to many of the PHIGS/PHIGS+ primitives.

Attributes are assigned to primitives either individually or bundled. During structure traversal, the current state of the ASF (Aspect Source Flag) indicates if

the attributes are individual or bundled. The CSF (Color Source Flag) specifies whether indexed or direct color is used.

Logical Input Devices

PHIGS specifies six logical input device classes: choice, locator, pick, string, stroke, and valuator. Each of these devices can operate in request, sample, and event mode. Request mode waits until the user responds to the requested input. Sample mode simply returns the current state of the input device. Event mode is generally more suitable for user-friendly applications, because the user controls the sequence of input. Table 2 shows how the logical input devices map to the physical devices available on the GX4000 workstation.

Table 2. PHIGS Logical Input Devices

Logical Input Device Class	Description of Input Device	GX4000 Physical Input Devices
Choice	Returns a non-negative integer indicating selection from a set	NeWS Text Cycle NeWS Static Menu NeWS Button NeWS Check Box
Locator	Returns position in world coordinates	Mouse
Pick	Returns integer pick identifier of selected graphics object	Mouse
String	Returns a character string	NeWS Text Item
Stroke	Returns a sequence of points in world coordinates	Mouse (not implemented)
Valuator	Returns a real number	Dial Box NeWS Slider Item

State Lists and Operating States

PHIGS has six state lists; they are:

1. the PHIGS system state list
2. the workstation state list
3. the structure state list
4. the archive state list
5. the traversal state list
6. the error state list

The first four state lists contain operating states that are either opened (active) or closed (inactive). For example, the PHIGS system must be open and a workstation must be open in order to post a structure to a workstation. The traversal state list only exists during structure traversal, and the error state list contains current error information.

Name Sets, Filters, and Pick Identifiers

Name sets allow grouping of related output primitives by assigning a common name (integer value). By associating primitives with name sets in different parts of the structure network, and changing the inclusion and exclusion filters on the workstation, the programmer can control which primitives, or groups of primitives, are invisible, highlighted, or eligible for picking.

Inquiry

PHIGS has many functions for inquiring about the capabilities and state of the graphics system. For example, inquire functions can be called to see if a workstation is opened or closed. There are functions that report the extent of PHIGS/PHIGS+ facilities available. Use of inquire functions is necessary to develop truly device-independent graphics applications.

Error Processing

The PHIGS system has few error states. The results of all error conditions are well-defined. This prevents the error handler from slowing the interactive response of PHIGS. Application programs can use either the default PHIGS error handler or a programmer supplied function.

Transformation and Viewing Pipeline

The PHIGS transformation and viewing pipeline is shown in Figure 5. PHIGS contains 2-D shorthand versions of all 3-D transformation and viewing functions. The 2-D functions work identically to the 3-D transformations except that the *z* coordinate is set to zero. The stages in the transformation and viewing pipeline are as follows:

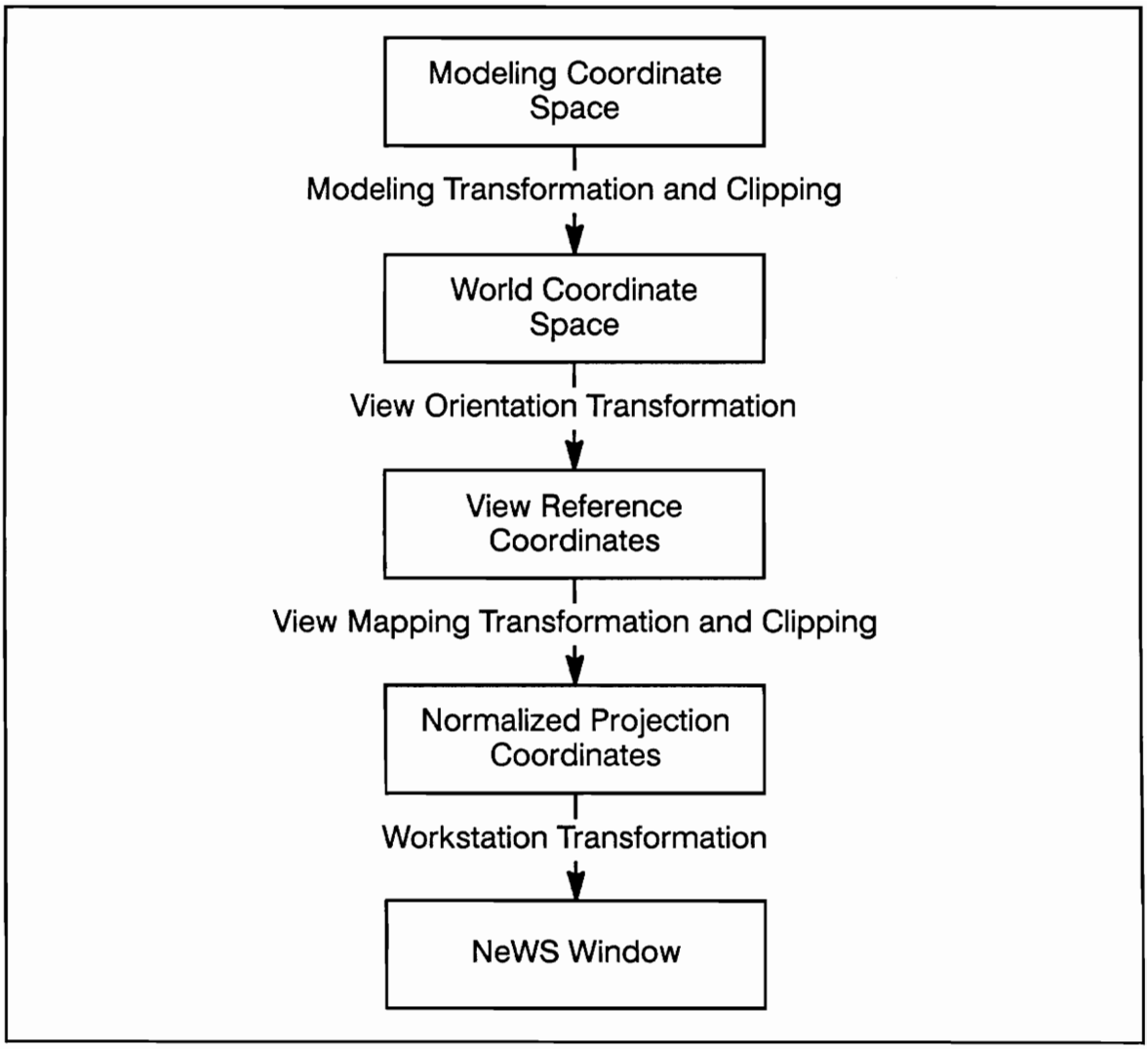

Figure 5. PHIGS Transformation and Viewing Pipeline

- *Modeling transformations* convert the modeling coordinate space of the output primitives to world coordinate space. Utility functions produce and perform computations on the 4 x 4 matrices that describe the modeling transformations. Modeling transformations may be replaced or concatenated to previous transformations. With modeling transformations, a programmer can define objects in their natural modeling coordinates and then position them in a world coordinate system.

- *Viewing orientation transformations* describe the orientation of the view of the scene. World coordinate space is mapped to the viewing coordinate system. Viewing transformations are described by 4 x 4 matrices in a workstation view table.
- *View mapping transformations* map the viewing coordinate system to the NPC (normalized projection coordinate) space. Information required for this stage includes the view plane distance, 2-D window, front and back clipping planes, projection reference point, 3-D viewport, clipping indicators, and projection type (perspective or parallel).
- *Workstation transformations* map the NPC space to the DC (Device Coordinate) space of a workstation. These transformations are accomplished by selecting a volume in NPC space (the workstation window) and mapping it to a volume in DC space (the workstation viewport). This allows selective positioning of an image within a workstation's DC space. This transformation is similar to windowing on a workstation.

Color

PHIGS uses only the RGB (Red/Green/Blue) color model. PHIGS+ also supports HSV (Hue/Saturation/Value), HLS (Hue/Lightness/Saturation), and CIE (Commission Internationale de l'Eclairage; x and y coordinates and luminance) color specifications. Figure 6 shows the four different color models. Color can be assigned to all graphics primitives, light sources, background, depth cueing, and specular highlighting using any of the four color models. PHIGS+ allows both direct color specification and the use of PHIGS color index tables.

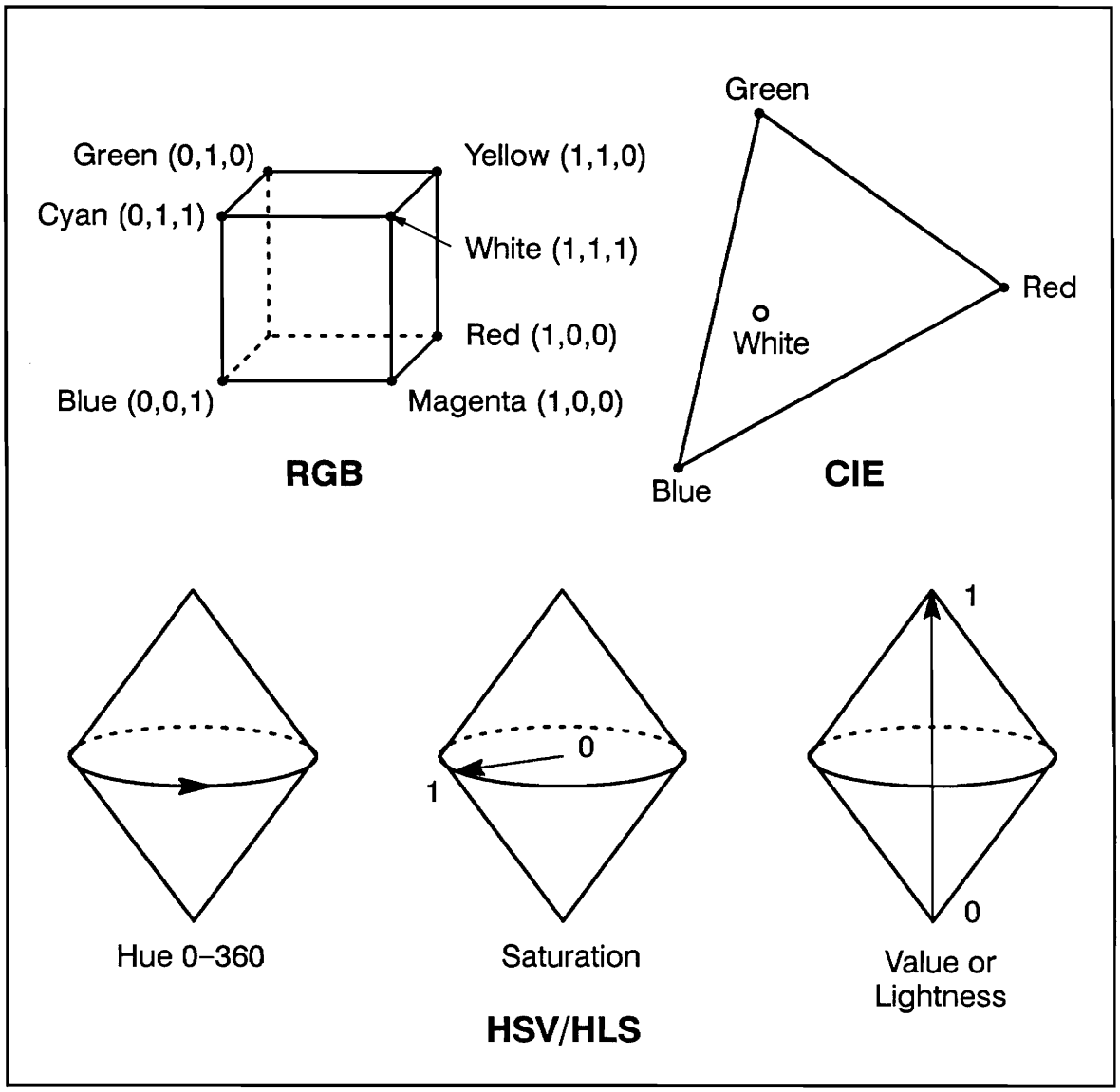

Figure 6. PHIGS+ Color Models

Lighting

PHIGS+ specifies eight light sources for lighting filled areas and surfaces. Light source characteristics include color, location, direction, color exponent, and cone of influence. The four different lighting types (in addition to no light) are:

- *Ambient* light is the overall light in space (background light).
- *Infinite (directional)* light has a constant direction (e.g. sunlight).
- *Point* light emanates from a single point in space (e.g. light bulb).
- *Spot* light is restricted to a semi-infinite cone along a ray emanating from the light source (e.g. stage light).

The lighting mode of a structure can be set to none, ambient, diffuse/ambient, or specular/diffuse/ambient. The surface properties of primitives that affect lighting include the ambient coefficient, the diffuse coefficient, the transparency coefficient, and the specular color, coefficient, and exponent.

Shading

Shading is also a PHIGS+ extension to the rendering pipeline. Fill areas and advanced surfaces may be shaded. Gouraud and Phong shading require that vertex normal data is added to the definition of primitives. PHIGS+ specifies four types of shading (in addition to no shading):

- *Flat (constant) shading:* each polygonal surface patch is drawn with a single calculated color.
- *Color (Gouraud, bilinear interpolation, or smooth) shading:* lit color values are calculated at each vertex of the primitive and then interpolated across the primitive to shade a surface with smoothly varying color.
- *Dot product (Partial Phong) shading:* the dot product of the surface normals and light source directions at each vertex are calculated, the results are interpolated across the primitive, and finally the lighting calculation is performed to determine a color at each pixel.

- *Normal (Full Phong) shading:* both the surface normals and the specified colors at each vertex are interpolated across the primitive and then the lighting calculation is performed to determine a color at each pixel.

Depth Cueing

Depth cueing is a PHIGS+ extension for fading color at each point on a primitive based on its distance from the observer. Front and back depth cue reference planes are specified in NPC space. Depth cue scale factors indicate how much depth cue color (generally the background color) is combined with the primitive color at these planes. Attenuation is accomplished by linearly interpolating the colors between the front and back depth cue planes. Figure 7 shows the recommended implementation of linear interpolation for depth cueing.

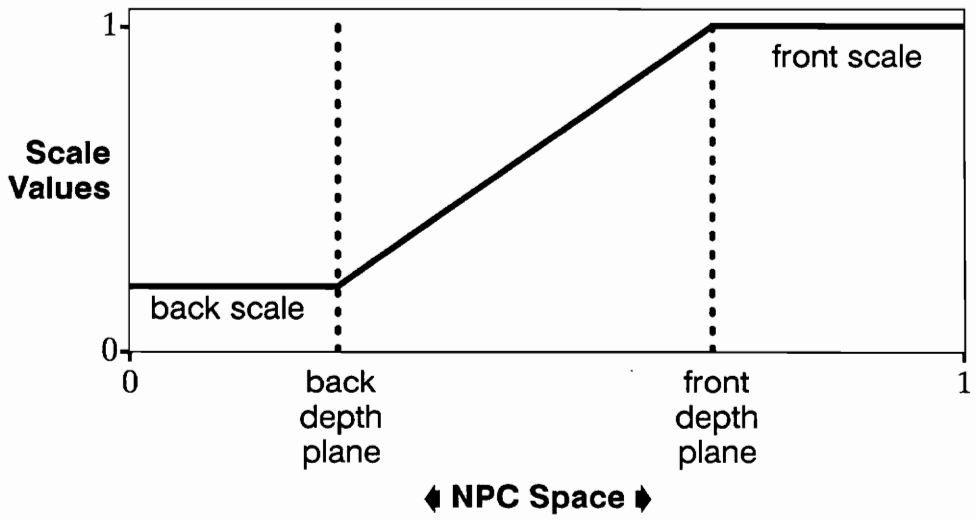

Figure 7. Linear Interpolation of Depth Cueing

Chapter 6

PriSM Conceptual Design

PriSM is a device independent graphics program providing an interactive scene rendering environment specifically for animation of spatial mechanism models. Figure 8 shows the PriSM display with three PHIGS workstations. The PHIGS logical workstations are implemented as windows with SUN NeWS. These windows can be moved and resized using a mouse. The functionality of the NeWS windows is entirely independent of the windows within the PHIGS viewing system.

Screen layout of graphics and text data is a discipline in and of itself. Galitz [52] offered constructive guidelines on screen layout for interactive graphics systems. Galitz favored menu oriented input systems rather than command oriented ones. A logically constructed menu system with related information grouped together will reduce the amount of information users must remember. This philosophy influenced the arrangement of choice items in the PriSM menus as well as valuator input in the dial groups. Studies have also shown that menu search speed and selection accuracy improve with greater menu breadth (less menu depth) [53]. For this reason, all menu choices are available in PriSM at all times.

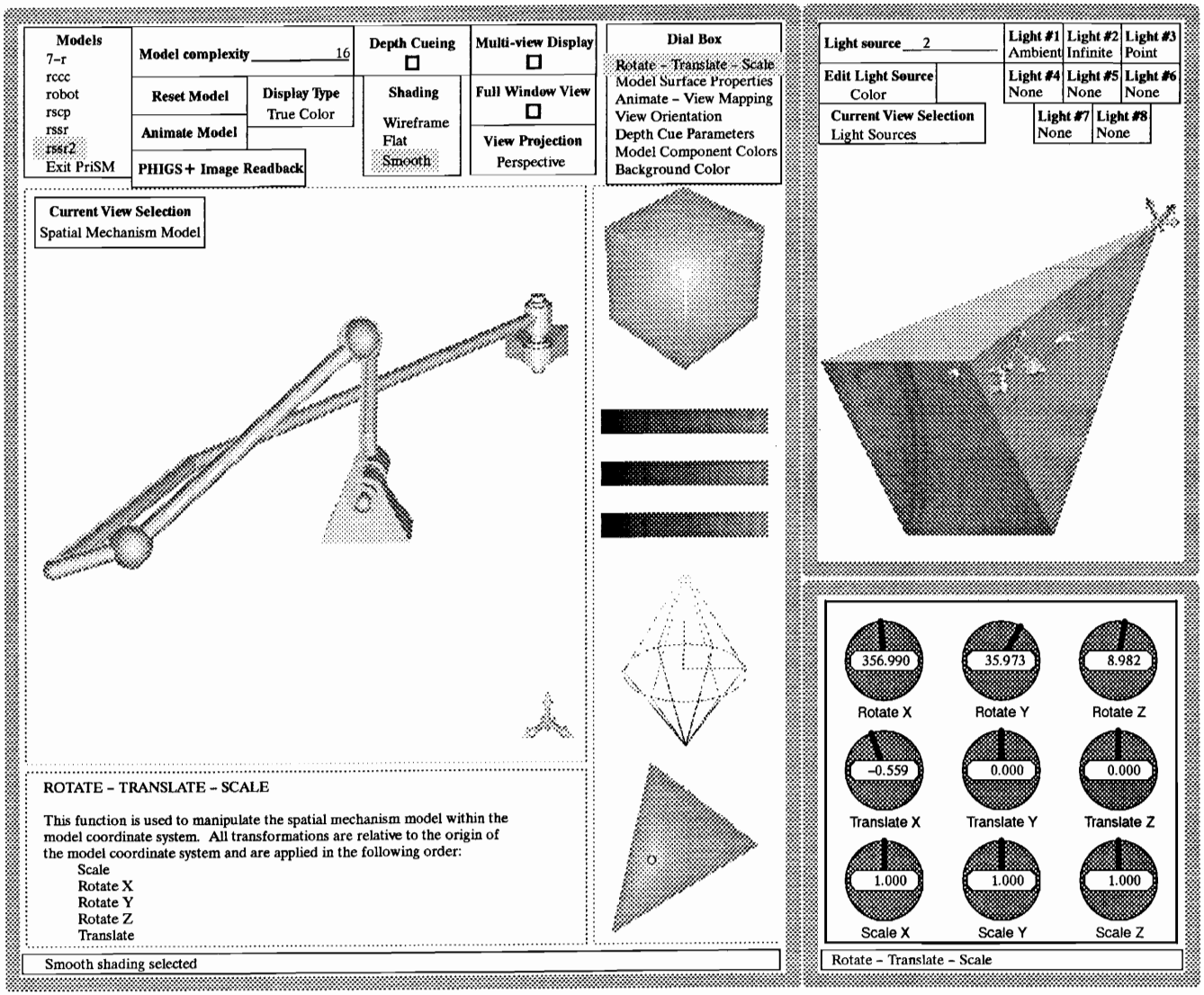

Figure 8. PriSM Workstation Windows Layout

Galitz also reported that eyeball fixation studies indicate that the human eye usually moves to the upper left of a display and then proceeds clockwise around the display. This visual scanning pattern is also influenced by the symmetrical balance and highlighting of the displayed graphics and text. This observation affected the screen layout of PriSM. When scanning from the upper left of the display clockwise, a user will first see the main window, followed by the auxiliary window, the dial box window, and finally the help text window. This is the intended order of visual priority.

UIMS (User Interface Management Systems) have developed into a discipline of their own. Typically a UIMS consists of dialogue software that controls the user interactions separately from the application software. This is not the case with PriSM, but 3-D graphics (such as PHIGS), and a window manager (such as NeWS) are considered to be part of a UIMS. One should refer to Ero and van Liere [54] for additional insight on the role of a UIMS.

This thesis did not concentrate on the theories and methods of a UIMS, but many of the ergonomic guidelines that direct the design of a UIMS were applied. The entire input system is event mode driven. The user has complete freedom of task order. NeWS "point and click" menus serve as PHIGS logical choice input devices. Help information relevant to the latest menu selection can be toggled on and off with the rightmost (third) mouse button. The help text for all PriSM functions is listed in Appendix B. User feedback is continuous; whenever the dials are rotated (PHIGS logical valuators), the current data values are updated in the simulated dial box window. The mouse can also be used to "rotate" the simulated dials in the dial box window. The PriSM user interface is entirely independent of

keying numeric data. All input to PriSM can be performed through the mouse, but the dial box provides faster interaction. This feature contributes to easy parameter specification with immediately displayed results. Also, errors are handled gracefully by displaying error messages with the PHIGS message function.

Whenever a user desires to modify a color within PriSM, a color system with an RGB color cube and slider bars, an HSV double color cone, and a CIE chromaticity diagram is presented with the current color as shown in Figure 9.

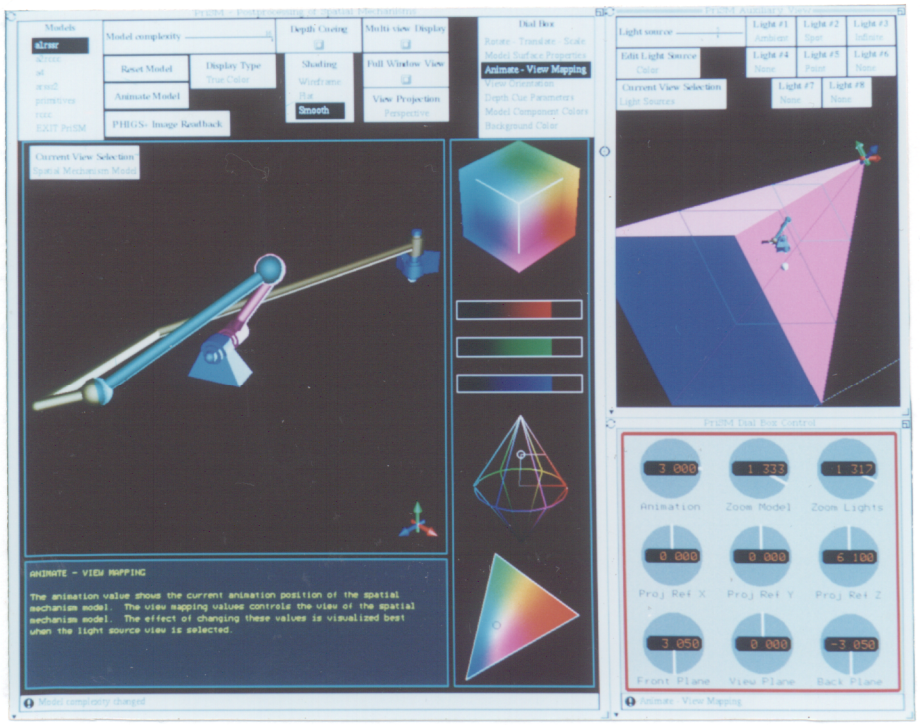

Figure 9. PriSM Windows with Color Models

The chromaticity coordinates for the GX4000 monitor shown in Table 4 of Appendix A were used in the PriSM CIE triangle. Elements whose colors can be modified include the background, each individual mechanism link, and the light sources. Whenever any one of the variables in the three color systems are modified, all other color system values are updated to match the current color

specification. Many effects of modifying the colors are not intuitive, but this system helps to instill an understanding of the relationships among the color models. In addition to changing the color parameters with the dial box or rotating the simulated dials with the mouse, two of the color models can be modified by using the mouse as a logical PHIGS locator device. The mouse cursor can be pointed at a location in the RGB slider bars or the CIE color triangle, and then the position is indicated by "clicking" the middle mouse button. Rogers [33] documented details on the mathematical conversions to and from the RGB and HSV color models and the RGB and CIE color models.

Berk, Brownston, and Kaufman [55] developed an English-based color notation system that proved experimentally easier to learn and more accurate than quantitative systems such as RGB and HSV. Their new CNS (Color Naming System) is similar to the HSV system; it includes terms for lightness (value), saturation, and hue. CNS is also very similar to the ISCC–NBS (International Society Color Council – National Bureau of Standards) lexicon of 267 distinct English names for regions of the color space. Although an English language color notation system would be helpful in specifying colors, the interactive method of indicating colors using PriSM is a definite improvement for quantitative systems. One should refer to *Color and the Computer* [56] for a thorough review of the science of color, human perception, CRT and printing technology, history of the color models, ergonomics, and applications.

The main and auxiliary workstation windows can display the mechanism model, a global view of the model with its viewing volume and the light sources, or any of the four color models. The main workstation window can also show four

views of the mechanism model (top, front, right side, and modified isometric view). Figure 10 shows four views of an RSCP spatial mechanism. The main window can be configured with or without the color models displayed down the right side.

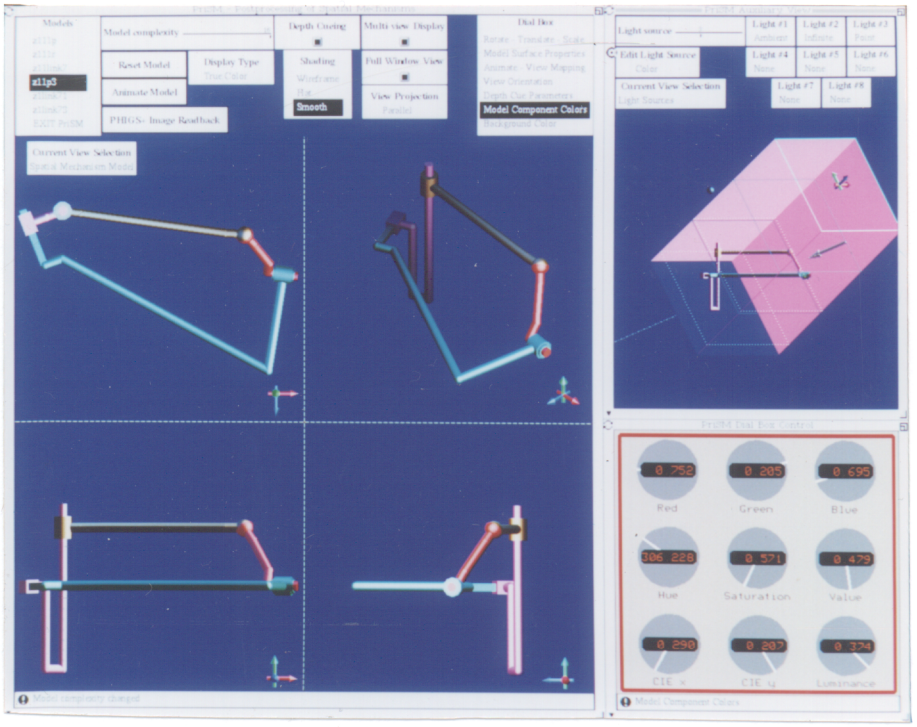

Figure 10. Four Views of an RSCP Mechanism

All parameters of the viewing volume (perspective projection in Figure 9) can be interactively modified by the user. Figure 11 is an example of a seven revolute (7-R) mechanism with the main window set to display only the mechanism model and the auxiliary window showing the parallel projection viewing volume. With this configuration, the user can easily see the effects of modifying viewing parameters on both the model display and the shape of the viewing volume. Since the viewing volume would normally enclose the model, only the outer faces that point away from the user are displayed. The solid lines parallel to the blue back

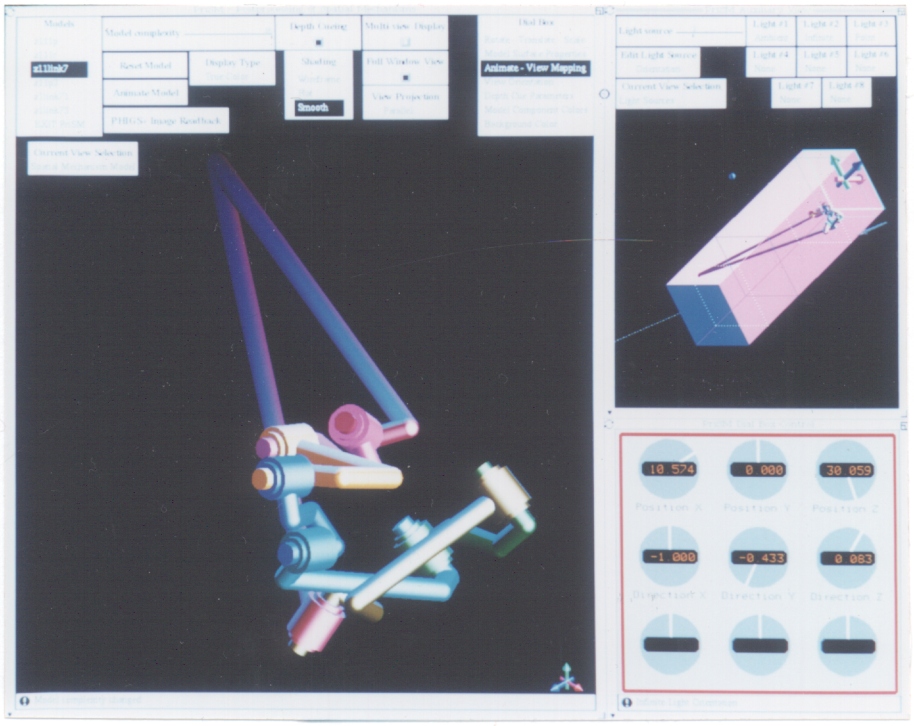

Figure 11. 7-R Mechanism with Parallel Projection

plane in the viewing volume represent the front clipping plane and the viewing plane. The dotted lines represent the front and back depth cueing planes.

Meads [57] expressed concern about the *set view index* structure element that is inserted into a structure to establish a view. He asserted that the viewing information should be entirely separate from the structure. This could be considered a deficiency with PHIGS. The PHIGS standard goes to great lengths to separate the graphics data structure from the viewing pipeline, yet a view index must be inserted into a PHIGS structure. PriSM creates some structures to simply set a view index and then execute other structures that contain all of the graphics. For example, the main PriSM structure (highest node) sets a view index for the mechanism model and then executes the structure network containing the elements that represent the mechanism.

The four PHIGS+ light source types are modeled in PriSM. Up to eight active light sources are displayed with the mechanism viewing volume. Ambient lights can be activated but not graphically modeled. Infinite lights are modeled with arrows. Point lights are depicted with spheres. Spot light sources are modeled with cones. The color and orientation of each activated light source can be interactively modified (ambient light has no orientation). The full color editing features described earlier are available. The corresponding model for each light source is consistently rendered and oriented according to its current values. Figure 9 shows a white point light source producing a back lighting effect. Figure 12 shows a dark gray infinite light source and a red spot light shining on an RCCC mechanism.

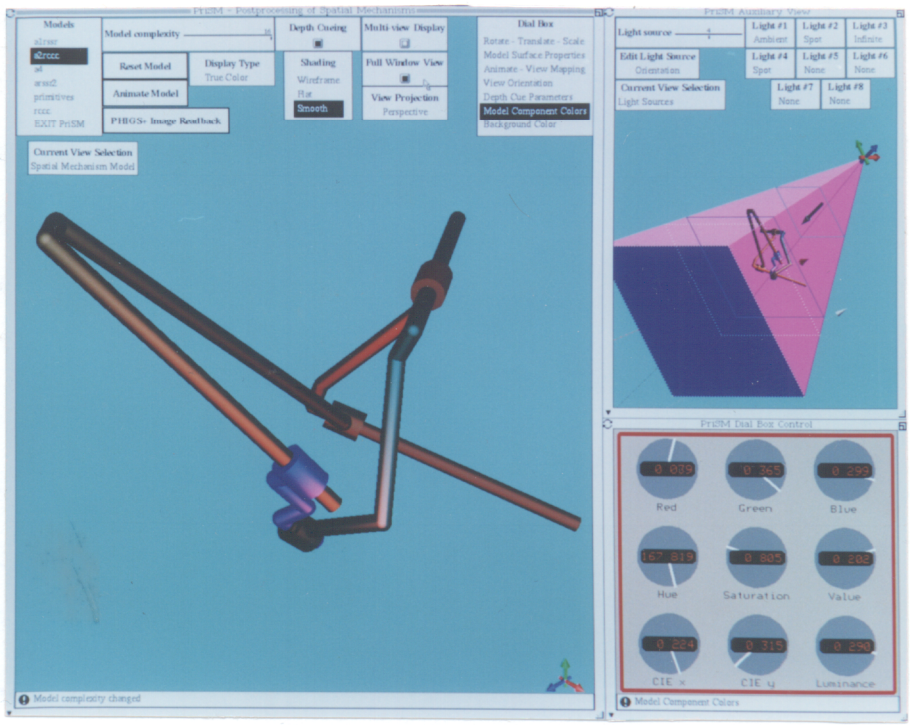

Figure 12. RCCC Mechanism with Red Spot Light

The natural hierarchical structures of PHIGS are well suited for modeling and animating spatial mechanisms. Cylinders, prisms, and spheres are used to model internal and external joints. Two or more joints are then coupled with connecting element(s) to form a mechanism link. A set of mechanism links are oriented to form a mechanism at a single position. The same mechanism links can then be reoriented to new positions. A sequence of consecutive positions displayed quickly on the screen then becomes a real-time animation sequence.

Following Table 3 and Figure 13 from the bottom up, PriSM begins with the construction of six unit primitives: cylinder, cone, prism, half sphere, circle, and frustum. These unit primitives, as shown in Figure 14, are constructed with either polylines for wireframe modeling or polygonal fill areas for flat or smooth shaded surface modeling. These six primitives are used to construct the spatial mechanism models, the light source models, and the XYZ axes sets. Sample vertex normals for lighting and shading calculations are shown on the cylinder, cone, and half sphere. The vertex normals are perpendicular to the surfaces being modeled; they are not perpendicular to the approximating fill areas.

In the next level, a set of routines is used internally by PriSM to scale these unit primitives to form different copies of five distinct objects: cylinder, cone, prism, half sphere, and frustum. The cylinder, cone, and half sphere objects can optionally have a circle as an endcap.

The objects are used to model internal and external joints, connecting elements, pillow blocks, and the arrows of the XYZ axes set (Figure 15). This level of mechanism generation is driven by the input model file. Mechanisms are modeled by specifying size and orientation (rotation and position) of these objects.

Table 3. Generating a Mechanism with PHIGS Graphics Structures

Spatial Mechanism Component Levels	PHIGS Structure IDs	Essential PHIGS Functions in the Structure
Mechanism at the current animation step	STRUCT_MECHANISM	set global transformation execute structure of an animation step (STRUCT_MECHANISM + step_id[n])
Mechanism at each step	STRUCT_MECHANISM + step_id[n] (from position file)	set local transformation execute structure of link 1 (STRUCT_ELEMENTS + link_id[1]) . . execute structure of link n (STRUCT_ELEMENTS + link_id[n])
Mechanism links	STRUCT_ELEMENTS + link_id[n] (from model file)	execute structure(s) of joint(s) (STRUCT_ELEMENTS + joint_id[x]) execute structure(s) of connecting element(s) (STRUCT_ELEMENTS + joint_id[x])
Internal and external joints, connecting element and pillow block	STRUCT_ELEMENTS + joint_id[x] (from model file)	set local transformation execute structure of outer cylinder (example: id_cyl_out) execute structure of inner cylinder (example: id_cyl_in)
Objects (cylinder, cone, prism, half sphere, frustum)	example: id_cyl_out (these IDs are generated internally)	set local transformation execute STRUCT_UNIT_CYLINDER set local transformation execute STRUCT_UNIT_CIRCLE set local transformation execute STRUCT_UNIT_CIRCLE
Unit Primitives (cylinder, cone, circle, half sphere, prism, frustum)	example: STRUCT_UNIT_CYLINDER	polyline 3 *or* fill area 3 with data PHIGS + graphics primitives

The model file also specifies how groups of the joints and connecting elements are related as links. All PHIGS structures in a link are assigned the same pick identifier. When modifying model component colors, a link may be selected by pointing with the mouse cursor and "clicking" the left mouse button. This triggers a PHIGS event mode pick which returns an identifier for the chosen link.

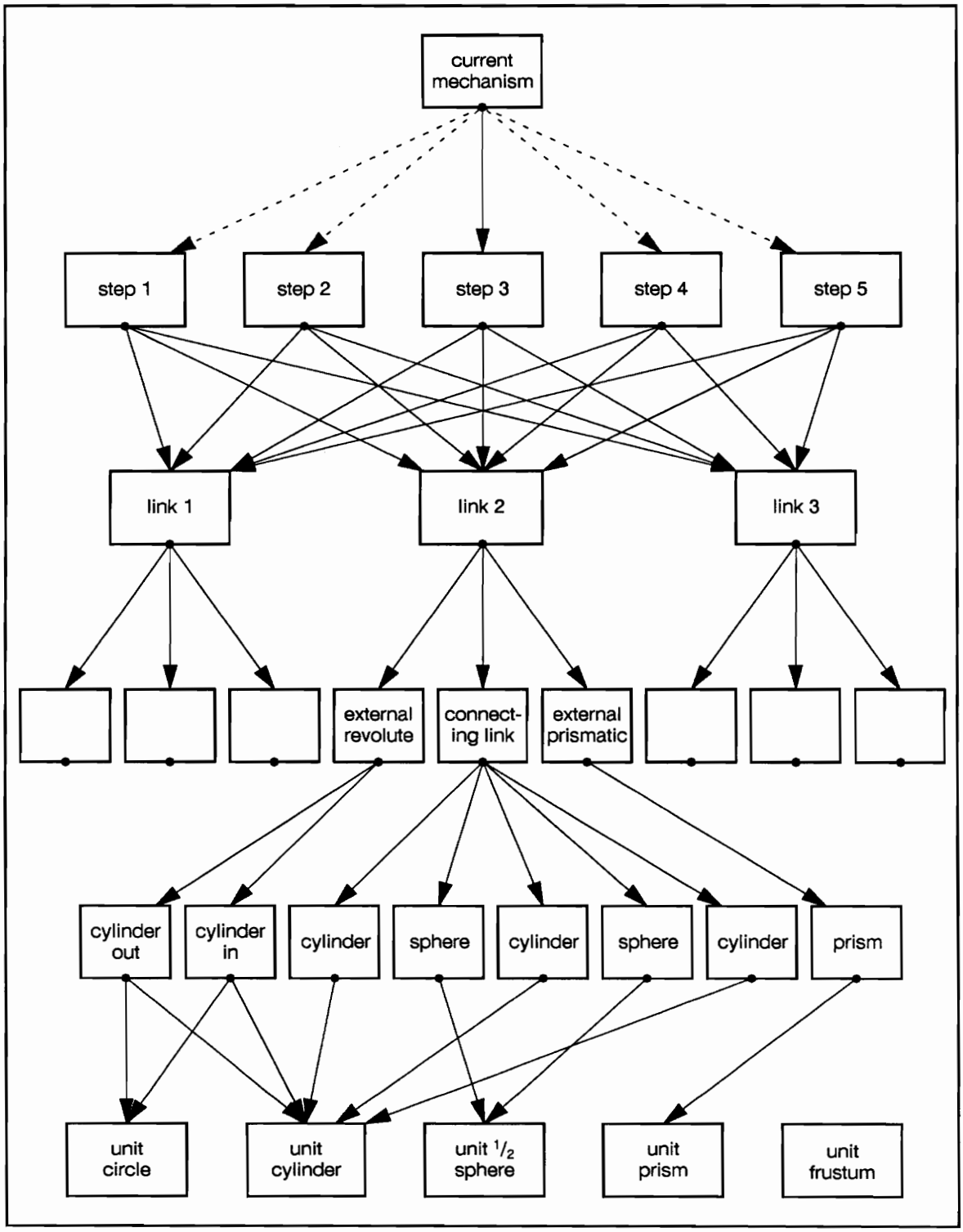

Figure 13. Example Hierarchical Structure Network for Part of a Mechanism

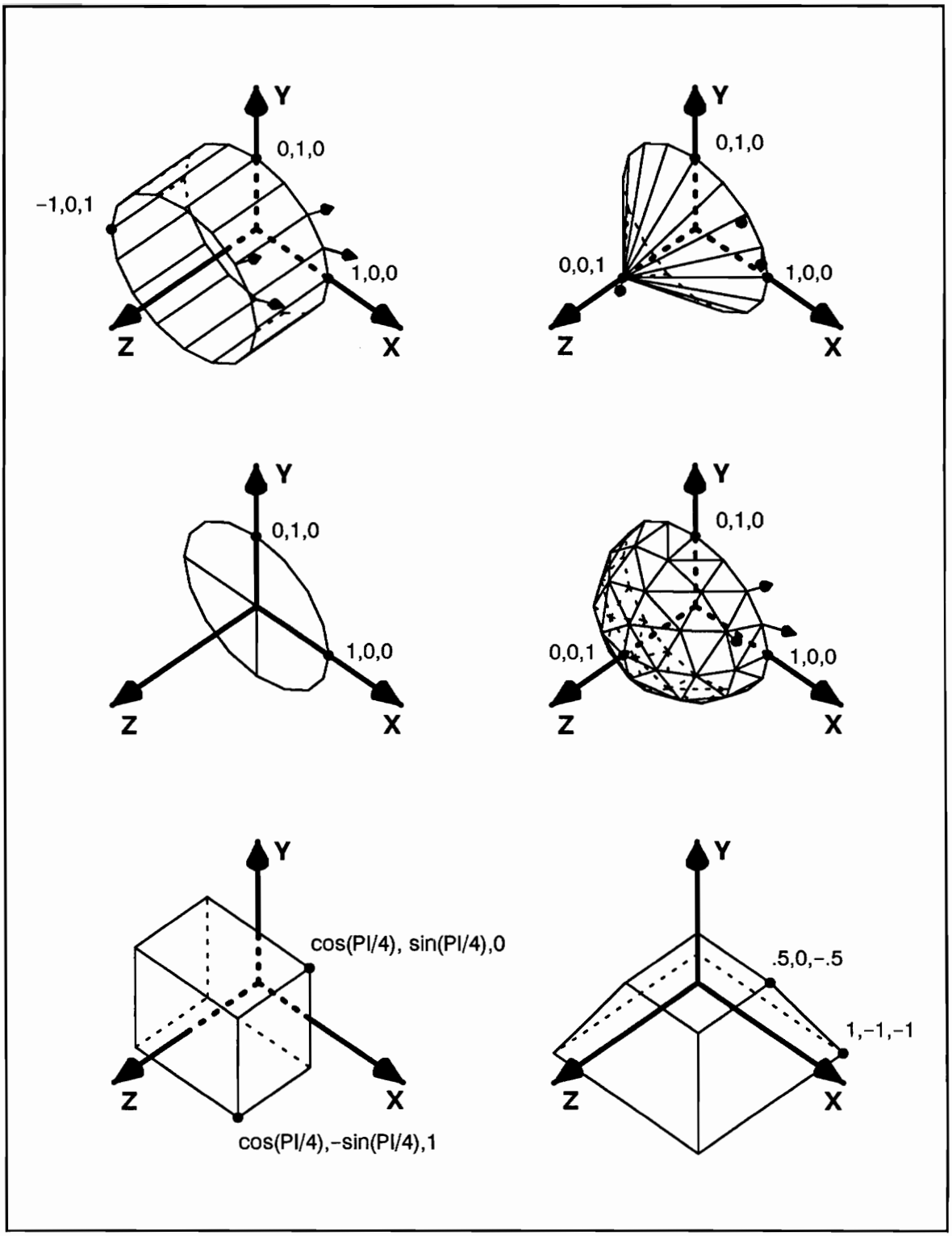

Figure 14. PriSM Unit Primitives

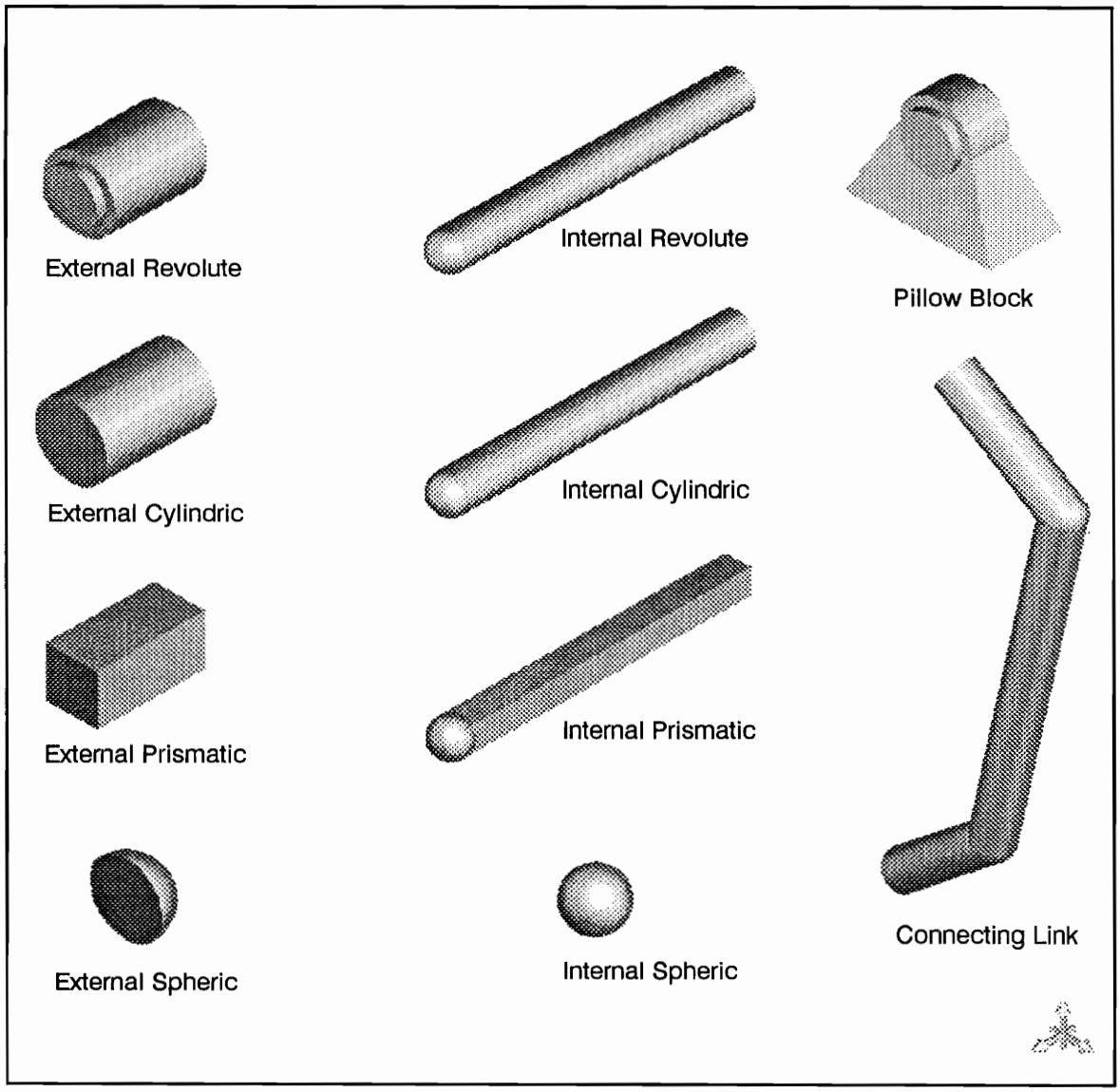

Figure 15. PriSM Joints and Connecting Link

The next level of mechanism generation is driven by the input position file data. The position file specifies the orientation of each link at all positions in the animation sequence.

The final level of mechanism generation identifies a single mechanism position or step in the animation sequence to display on the PHIGS workstation.

During the animation process, this top level mechanism structure is constantly modified to execute the PHIGS structure for the next consecutive position.

One of the greatest strengths of the PHIGS hierarchical structure demonstrated by PriSM is that the six unit primitives are generated and stored only once in graphics memory. Consequently, for all the animation steps modeled, every joint and connecting element is a scaled and oriented instance of the same set of PHIGS graphics primitives. Changing between wireframe and shaded objects in PriSM is simple, because only the PHIGS structures of these six primitives must be modified. The same technique is applied to update the number of lines or fill areas per ninety degrees of arc. The six unit primitives are simply regenerated while the rest of the PHIGS hierarchical structures are left intact. Figure 16 shows an RSRC mechanism with wireframe components (composed of polylines). Figure 17 shows the same RSRC mechanism with flat shaded elements (composed of fill areas), and Figure 18 shows smooth shaded elements (also fill areas).

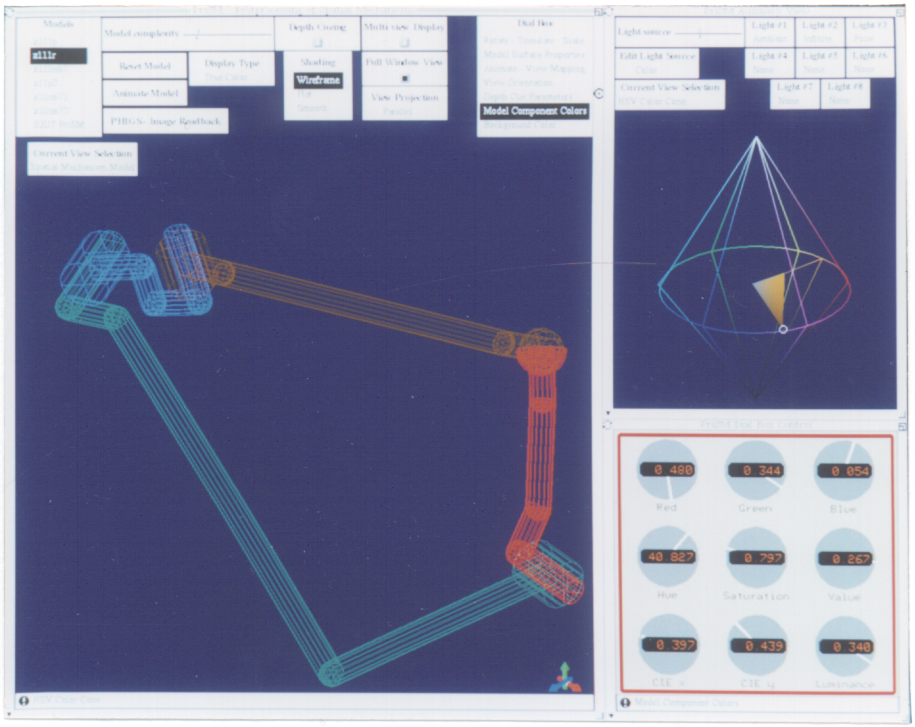

Figure 16. RSRC Mechanism Modeled with Wireframe

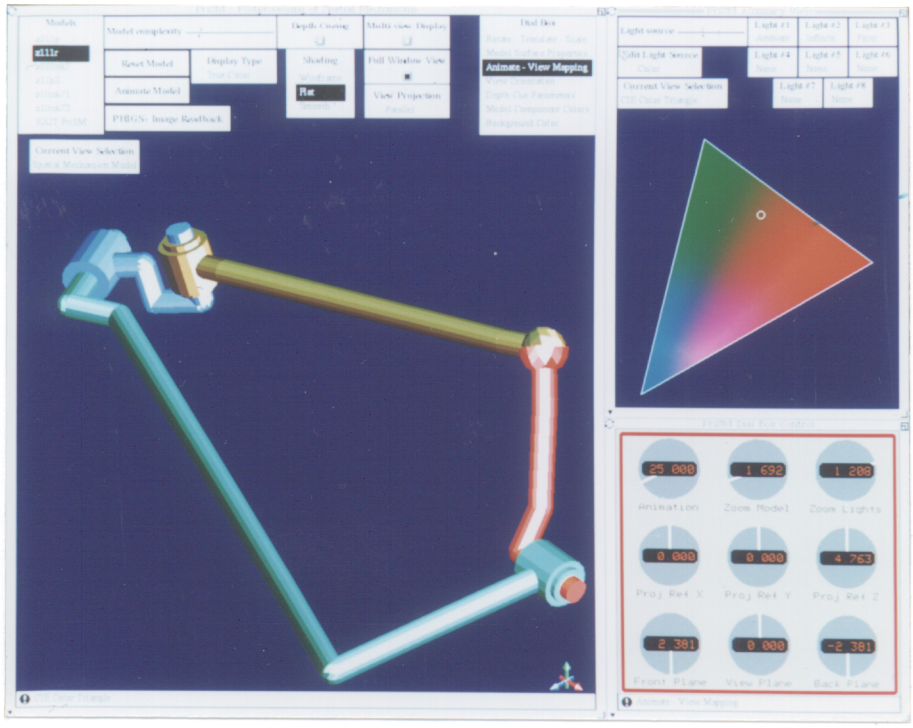

Figure 17. RSRC Mechanism Modeled with Flat Shading

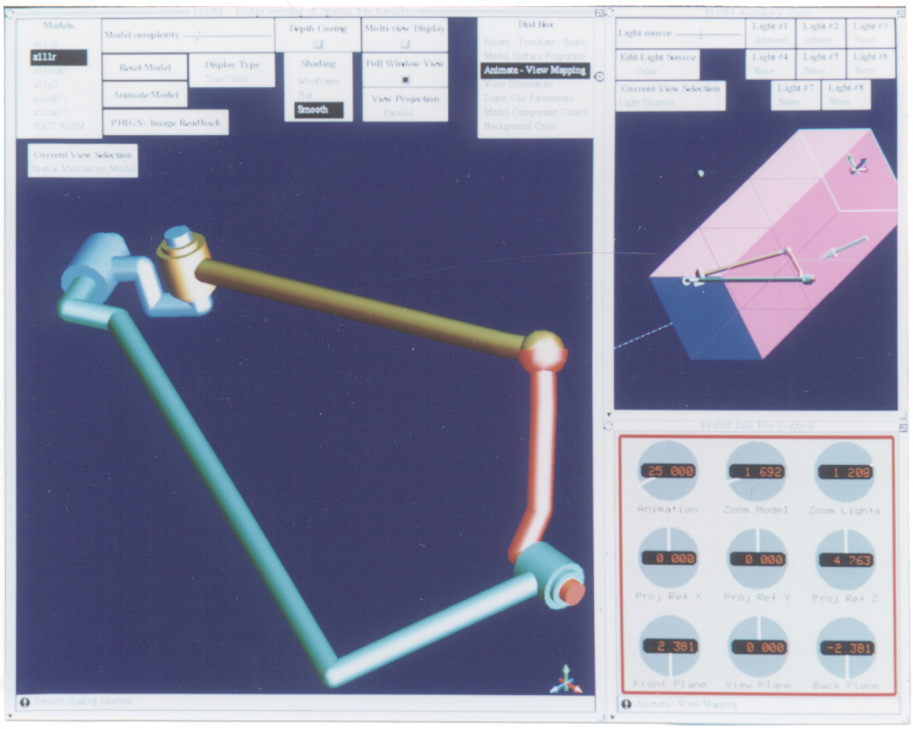

Figure 18. RSRC Mechanism Modeled with Smooth Shading

Chapter 7

PriSM Program Structure

PriSM was written in the C programming language, and consequently used the C version of the Raster Technologies PHIGS/PHIGS+ library. A principal reference for the development of PriSM was the concise 1978 edition of *The C Programming Language* by Kernighan and Ritchie [58] of Bell Laboratories. They revised the book in 1988, but the Sun Microsystems C compiler was based on the original edition. The *C Subroutine Reference Manual* [59] for the Visualization Series GX4000 workstation was the primary PHIGS/PHIGS+ reference.

PriSM comprises 118 C functions which are grouped into 10 modules. Each module contains a related set of functions. For example, the module *psm_color.c* contains all the functions for defining, creating, and manipulating the color models. A complete PriSM source code listing is provided in Appendix C.

Many C data structures are defined for PriSM. These structures combine logically related variables into one unit. As an example, *PSM_global_transformation* is a structure that contains 3-D *rotate, translate*, and *scale* vectors. These vectors are predefined PHIGS structures composed of floating point variables for

x, *y*, and *z*. The use of structures in C simplifies passing data between functions and insures that data is handled correctly.

Numerous PHIGS/PHIGS+ function calls require numeric constants. Some common examples are structure identifiers, labels, and view numbers. To make the PriSM code more readable and easier to understand, descriptive constants were defined in the C include files. The PHIGS library also contained many predefined numeric constants. The PHIGS constants were always used instead of the actual numeric values.

The top level program structure of PriSM is simple. Chart 1 in Figure 19 shows that the main function calls only two PriSM functions. Function *psm_init* manages all start-up initialization. Function *psm_get_input* enters a loop that continually responds to all user initiated input. The debug mode was used extensively during the program development to activate print statements. Some debug print statements remain in the input data parsing functions.

Chart 1.2 in Figures 20 and 21 shows further detail of the program flow in function *psm_init*. The final step of successful program initialization is to display the user selected model in the main workstation window. Chart 1.5 in Figure 22 and Chart 1.5.5 in Figure 23 show the event driven input loop executed by *psm_get_input*. When animation is active, the model is set to the next consecutive step. Then the next (if any) user input is acted upon, and finally all modified PHIGS structures are redrawn.

The compilation and linking of PriSM was simplified by using a UNIX makefile. The makefile for PriSM is listed in Appendix D.

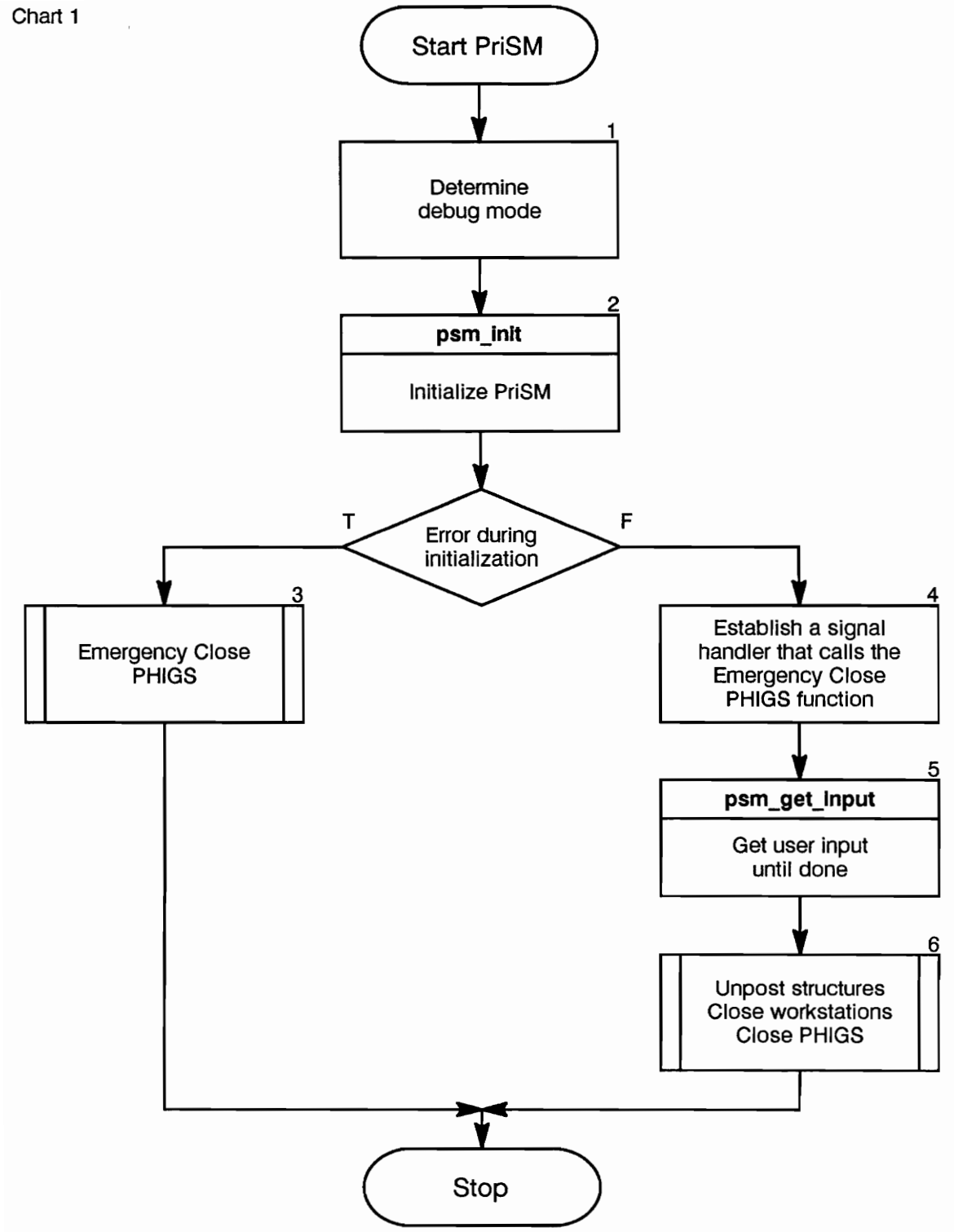

Figure 19. PriSM Main Program Structure (Chart 1)

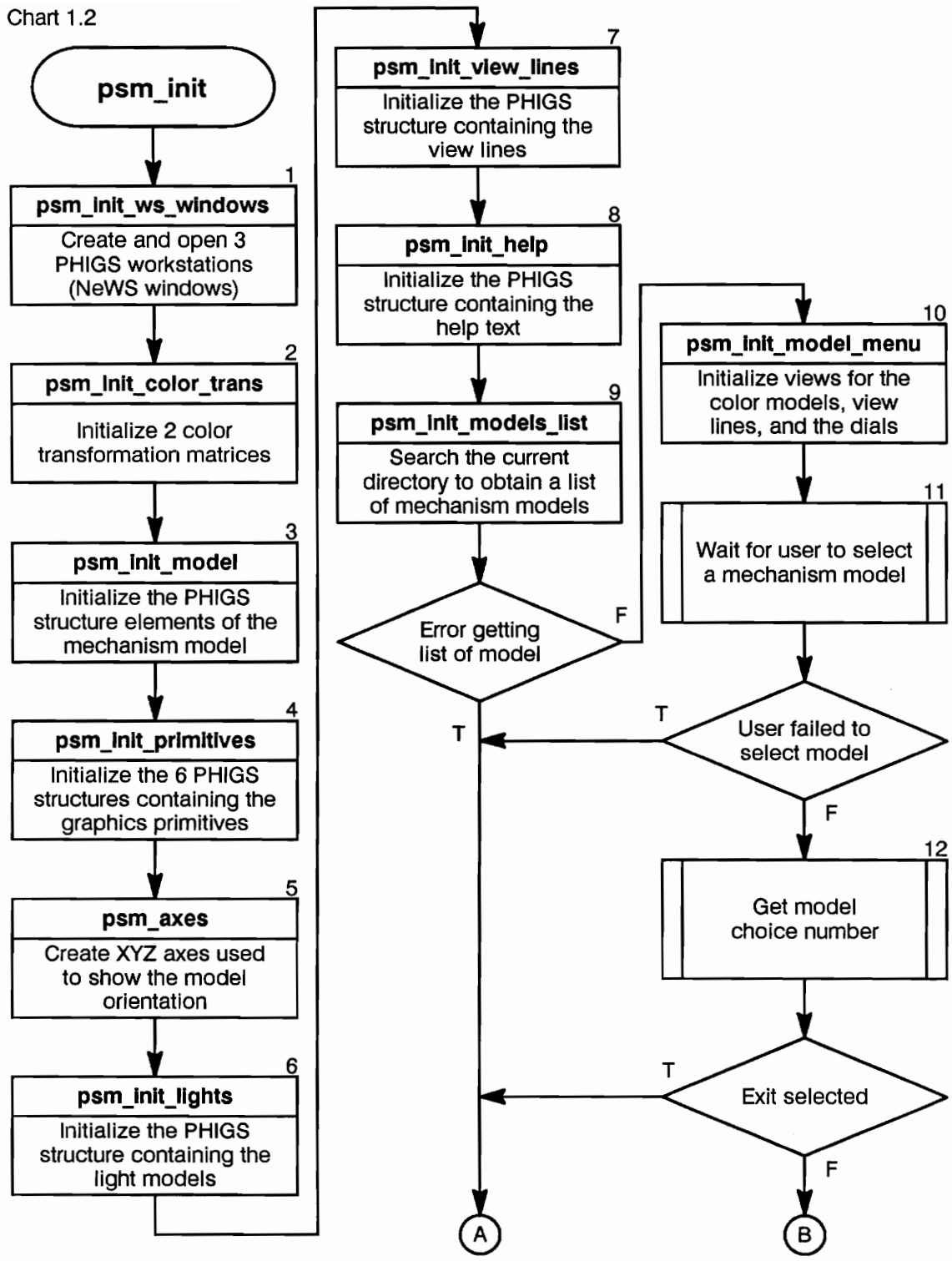

Figure 20. PriSM Initialization Function (Chart 1.2)

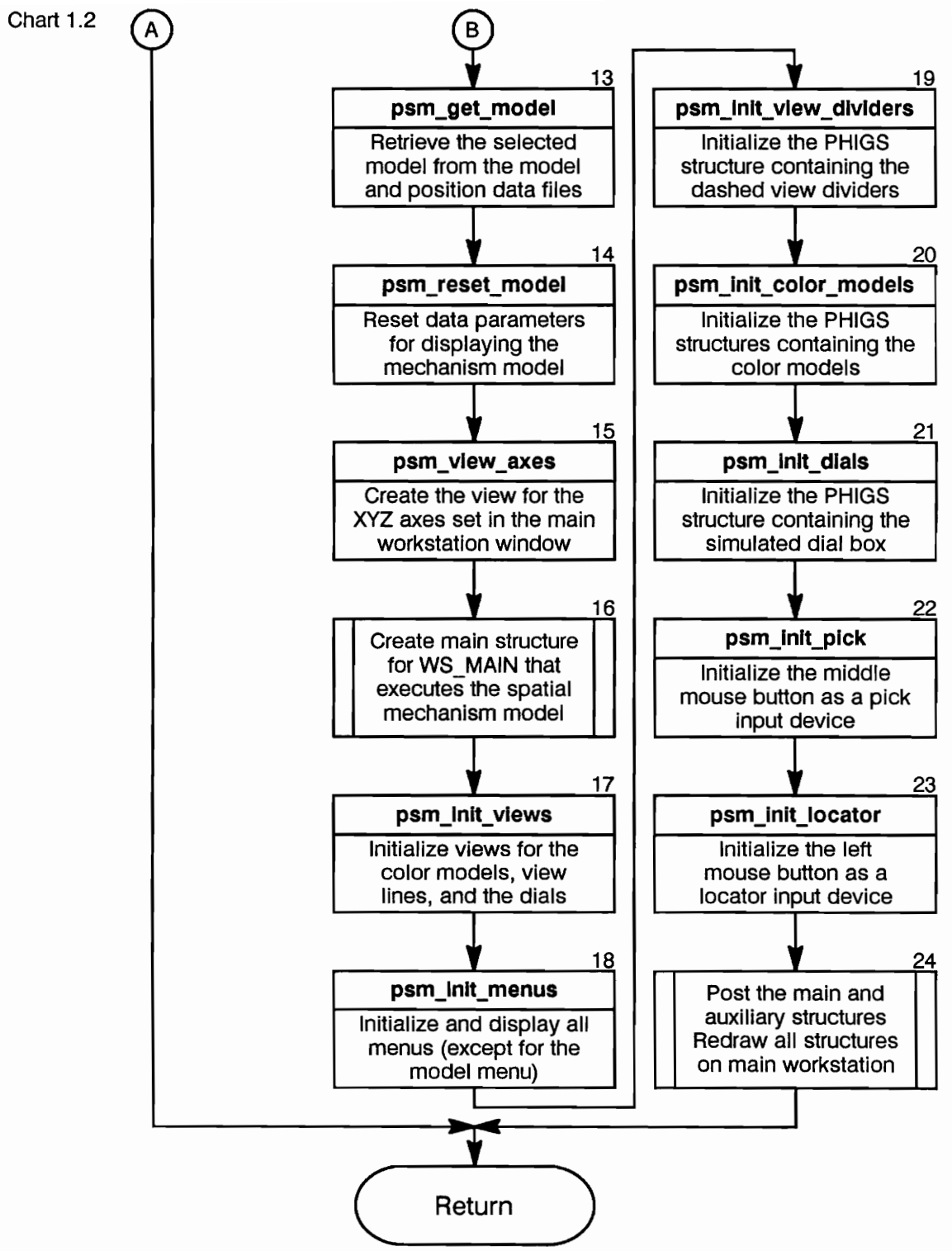

Figure 21. PriSM Initialization Function (Chart 1.2) Continued

Figure 22. PriSM Interactive Input Function (Chart 1.5)

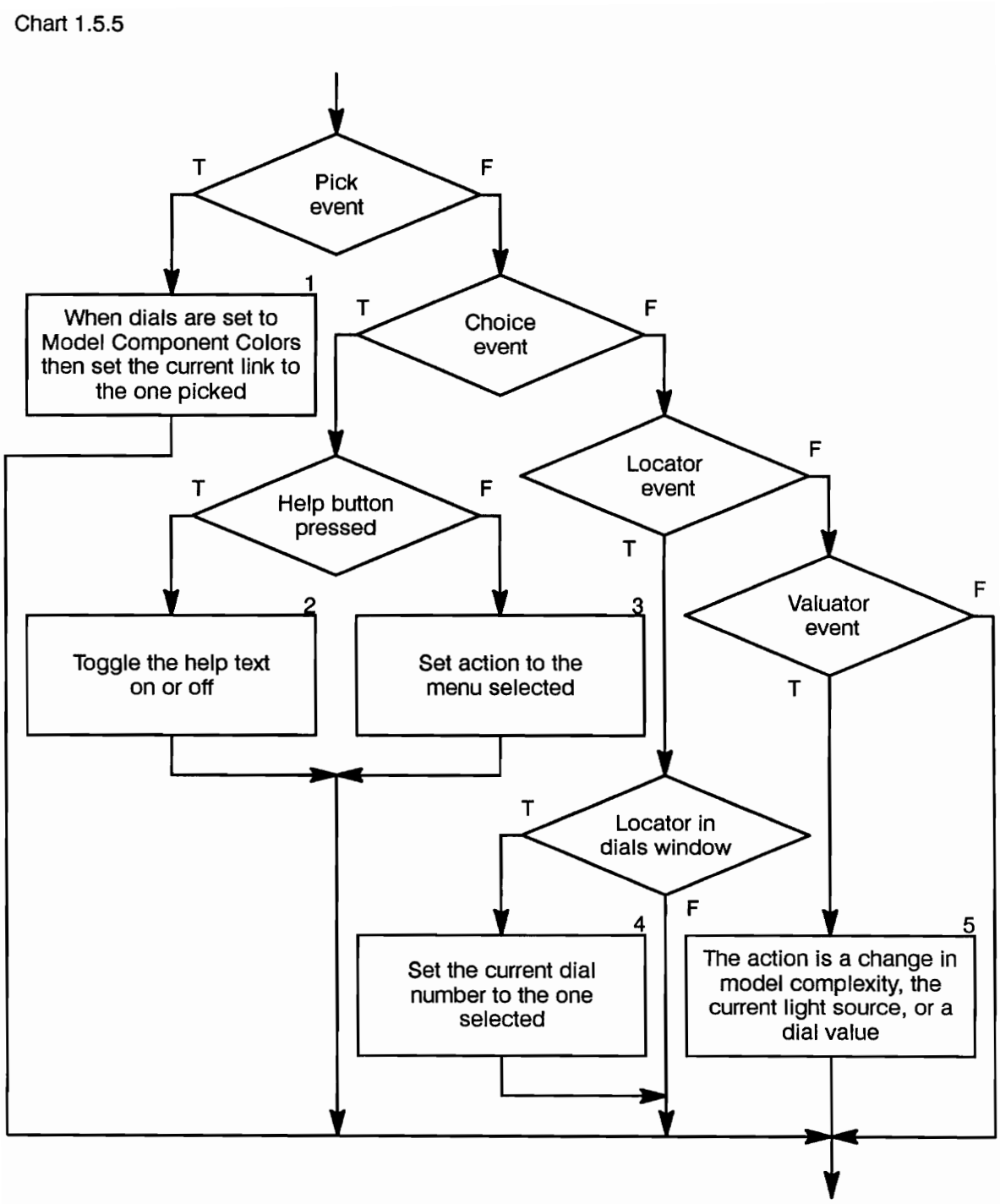

Figure 23. PriSM Definition of Action Variable (Chart 1.5.5)

Chapter 8

PriSM Model Input and Examples

The following mechanism internal and external joint types can be modeled with PriSM:

- *Revolute:* allows only relative rotation about revolute axis
- *Prismatic*: allows only relative translation along prismatic axis
- *Cylindric:* allows rotation about, and translation along axis, both independent of the other
- *Spheric*: allows three simultaneous rotations about a point
- *Screw:* allows rotation about screw axis and translation along axis proportional to rotation by screw pitch (this is a controlled version of the cylindric joint; therefore, it can be modeled by PriSM)
- *Planar:* allows two simultaneous translations along a plane and a rotation about any axis perpendicular to plane (this can be modeled by a revolute joint in PriSM that also moves in the plane perpendicular to its rotation axis)

Two input files are required to model a mechanism. The first file is called the model file; its filename ends with *.mod*. The data in the model file define the

joints, connecting elements, and possibly pillow blocks that constitute links of a mechanism. At this stage, each link is defined in its own local coordinate system. Figure 24 contains the input format for the model file.

The second file is called the position file; its filename ends with *.pos*. The information in the position file specifies the orientation (position and rotation) of each link at each step in an animation sequence. Figure 25 contains the input format for the position file.

Figure 26 shows the coordinate systems and dimensions of three representative joints and the pillow block. Appendix E contains the model and position files for the RSSR spatial mechanism shown earlier in Figure 9. Following are descriptions of the variables used in the model and position files.

- *Ro:* outer radius of external revolute, cylindric, and spheric joints; also the radius of a bounding cylinder for external prismatic joints (the diagonal from the centerline to an outer edge)
- *Ri:* inner radius of external revolute, cylindric, and spheric joints; also the diagonal from the centerline to an edge of the hole through an external prismatic joint
- *Len:* length (along centerline) of external revolute, cylindric, and prismatic joints
- *Tz, Ty, Tx:* rotation angles around the origin applied to the joints in the order Z, Y, X
- *Px, Py, Pz:* position (offset) of the joints from the origin (applied after the rotations)
- *R:* outer radius of the internal revolute, cylindric, and spheric joints; the radius of a bounding cylinder for external prismatic joints (the diagonal from the centerline to an outer edge); the outer radius of the connecting elements

1	Character string description of the model				
2	Element Type	Element ID	Element Description		
3	Parameter 1	Parameter 2	Parameter 3	Parameter 4	. . .
.					
.					
.	Element Type	Element ID	Element Description		
N	Parameter 1	Parameter 2	Parameter 3	Parameter 4	. . .

Supported Element Types

1	ID #	External Revolute Joint						
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz
2	ID #	Internal Revolute Joint						
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz
3	ID #	Connecting Element						
R	Num	P1x	P1y	P1z	P2x	P2y	P2z	. . .
4	ID #	Link (Joint, Joint, Connector,...)						
Num	ID 1	ID 2	ID 3	. . .				
5	ID #	External Prismatic Joint						
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz
6	ID #	Internal Prismatic Joint						
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz
7	ID #	External Cylindric Joint						
Ro	Ri	Len	Tz	Ty	Tx	Px	Py	Pz
8	ID #	Internal Cylindric Joint						
R	Z1	Z2	Tz	Ty	Tx	Px	Py	Pz
9	ID #	External Spheric Joint						
Ro	Ri	Tz	Ty	Tx	Px	Py	Pz	
10	ID #	Internal Spheric Joint						
R	Px	Py	Pz					
11	ID #	Ground Element (Pillow Block)						
Ro	Ri	Z	Ly	Lz	Tz	Ty	Tx	Px	Py	Pz

Figure 24. PriSM Model File Format

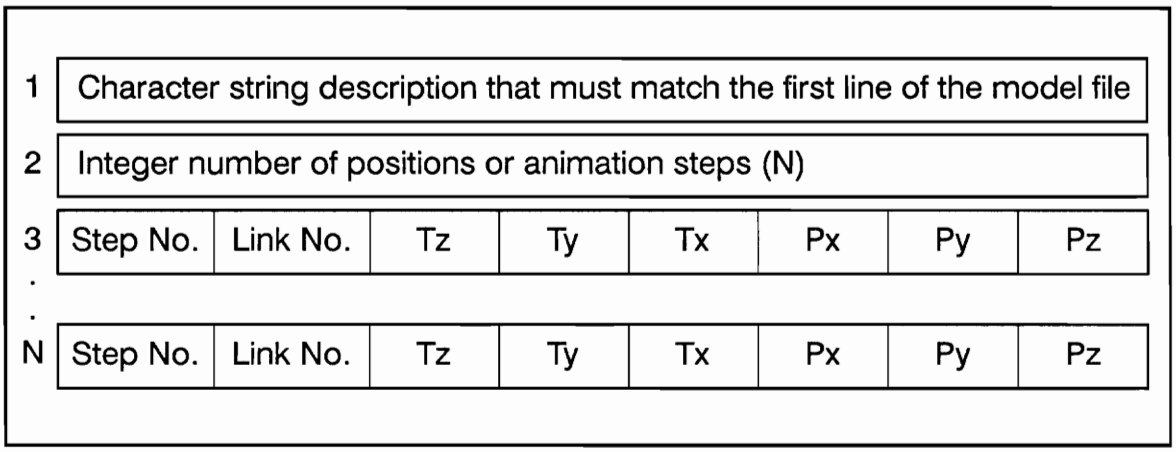

Figure 25. PriSM Position File Format

- *Z1:* end point (with a sphere) of the internal revolute, prismatic, and cylindric joints along the local Z-axis
- *Z2:* end point (without a sphere) of the internal revolute, prismatic, and cylindric joints along the local Z-axis
- *Num:* number of XYZ positions along connecting elements; the number of elements in a link
- *P1x, P1y, P1z:* XYZ coordinates of positions that constitute connecting links
- *ID#:* element IDs of joints and connectors
- *Z:* center point of the cylinders in the pillow block along the Z-axis
- *Ly:* length (height) of the pillow block frustum in the Y-direction
- *Lz:* length of the pillow block (cylinders and frustum) in the Z-direction

Appendix F contains the Guide to Running PriSM. It explains the steps for logging on the GX4000 workstation, starting and stopping PriSM, and logging off

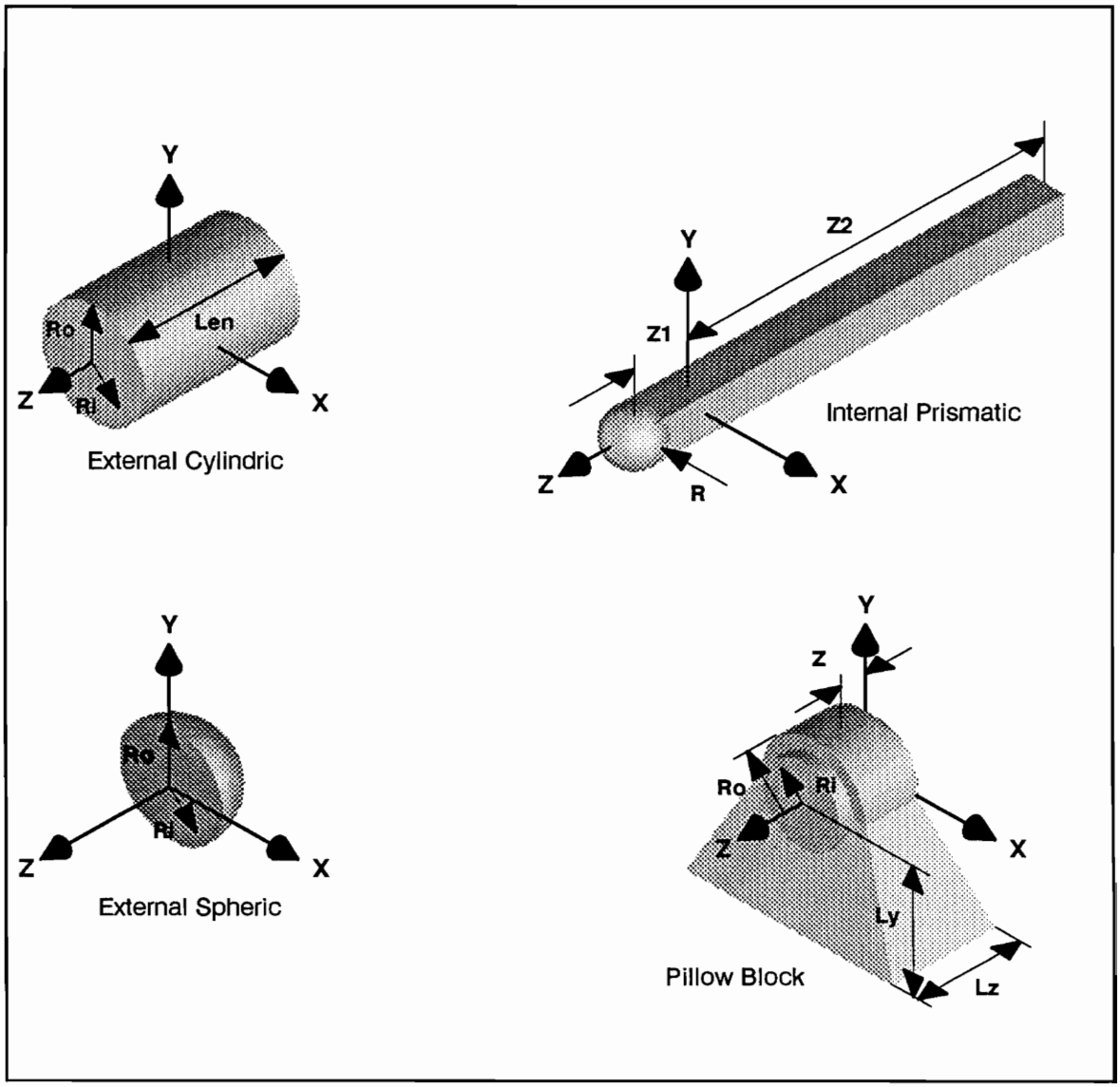

Figure 26. Joint Dimensions

the GX4000 workstation. Following are some additional examples of the model and scene rendering capabilities of PriSM.

The RCCC model in the upper left of Figure 27 has its ambient and diffuse lighting surface property coefficients set to 1.0 (full influence) and its specular lighting coefficient set to 0.0 (no specular light influence). The model in the upper

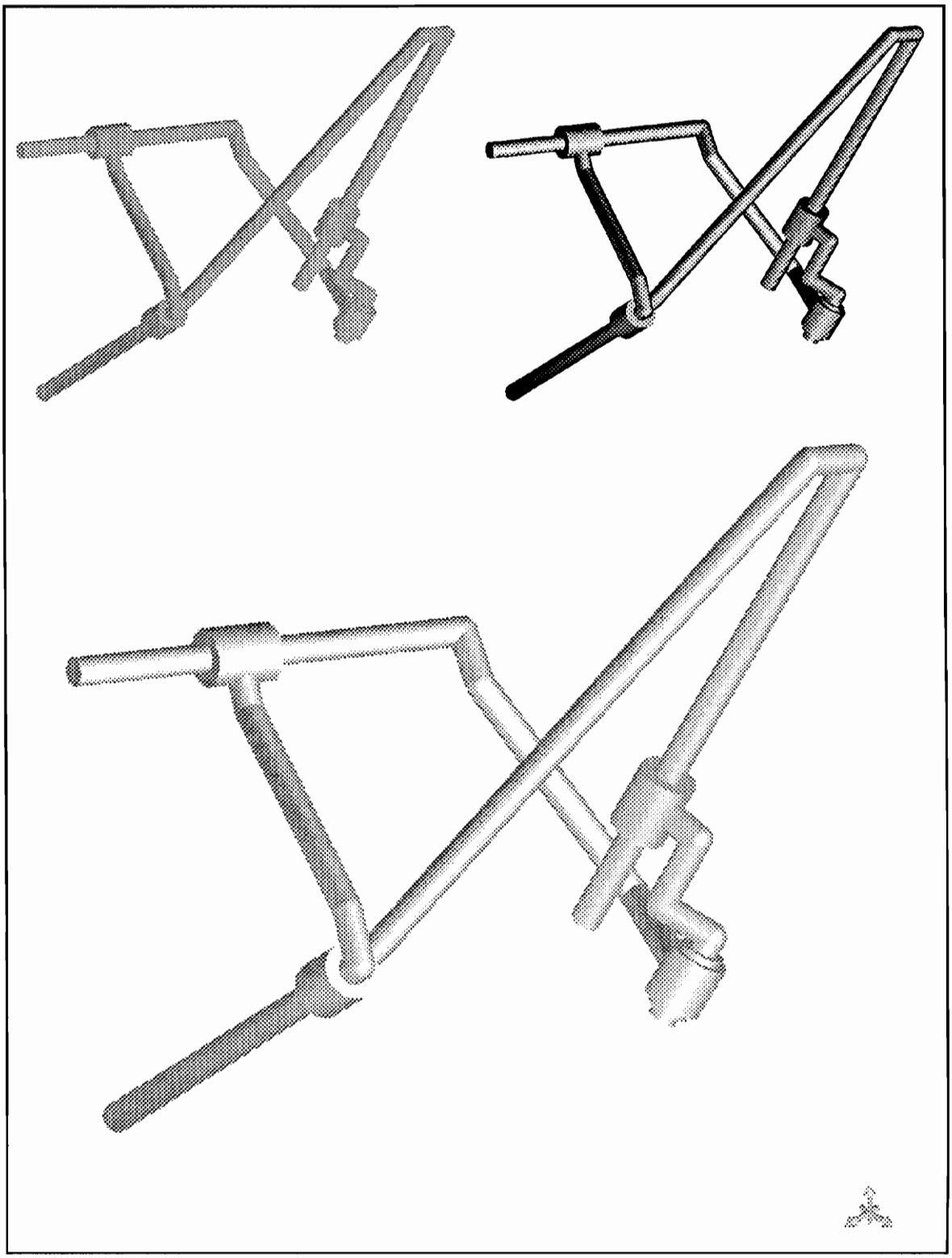

Figure 27. RCCC Mechanism with Various Lighting Surface Properties

right has the specular lighting coefficient set to 1.0 with the others set to 0.0. The combined influence of all three types of lighting is shown in the model at the bottom with the coefficients all set to 1.0. Again, these surface property lighting coefficients are modified interactively through PHIGS logical valuators, by rotating the appropriate dials on the dial box, or through a PHIGS logical locator, by simulating the rotation of the dials with the mouse. No numeric data entry is required with the keyboard. Figure 28 shows a color picture of the RCCC mechanism with specular lighting. Figure 29 shows the same RCCC model without the specular highlights.

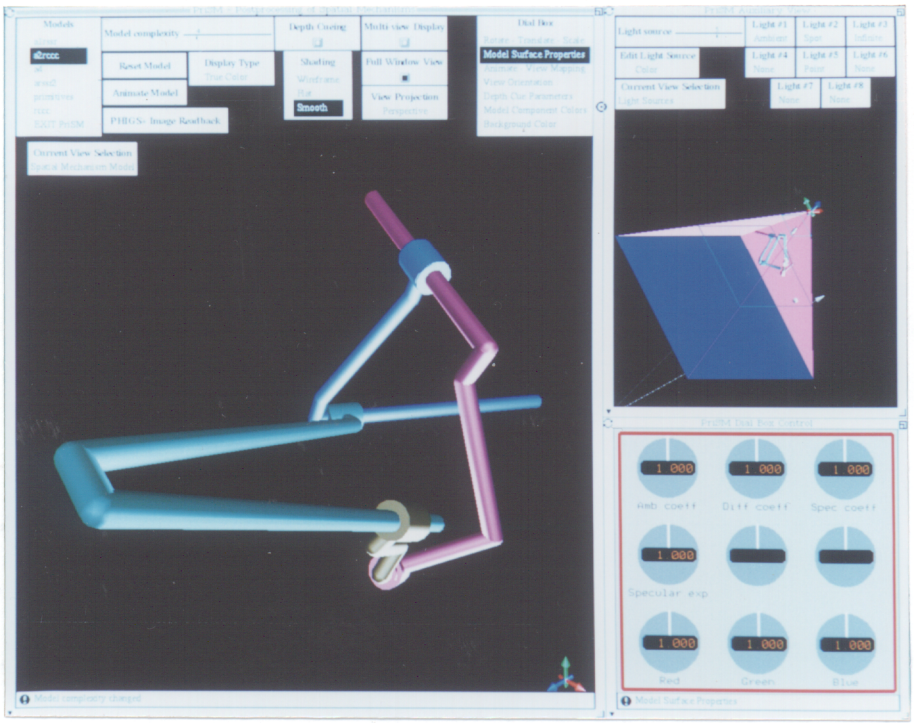

Figure 28. RCCC Mechanism with Specular Highlights

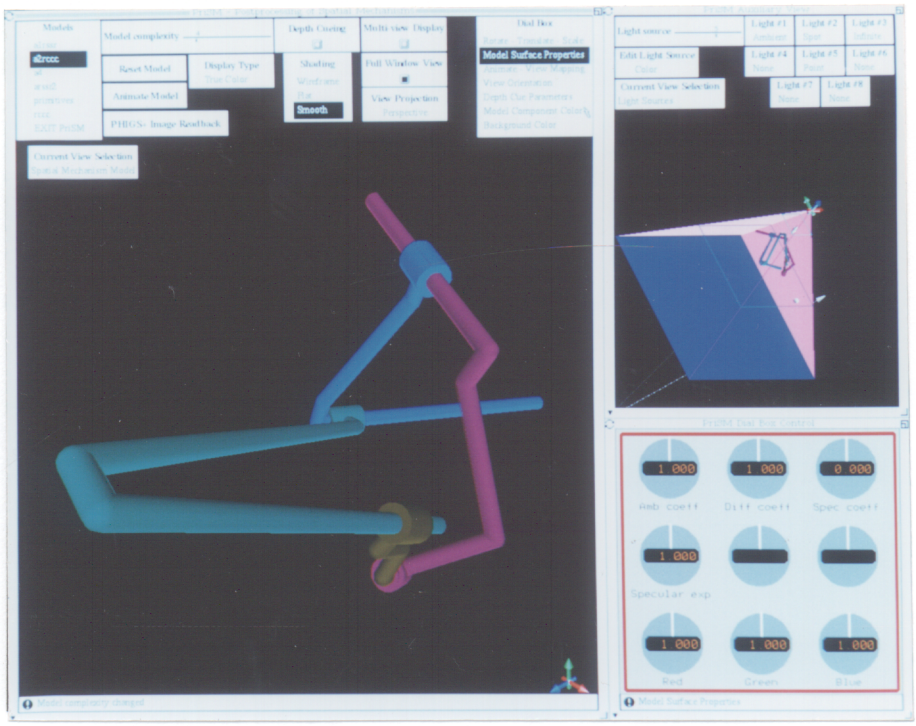

Figure 29. RCCC Mechanism without Specular Highlights

Figure 30 shows the effect of perspective projection on the same RCCC mechanism. This feature greatly enhances the depth perception of the model. Parameters that control the viewing volume (back plane, front plane, viewing plane, eye point, etc.) are all accessible for interactive modification. Depth cueing has also been applied to this model. The portions of the model that are farther away from the eye point are set to fade into a dark color. On the CRT display, the background would probably be set to a dark color, so the model would appear to fade away. The depth cue color, depth cue planes, and the depth cue scale values for the front and back depth cue planes are also available for interactive modification by the user.

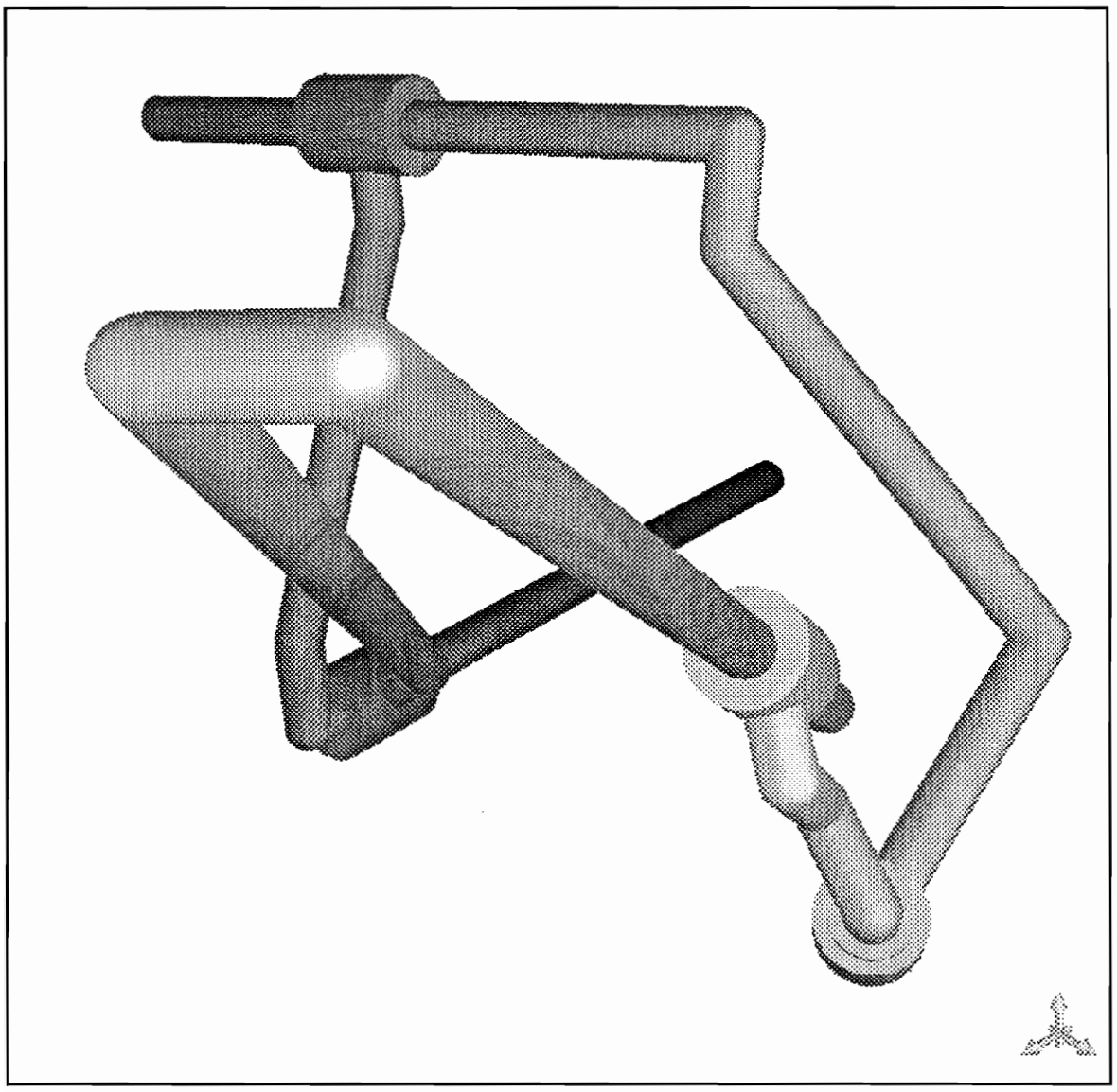

Figure 30. RCCC Mechanism with Perspective Projection and Depth Cueing

Figure 31 shows the result of two directional lights shining on a spheric joint. The light source colors have different intensities; therefore, they produce different levels of specular reflections on the model.

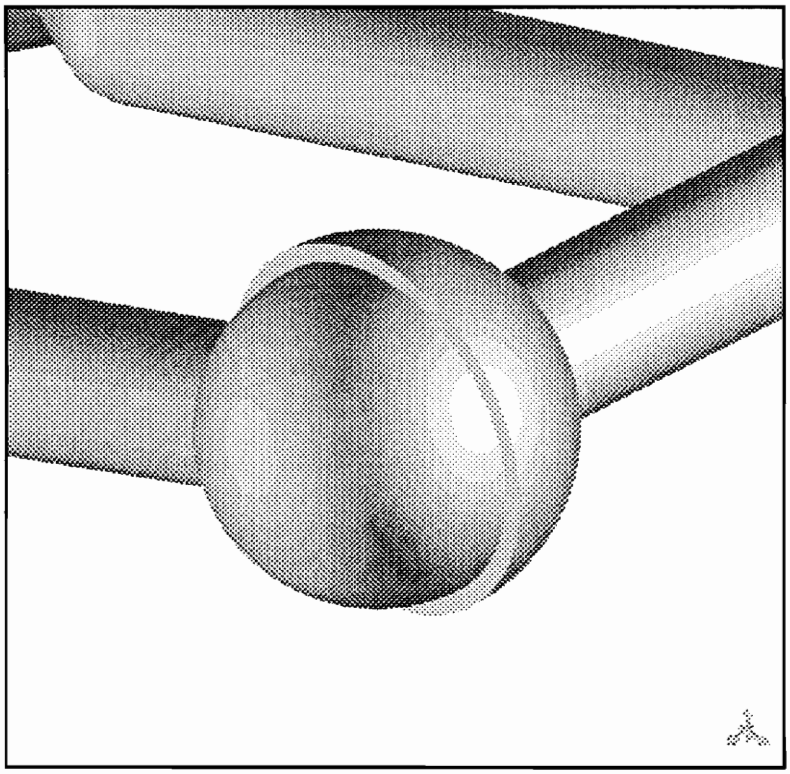

Figure 31. Two Directional Light Sources Shining on a Spheric Joint

Figure 32 shows the effect of modifying the view orientation on a model of a simple robot. The view orientation parameters include the view reference point position; this is generally the center of the model.

Figure 33 is a model of a 7-R mechanism showing a visible collision of the green and dark blue links. During animation of this mechanism, the links can be clearly seen passing through each other.

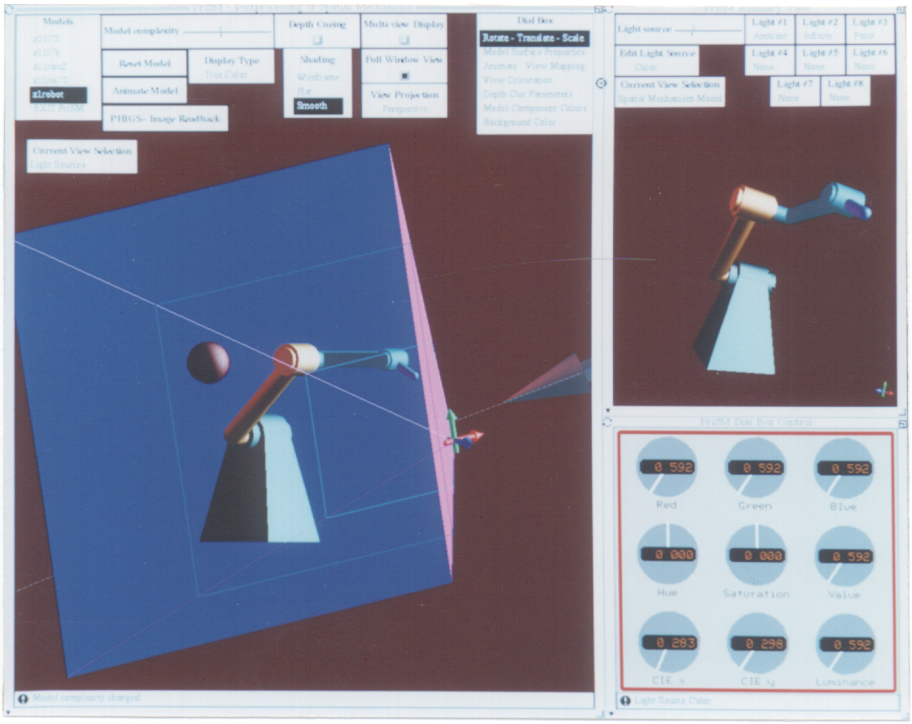

Figure 32. Robot Model with Modified View Orientation

Figure 33. 7-R Mechanism with Visible Collision

Chapter 9

PriSM Enhancement Recommendations

Couple with Synthesis and Analysis Code

PriSM is a stand-alone application. It serves to postprocess spatial mechanism models that comply with the PriSM input format. Since PriSM constitutes only a portion of the spatial mechanism design cycle, it would be more useful directly integrated with mechanism synthesis and analysis code. This thesis demonstrated value in realistic visualization, but it is only one part of a broader goal to support computer aided design of spatial mechanisms from concept to computerized prototype.

Interference Detection and Elimination

It is no longer sufficient to only synthesize and analyze mechanisms, especially spatial mechanisms. Many people have considered certain classes of spatial mechanisms to be impractical. The RCCC and the 7-R mechanisms are certainly among those. One reason might be the difficulty in visualizing their

motion. It is also difficult to determine if collision-free configurations of some mechanisms are possible. PriSM addresses the first problem by providing the designer a powerful visualization tool which eliminates the need for tedious modeling and analysis. The doctoral work of Keil [24] resulted in a program called SLIDE (Spatial Linkage Interference Detection and Elimination) which addresses the second problem.

SLIDE is a group of algorithms that generate mathematical models of binary linked mechanisms. SLIDE can construct models of mechanisms which contain joints of the same types as PriSM: revolute, prismatic, cylindric, and spheric. SLIDE sizes and orients the joint elements, eliminates any interferences between joints, and then proceeds to construct links between the joints. The collision detection, reshaping phase of SLIDE analyzes the mechanism at each discrete step of its motion. If a collision is detected, the links are reshaped in a manner which maintains the attachment constraints for each link section. All of this is accomplished while retaining the original kinematic properties of the mechanism. Some mechanisms cannot be reshaped without re-configuration. To aid in this re-configuration, the data from the initial construction phase can be passed to PriSM for visualization.

Most of the spatial mechanism examples in this thesis were reshaped by SLIDE. Figure 34 shows an RSSR mechanism before and after reshaping.

Although PriSM and SLIDE have been interfaced to demonstrate both model rendering and collision avoidance of spatial mechanisms, they were implemented independently of each other. SLIDE has been interfaced to the IMP mechanism analysis code, but it also runs independently of mechanism synthesis

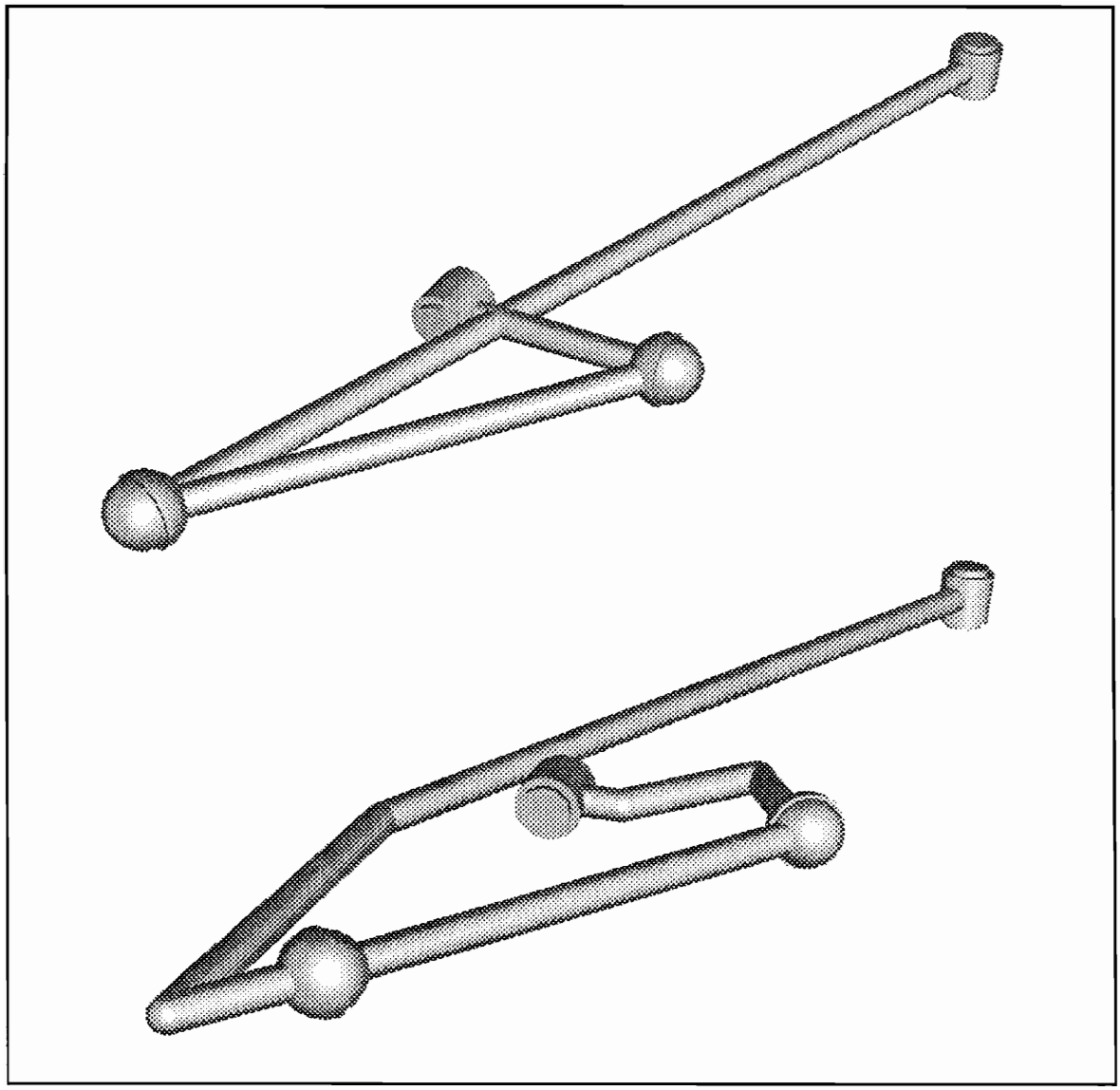

Figure 34. RSSR Mechanism Before and After Resphaping by SLIDE

and analysis programs. A proposed system that merges the interactive model rendering and animation of spatial mechanisms with algorithms for interference detection and elimination would further improve the overall mechanism design cycle. Many of the collision avoidance algorithms could be accelerated by the interactive assistance of the user. The user could decide which reshaping methods

to apply, while the system could automatically indicate the locations of collisions throughout the spatial mechanisms motion cycle.

Solid Modeling

PriSM incorporates a basic surface model technique for modeling the components of spatial mechanisms. This fulfills the visualization requirements of this thesis. Incorporating solid modeling geometry would enhance PriSM by modeling more properties of spatial mechanisms. These properties would include mass properties and intersecting volumes. The capability to detect intersecting volumes of the spatial mechanism models would allow the automatic indication of interferences and collisions within a model. This would be an alternative to interference detection by visual inspection of the mechanism models or by a separate program.

Robotics

The application of the PriSM model rendering system to robotic manipulators is very straightforward. A robot, which generally consists of an open loop kinematic chain, can be modeled as a spatial mechanism by adding one more link that "stretches" as the robot moves. The link from the end of the hand to the base would not be displayed. This technique would require that predefined motion sequences be generated for animation by PriSM. A more suitable application would be to extend PriSM to read a set of rules that define motion constraints for each joint. Users could interactively move the robot by rotating dials that corresponded to joint rotations or positioned the end effector in space. Additional

geometric components that more closely model robotics mechanisms would be desirable; however, the simple geometries used in PriSM were sufficient to produce the model in Figure 35.

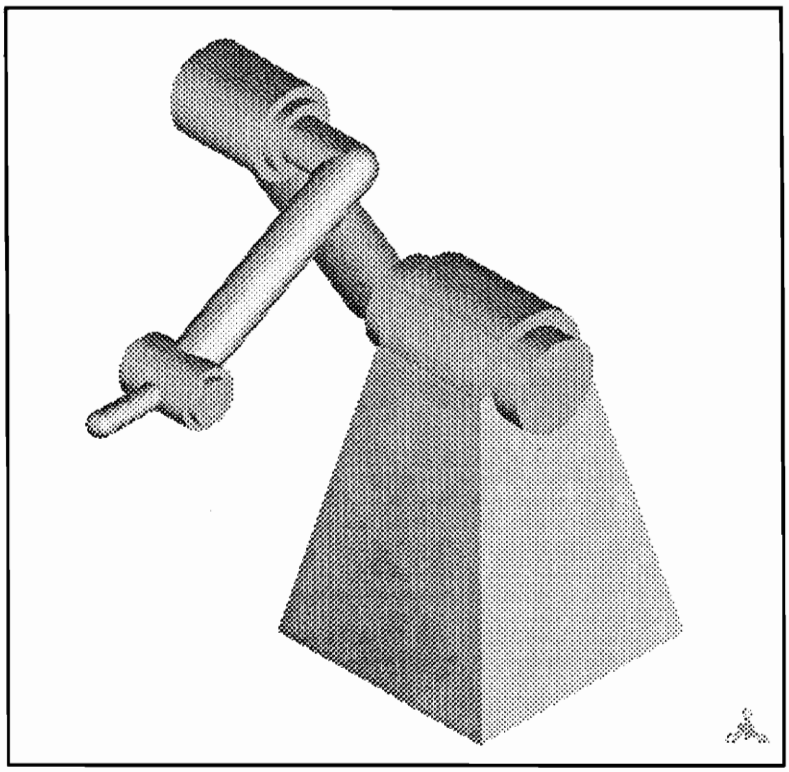

Figure 35. Simple Open Loop Robot Modeled with PriSM

Save Scene Parameters

A function to save and recall the parameters used to create a scene would be a useful enhancement to PriSM. Users can quickly modify the default settings for a scene, but the ability to recall previously defined scene parameters is needed. In addition, this capability would be beneficial for quickly comparing different scenes of the same model.

Phong Shading

The smooth shading algorithm of PHIGS+ invoked by PriSM is currently Gouraud shading. Phong shading is similar, but it can produce models that appear even more realistic. The version of the GX4000 PHIGS+ library (1.3.6) used for this thesis did not support Phong shading. A simple enhancement to PriSM would be to invoke both the partial (dot product) and/or full (normal) Phong shading options of PHIGS+.

Advanced Surfaces

The Raster Technologies version of PHIGS+ used for this work did not contain advanced surfaces. All surfaces that require lighting and shading properties are approximated with polygons using the *fill area 3 with data* primitive. The PHIGS+ functional specification [34] defines functions for non-uniform B-spline surfaces and uniform parametric polynomial B-spline and Bezier surfaces. Incorporating these advanced surfaces would simplify the modeling and provide more accurate representation of the cylinders, cones, and spheres modeled in PriSM.

Transparency

The transparency primitive attribute was not functional in the GX4000 PHIGS+ library. Transparency would provide additional visual aid to the mechanism designer by allowing the viewer to see internal joints and portions of the mechanism blocked by the mechanism itself. The use of transparency would require more detailed modeling of the external joints. These joints were not

modeled in PriSM with holes for the internal joints to fit in. Hidden surface elimination precluded the need to model the holes; however, transparency would expose this shortcut in modeling the external joints.

The display of the viewing volume could also be improved with the transparency attribute. The current viewing volume display does not draw the external sides of the viewing volume that face the viewer. This allows the viewer to see "inside" the viewing volume. Making all sides visible, but transparent, would keep the mechanism model visible at all times, regardless of the location of the viewing volume.

Light Source Attenuation

The light source attenuation coefficients of point and spot light source types control the effect of weakening light. The attenuation is a function of the distance between a light source and objects in the display. No provisions were made in PriSM for the interactive modification of the XYZ attenuation coefficients since they were not yet supported by the GX4000 PHIGS+ graphics library. This capability may not be a major feature of the PHIGS+ lighting model, but light source attenuation would further enhance realistic scene rendering.

Relative Rotations

Regardless of the order that the user modifies the X, Y, and Z rotation angles in PriSM, they are always applied to the model in XYZ order. This creates some unusual effects when two or three of the rotation angles have a non-zero value and the X or Y rotations are modified.

Relative rotations would be a more intuitive approach to rotating the model. Each new X, Y, or Z rotation of the model would be applied to the current angular orientation of the model. Then the model would appear to revolve around the axis of the angle being modified. A drawback to this approach is that the XYZ rotation angle values displayed on the simulated dial box would be meaningless. However, the current angular orientation of the model could be shown by back solving for the XYZ angles in a preselected order.

Space Ball Input

Although the application of relative rotation angles would reduce confusion in rotating a model, rotating a dial is not a natural motion for rotating and translating a model or the light sources. A six degree-of–freedom input device like the recently available *space ball* would allow natural movements of pushing and turning the *space ball* to match corresponding movements of the spatial mechanism model. Presently, the user might turn a dial in one direction expecting the model to move one way, but instead it moves in the opposite direction. The *space ball* would be interfaced as a PHIGS logical input valuator in the same manner as the dial box.

Replace ppolyline3data Patch for Hidden Line Problem

The Raster Technologies version of PHIGS+ used for this thesis did not perform hidden line elimination. Hidden surface elimination worked well, but the PHIGS+ structures containing lines and surfaces displayed obvious flaws in hidden line elimination. This problem was evident in PriSM with the HSV color

cone and the light source and viewing volume display. To overcome this limitation, lines were inserted in the PHIGS structures starting with lines farther from the view point first and the ones closer to the view point later. In a general sense, that is how HLHSR is performed, so the results were satisfactory. Some problems are still evident with the light source and viewing volume display. Some concise code was included as comments in the function *psm_init_hsv_cone* (Appendix C) for building the HSV color cone structure with properly working HLHSR.

Chapter 10

Conclusions

This thesis presents new ideas and methods for simplifying model and scene rendering which improve the visualization of closed loop spatial mechanism designs. The ability to interactively specify a realistic graphic scene without keyboard entry simplifies the mechanism design cycle. The PriSM graphical postprocessor was developed independently of spatial mechanism synthesis and design codes. Some of the benefits of PriSM include better designs, a reduction in design time, and thus more time to concentrate on the design process.

PriSM demonstrates the credibility of using the PHIGS international 3-D graphics standard for advanced interactive graphics of specialized computer-aided engineering software. Perception and evaluation of spatial mechanism designs are significantly improved by the use of PHIGS+ functionality to produce animated models that are shaded, lighted, and depth cued.

References

1. Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System," *AFIPS Conference Proceedings*, Vol. 23, 1963, pp. 329–346.

2. Johnson, T. E., "Sketchpad III: A Computer Program for Drawing in Three Dimensions," *AFIPS Conference Proceedings*, Vol. 23, 1963, pp. 347–353.

3. Sheth, P. N., and Uicker, J. J., "IMP (Integrated Mechanism Program), A Computer-Aided Design Analysis System for Mechanisms and Linkages," *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 94, May 1972, pp. 454–464.

4. Myklebust, A., and Tesar, D., "The Analytical Synthesis of Complex Mechanisms for Combinations of Specified Geometric or Time Derivatives up to the Fourth Order," *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 97, May 1975, pp. 714–722.

5. Sivertsen, O., and Myklebust, A., "MECSYN: An Interactive Computer Graphics System for Mechanism Synthesis by Algebraic Means," ASME Design Engineering Technical Conference, Los Angeles, California, Paper No. 80–DET–68, September 1980.

6. Rubel, A. J., and Kaufman, R. E., "KINSYN III: A New Human-Engineered System for Interactive Computer-Aided Design of Planar Linkages," *Transactions of the ASME, Journal of Engineering for Industry*, Vol. 99, No. 2, May 1977, pp. 440–8.

7. Erdman, A. G., and Gustafson, J. E., "LINCAGES: Linkage INteractive Computer Analysis and Graphically Enhanced Synthesis Package," ASME Design Engineering Technical Conference, Chicago, Illinois, Paper No. 77–DET–5, September 1977.

8. Paul, B., "Dynamic Analysis of Machinery via Program DYMAC," Society of Automotive Engineers, Paper No. 770049, 1977.

9. Chuang, J. C., Strong, R. T., and Waldron, K. J., "Implementation of Solution Rectification Techniques in an Interactive Linkage Synthesis Program," *ASME Journal of Mechanical Design*, July 1981, pp. 657–664.

10. Chace, M. A., "Computer-Aided Engineering of Large-Displacement Dynamic Mechanical Systems," National Conference on University Programs in Computer-Aided Engineering, Design and Manufacturing, Brigham Young University, Salt Lake City, Utah, April, 1983.

11. Reinholtz, C. F., *Optimization of Spatial Mechanisms,* Doctoral Dissertation, University of Florida, Gainsville, Florida, 1983.

12. Tilove, R. B., "Extending Solid Modeling Systems for Mechanisms Design and Kinematic Simulation," *IEEE Computer Graphics and Applications,* Vol. 3, No. 3, May/June 1983, pp. 9–19.

13. Coats, B. A., and Cipra, R. J., "A Computer Graphics Technique for Creating and Animating Spatial Linkages using IMP (Integrated Mechanisms Program)," *Proceedings of the Eighth Applied Mechanisms Conference*, St. Louis, Missouri, September 1983.

14. Keil, M. J., *A Method for Automatically Generating and Animating 3-D Models of Planar Linkages,* Master's Thesis, Florida Atlantic University, Boca Raton, Florida, 1984.

15. Keil, M. J., and Myklebust, A., "Automatic 3-D Geometric Modelling and Animation of Planar Mechanisms," *Proceedings JSPE International Symposium on Design and Synthesis,* JSPE, Tokyo, 1984, pp. 395–9.

16. Myklebust, A., Keil, M. J., and Reinholtz, C. F., "MECSYN-IMP-ANIMEC: Foundation for a New Computer-Aided Spatial Mechanism Design System," *Mechanism and Machine Theory*, Vol. 20, No. 4, 1985, pp. 257–269.

17. Myklebust, A., Reinholtz, C. F., Francis, W. H., and Keil, M. J., "Design of a Radar Guidance Mechanism Using MECSYN-ANIMEC," ASME Design Engineering Technical Conference, Cambridge, Mass., Paper No. 84–DET–139, 1984.

18. Keil, M. J., Myklebust, A., and Reinholtz, C. F., "Prediction of Link Interference in Planar Mechanisms," *Proceedings of the Ninth Applied Mechanisms Conference*, Kansas City, October 1985.

19. Barris, W. C., and Riley, D. R., "The Impact of the Workstation Environment on Mechanism Synthesis Strategy," ASME Design Engineering Technical Conference, Columbus, Ohio, Paper No. 86–DET–150, 1986.

20. Barris, W. C., Kota, S., Riley, D. R., and Erdman, A. G., "Mechanism Synthesis Using the Workstation Environment," *IEEE Computer Graphics and Applications,* March 1988, pp. 39–50.

21. Pennington, S. L., *Automatic Geometric Modeling of Spatial Mechanism Links,* Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1986.

22. Pennington, S. L., Keil, M. J., and Myklebust, A., "Automatic Generation of Spatial Mechanisms Geometric Models to Avoid Link Interference," *Proceedings of the Seventh World Congress on the Theory of Machines and Mechanisms*, Vol. 1, Sevilla, Spain, 1987, pp. 285–8.

23. Keil, M. J., and Myklebust, A., "An Improved Interference Detection Method for Spatial Mechanism Geometric Models," *ASME 20th Biennial Mechanisms Conference*, Vol. 2, Orlando, Florida, September 1988, pp. 83–9.

24. Keil, M. J., Personal communications – doctoral research on *Automatic Generation of Interference Free Geometric Models of Spatial Mechanisms*, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1989–1990.

25. Thatch, B. R., *A PHIGS Based Interactive Graphical Preprocessor for Spatial Mechanism Analysis and Synthesis*, Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1987.

26. Thatch, B. R., and Myklebust, A., "A PHIGS Based Graphics Input Interface for Spatial-Mechanism Design," *IEEE Computer Graphics and Applications,* Vol. 8, No. 2, March 1988, pp. 26–38.

27. Thompson, J. M., *Computer Aided Design and Synthesis of the RSCR Spatial Mechanism,* Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1987.

28. Crane, C. D., Duffy, J., "Applications of Computer Graphics to Robot Path Planning and Control," *Proceedings of the Seventh World Congress on the Theory of Machines and Mechanisms*, Vol. 2, Sevilla, Spain, 1987, pp. 1033–6.

29. Choi, Y. J., Crane III, C. D., Matthew, G. K., "Interactive Off-line Robot Path Processor," *Proceedings of the 1988 ASME International Computers in Engineering Conference*, Vol. 2, San Francisco, July-August 1988, pp. 249–252.

30. Crane, C. D., Duffy, J., "An Interactive Animated Display of Man-Controlled and Autonomous Robots," *Proceedings of the 1986 ASME International Computers in Engineering Conference*, Vol. 1, Chicago, Illinois, July, 1986, pp. 35–8.

31. Kahloun, F., Malowany, A., Angeles, J., "A Graphic Simulator for Robotics," *Proceedings of the 1988 ASME International Computers in Engineering Conference*, Vol. 2, San Francisco, July-August 1988, pp. 227–231.

32. Foley, J. D., and van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1982.

33. Rogers, D. F., "Procedural Elements for Computer Graphics," McGraw-Hill Inc., New York, 1985.

34. "PHIGS+ Functional Description Revision 3.0," *Computer Graphics*, Vol. 22, No. 3, July 1988, pp. 125–218.

35. Jern, M., "Unravelling Graphics Standards," *Datamation*, Vol. 33, December 1, 1987, pp. 56+.

36. Bono, P. R., Encarnacao, J. L., Hopgood, F. R. A., and ten Hagen P. J. W., "GKS – The First Graphics Standard," *IEEE Computer Graphics and Applications,* Vol. 2, No. 5, July 1982, pp. 9–22.

37. Bono, P. R., "Guest Editor's Introduction, Graphics Standards," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 12–6.

38. Carson, G. S., and McGinnis, E., "The Reference Model for Computer Graphics," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 17–23.

39. Carson, G. S., "The Future of ISO Graphics Standards," *IEEE Computer Graphics and Applications,* Vol. 8, No. 4, July 1988, pp. 82–3.

40. Meads, J. A., and Puk, R. F., "The Standards Pipeline," *Computer Graphics*, Vol. 22, No. 3, October 1988, pp. 292–3.

41. Henderson, L., Journey, M., and Osland, C., "The Computer Graphics Metafile," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 24–32.

42. Powers, T., Frankel A., and Arnold, D., "The Computer Graphics Virtual Device Interface," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 33–41.

43. Meads, J. A., and Puk, R. F., "The Standards Pipeline," *Computer Graphics*, Vol. 22, No. 1, February 1988, pp. 44–7.

44. Puk, R. F., and McConnell, J. I., "GKS-3D: A Three-Dimensional Extension to the Graphical Kernel System," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 42–9.

45. Williams, T., "Hardware advances fuel graphics standards controversy," *Computer Design*, Vol. 25, No. 15, August 15, 1986, pp. 28–32.

46. Chin, J. S., "Graphics Standards – The Standards Conformance Game," *IEEE Computer Graphics and Applications,* Vol. 9, No. 3, May 1989, pp. 76–7.

47. Williams, T., "3-D Modeling Screams for Super Performance in Graphics Workstations," *Computer Design*, August 1, 1987, pp. 53+.

48. Rost, R. J., Friedberg, J. D., Nishimoto, P. L., "PEX: A Network-Transparent 3D Graphics System," *IEEE Computer Graphics and Applications,* Vol. 9, No. 4, July 1989, pp. 14–26.

49. Plaehn, M., "PHIGS: Programmer's Hierarchical Interactive Graphics Standard," *Byte*, Vol. 12, November, 1987, pp. 275–6+.

50. Shuey, D., Bailey, D., Morrissey, T. P., "PHIGS: A Standard, Dynamic, Interactive Graphics Interface," *IEEE Computer Graphics and Applications,* Vol. 6, No. 8, August 1986, pp. 50–7.

51. Alliant Computer Systems Corporation, Visualization Series Programming Guide, Part No. 301–02011 Revision A, Littleton, Massachusetts, March 1989.

52. Galitz, W., "Human Engineering in Screen Design," *Journal of Systems Management*, Vol. 34, No. 5, May 1983, pp. 6–11.

53. Snowberry, K., Parkinson, S. R., and Sisson, N., "Computer Display Menus," *Ergonomics*, Vol. 26, No. 7, 1983, pp. 699–712.

54. Ero, J., and van Liere, R., "User Interface Management Systems," *Advances in Computer Graphics III,* Springer-Verlag, Berlin, 1988, pp. 99–131.

55. Berk, T. S., Brownston, L. S., and Kaufman, A., "A New Color-Naming System for Graphics Languages," *IEEE Computer Graphics and Applications,* Vol. 2, No. 3, May 1982.

56. Durrett, J. H., editor, *Color and the Computer*, Academic Press, Inc., Boston, Massachusetts, 1987.

57. Meads, J., "The Standards Pipeline," *Computer Graphics*, Vol. 20, No. 3, July 1986, pp. 164–6.

58. Kernighan, B. K., and Ritchie, D. M., *The C Programming Language*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1978.

59. Alliant Computer Systems Corporation, Visualization Series C Subroutine Reference Manual, Part No. 302–00032 Revision A, Littleton, Massachusetts, March 1989.

60. Abi-Ezzi, S. S., Bunshaft, A. J., "An Implementer's View of PHIGS," *IEEE Computer Graphics and Applications,* Vol. 6, No. 2, February 1986, pp. 12–23.

61. Torborg, J. G., "A Parallel Processor Architecture for Graphics Arithmetic Operations," *Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 197–204.

Appendix A

Raster Technologies GX4000 Workstation

PriSM was implemented on a Raster Technologies *(Alliant)* GX4000 workstation. The GX4000 is a Sun Microsystems 4/260 supercomputing workstation with a proprietary Raster Technologies graphics subsystem installed on the VME bus (Figure 36). The high performance graphics subsystem has a parallel multiprocessor graphics architecture that was designed and optimized for PHIGS+. Abi-Ezzi and Bunshaft [60] confirm the value of hardware designed for PHIGS. They conclude that the best implementation of PHIGS is achieved by mirroring the functionality of PHIGS in hardware. The GX4000 graphics subsystems is a hardware implementation of PHIGS/PHIGS+. A condensed description of the GX4000 workstation follows, and more detailed information is covered by Torberg [61] and the Visualization Series Programming Guide [51].

The Sun 4/260 supercomputing workstation utilizes a 10-MIPS SPARC™ (Scalable Processor ARChitecture) CPU running the UNIX operating system with a C language compiler and Sun's NeWS (Network/extensible Window System).

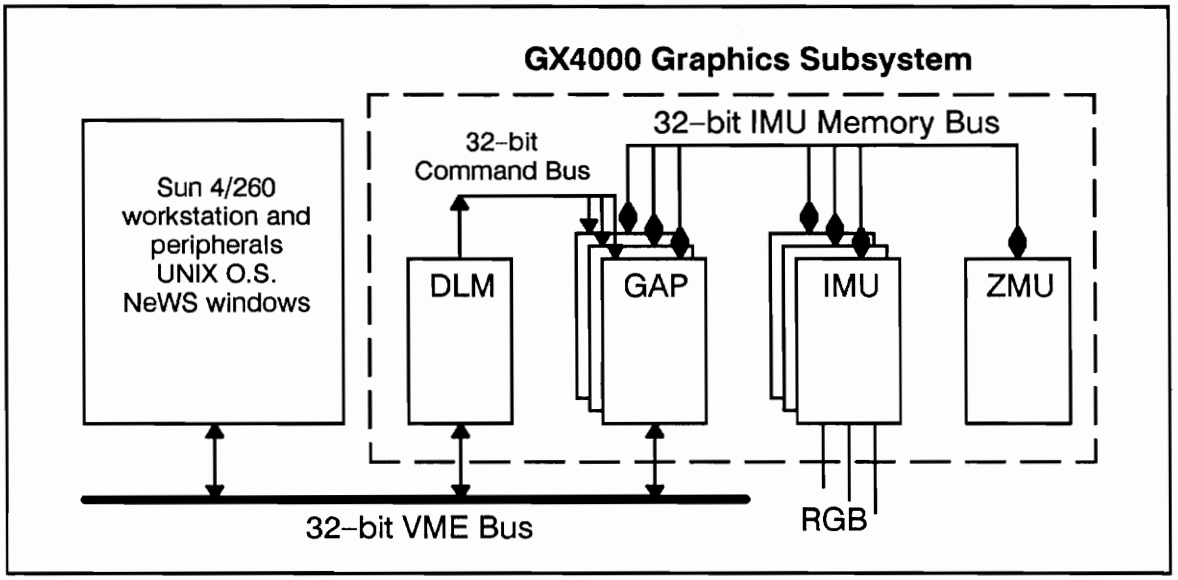

Figure 36. Raster Technologies GX4000 Workstation

The SPARC processor is a reduced instruction set computer. It contains the industry standard 32-bit VME bus for integrating high performance devices including the GX4000 graphics subsystem. An adaptation of NeWS, called RNeWS, provides windowing capability on the GX4000 graphics monitor. The monitor is the Sony model GDM–1950 1280 x 1024 pixel, 60 Hz, non-interlaced display. The chromaticity coordinates of the phosphors used in the GDM–1950 are listed in Table 4. Figure 37 contains the CIE triangle that corresponds to these values.

The GX4000 graphics subsystem used for this project consists of an 8 MB DLM (Display List Manager), 2 GAPs (Graphics Arithmetic Processors), an IMU (Image Memory Unit), and a ZMU (Z-buffer Memory Unit). The workstation had a 327MB hard disk drive, a 60MB cartridge tape drive, 8MB of main memory, a dial box with nine knobs, and an optical mouse.

Table 4. Chromaticity Coordinates of the Sony GDM–1950 Graphics Monitor

	X	Y
Red	0.625 ± 0.030	0.340 ± 0.030
Green	0.280 ± 0.030	0.595 ± 0.030
Blue	0.155 ± 0.016	0.070 ± 0.015
White (9300 K)	0.283 ± 0.030	0.298 ± 0.030

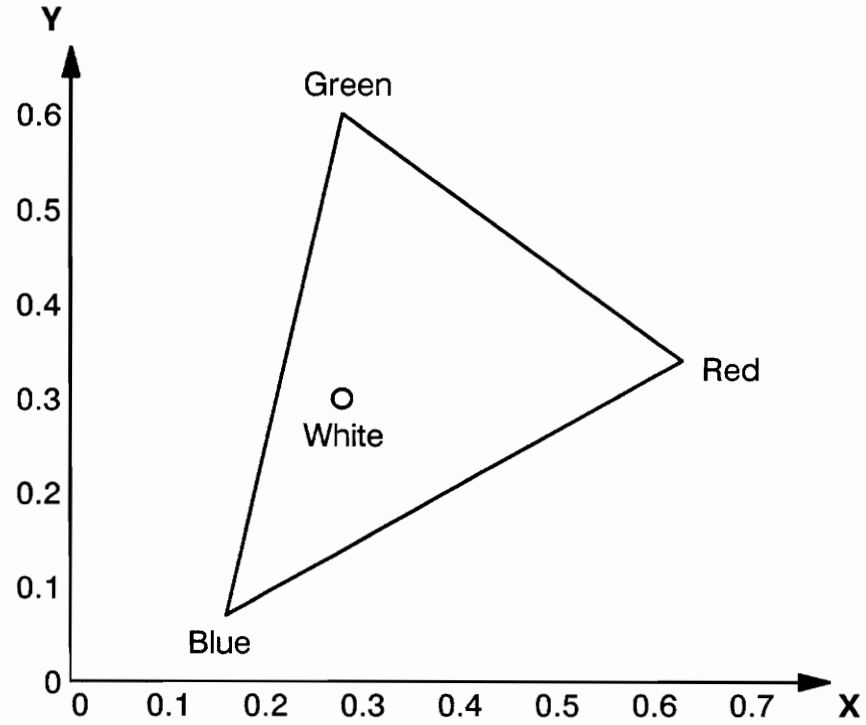

Figure 37. CIE Color Triangle for the GDM–1950 Graphics Monitor

Display List Manager

The DLM is a dedicated 16-bit 20 MHz memory management processor for processing display lists of graphics commands. It can be configured for 8 to 32 MB of dual-ported memory to support very large graphics objects. It contains the PHIGS hierarchical graphics structures. This memory is separate from the Sun 4/260 workstation memory, but it is directly accessible by the workstation processor and workstation memory management processor as well as the graphics system.

Graphics Arithmetic Processors

The GAPs are dedicated to graphics output calculations. A 256K x 64-bit microstore on each GAP contains a native PHIGS+ instruction set providing optimized performance of a standard graphics library. The GAPs contain a 10 MIPS (Million Instructions Per Second) 32-bit integer processor and a 20 MFLOPS (Million FLoating point Operations Per Second) 32-bit floating point processor. Each GAP even contains 1 MB of local RAM. The graphics calculations are initiated by commands generated when the DLM is traversed. The GAPs process transformations, clipping, lighting, shading, picking, and other general commands. Each GX4000 graphics subsystem can coordinate up to eight GAPs in parallel. The synchronization and arbitration of multiple GAPs is handled over the command bus. Each GAP has a command input FIFO (First-In First-Out) stack, and the GAP with the command input FIFO requiring the least amount of processing time gets the next command. Additional GAPs provide a near linear increase in graphics performance. Conceptually, graphics rendering commands are executed by multiple GAPs as if all commands were processed by one GAP.

Image Memory Units

The IMUs contain the image memory, also known as the frame buffer memory. The IMUs consist of parallel VLSI (Very Large Scale Integration) drawing processors that execute low level graphics primitives (vectors, smooth shaded patches, rectangle fills, block moves, and image transfers). Since low level graphics are handled by the IMUs instead of the GAPs, the GAPs process independently of the number of pixels required to display each graphics object. The IMUs have hardware capabilities for panning and zooming. Each IMU supports 1280 x 1024 true color 24-bit images (16.7 million possible colors) or double buffered pseudo color 8-bit images. The true or pseudo color option is selectable on a window-by-window basis. Double buffered true color images require three IMUs. The IMUs also contain a 4-bit overlay plane for text windows, cursors, etc. The GX4000 workstation will support up to 6 IMUs which allows for various combinations of multi-headed (multiple monitors) true color and true color double buffered displays. Rendering performance specifications include: a peak drawing rate of 60 million pixels per second, 1 million short vectors/second, or 330,000 small triangles/second with interpolated shading.

Z-buffer Memory Unit

The ZMU is an optional 1280 x 1024 x 16 bit VLSI depth-buffer processor that executes in parallel with IMUs to perform hidden line and hidden surface removal. Since the ZMU works in parallel with the IMUs, throughput of the graphics commands is not reduced when hidden line and surface removal is enabled.

Appendix B

PriSM Help Text

PriSM: Postprocessing of Spatial Mechanisms

PriSM is a highly interactive graphics model rendering system. Only the optical mouse is required for input. The dial box is optional, because the dials can be "turned" by using the mouse. To turn a dial with the mouse, first select the dial you want to "turn" by pointing at a dial with the mouse arrow and clicking the first button. Next, move the mouse arrow in a circular motion around the dial within the gray area. Either click the first button again or move the mouse arrow outside the dial area to stop the active rotation. Note: the small area in the center of each dial is inactive.

MODEL COMPLEXITY

The model complexity is the number of facets per 90 degree arc used to form the primitives (cylinder, cone, prism, half-sphere, circle, and frustum). These primitives comprise the spatial mechanism model, axes, and light sources.

MODEL SELECTION

This menu contains the list of available spatial mechanism models and animation sequences (positions).

RESET MODEL

This function resets the following model rendering attributes:
- Rotate - Translate - Scale
- Model Surface Properties
- Animation Step
- Model View Mapping
- Model View Orientation
- Depth Cue Parameters
- Model Component Colors

ANIMATE MODEL

This function turns automatic animation on and off. All other input functions may still be used during automatic animation.

IMAGE READBACK

This function creates a file ("bitmap.gray") containing a 1000 by 1000 grayscale pixel bitmap image of the current screen. Each byte of data in the file corresponds to a single pixel and has values from 0 (black) to 255 (white). Since the grayscale data is derived from the RGB values for each pixel, this image readback data will only be correct if the display type is TRUE COLOR.

DISPLAY TYPE

Two display types are available for PriSM. True Color provides a 24-bit color approximation of the graphics entities displayed. This display type is currently NOT double buffered. Grayscale DB provides an 8-bit pseudo color approximation of the graphics entities displayed. This display type is double buffered; therefore it is more pleasing to the viewer during all redraw functions (especially animation).

DEPTH CUEING

Depth cueing is the process of "fading" colors at each point on a graphics entity, based on its distance from the observer. The most common use for depth cueing is to have the model "fade" into the background color.

SHADING SELECTION

The shading selection controls how the basic primitives used in PriSM (cylinder, cone, prism, half-sphere, circle, and frustum) are formed. When wireframe is selected, the primitives are formed with polylines. When either flat or smooth shading is selected, the primitives are formed with fill areas. Smooth shading is an implementation of Gouraud shading, also known as bilinear interpolated shading. This method applies the lighting calculation at each vertex of a fill area and then interpolates the resulting colors across the fill area to draw a polygonal patch with smoothly varying colors.

MULTI-VIEW DISPLAY

The multi-view display for PriSM generates four views for the spatial mechanism model: top, front, side, and "isometric". The fourth view is actually the view displayed in single view mode and it is manipulated by all the viewing controls.

VIEW SIZE

The full window view simply uses the entire main window (a PHIGS workstation) for displaying the spatial mechanism model or other objects.

VIEW PROJECTION

The view projection is either parallel or perspective. This function only controls the view projection of the spatial mechanism model.

Note: due to a problem in the pick input devices, picking of model components (for setting the model component colors) only works with parallel projection.

ROTATE - TRANSLATE - SCALE

This function is used to manipulate the spatial mechanism model within the model coordinate system. All transformations are relative to the origin of the model coordinate system and are applied in the following order:

Scale
Rotate X
Rotate Y
Rotate Z
Translate

MODEL SURFACE PROPERTIES

Model surface properties affect the lighting calculations for rendering the spatial mechanism model with flat or smooth shading.

The RGB values may be modified by rotating the dials (or using the mouse). The color can also be modified by pointing at a location in the RGB bars or the CIE triangle and clicking the first button on the mouse.

ANIMATE - VIEW MAPPING

The animation value shows the current animation position of the spatial mechanism model. The view mapping values controls the view of the spatial mechanism model. The effect of changing these values is visualized best when the light source view is selected.

VIEW ORIENTATION

These values control the orientation of the viewing volume for the spatial mechanism model.

DEPTH CUE PARAMETERS

The depth cue parameters are only applied to the display when depth cueing is on. A scale value of 0.0 means that none of the spatial mechanism model color is combined with the depth cue color. Conversely, a scale value of 1.0 means that none of the depth cue color is combined with the model color.

The RGB values may be modified by rotating the dials (or using the mouse). The color can also be modified by pointing at a location in the RGB bars or the CIE triangle and clicking the first button on the mouse.

MODEL COMPONENT COLORS

Each link of the spatial mechanism can be selected separately via a pick operation for setting individual link colors. To pick a link, position the mouse arrow over the desired link and click the second mouse button. To select the entire mechanism, simply pick while not pointing at any link.

The RGB, HSV, and CIE values may be modified by rotating the dials (or using the mouse). The color can also be modified by pointing at a location in the RGB bars or the CIE triangle and clicking the first button on the mouse.

BACKGROUND COLOR

The background color may be changed with this option.

The RGB, HSV, and CIE values may be modified by rotating the dials (or using the mouse). The color can also be modified by pointing at a location in the RGB bars or the CIE triangle and clicking the first button on the mouse.

MAIN VIEW OBJECT

One of the following objects may be selected for viewing in the main workstation window:

- Spatial Mechanism Model
- Light Sources (with the model viewing volume)
- RGB Color Cube
- RGB Slider Bars
- HSV Color Cone
- CIE Color Triangle

LIGHT COLOR

This function is used to modify the color of the 8 light sources. For this function to be active, a light source number 1-8 must be selected for editing, and the light source must be on (Ambient, Infinite, Point, or Spot).

The RGB, HSV, and CIE values may be modified by rotating the dials (or using the mouse). The color can also be modified by pointing at a location in the RGB bars or the CIE triangle and clicking the first button on the mouse.

LIGHT ORIENTATION

This function is used to modify the orientation of the 8 light sources. This function does not apply to ambient lights, because they do not have an orientation. The following values can be modified for each light source:

- Infinite: position (for viewing purposes only), direction
- Point: position
- Spot: position, direction, concentration exponent, spread angle

LIGHT TYPE

Each of the 8 light sources may be set to one of the following types:

- Ambient: color only; affects all objects uniformly
- Infinite: (also known as directional) color and direction
- Point: color and position
- Spot: color, position, direction, concentration exponent, and spread angle

AUX VIEW OBJECT

One of the following objects may be selected for viewing in the auxiliary workstation window:

- Spatial Mechanism Model
- Light Sources (with the model viewing volume)
- RGB Color Cube
- RGB Slider Bars
- HSV Color Cone
- CIE Color Triangle

Appendix C

PriSM Program Listing

Include file: chrom_coord.h

```
#define XR 0.625
#define YR 0.340
#define XG 0.280
#define YG 0.595
#define XB 0.155
#define YB 0.070
#define XW 0.283
#define YW 0.298
```

Include file: common.h

```
#define TRUE                    1
#define FALSE                   0
#define UNDEFINED               -1
#define ON                      1
#define OFF                     0
#define BEGINNING               0
#define PI                      4.0*atan(1.0)
#define MAXLEN                  120
#define MAXLINKS                10
#define MAXLIGHTS               8

#define WS_MAIN                 1
#define WS_AUX                  2
#define WS_DIALS                3
```

```
#define MAX_RED_Y                       0.75
#define MIN_RED_Y                       0.60
#define MAX_GREEN_Y                     0.45
#define MIN_GREEN_Y                     0.30
#define MAX_BLUE_Y                      0.15
#define MIN_BLUE_Y                      0.00

#define PART_VIEW                       0
#define FULL_VIEW                       1
#define SIDE_VIEW                       2

#define BACKGROUND                      0
#define RED                             1
#define GREEN                           2
#define BLUE                            3
#define YELLOW                          4
#define CYAN                            5
#define MAGENTA                         6
#define ORANGE                          7
#define WHITE                           8
#define BLACK                           9
#define MODEL_COLOR_INDEX               10
#define LIGHT_COLOR_INDEX               21

#define MODEL_DEPTH_CUE                 1

#define STRUCT_MAIN                     5
#define STRUCT_AUX                      6
#define STRUCT_OUTLINES                 10
#define STRUCT_VIEW_DIVIDERS            11
#define STRUCT_HELP_TEXT                12
#define STRUCT_COLOR_MODELS             20
#define STRUCT_RGB_CUBE                 21
#define STRUCT_RGB_BARS                 22
#define STRUCT_HSV_CONE                 23
#define STRUCT_CIE_TRIANGLE             24
#define STRUCT_DIALS                    30
#define STRUCT_SINGLE_DIAL              31
#define STRUCT_UNIT_CYLINDER            40
#define STRUCT_UNIT_CONE                41
#define STRUCT_UNIT_PRISM               42
#define STRUCT_UNIT_HALF_SPHERE         43
#define STRUCT_UNIT_1_8TH_SPHERE        44
#define STRUCT_UNIT_CIRCLE              45
#define STRUCT_UNIT_FRUSTUM             46
#define STRUCT_LIGHT_SOURCES            50
#define STRUCT_LIGHT_1                  51
#define STRUCT_LIGHT_2                  52
#define STRUCT_LIGHT_3                  53
#define STRUCT_LIGHT_4                  54
#define STRUCT_LIGHT_5                  55
#define STRUCT_LIGHT_6                  56
#define STRUCT_LIGHT_7                  57
#define STRUCT_LIGHT_8                  58
#define STRUCT_INFINITE_LIGHT           60
#define STRUCT_POINT_LIGHT              61
#define STRUCT_SPOT_LIGHT               62
#define STRUCT_LIGHT_RAY                63
#define STRUCT_AXES                     70
#define STRUCT_AXIS_ARROW               71
#define STRUCT_AXIS_CONE                72
#define STRUCT_AXIS_CYLINDER            73
#define STRUCT_VIEW_LINES               80
#define STRUCT_MECHANISM                100
#define STRUCT_ID_FREE                  1000
#define STRUCT_ELEMENTS                 2000
```

```
#define LABEL_DIAL_ATTR_NAME            1
#define LABEL_DIAL_ATTR_VALUE           2
#define LABEL_DIAL_ROTATION             3
#define LABEL_RGB_FACES                 4
#define LABEL_RED_BAR                   5
#define LABEL_GREEN_BAR                 6
#define LABEL_BLUE_BAR                  7
#define LABEL_HSV_TRIANGLE              8
#define LABEL_HSV_MARKER                9
#define LABEL_CIE_TRIANGLE              10
#define LABEL_CIE_MARKER                11
#define LABEL_GLOBAL_TRANSFORMATION     12
#define LABEL_SURFACE_PROPERTIES        13
#define LABEL_STRUCT_MECHANISM          14
#define LABEL_SHADING_METHOD            15
#define LABEL_LIGHT_SOURCE_STATE        16
#define LABEL_LOCAL_TRANSFORMATION      17
#define LABEL_HELP_TEXT                 18
#define LABEL_VIEW_LINES                19

#define VIEW_DEFAULT                    0
#define VIEW_RGB_CUBE                   1
#define VIEW_RGB_BARS                   2
#define VIEW_HSV_CONE                   3
#define VIEW_CIE_TRIANGLE               4
#define VIEW_AXES_MAIN                  5
#define VIEW_AXES_TOP                   6
#define VIEW_AXES_FRONT                 7
#define VIEW_AXES_SIDE                  8
#define VIEW_MODEL_MAIN                 9
#define VIEW_MODEL_TOP                  10
#define VIEW_MODEL_FRONT                11
#define VIEW_MODEL_SIDE                 12
#define VIEW_MAIN                       13
#define VIEW_AUX                        14
#define VIEW_LIGHTS                     15
#define VIEW_LINES                      16

#define VIEW_DIALS                      1

typedef struct {
    Pcobundl rgb;
    Pcobundl hsv;
    Pcobundl cie;
} PSM_color_models;

extern int struct_id_free;
extern int DEBUG;
```

Include file: input.h

```
#define PET_TEXT_CYCLE                  1
#define PET_STATIC_MENU                 3
#define PET_BUTTON                      -1
#define PET_CHECK_BOX                   -2
#define PET_PICK                        1
#define PET_CURSOR                      1
```

```
#define DEV_CHOICE_MODEL_SELECTION      0
#define DEV_CHOICE_RESET_MODEL          1
#define DEV_CHOICE_ANIMATE_MODEL        2
#define DEV_CHOICE_IMAGE_READBACK       3
#define DEV_CHOICE_DISPLAY_TYPE         4
#define DEV_CHOICE_DEPTH_CUEING         5
#define DEV_CHOICE_SHADING_SELECTION    6
#define DEV_CHOICE_MULTI_VIEW_DISPLAY   7
#define DEV_CHOICE_VIEW_SIZE            8
#define DEV_CHOICE_VIEW_PROJECTION      9
#define DEV_CHOICE_DIAL_SELECTION       10
#define DEV_CHOICE_MAIN_VIEW_OBJECT     11
#define DEV_CHOICE_HELP                 14

#define DEV_CHOICE_EDIT_LIGHT_SOURCE    0
#define DEV_CHOICE_LIGHT_1              1
#define DEV_CHOICE_LIGHT_2              2
#define DEV_CHOICE_LIGHT_3              3
#define DEV_CHOICE_LIGHT_4              4
#define DEV_CHOICE_LIGHT_5              5
#define DEV_CHOICE_LIGHT_6              6
#define DEV_CHOICE_LIGHT_7              7
#define DEV_CHOICE_LIGHT_8              8
#define DEV_CHOICE_AUX_VIEW_OBJECT      9

#define DEV_LOCATOR_POSITION            0
#define DEV_LOCATOR_DIAL                0
#define DEV_LOCATOR_DIAL_ANGLE          3

#define DEV_PICK_MODEL_COMPONENT        1

#define DEV_VALUATOR_MODEL_COMPLEXITY   0
#define DEV_VALUATOR_LIGHT_SOURCE       1

#define CHOICE_PSEUDO_COLOR_DB          0
#define CHOICE_TRUE_COLOR_DB            1

#define CHOICE_WIREFRAME                0
#define CHOICE_SHADING_FLAT             1
#define CHOICE_SHADING_SMOOTH           2

#define CHOICE_ROTATE_TRANSLATE_SCALE   0
#define CHOICE_MODEL_SURFACE_PROPERTIES 1
#define CHOICE_ANIMATE_VIEW_MAPPING     2
#define CHOICE_VIEW_ORIENTATION         3
#define CHOICE_DEPTH_CUE_PARAMETERS     4
#define CHOICE_MODEL_COMPONENT_COLORS   5
#define CHOICE_BACKGROUND_COLOR         6

#define CHOICE_LIGHT_COLOR              0
#define CHOICE_LIGHT_ORIENTATION        1

#define CHOICE_SPATIAL_MECHANISM_MODEL  0
#define CHOICE_LIGHT_SOURCES            1
#define CHOICE_RGB_COLOR_CUBE           2
#define CHOICE_RGB_SLIDER_BARS          3
#define CHOICE_HSV_COLOR_CONE           4
#define CHOICE_CIE_COLOR_TRIANGLE       5
#define CHOICE_NONE                     6
```

```
#define ACTION_NONE                     -2
#define ACTION_DIALS                    -1
#define ACTION_STARTUP                  0
#define ACTION_MODEL_COMPLEXITY         1
#define ACTION_MODEL_SELECTION          2
#define ACTION_RESET_MODEL              3
#define ACTION_ANIMATE_MODEL            4
#define ACTION_IMAGE_READBACK           5
#define ACTION_DISPLAY_TYPE             6
#define ACTION_DEPTH_CUEING             7
#define ACTION_SHADING_SELECTION        8
#define ACTION_MULTI_VIEW_DISPLAY       9
#define ACTION_VIEW_SIZE                10
#define ACTION_VIEW_PROJECTION          11
#define ACTION_ROTATE_TRANSLATE_SCALE   12
#define ACTION_MODEL_SURFACE_PROPERTIES 13
#define ACTION_ANIMATE_VIEW_MAPPING     14
#define ACTION_VIEW_ORIENTATION         15
#define ACTION_DEPTH_CUE_PARAMETERS     16
#define ACTION_MODEL_COMPONENT_COLORS   17
#define ACTION_BACKGROUND_COLOR         18
#define ACTION_MAIN_VIEW_OBJECT         19
#define ACTION_LIGHT_COLOR              20
#define ACTION_LIGHT_ORIENTATION        21
#define ACTION_LIGHT_TYPE               22
#define ACTION_AUX_VIEW_OBJECT          23

#define DIALS_ROTATE_TRANSLATE_SCALE    0
#define DIALS_MODEL_SURFACE_PROPERTIES  1
#define DIALS_ANIMATE_VIEW_MAPPING      2
#define DIALS_VIEW_ORIENTATION          3
#define DIALS_DEPTH_CUE_PARAMETERS      4
#define DIALS_MODEL_COMPONENT_COLORS    5
#define DIALS_BACKGROUND_COLOR          6
#define DIALS_LIGHT_COLOR               7
#define DIALS_INFINITE_LIGHT            8
#define DIALS_POINT_LIGHT               9
#define DIALS_SPOT_LIGHT                10

#define EXTERNAL_REVOLUTE_JOINT         1
#define INTERNAL_REVOLUTE_JOINT         2
#define CONNECTING_ELEMENT              3
#define MECHANISM_LINK                  4
#define EXTERNAL_PRISMATIC_JOINT        5
#define INTERNAL_PRISMATIC_JOINT        6
#define EXTERNAL_CYLINDRIC_JOINT        7
#define INTERNAL_CYLINDRIC_JOINT        8
#define EXTERNAL_SPHERIC_JOINT          9
#define INTERNAL_SPHERIC_JOINT          10
#define PILLOW_BLOCK                    11
```

Include file: lights.h

```
#define PLT_NONE 4

typedef struct {
    Ppoint3 position;
    Pvector3 direction;
    Pfloat exponent;
    Pfloat angle;
    Pfloat temp;
} PSM_light_orientation;
```

```
typedef struct {
    Pint on;
    Pint type;
    PSM_color_models color;
    PSM_light_orientation orient;
} PSM_light_representation;

typedef struct {
    Pint on[MAXLIGHTS];
    Pint off[MAXLIGHTS];
    Pintlst onlst;
    Pintlst offlst;
    PSM_light_representation source[MAXLIGHTS+1];
} PSM_light_sources;
```

Include file: model.h

```
#define MAXMODELS 6
char *model_names[MAXMODELS+1];
int num_models;

typedef struct {
    Pvector3 rotate;
    Pvector3 translate;
    Pvector3 scale;
} PSM_global_transformation;

typedef struct {
    Pfloat ambient;
    Pfloat diffuse;
    Pfloat specular;
    Pfloat exponent;
    Pfloat temp[2];
    PSM_color_models color;
} PSM_surface_properties;

typedef struct {
    Pfloat step;
    Pfloat zoom_model;
    Pfloat zoom_lights;
    Ppoint3 prp;
    Pfloat front_plane;
    Pfloat view_plane;
    Pfloat back_plane;
    Pfloat fractional_step;
} PSM_animate_view_mapping;

typedef struct {
    Ppoint3 vrp;
    Pvector3 vpn;
    Pvector3 vup;
} PSM_view_orientation;

typedef struct {
    Pfloat planes[2];
    Pfloat temp;
    Pfloat scaling[2];
    Pfloat temp2;
    PSM_color_models color;
} PSM_depth_cueing;
```

```
typedef struct {
    Pint nsteps;
    Pint nlinks;
    Pchar *link_names[MAXLINKS+1];
    Ppoint3 center;
    Pfloat view_edge_len;
    Pint nsegments;
    Pint depth_cue_on;
    Pint shading_method;
    Pint multi_view;
    Pint view_location;
    Pint view_projection;
    PSM_global_transformation transformation;
    PSM_surface_properties surface;
    PSM_animate_view_mapping view_map;
    PSM_view_orientation view_orient;
    PSM_depth_cueing depth_cue;
    PSM_color_models link_colors[MAXLINKS+1];
    PSM_color_models background;
} PSM_model_rendering;
```

Module: prism.c

```
/*
 * PROGRAM:     PriSM - Postprocessing of Spatial Mechanisms
 *
 *      PriSM provides interactive model rendering and animation capabilities
 *      for spatial mechanism designs with revolute, cylindric, prismatic,
 *      and spheric joint types. The graphical input and output for this
 *      program is supported solely by the device independent PHIGS/PHIGS+
 *      graphics subroutine library.
 *
 * INPUT ARGUMENTS:
 *
 *      argc:   Number of command line arguments
 *      argv:   Command line argument value (only "d" for DEBUG mode is valid
 *              for this program)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

#include <stdio.h>
#include <phigs.h>
#include <signal.h>
#include "common.h"
#include "model.h"
#include "lights.h"

int DEBUG;
int struct_id_free;
PSM_model_rendering model;
PSM_light_sources lights;

int signal_handler ();

main (argc, argv)
int argc;
char *argv[];
{
    int ierr;
    char *s;
```

```
    DEBUG = FALSE;
    while (--argc>0 && (*++argv)[0]=='-')
        for (s=argv[0]+1; *s!='\0'; s++)
            switch (*s) {
            case 'd':
                DEBUG = TRUE;
                break;
            default:
                fprintf (stderr, "***prism: illegal option %c\n", *s);
                argc = 0;
            }

    psm_init (&ierr);
    if (ierr) {
        pemergencyclosephigs ();
        exit (1);
    }

    signal (SIGINT,signal_handler);

    psm_get_input ();

    punpostallstruct (WS_MAIN);
    punpostallstruct (WS_AUX);
    punpostallstruct (WS_DIALS);
    pclosews (WS_MAIN);
    pclosews (WS_AUX);
    pclosews (WS_DIALS);
    pclosephigs ();
}
```

```
/*
 * FUNCTION:    signal_handler
 *
 *      This function is the interrupt handler for PriSM. The user initiates
 *      the interrupt by typing CTRL-C in the window where PriSM was
 *      started.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

int signal_handler ()
{
    char inbuf[100];

    signal (SIGINT,SIG_IGN);
    printf ("\nInterrupt (y/n)? ");
    gets (inbuf);
    if (inbuf[0]=='y' || inbuf[0]=='Y') {
        pemergencyclosephigs ();
        exit (1);
    }
    signal (SIGINT,signal_handler);
}
```

Module: psm_color.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "chrom_coord.h"

float max (a,b)
float a, b;
{
    return ((a>b) ? a : b);
}

float min (a,b)
float a, b;
{
    return ((a<b) ? a : b);
}
```

```
/*
 * FUNCTION:    psm_rgb_to_hsv_cie
 *
 *      psm_rgb_to_hsv_cie converts an rgb color to hsv and cie by calling
 *      psm_rgb_to_hsv and psm_rgb_to_cie.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  &color->rgb is the input color
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  &color->hsv and &color->cie are the output colors that
 *              correspond to the rgb input
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_rgb_to_hsv_cie (color)
PSM_color_models *color;
{
    psm_rgb_to_hsv (&color->rgb,&color->hsv);
    psm_rgb_to_cie (&color->rgb,&color->cie);
}
```

```
/*
 * FUNCTION:    psm_rgb_to_hsv
 *
 *      psm_rgb_to_hsv converts an rgb color to the corresponding hsv color.
 *      Given: r,g,b each in [0,1]
 *      Desired: h in [0,360), s and v in [0,1],
 *               except if s=0, then h is UNDEFINED
 *      In this function, h is not set to UNDEFINED so that the last
 *      value of the hue will not be lost.
 *
 * INPUT ARGUMENTS:
 *
 *      rgb:    the input rgb color components
 *
 * OUTPUT ARGUMENTS:
 *
 *      hsv:    the corresponding hsv color components
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_rgb_to_hsv (rgb,hsv)
Pcobundl *rgb, *hsv;
{
    float maxrgb, minrgb, rc, gc, bc;

    maxrgb = max (max (rgb->x,rgb->y), rgb->z);
    minrgb = min (min (rgb->x,rgb->y), rgb->z);

    hsv->z = (maxrgb+minrgb)/2.0;

    if (maxrgb == minrgb) {
        hsv->y = 0.0;
/*        hsv->x = UNDEFINED;  */

    } else {
        if (hsv->z <= 0.5)
            hsv->y = (maxrgb-minrgb)/(maxrgb+minrgb);
        else
            hsv->y = (maxrgb-minrgb)/(2.0-maxrgb-minrgb);
        rc = (maxrgb-rgb->x)/(maxrgb-minrgb);
        gc = (maxrgb-rgb->y)/(maxrgb-minrgb);
        bc = (maxrgb-rgb->z)/(maxrgb-minrgb);

        if (rgb->x == maxrgb)
            hsv->x = bc-gc;
        else if (rgb->y == maxrgb)
            hsv->x = 2.0+rc-bc;
        else if (rgb->z == maxrgb)
            hsv->x = 4.0+gc-rc;
        hsv->x *= 60.0;
        if (hsv->x < 0.0) hsv->x += 360.0;
    }
}
```

```
/*
 * FUNCTION:    psm_hsv_to_rgb
 *
 *      psm_hsv_to_rgb converts an hsv color to the corresponding rgb color.
 *      Given: h in [0,360] or UNDEFINED, s and v in [0,1]
 *      Desired: r,g,b each in [0,1]
 *
 * INPUT ARGUMENTS:
 *
 *      hsv:    the input hsv color components
 *
 * OUTPUT ARGUMENTS:
 *
 *      rgb:    the corresponding rgb color components
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_hsv_to_rgb (hsv,rgb)
Pcobundl *hsv, *rgb;
{
    float m1, m2, psm_value ();

    if (hsv->z <= 0.5)
        m2 = hsv->z*(1.0+hsv->y);
    else
        m2 = hsv->z+hsv->y-(hsv->z * hsv->y);
    m1 = (2.0 * hsv->z)-m2;

    if (hsv->y == 0.0) {
        rgb->x = rgb->y = rgb->z = hsv->z;
    } else {
        rgb->x = psm_value (m1,m2,hsv->x+120.0);
        rgb->y = psm_value (m1,m2,hsv->x);
        rgb->z = psm_value (m1,m2,hsv->x-120.0);
    }
}
```

```
/*
 * FUNCTION:    psm_value
 *
 *      psm_value returns one of the component rgb values when converting
 *      from hsv to rgb.
 *
 * INPUT ARGUMENTS:
 *
 *      n1:
 *      n2:
 *      hue:    the hue of the hsv color specified in the range of 0.0 to
 *              360.0 degrees
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      The rgb color component value
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
float psm_value (n1,n2,hue)
float n1, n2, hue;
{
    if (hue > 360.0) hue -= 360.0;
    if (hue < 0.0) hue += 360.0;
    if (hue < 60.0)
        return (n1+(n2-n1)*hue/60.0);
    else if (hue < 180.0)
        return (n2);
    else if (hue < 240.0)
        return (n1+(n2-n1)*(240.0-hue)/60.0);
    else
        return (n1);
}
```

```
float c_rgbcie[3][3], c_ciergb[3][3];
float SLOPE_BR, SLOPE_BG, SLOPE_GR;

/*
 * FUNCTION:    psm_init_color_trans
 *
 *      This function initializes two matrices used in the color conversions
 *      from rgb to cie and vice versa.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_color_trans ()
{
    float d, cr, cg, cb;
    int i, j;

    d = XR*(YG-YB) + XG*(YB-YR) + XB*(YR-YG);
    cr = (XW*(YG-YB) - YW*(XG-XB) + XG*YB - XB*YG)/(YW*d);
    cg = (XW*(YB-YR) - YW*(XB-XR) - XR*YB + XB*YR)/(YW*d);
    cb = (XW*(YR-YG) - YW*(XR-XG) + XR*YG - XG*YR)/(YW*d);

    c_rgbcie[0][0] = XR*cr;
    c_rgbcie[0][1] = XG*cg;
    c_rgbcie[0][2] = XB*cb;
    c_rgbcie[1][0] = YR*cr;
    c_rgbcie[1][1] = YG*cg;
    c_rgbcie[1][2] = YB*cb;
    c_rgbcie[2][0] = (1.0-XR-YR)*cr;
    c_rgbcie[2][1] = (1.0-XG-YG)*cg;
    c_rgbcie[2][2] = (1.0-XB-YB)*cb;

    psm_matrix_inv (c_rgbcie,c_ciergb);

    SLOPE_BR = (YR-YB)/(XR-XB);
    SLOPE_BG = (YG-YB)/(XG-XB);
    SLOPE_GR = (YR-YG)/(XR-XG);
}
```

```
/*
 * FUNCTION:    psm_rgb_to_cie
 *
 *      psm_rgb_to_cie converts an rgb color to the corresponding cie color.
 *
 * INPUT ARGUMENTS:
 *
 *      rgb:    the input rgb color components
 *
 * OUTPUT ARGUMENTS:
 *
 *      cie:    the corresponding cie color components
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_rgb_to_cie (rgb,cie)
Pcobundl *rgb, *cie;
{
    float ciexyz[3], trgb[3], tot;
    int i, j;

    tot = 0.0;
    trgb[0] = rgb->x;
    trgb[1] = rgb->y;
    trgb[2] = rgb->z;

    for (i=0; i<3; i++) {
        ciexyz[i] = 0.0;
        for (j=0; j<3; j++)
            ciexyz[i] = ciexyz[i] + c_rgbcie[i][j]*trgb[j];
        tot += ciexyz[i];
    }

    if (tot != 0.00) {
        cie->x = ciexyz[0]/tot;
        cie->y = ciexyz[1]/tot;
    }
    cie->z = ciexyz[1];
}
```

```
/*
 * FUNCTION:    psm_cie_to_rgb
 *
 *        psm_cie_to_rgb converts a cie color to the corresponding rgb color.
 *
 * INPUT ARGUMENTS:
 *
 *      cie:    the input cie color components
 *
 * OUTPUT ARGUMENTS:
 *
 *      rgb:    the corresponding rgb color components
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_cie_to_rgb (cie,rgb)
Pcobundl *cie, *rgb;
{
    float ciexyz[3], trgb[3], maxrgb;
    int i, j;

    ciexyz[0] = cie->x * cie->z / cie->y;
    ciexyz[1] = cie->z;
    ciexyz[2] = (1.0 - cie->x - cie->y) * cie->z / cie->y;
    maxrgb = 0.0;

    for (i=0; i<3; i++) {
        trgb[i] = 0.0;
        for (j=0; j<3; j++)
            trgb[i] = trgb[i] + c_ciergb[i][j]*ciexyz[j];
        if (trgb[i] > maxrgb) maxrgb = trgb[i];
    }

    for (i=0; i<3; i++) {
        if (maxrgb > 1.0) trgb[i] = trgb[i]/maxrgb;
        if (trgb[i] < 0.0) trgb[i] = 0.0;
    }

    rgb->x = trgb[0];
    rgb->y = trgb[1];
    rgb->z = trgb[2];
}
```

```
/*
 * FUNCTION:    psm_matrix_inv
 *
 *      psm_matrix_inv performs a matrix inversion on a 3x3 matrix.
 *
 * INPUT ARGUMENTS:
 *
 *      mi:     the 3x3 matrix to be inverted
 *
 * OUTPUT ARGUMENTS:
 *
 *      mo:     the inverted 3x3 matrix
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_matrix_inv (mi,mo)
float mi[3][3], mo[3][3];
{
    int i, j, k;
    float mt[3][3], a;

    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) {
            mt[i][j] = mi[i][j];
            mo[i][j] = 0.0;
        }
        mo[i][i] = 1.0;
    }
```

```
    for (k=0; k<2; k++)
        for (i=k+1; i<3; i++) {
            a = -mt[i][k] / mt[k][k];
            for (j=0; j<3; j++) {
                mt[i][j] = mt[k][j]*a + mt[i][j];
                mo[i][j] = mo[k][j]*a + mo[i][j];
            }
        }

    for (k=2; k>0; k--)
        for (i=k-1; i>=0; i--) {
            a = -mt[i][k] / mt[k][k];
            mt[i][k] = 0.0;
            for (j=2; j>=0; j--)
                mo[i][j] = mo[k][j]*a + mo[i][j];
        }

    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            mo[i][j] = mo[i][j] / mt[i][i];

}
```

```
/*
 * FUNCTION:    psm_update_color_values
 *
 *      This function updates the color values of the rgb, hsv, and cie
 *      color models after one component of one of the three models has
 *      been modified.
 *
 * INPUT ARGUMENTS:
 *
 *      n:      the number of the color component that was modified
 *              0 - red (rgb)
 *              1 - green (rgb)
 *              2 - blue (rgb)
 *              3 - hue (hsv)
 *              4 - saturation (hsv)
 *              5 - value (hsv)
 *              6 - x (cie)
 *              7 - y (cie)
 *              8 - luminance (cie)
 *      color:  contains all three color models (with the one modified
 *              component)
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  contains the updated color models
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_color_values (n,color)
int n;
PSM_color_models *color;
{
    if (n <= 2)
        psm_update_hsv_cie (n,color);
    else if (n <= 5)
        psm_update_rgb_cie (n,color);
    else
        psm_update_rgb_hsv (n,color);
}
```

```
/*
 * FUNCTION:    psm_update_hsv_cie
 *
 *      This function updates the hsv and cie color models from the values
 *      of the rgb color model.  Any out-of-range values for the rgb model
 *      are properly adjusted.
 *
 * INPUT ARGUMENTS:
 *
 *      n:      the number of the color component that was modified
 *              0 - red (rgb)
 *              1 - green (rgb)
 *              2 - blue (rgb)
 *      color:  contains all three color models (with the one modified
 *              component)
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  will contains the updated color models
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_hsv_cie (n,color)
int n;
PSM_color_models *color;
{
    Pcobundl cie;

    if (n<0 || 2<n) return;

    if (color->rgb.x < 0.0) color->rgb.x = 0.0;
    if (color->rgb.x > 1.0) color->rgb.x = 1.0;
    if (color->rgb.y < 0.0) color->rgb.y = 0.0;
    if (color->rgb.y > 1.0) color->rgb.y = 1.0;
    if (color->rgb.z < 0.0) color->rgb.z = 0.0;
    if (color->rgb.z > 1.0) color->rgb.z = 1.0;

    psm_rgb_to_hsv (&color->rgb,&color->hsv);
    psm_rgb_to_cie (&color->rgb,&cie);
    if (!((color->rgb.x+color->rgb.y+color->rgb.z) == 0.00)) {
        color->cie = cie;
    }
    color->cie.z = cie.z;
}
```

```
/*
 * FUNCTION:    psm_update_rgb_cie
 *
 *        This function updates the rgb and cie color models from the values
 *        of the hsv color model.  Any out-of-range values for the hsv model
 *        are properly adjusted.
 *
 * INPUT ARGUMENTS:
 *
 *      n:      the number of the color component that was modified
 *              3 - hue (hsv)
 *              4 - saturation (hsv)
 *              5 - value (hsv)
 *      color:  contains all three color models (with the one modified
 *              component)
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  will contains the updated color models
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_rgb_cie (n,color)
int n;
PSM_color_models *color;
{
    if (n<3 || 5<n) return;

    if (color->hsv.x < 0.0) color->hsv.x = fmod(color->hsv.x,360.0) + 360.0;
    if (color->hsv.x > 360.0) color->hsv.x = fmod(color->hsv.x,360.0);
    if (color->hsv.y < 0.0) color->hsv.y = 0.0;
    if (color->hsv.y > 1.0) color->hsv.y = 1.0;
    if (color->hsv.z < 0.0) color->hsv.z = 0.0;
    if (color->hsv.z > 1.0) color->hsv.z = 1.0;

    psm_hsv_to_rgb (&color->hsv,&color->rgb);
    psm_rgb_to_cie (&color->rgb,&color->cie);
}
```

```
/*
 * FUNCTION:    psm_update_rgb_hsv
 *
 *      This function updates the rgb and hsv color models from the values
 *      of the cie color model.  Any out-of-range values for the cie model
 *      are properly adjusted.
 *
 * INPUT ARGUMENTS:
 *
 *      n:      the number of the color component that was modified
 *              6 - x (cie)
 *              7 - y (cie)
 *              8 - luminance (cie)
 *      color:  contains all three color models (with the one modified
 *              component)
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  will contains the updated color models
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_rgb_hsv (n,color)
int n;
PSM_color_models *color;
{
    float x, y;

    if (n<6 || 8<n) return;

    x = color->cie.x;
    y = color->cie.y;

    if (n != 7) {
        if (x < XB) {
            x = XB;
            y = YB;
        } else if (x > XR) {
            x = XR;
            y = YR;
        } else if (y < (SLOPE_BR*(x-XB)+YB)) {
            y = SLOPE_BR*(x-XB)+YB;
        } else if (x < XG) {
            if (y > (SLOPE_BG*(x-XB)+YB))
                y = SLOPE_BG*(x-XB)+YB;
        } else {
            if (y > (SLOPE_GR*(x-XG)+YG))
                y = SLOPE_GR*(x-XG)+YG;
        }
```

```
    } else {
        if (y < YB) {
            x = XB;
            y = YB;
        } else if (y > YG) {
            x = XG;
            y = YG;
        } else if (x < ((y-YB)/SLOPE_BG+XB)) {
            x = (y-YB)/SLOPE_BG+XB;
        } else if (y < YR) {
            if (x > ((y-YB)/SLOPE_BR+XB))
                x = (y-YB)/SLOPE_BR+XB;
        } else {
            if (x > ((y-YG)/SLOPE_GR+XG))
                x = (y-YG)/SLOPE_GR+XG;
        }
    }

    if (n == 8) {
        if (color->cie.z < 0.0) color->cie.z = 0.0;
        if (color->cie.z > 1.0) color->cie.z = 1.0;
    }

    color->cie.x = x;
    color->cie.y = y;
    psm_cie_to_rgb (&color->cie,&color->rgb);
    psm_rgb_to_hsv (&color->rgb,&color->hsv);
}
```

```
Pfasdata3 fadata;

/*
 * FUNCTION:    psm_init_color_models
 *
 *      psm_init_color_models initializes the PHIGS structure that contains
 *      the rgb color cube, rgb color bars, hsv color cone, and cie color
 *      triangle.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_color_models ()
{
    static Pcobundl black_rep = {0.00,0.00,0.00};

    psetcolourrep (WS_MAIN,BLACK,&black_rep);
    psetcolourrep (WS_AUX,BLACK,&black_rep);
    psetcolourrep (WS_DIALS,BLACK,&black_rep);

    psm_init_rgb_cube ();
    psm_init_rgb_bars ();
    psm_init_hsv_cone ();
    psm_init_cie_triangle ();
```

```
    popenstruct (STRUCT_COLOR_MODELS);
    psetviewind (VIEW_RGB_CUBE);
    pexecutestruct (STRUCT_RGB_CUBE);
    psetviewind (VIEW_RGB_BARS);
    pexecutestruct (STRUCT_RGB_BARS);
    psetviewind (VIEW_HSV_CONE);
    pexecutestruct (STRUCT_HSV_CONE);
    psetviewind (VIEW_CIE_TRIANGLE);
    pexecutestruct (STRUCT_CIE_TRIANGLE);
    pclosestruct ();
    ppoststruct (WS_MAIN,STRUCT_COLOR_MODELS,0.0);
}
```

```
/*
 * FUNCTION:    psm_update_color_models
 *
 *      This function updates the 4 PHIGS graphical color models.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  contains the rgb, hsv, and cie color model data
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 if successful; 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_update_color_models (color)
PSM_color_models color;
{
    static PSM_color_models last_color = {0.00,0.00,0.00,
                                          0.00,0.00,0.00,
                                          0.00,0.00,0.00};

    if (last_color.rgb.x == color.rgb.x &&
        last_color.rgb.y == color.rgb.y &&
        last_color.rgb.z == color.rgb.z &&
        last_color.hsv.x == color.hsv.x &&
        last_color.hsv.y == color.hsv.y &&
        last_color.hsv.z == color.hsv.z &&
        last_color.cie.x == color.cie.x &&
        last_color.cie.y == color.cie.y &&
        last_color.cie.z == color.cie.z) return (0);

    psm_update_rgb_cube (color);
    psm_update_rgb_bars (color);
    psm_update_hsv_cone (color);
    psm_update_cie_triangle (color);

    last_color = color;
    return (1);
}
```

```
/*
 * FUNCTION:    psm_init_rgb_cube
 *
 *      psm_init_rgb_cube initializes the PHIGS structure for the rgb color
 *      cube.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_rgb_cube ()
{
    int i;
    Pfasvdata3 cube_face;
    Pptco3 face_ptco[4];
    static Pptco3 red_ptco =       {1.00,0.00,0.00, 1.00,0.00,0.00};
    static Pptco3 green_ptco =    {0.00,1.00,0.00, 0.00,1.00,0.00};
    static Pptco3 blue_ptco =     {0.00,0.00,1.00, 0.00,0.00,1.00};
    static Pptco3 cyan_ptco =     {0.00,1.00,1.00, 0.00,1.00,1.00};
    static Pptco3 magenta_ptco =  {1.00,0.00,1.00, 1.00,0.00,1.00};
    static Pptco3 yellow_ptco =   {1.00,1.00,0.00, 1.00,1.00,0.00};
    static Pptco3 black_ptco =    {0.00,0.00,0.00, 0.00,0.00,0.00};

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_RGB_CUBE);
    psethlhsrid (ON);
    psetintstyle (PSOLID);
    psetintshademethod (PSH_COLOUR);
    psetlinecolourind (WHITE);

    face_ptco[0] = black_ptco;
    face_ptco[1] = red_ptco;
    face_ptco[2] = yellow_ptco;
    face_ptco[3] = green_ptco;
    cube_face.ptco = face_ptco;
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    face_ptco[1] = blue_ptco;
    face_ptco[2] = cyan_ptco;
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    face_ptco[2] = magenta_ptco;
    face_ptco[3] = red_ptco;
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    for (i=0;i<7; i++)
        plabel (LABEL_RGB_FACES);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_update_rgb_cube
 *
 *      This function updates the PHIGS structure for the rgb color cube.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  contains the rgb color model data
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_rgb_cube (color)
PSM_color_models color;
{
    int i;
    Pfasvdata3 cube_face;
    Pptco3 face_ptco[4], rgb_ptco;
    Ppoint3 lnpt3[2];

    rgb_ptco.point.x = rgb_ptco.colour.x = color.rgb.x;
    rgb_ptco.point.y = rgb_ptco.colour.y = color.rgb.y;
    rgb_ptco.point.z = rgb_ptco.colour.z = color.rgb.z;

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_RGB_CUBE);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_RGB_FACES);

    for (i=0; i<4; i++)
        face_ptco[i] = rgb_ptco;

    face_ptco[1].point.z = face_ptco[1].colour.z = 0.00;
    face_ptco[2].point.y = face_ptco[2].colour.y = 0.00;
    face_ptco[2].point.z = face_ptco[2].colour.z = 0.00;
    face_ptco[3].point.y = face_ptco[3].colour.y = 0.00;
    cube_face.ptco = face_ptco;
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    face_ptco[1] = rgb_ptco;
    face_ptco[2] = rgb_ptco;
    face_ptco[1].point.x = face_ptco[1].colour.x = 0.00;
    face_ptco[2].point.x = face_ptco[2].colour.x = 0.00;
    face_ptco[2].point.y = face_ptco[2].colour.y = 0.00;
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    face_ptco[2] = rgb_ptco;
    face_ptco[3] = rgb_ptco;
    face_ptco[2].point.x = face_ptco[2].colour.x = 0.00;
    face_ptco[2].point.z = face_ptco[2].colour.z = 0.00;
    face_ptco[3].point.z = face_ptco[3].colour.z = 0.00;
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&cube_face);

    lnpt3[0] = face_ptco[0].point;
    lnpt3[1] = face_ptco[1].point;
    poffsetelemptr (1);
    ppolyline3 (2,lnpt3);
```

```
    lnpt3[1] = face_ptco[3].point;
    poffsetelemptr (1);
    ppolyline3 (2,lnpt3);

    lnpt3[1] = face_ptco[0].point;
    lnpt3[1].y = 0.00;
    poffsetelemptr (1);
    ppolyline3 (2,lnpt3);

    pclosestruct ();
}
```

```
Pfasvdata3 red_bar, green_bar, blue_bar;
Pptco3 red_ptco[4], green_ptco[4], blue_ptco[4];

/*
 * FUNCTION:    psm_init_rgb_bars
 *
 *      psm_init_rgb_bars initializes the PHIGS structure for the rgb color
 *      bars.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_rgb_bars ()
{
    int i;
    static Ppoint3 red_pts[5] =
                            {0.00,MAX_RED_Y,0.00,   0.00,MIN_RED_Y,0.00,
                              1.00,MIN_RED_Y,0.00,   1.00,MAX_RED_Y,0.00,
                              0.00,MAX_RED_Y,0.00};
    static Ppoint3 green_pts[5] =
                            {0.00,MAX_GREEN_Y,0.00, 0.00,MIN_GREEN_Y,0.00,
                               1.00,MIN_GREEN_Y,0.00, 1.00,MAX_GREEN_Y,0.00,
                             0.00,MAX_GREEN_Y,0.00};
    static Ppoint3 blue_pts[5] =
                            {0.00,MAX_BLUE_Y,0.00,  0.00,MIN_BLUE_Y,0.00,
                               1.00,MIN_BLUE_Y,0.00,  1.00,MAX_BLUE_Y,0.00,
                             0.00,MAX_BLUE_Y,0.00};
    static Pcobundl black = {0.00,0.00,0.00};

    for (i=0; i<4; i++) {
        red_ptco[i].point = red_pts[i];
        red_ptco[i].colour = black;
        green_ptco[i].point = green_pts[i];
        green_ptco[i].colour = black;
        blue_ptco[i].point = blue_pts[i];
        blue_ptco[i].colour = black;
    }
    red_bar.ptco = red_ptco;
    green_bar.ptco = green_ptco;
    blue_bar.ptco = blue_ptco;
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_RGB_BARS);
    psetintstyle (PSOLID);
    psetintshademethod (PSH_COLOUR);
    psetlinecolourind (WHITE);

    plabel (LABEL_RED_BAR);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&red_bar);
    ppolyline3 (5,red_pts);
    plabel (LABEL_GREEN_BAR);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&green_bar);
    ppolyline3 (5,green_pts);
    plabel (LABEL_BLUE_BAR);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&blue_bar);
    ppolyline3 (5,blue_pts);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_update_rgb_bars
 *
 *      This function updates the PHIGS structure for the rgb color bars.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  contains the rgb color model data
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_rgb_bars (color)
PSM_color_models color;
{
    red_bar.ptco[2].point.x   = red_bar.ptco[2].colour.x   = color.rgb.x;
    red_bar.ptco[3].point.x   = red_bar.ptco[3].colour.x   = color.rgb.x;
    green_bar.ptco[2].point.x = green_bar.ptco[2].colour.y = color.rgb.y;
    green_bar.ptco[3].point.x = green_bar.ptco[3].colour.y = color.rgb.y;
    blue_bar.ptco[2].point.x  = blue_bar.ptco[2].colour.z  = color.rgb.z;
    blue_bar.ptco[3].point.x  = blue_bar.ptco[3].colour.z  = color.rgb.z;

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_RGB_BARS);
    psetelemptr (BEGINNING);

    psetelemptrlabel (LABEL_RED_BAR);
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&red_bar);
    psetelemptrlabel (LABEL_GREEN_BAR);
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&green_bar);
    psetelemptrlabel (LABEL_BLUE_BAR);
    poffsetelemptr (1);
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&blue_bar);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_init_hsv_cone
 *
 *      psm_init_rgb_bars initializes the PHIGS structure for the hsv color
 *      cone.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_hsv_cone ()
{
    int i;
    Ppoint3 hsv_pt[19];
    Pcobundl hsv_co[19];
    Pcobundl hsv;
    float phi;

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_HSV_CONE);
    psethlhsrid (ON);
    psetintshademethod (PSH_COLOUR);

    psetmarkertype (PMK_O);
    psetmarkercolourind (WHITE);

    /*  CODE FOR WORKING HIDDEN LINE ELIMINATION --

    hsv_pt[0].x = 0.00; hsv_pt[0].y = 2.00; hsv_pt[0].z = 0.00;
    hsv.x = 0.00; hsv.y = 1.00; hsv.z = 1.00;
    psm_hsv_to_rgb (&hsv,&hsv_co[0]);

    hsv_pt[8].x = 0.00; hsv_pt[8].y = -2.00; hsv_pt[8].z = 0.00;
    hsv.x = 0.00; hsv.y = 0.00; hsv.z = 0.00;
    psm_hsv_to_rgb (&hsv,&hsv_co[8]);

    hsv.y = 1.00; hsv.z = 0.50;
    for (i=0; i<6; i++) {
        phi = PI*i/3.0;
        hsv_pt[1].x = cos(phi);
        hsv_pt[1].y = 0.00;
        hsv_pt[1].z = -sin(phi);
        hsv.x = 60.0*i;
        psm_hsv_to_rgb (&hsv,&hsv_co[1]);

        for (j=0; j<6; j++) {
            phi += PI/18.0;
            hsv_pt[j+2].x = cos(phi);
            hsv_pt[j+2].y = 0.00;
            hsv_pt[j+2].z = -sin(phi);
            hsv.x += 10.0;
            psm_hsv_to_rgb (&hsv,&hsv_co[j+2]);
        }
        ppolyline3data (9,PVERT_COLOUR,hsv_pt,hsv_co);
    }
```

```
plabel (LABEL_HSV_TRIANGLE);
psetintstyle (PSOLID);
plabel (LABEL_HSV_TRIANGLE);
psetintstyle (PHOLLOW);
plabel (LABEL_HSV_TRIANGLE);
plabel (LABEL_HSV_MARKER);

*/

/*  CODE FOR TEMPORARY HIDDEN LINE ELIMINATION  */

hsv_pt[0].x = 1.00; hsv_pt[0].y = 0.00; hsv_pt[0].z = 0.00;
hsv.x = 0.00; hsv.y = 1.00; hsv.z = 0.50;
psm_hsv_to_rgb (&hsv,&hsv_co[0]);
for (i=1; i<=18; i++) {
    phi = PI*i/18.0;
    hsv_pt[i].x = cos(phi);
    hsv_pt[i].y = 0.00;
    hsv_pt[i].z = -sin(phi);
    hsv.x = 10.0*i;
    psm_hsv_to_rgb (&hsv,&hsv_co[i]);
}
ppolyline3data (19,PVERT_COLOUR,hsv_pt,hsv_co);

hsv_pt[0].x = 0.00; hsv_pt[0].y = 2.00; hsv_pt[0].z = 0.00;
hsv.x = 0.00; hsv.y = 1.00; hsv.z = 1.00;
psm_hsv_to_rgb (&hsv,&hsv_co[0]);
hsv_pt[1].y = 0.00;
hsv.y = 1.00; hsv.z = 0.50;
for (i=1; i<3; i++) {
    phi = PI*i/3.0;
    hsv_pt[1].x = cos(phi);
    hsv_pt[1].z = -sin(phi);
    hsv.x = 60.0*i;
    psm_hsv_to_rgb (&hsv,&hsv_co[1]);
    ppolyline3data (2,PVERT_COLOUR,hsv_pt,hsv_co);
}

hsv_pt[0].x = 0.00; hsv_pt[0].y = -2.00; hsv_pt[0].z = 0.00;
hsv.x = 0.00; hsv.y = 0.00; hsv.z = 0.00;
psm_hsv_to_rgb (&hsv,&hsv_co[0]);
hsv_pt[1].y = 0.00;
hsv.y = 1.00; hsv.z = 0.50;
for (i=0; i<4; i++) {
    phi = PI*i/3.0;
    hsv_pt[1].x = cos(phi);
    hsv_pt[1].z = -sin(phi);
    hsv.x = 60.0*i;
    psm_hsv_to_rgb (&hsv,&hsv_co[1]);
    ppolyline3data (2,PVERT_COLOUR,hsv_pt,hsv_co);
}

plabel (LABEL_HSV_TRIANGLE);
psetintstyle (PSOLID);
plabel (LABEL_HSV_TRIANGLE);
psetintstyle (PHOLLOW);
plabel (LABEL_HSV_TRIANGLE);
plabel (LABEL_HSV_MARKER);

hsv_pt[0].x = -1.00; hsv_pt[0].y = 0.00; hsv_pt[0].z = 0.00;
hsv.x = 180.00; hsv.y = 1.00; hsv.z = 0.50;
psm_hsv_to_rgb (&hsv,&hsv_co[0]);
for (i=1; i<=18; i++) {
    phi = PI*(1.0+i/18.0);
    hsv_pt[i].x = cos(phi);
    hsv_pt[i].y = 0.00;
    hsv_pt[i].z = -sin(phi);
    hsv.x = 180.0 + 10.0*i;
    psm_hsv_to_rgb (&hsv,&hsv_co[i]);
}
ppolyline3data (19,PVERT_COLOUR,hsv_pt,hsv_co);
```

```
    hsv_pt[0].x = 0.00; hsv_pt[0].y = 2.00; hsv_pt[0].z = 0.00;
    hsv.x = 0.00; hsv.y = 1.00; hsv.z = 1.00;
    psm_hsv_to_rgb (&hsv,&hsv_co[0]);
    hsv_pt[1].y = 0.00;
    hsv.y = 1.00; hsv.z = 0.50;
    for (i=3; i<7; i++) {
        phi = PI*i/3.0;
        hsv_pt[1].x = cos(phi);
        hsv_pt[1].z = -sin(phi);
        hsv.x = 60.0*i;
        psm_hsv_to_rgb (&hsv,&hsv_co[1]);
        ppolyline3data (2,PVERT_COLOUR,hsv_pt,hsv_co);
    }

    hsv_pt[0].x = 0.00; hsv_pt[0].y = -2.00; hsv_pt[0].z = 0.00;
    hsv.x = 0.00; hsv.y = 0.00; hsv.z = 0.00;
    psm_hsv_to_rgb (&hsv,&hsv_co[0]);
    hsv_pt[1].y = 0.00;
    hsv.y = 1.00; hsv.z = 0.50;
    for (i=4; i<6; i++) {
        phi = PI*i/3.0;
        hsv_pt[1].x = cos(phi);
        hsv_pt[1].z = -sin(phi);
        hsv.x = 60.0*i;
        psm_hsv_to_rgb (&hsv,&hsv_co[1]);
        ppolyline3data (2,PVERT_COLOUR,hsv_pt,hsv_co);
    }

    /*  END OF TEMPORARY HIDDEN LINE ELIMINATION  */

    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_update_hsv_cone
 *
 *      This function updates the PHIGS structure for the hsv color cone.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  contains the hsv color model data
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_hsv_cone (color)
PSM_color_models color;
{
    int i, j, k;
    Pfasvdata3 hsv_tri;
    Pptco3 hsv_ptco[5];
    Pfloat theta, val;
    Pcobundl hsv;
    Ppoint3 pt3;
```

```
    hsv = color.hsv;
    theta = hsv.x*PI/180;0;
    val = (hsv.z<0.5) ? hsv.z*2.0 : 2.0*(1.0-hsv.z);
    hsv_ptco[0].point.x = cos(theta)*val*hsv.y;
    hsv_ptco[0].point.y = 0.00;
    hsv_ptco[0].point.z = -sin(theta)*val*hsv.y;

    hsv.z = 0.5;
    psm_hsv_to_rgb (&hsv,&hsv_ptco[0].colour);
    hsv_ptco[1].point.x = hsv_ptco[1].point.y = hsv_ptco[1].point.z = 0.00;
    hsv_ptco[1].colour.x = hsv_ptco[1].colour.y = hsv_ptco[1].colour.z = 0.50;

    hsv.z = color.hsv.z;
    hsv_ptco[2].point.x = hsv_ptco[0].point.x;
    hsv_ptco[2].point.y = hsv.z*4.0 - 2.0;
    hsv_ptco[2].point.z = hsv_ptco[0].point.z;

    psm_hsv_to_rgb (&hsv,&hsv_ptco[2].colour);
    hsv_ptco[3].point.x = cos(theta)*val;
    hsv_ptco[3].point.y = hsv_ptco[2].point.y;
    hsv_ptco[3].point.z = -sin(theta)*val;

    hsv.y = 1.0;
    psm_hsv_to_rgb (&hsv,&hsv_ptco[3].colour);
    hsv_ptco[4].point.x = cos(theta);
    hsv_ptco[4].point.y = 0.00;
    hsv_ptco[4].point.z = -sin(theta);

    hsv.z = 0.5;
    psm_hsv_to_rgb (&hsv,&hsv_ptco[4].colour);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_HSV_CONE);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_HSV_TRIANGLE);
    poffsetelemptr (2);
    hsv_tri.ptco = hsv_ptco;
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,3,&hsv_tri);
    poffsetelemptr (2);
    hsv_tri.ptco = &hsv_ptco[1];
    pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,4,&hsv_tri);
    pt3 = hsv_ptco[2].point;
    poffsetelemptr (1);
    ppolymarker3 (1,pt3);
    pclosestruct ();
}

#define NDIV 20
#define NPTS 231
Pptco3 cie_ptco[NPTS];
```

```
/*
 * FUNCTION:    psm_init_cie_triangle
 *
 *      psm_init_cie_triangle initializes the PHIGS structure for the cie
 *      color triangle.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_cie_triangle ()
{
    int i, j, k, m, n;
    Pfasvdata3 cie_tri;
    Pptco3 tri_ptco[3];
    static Ppoint3 cie_tri_pts[4] = {XR,YR,0.00, XG,YG,0.00,
                                     XB,YB,0.00, XR,YR,0.00};
    Ppoint pnt, delq, delr;
    static Pcobundl black = {0.00,0.00,0.00};

    delq.x = (XR-XB)/NDIV;
    delq.y = (YR-YB)/NDIV;
    delr.x = (XG-XB)/NDIV;
    delr.y = (YG-YB)/NDIV;

    k = 0;
    for (i=0; i<=NDIV; i++) {
        pnt.x = XB + delr.x*i;
        pnt.y = YB + delr.y*i;
        for (j=0; j<=(NDIV-i); j++) {
            cie_ptco[k].point.x = pnt.x + delq.x*j;
            cie_ptco[k].point.y = pnt.y + delq.y*j;
            cie_ptco[k].point.z = 0.00;
            cie_ptco[k].colour = black;
            k++;
        }
    }

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_CIE_TRIANGLE);
    psetintstyle (PSOLID);
    psetintshademethod (PSH_COLOUR);
    psetmarkertype (PMK_O);
    psetlinecolourind (WHITE);
    plabel (LABEL_CIE_TRIANGLE);

    m = 0;
    for (i=0; i<NDIV; i++) {
        n = NDIV + 1 - i;
        for (j=0; j<(NDIV-i); j++) {
            tri_ptco[0] = cie_ptco[m];
            tri_ptco[1] = cie_ptco[m+1];
            tri_ptco[2] = cie_ptco[m+n];
            cie_tri.ptco = tri_ptco;
            pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,
                            3,&cie_tri);
```

```
            if ((m-n) > 0) {
                tri_ptco[2] = cie_ptco[m-n];
                cie_tri.ptco = tri_ptco;
                pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,
                                3,&cie_tri);
            }
            m++;
        }
        m++;
    }

    psetmarkercolourind (BLACK);
    plabel (LABEL_CIE_MARKER);
    ppolyline3 (4,cie_tri_pts);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_update_cie_triangle
 *
 *      This function updates the PHIGS structure for the cie color triangle.
 *
 * INPUT ARGUMENTS:
 *
 *      color:  contains the cie color model data
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_cie_triangle (color)
PSM_color_models color;
{
    int i, j, k, m, n;
    Pfasvdata3 cie_tri;
    Pptco3 tri_ptco[3];
    Ppoint pnt;

    k = 0;
    for (i=0; i<=NDIV; i++) {
        for (j=0; j<=(NDIV-i); j++) {
            cie_ptco[k].point.z = color.cie.z;
            psm_cie_to_rgb (&cie_ptco[k].point,&cie_ptco[k].colour);
            cie_ptco[k].point.z = 0.00;
            k++;
        }
    }

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_CIE_TRIANGLE);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_CIE_TRIANGLE);
```

```
    m = 0;
    for (i=0; i<NDIV; i++) {
        n = NDIV + 1 - i;
        for (j=0; j<(NDIV-i); j++) {
            tri_ptco[0] = cie_ptco[m];
            tri_ptco[1] = cie_ptco[m+1];
            tri_ptco[2] = cie_ptco[m+n];
            cie_tri.ptco = tri_ptco;
            poffsetelemptr (1);
            pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,
                            3,&cie_tri);

            if ((m-n) > 0) {
                tri_ptco[2] = cie_ptco[m-n];
                cie_tri.ptco = tri_ptco;
                poffsetelemptr (1);
                pfillarea3data (PFA_NONE,PVERT_COLOUR,PCM_RGB,&fadata,
                                3,&cie_tri);
            }
            m++;
        }
        m++;
    }

    poffsetelemptr (1);
    if (color.rgb.x+color.rgb.y+color.rgb.z > 2.2)
        psetmarkercolourind (BLACK);
    else
        psetmarkercolourind (WHITE);
    pnt.x = color.cie.x;
    pnt.y = color.cie.y;
    poffsetelemptr (1);
    ppolymarker (1,pnt);
    pclosestruct ();
}
```

Module: psm_dials.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "input.h"

Ppoint attr_name_loc = {0.00,-0.43};
Ppoint attr_value_loc = {0.00,0.00};

typedef struct {
    Pchar *names[9];
    Pfloat travel[9];
} PSM_attr_description;

PSM_attr_description attr[11]
                      = {"Rotate X","Rotate Y","Rotate Z",
                         "Translate X","Translate Y","Translate Z",
                         "Scale X","Scale Y","Scale Z",
                         360.0,360.0,360.0,
                          1.00, 1.00, 1.00,
                          1.00, 1.00, 1.00,
```

```
"Amb coeff","Diff coeff","Spec coeff",
"Specular exp"," "," ",
"Red","Green","Blue",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"Animation","Zoom Model","Zoom Lights",
"Proj Ref X","Proj Ref Y","Proj Ref Z",
"Front Plane","View Plane","Back Plane",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"View Ref X","View Ref Y","View Ref Z",
"View Norm X","View Norm Y","View Norm Z",
"View Up X","View Up Y","View Up Z",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"Front plane","Back plane"," ",
"Front scale","Back scale"," ",
"Red","Green","Blue",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"Red","Green","Blue",
"Hue","Saturation","Value",
"CIE x","CIE y","Luminance",
 1.00, 1.00, 1.00,
360.0, 1.00, 1.00,
 0.50, 0.50, 1.00,

"Red","Green","Blue",
"Hue","Saturation","Value",
"CIE x","CIE y","Luminance",
 1.00, 1.00, 1.00,
360.0, 1.00, 1.00,
 0.50, 0.50, 1.00,

"Red","Green","Blue",
"Hue","Saturation","Value",
"CIE x","CIE y","Luminance",
 1.00, 1.00, 1.00,
360.0, 1.00, 1.00,
 0.50, 0.50, 1.00,

"Position X","Position Y","Position Z",
"Direction X","Direction Y","Direction Z",
" "," "," ",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"Position X","Position Y","Position Z",
" "," "," ",
" "," "," ",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,

"Position X","Position Y","Position Z",
"Direction X","Direction Y","Direction Z",
"Exponent","Angle"," ",
 1.00, 1.00, 1.00,
 1.00, 1.00, 1.00,
 1.00,90.00, 1.00};
```

```
static Pint active[9];

Pchar *attr_labels[11] = {"Rotate - Translate - Scale",
                          "Model Surface Properties",
                          "Animate - View Mapping",
                          "View Orientation",
                          "Depth Cue Parameters",
                          "Model Component Colors",
                          "Background Color",
                          "Light Source Color",
                          "Infinite Light Orientation",
                          "Point Light Orientation",
                          "Spot Light Orientation"};
```

```
/*
 * FUNCTION:    psm_init_dials
 *
 *      psm_init_dials initializes the PHIGS structure that displays a
 *      simulated set a 9 dials.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_dials ()
{
    int i;
    Pint ierr;
    static Pcobundl background_color = {0.85,0.80,0.75};
    float theta;
    static float radius = 0.35;
    static Pcolour silver = {PCM_RGB, 0.60,0.60,0.65};
    static Pcolour dark_red = {PCM_RGB, 0.80,0.00,0.00};
    Ppoint dial[32];

    static Ppoint display[8] = { 0.28,-0.08,  0.31,-0.05,
                                 0.31, 0.05,  0.28, 0.08,
                                -0.28, 0.08, -0.31, 0.05,
                                -0.31,-0.05, -0.28,-0.08};
    static Ppoint outline[6] = { 0.02, 0.50,  0.02, 0.98,  0.98, 0.98,
                                 0.98, 0.07,  0.02, 0.07,  0.02, 0.50};
    static Ppoint rotation_marker[4] =          { 0.01, 0.00,  0.01, 0.36,
                                                 -0.01, 0.36, -0.01, 0.00};
    static Ppoint line[2] = {0.00,0.00, 0.00,0.36};
    Ptxalign text_align;
    Pfloat angle;
    Pmatrix3 rotation_matrix;

    psetcolourrep (WS_DIALS,BACKGROUND,&background_color);
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_SINGLE_DIAL);
    psetintstyle (PSOLID);
    psetindivcsf (PCO_INTERIOR,PDIRECT);
    psetintcolour (&silver);
    for (i=0; i<32; i++) {
        theta = PI*i/16.0;
        dial[i].x = radius*cos(theta);
        dial[i].y = radius*sin(theta);
    }
    pfillarea (32,dial);

    psetlinecolourind (WHITE);
    psetlinewidth (3.0);
    ppolyline (2,line);
    pclosestruct ();

    popenstruct (STRUCT_DIALS);
    psetviewind (VIEW_DEFAULT);
    psetindivcsf (PCO_POLYLINE,PDIRECT);
    psetlinecolour (&dark_red);
    psetlinewidth (5.0);
    ppolyline (6,outline);

    psetindivcsf (PCO_POLYLINE,PINDEXED);
    text_align.hor = PAH_CENTRE;
    text_align.ver = PAV_HALF;
    psettextalign (&text_align);
    psetintstyle (PSOLID);
    protatez (0.0,&ierr,rotation_matrix);
    if (ierr) printf ("psm_init_dials: rotate error = %d\n",ierr);

    for (i=0; i<9; i++) {
        psetviewind (VIEW_DIALS+i);
        plabel (LABEL_DIAL_ROTATION);
        psetlocaltran3 (rotation_matrix,PREPLACE);
        pexecutestruct (STRUCT_SINGLE_DIAL);
        psetlocaltran3 (rotation_matrix,PREPLACE);

        psetindivcsf (PCO_INTERIOR,PINDEXED);
        psetintcolourind (BLACK);
        pfillarea (8,display);

        psettextcolourind (BLACK);
        psetcharheight (0.075);
        plabel (LABEL_DIAL_ATTR_NAME);
        ptext (attr_name_loc," ");

        psettextcolourind (ORANGE);
        psetcharheight (0.085);
        plabel (LABEL_DIAL_ATTR_VALUE);
        ptext (attr_value_loc," ");
    }

    pclosestruct ();

    ppoststruct (WS_DIALS,STRUCT_DIALS,0.0);
    predrawallstruct (WS_DIALS,PALWAYS);
}
```

```
/*
 * FUNCTION:    psm_update_travel
 *
 *      This function updates the "travel" value of a dial that is used
 *      later in calculations when the dials are turned.
 *
 * INPUT ARGUMENTS:
 *
 *      ndef:   the number of the dials definition (0-10)
 *      ndial:  the number of the dial (0-8)
 *      travel: the travel value (must be positive)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_travel (ndef,ndial,travel)
Pint ndef, ndial;
Pfloat travel;
{
    if (travel <= 0.0) return;
    attr[ndef].travel[ndial] = travel;
}

static Pfloat last_values[9];
static Pfloat dial_values[9];
static Pint locator_dial;
static Pfloat dial_angle;
```

```
/*
 * FUNCTION:    psm_reinit_dials
 *
 *      psm_reinit_dials re-initializes the PHIGS structure that displays a
 *      simulated set a 9 dials with a new set of labels and values.
 *
 * INPUT ARGUMENTS:
 *
 *      ndef:   the number of the dials definition (0-10)
 *      attr_values:  the values of the attributes represented by dials 0-8
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_reinit_dials (ndef,attr_values)
int ndef;
Pfloat attr_values[9];
{
    Pvalrec dials_rec;
    static Plimit dials_area = {0.25,0.50,0.25,0.50};
    int i;
    Pmatrix3 rotation_matrix;
    Pint ierr;

    psetlocmode (WS_DIALS,DEV_LOCATOR_DIAL_ANGLE,PREQUEST,PES_ECHO);
    locator_dial = 99;

    dials_rec.valpet1_datarec.low = -100;
    dials_rec.valpet1_datarec.high = 100;
    dials_rec.valpet1_datarec.data = "";

    protatez (0.0,&ierr,rotation_matrix);
    if (ierr) printf ("psm_reinit_dials: rotate error = %d\n",ierr);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_DIALS);
    psetelemptr (BEGINNING);

    for (i=0; i<=8; i++) {
        psetelemptrlabel (LABEL_DIAL_ROTATION);
        poffsetelemptr (1);
        psetlocaltran3 (rotation_matrix,PREPLACE);
        psetelemptrlabel (LABEL_DIAL_ATTR_NAME);
        poffsetelemptr (1);
        ptext (attr_name_loc,attr[ndef].names[i]);
        psetvalmode (WS_MAIN,i+3,PREQUEST,PES_NOECHO);
        psetelemptrlabel (LABEL_DIAL_ATTR_VALUE);
        poffsetelemptr (1);

        if (strcmp(attr[ndef].names[i]," ") == 0) {
            active[i] = FALSE;
            ptext (attr_value_loc," ");
        } else {
            active[i] = TRUE;
            ptext (attr_value_loc,"999.999");
            last_values[i] = 99999.999;
        }

        dial_values[i] = 0.0;
        pinitval (WS_MAIN,i+3,dial_values[i],1,&dials_area,&dials_rec);
        psetvalmode (WS_MAIN,i+3,PEVENT,PES_NOECHO);
    }

    pclosestruct ();
    psm_update_dials (ndef,attr_values);
    pmessage (WS_DIALS,attr_labels[ndef]);
}
```

```
/*
 * FUNCTION:        psm_evaluate_dials
 *
 *      This function evaluates the new value of a dial based on input from
 *      either a locator event (mouse movement) or valuator event (dial
 *      rotation).
 *
 * INPUT ARGUMENTS:
 *
 *      event:    data structure identifying the workstation, device class,
 *              and device number of the device that triggered the input
 *              event.
 *      ndef:     the number of the dials definition (0-10)
 *      ndial:    the number of the dial (0-8)
 *
 * OUTPUT ARGUMENTS:
 *
 *      attr_values:  the updated values of the attributes represented by
 *                    dials 0-8
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_evaluate_dials (event,ndial,ndef,attr_values)
Pevent event;
int *ndial, ndef;
Pfloat attr_values[9];
{
    Ploc locator;
    Pfloat valuator, radius, angle, delta;

    if (event.class == PI_LOCATOR) {
        pgetloc (&locator);
        *ndial = locator.view_index-VIEW_DIALS;
        radius = sqrt(locator.position.x*locator.position.x +
                      locator.position.y*locator.position.y);

        if (event.dev == DEV_LOCATOR_DIAL) {
            if (locator_dial == 99) {
                if (radius >= 0.05 && radius <= 0.40) {
                    locator_dial = *ndial;
                    dial_angle = -acos(locator.position.x/radius);
                    if (locator.position.y != 0.00) dial_angle *=
                                locator.position.y/fabs(locator.position.y);
                    psetlocmode (WS_DIALS,DEV_LOCATOR_DIAL_ANGLE,
                                  PEVENT,PES_ECHO);
                }

            } else {
                psetlocmode (WS_DIALS,DEV_LOCATOR_DIAL_ANGLE,
                             PREQUEST,PES_ECHO);
                locator_dial = 99;
            }
```

```
        } else if (event.dev == DEV_LOCATOR_DIAL_ANGLE) {
            if (*ndial == locator_dial && radius >= 0.05 && radius <= 0.40) {
                angle = -acos(locator.position.x/radius);
                if (locator.position.y != 0.00) angle *=
                            locator.position.y/fabs(locator.position.y);
                if (fabs(angle-dial_angle) < PI)
                    delta = (angle-dial_angle)/(2.0*PI);
                else if (angle < 0.0)
                    delta = (2.0*PI+angle-dial_angle)/(2.0*PI);
                else
                    delta = (-2.0*PI+angle-dial_angle)/(2.0*PI);
                dial_angle = angle;
                attr_values[*ndial] += attr[ndef].travel[*ndial] * delta;

            } else {
                psetlocmode (WS_DIALS,DEV_LOCATOR_DIAL_ANGLE,
                             PREQUEST,PES_ECHO);
                locator_dial = 99;
            }
        }

    } else if (event.class == PI_VALUATOR) {
        pgetval (&valuator);
        *ndial = event.dev - 3;
        delta = valuator - dial_values[*ndial];
        dial_values[*ndial] = valuator;
        if (*ndial < 9)
            attr_values[*ndial] += attr[ndef].travel[*ndial] * delta;
    }
}
```

```
/*
 * FUNCTION:    psm_update_dials
 *
 *      psm_update_dials updates the PHIGS structure that simulates the
 *      dials.  The rotation angle and displayed value are updated for each
 *      dial that has a new attribute value.
 *
 * INPUT ARGUMENTS:
 *
 *      ndef:   the number of the dials definition (0-10)
 *      attr_values:  the values of the attributes represented by dials 0-8
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_update_dials (ndef,attr_values)
int ndef;
Pfloat attr_values[9];
{
    Pint ierr;
    int i, update;
    Pchar text_value[10];
    Pfloat angle;
    Pmatrix3 rotation_matrix;

    update = FALSE;
    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_DIALS);
    psetelemptr (BEGINNING);
```

```
    for (i=0; i<=8; i++) {
        psetelemptrlabel (LABEL_DIAL_ROTATION);
        if (active[i] && attr_values[i]!=last_values[i]) {
            poffsetelemptr (1);
            angle = -(fmod(attr_values[i],attr[ndef].travel[i])
                      /attr[ndef].travel[i]) * 2.0*PI;
            protatez (angle,&ierr,rotation_matrix);
            if (ierr) printf ("psm_update_dials: rotate error = %d\n",ierr);
            psetlocaltran3 (rotation_matrix,PREPLACE);
            psetelemptrlabel (LABEL_DIAL_ATTR_VALUE);
            poffsetelemptr (1);
            sprintf (text_value,"%7.3f",attr_values[i]);
            ptext (attr_value_loc,text_value);
            last_values[i] = attr_values[i];
            update = TRUE;
        }
    }

    pclosestruct ();
    if (update) predrawallstruct (WS_DIALS,PALWAYS);
}
```

Module: psm_help.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "input.h"

#define MAXLINES 250

static Pchar *help_text[MAXLINES];
static int len_help_text;
static Ppoint help_text_loc[10];

/*
 * FUNCTION:    psm_init_help
 *
 *      psm_init_help initializes the PHIGS structure that displays the
 *      help text.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_help ()
{
    static Pcolour gray = {PCM_RGB, 0.30,0.30,0.30};
    Ptxalign text_align;
    int i;

    psm_read_help_text ();
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_HELP_TEXT);
    psetviewind (VIEW_DEFAULT);
    psetintstyle (PSOLID);
    psetindivcsf (PCO_INTERIOR,PDIRECT);
    psetintcolour (&gray);
    psm_box (0.01,0.74,0.04,0.26);
    text_align.hor = PAH_LEFT;
    text_align.ver = PAV_BOTTOM;
    psettextalign (&text_align);
    psetcharheight (0.009);
    psettextcolourind (YELLOW);
    plabel (LABEL_HELP_TEXT);

    for (i=0; i<10; i++) {
        help_text_loc[i].x = 0.02;
        help_text_loc[i].y = 0.23 - (0.02*i);
        if (i < len_help_text) {
            ptext (help_text_loc[i],help_text[i]);
        } else
            ptext (help_text_loc[i]," ");
    }

    pclosestruct ();
    ppoststruct (WS_MAIN,STRUCT_HELP_TEXT,0.0);
}
```

```
/*
 * FUNCTION:    psm_read_help_text
 *
 *      psm_read_help_text reads the help text for PriSM from the file
 *      help.txt. Each section of the help text is expected to be 10 lines
 *      long, and the first line of each section corresponds to a text
 *      string that identifies the help topic.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_help_text ()
{
    FILE *fp, *fopen();
    char line[MAXLEN], *p;
    int len;

    len_help_text = 0;
    if ((fp=fopen("help.txt","r")) == NULL) {
        pmessage (WS_MAIN,"***CANNOT OPEN help.txt***");
        return;
```

```
    } else {
        while (len_help_text < MAXLINES && fgets (line,MAXLEN,fp) != NULL) {
            len = strlen (line);
            if (len == 1)
                line[0] = ' ';
            else
                line[len-1] = '\0';
            p = (char *) malloc (len);
            if (p == NULL) {
                break;
            } else {
                strcpy (p,line);
                help_text[len_help_text++] = p;
            }
        }
    }
}
```

```
/*
 * FUNCTION:    psm_display_help
 *
 *      psm_display_help searches for the identifying text string for the
 *      specified help topic in the help_text array. Once the help topic is
 *      found, the corresponding help text is used to replace the current
 *      text in the PHIGS help text structure.
 *
 * INPUT ARGUMENTS:
 *
 *      topic:  an integer constant that represents the desired help topic
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 if successful; 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_display_help (topic)
int topic;
{
    int lineno, i;
    Pchar search_text[MAXLEN];
    static int last_topic = 0;

    if (topic == last_topic || topic < 0)
        return (0);
    else
        last_topic = topic;
```

```
    if      (topic == ACTION_MODEL_COMPLEXITY)
        strcpy (search_text,"MODEL COMPLEXITY");
    else if (topic == ACTION_MODEL_SELECTION)
        strcpy (search_text,"MODEL SELECTION");
    else if (topic == ACTION_RESET_MODEL)
        strcpy (search_text,"RESET MODEL");
    else if (topic == ACTION_ANIMATE_MODEL)
        strcpy (search_text,"ANIMATE MODEL");
    else if (topic == ACTION_IMAGE_READBACK)
        strcpy (search_text,"IMAGE READBACK");
    else if (topic == ACTION_DISPLAY_TYPE)
        strcpy (search_text,"DISPLAY TYPE");
    else if (topic == ACTION_DEPTH_CUEING)
        strcpy (search_text,"DEPTH CUEING");
    else if (topic == ACTION_SHADING_SELECTION)
        strcpy (search_text,"SHADING SELECTION");
    else if (topic == ACTION_MULTI_VIEW_DISPLAY)
        strcpy (search_text,"MULTI-VIEW DISPLAY");
    else if (topic == ACTION_VIEW_SIZE)
        strcpy (search_text,"VIEW SIZE");
    else if (topic == ACTION_VIEW_PROJECTION)
        strcpy (search_text,"VIEW PROJECTION");
    else if (topic == ACTION_ROTATE_TRANSLATE_SCALE)
        strcpy (search_text,"ROTATE - TRANSLATE - SCALE");
    else if (topic == ACTION_MODEL_SURFACE_PROPERTIES)
        strcpy (search_text,"MODEL SURFACE PROPERTIES");
    else if (topic == ACTION_ANIMATE_VIEW_MAPPING)
        strcpy (search_text,"ANIMATE - VIEW MAPPING");
    else if (topic == ACTION_VIEW_ORIENTATION)
        strcpy (search_text,"VIEW ORIENTATION");
    else if (topic == ACTION_DEPTH_CUE_PARAMETERS)
        strcpy (search_text,"DEPTH CUE PARAMETERS");
    else if (topic == ACTION_MODEL_COMPONENT_COLORS)
        strcpy (search_text,"MODEL COMPONENT COLORS");
    else if (topic == ACTION_BACKGROUND_COLOR)
        strcpy (search_text,"BACKGROUND COLOR");
    else if (topic == ACTION_MAIN_VIEW_OBJECT)
        strcpy (search_text,"MAIN VIEW OBJECT");
    else if (topic == ACTION_LIGHT_COLOR)
        strcpy (search_text,"LIGHT COLOR");
    else if (topic == ACTION_LIGHT_ORIENTATION)
        strcpy (search_text,"LIGHT ORIENTATION");
    else if (topic == ACTION_LIGHT_TYPE)
        strcpy (search_text,"LIGHT TYPE");
    else if (topic == ACTION_AUX_VIEW_OBJECT)
        strcpy (search_text,"AUX VIEW OBJECT");
    else
        strcpy (search_text,"PriSM: Postprocessing of Spatial Mechanisms");

    lineno = 0;
    while (lineno < len_help_text &&
           strcmp (search_text,help_text[lineno]) != 0)
        lineno++;
    if (lineno == len_help_text)
        lineno = 0;

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_HELP_TEXT);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_HELP_TEXT);
    for (i=0; i<10; i++) {
        poffsetelemptr (1);
        if (lineno+i < len_help_text) {
            ptext (help_text_loc[i],help_text[lineno+i]);
        } else
            ptext (help_text_loc[i]," ");
    }
    pclosestruct ();
    return (1);
}
```

Module: psm_init.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "model.h"
#include "lights.h"
#include "input.h"

extern PSM_model_rendering model;
extern PSM_light_sources lights;

/*
 * FUNCTION:    psm_init
 *
 *      This function performs all startup initialization of PriSM.  It is
 *      logically done in two parts:
 *      1:  All initialization leading to the display of a single menu
 *          requesting the user to select a model or to exit.
 *      2:  After a model is selected, initialization is completed.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      ierr:    0 if no error; otherwise an error has occurred
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init (ierr)
int *ierr;
{
    Pevent event;
    Pchoice choice;
    int i;

    psm_init_ws_windows ();
    psm_init_color_trans ();
    psm_init_model ();
    psm_init_primitives ();
    psm_axes ();
    psm_init_lights ();
    psm_init_view_lines ();
    psm_init_help ();

    psm_init_models_list (ierr);
    if (*ierr) return;
    psm_init_model_menu ();
    pawaitevent (600.0,&event);
    if (event.class != PI_CHOICE) {
        *ierr = 1;
        return;
    }
    pgetchoice (&choice);
    if (choice.choice == num_models) {
        *ierr = 1;
        return;
    }
```

```
    psm_get_model (choice.choice);
    psm_reset_model ();
    psm_view_axes (WS_MAIN);

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_MAIN);
    psetviewind (VIEW_AXES_MAIN);
    pexecutestruct (STRUCT_AXES);
    psetviewind (VIEW_MODEL_MAIN);
    psetdcueind (MODEL_DEPTH_CUE);
    pexecutestruct (STRUCT_MECHANISM);
    pclosestruct ();

    psm_init_views ();
    psm_init_menus ();
    psm_init_view_dividers ();
    psm_init_color_models ();
    psm_init_dials ();
    psm_init_pick ();
    psm_init_locator ();

    ppoststruct (WS_MAIN,STRUCT_MAIN,1.0);
    ppoststruct (WS_AUX,STRUCT_AUX,1.0);
    predrawallstruct (WS_MAIN,PALWAYS);
}
```

```
/*
 * FUNCTION:    psm_init_ws_windows
 *
 *      This function initializes the 3 workstation windows used in PriSM.
 *      These workstation windows are WS_MAIN, WS_AUX, and WS_DIALS.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_ws_windows ()
{

    Puescapeinrec0002 window_location;
    Puescapeinrec0003 window_label;
    Pescapeout out;

    if (DEBUG) printf ("psm_init_ws_windows: opening phigs+\n");
    popenphigs ("/dev/tty",0);

    window_location.ws_xpos = 0;
    window_location.ws_ypos = 0;
    if (DEBUG) window_location.ws_ypos = 100;
    window_location.ws_wid  = 817;
    window_location.ws_hgt  = 992;
    if (DEBUG) window_location.ws_hgt  = 892;
    window_label.ws_flabel  = "PriSM - Postprocessing of Spatial Mechanisms";
    window_label.ws_showlabel = TRUE;
```

```
    pescape (puescape0002,&window_location,&out);
    pescape (puescape0003,&window_label,&out);

    popenws (WS_MAIN,NULL,PWST_OUTIN_TRUE);
    psethlhsrmode (WS_MAIN,ON);
    pmessage (WS_MAIN,"Please wait...");

    window_location.ws_xpos = 848;
    window_location.ws_ypos = 431;
    window_location.ws_wid  = 400;
    window_location.ws_hgt  = 561;
    window_label.ws_flabel  = "PriSM Auxiliary View";
    window_label.ws_showlabel = TRUE;

    pescape (puescape0002,&window_location,&out);
    pescape (puescape0003,&window_label,&out);

    popenws (WS_AUX,NULL,PWST_OUTIN_TRUE);
    psethlhsrmode (WS_AUX,ON);

    window_location.ws_xpos = 848;
    window_location.ws_ypos = 0;
    window_location.ws_wid  = 400;
    window_location.ws_hgt  = 400;
    window_label.ws_flabel  = "PriSM Dial Box Control";
    window_label.ws_showlabel = TRUE;

    pescape (puescape0002,&window_location,&out);
    pescape (puescape0003,&window_label,&out);

    popenws (WS_DIALS,NULL,PWST_OUTIN_TRUE);
}
```

```
/*
 * FUNCTION:    psm_init_models_list
 *
 *      psm_init_models_list reads the current directory to obtain a list
 *      of models.  A model requires a ".mod" and a ".pos" file to be
 *      completely specified.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      ierr:    0 if no error; otherwise an error has occurred
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_models_list (ierr)
int *ierr;
{
    FILE *fp, *fopen();
    char line1[MAXLEN], line2[MAXLEN], *p;
    static char *exit = {"EXIT PriSM"};
    int len1, len2, i;

    num_models = 0;
    *ierr = 0;
    system ("ls -1 *.mod *.pos > directory.lis");
```

```
    if ((fp=fopen("directory.lis","r")) == NULL) {
        pmessage (WS_MAIN,"***CANNOT OPEN directory.lis***");
        *ierr =  1;
        return;

    } else if (fgets (line1,MAXLEN,fp) != NULL) {
        len1 = strlen (line1);
        while (num_models<MAXMODELS && fgets (line2,MAXLEN,fp) != NULL) {
            len2 = strlen (line2);

            if (len1==len2 && len1>=6 &&
                strncmp (line1,line2,len1-5)==0 &&
                strcmp (&line1[len1-5],".mod\n")==0 &&
                strcmp (&line2[len2-5],".pos\n")==0) {
                p = (char *) malloc (len1-5);
                if (p==NULL) {
                    break;
                } else {
                    strncpy (p,line1,len1-5);
                    model_names[num_models++] = p;
                    if (fgets (line1,MAXLEN,fp) != NULL)
                        len1 = strlen (line1);
                    else
                        break;
                }

            } else {
                strcpy (line1,line2);
                len1 = len2;
            }
        }
    }

    system ("rm -f directory.lis");
    if (num_models==0) {
        pmessage (WS_MAIN,"***NO MODEL FILES IN CURRENT DIRECTORY");
        *ierr = 0;
        return;
    } else {
        model_names[num_models] = exit;
    }
}
```

```
/*
 * FUNCTION:    psm_init_model_menu
 *
 *      This function initializes the PHIGS input choice menu containing
 *      the list of available models.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_model_menu ()
{
    Pchoicerec model_menu_rec;
    static Plimit model_menu_area         = {0.00,0.15,1.00,1.00};
```

```
    model_menu_rec.choicepet3_datarec.number = num_models+1;
    model_menu_rec.choicepet3_datarec.data = "Models";
    model_menu_rec.choicepet3_datarec.strings = model_names;
    pinitchoice (WS_MAIN,DEV_CHOICE_MODEL_SELECTION,PCH_OK,0,PET_STATIC_MENU,
                 &model_menu_area,&model_menu_rec);
    psetchoicemode (WS_MAIN,DEV_CHOICE_MODEL_SELECTION,PEVENT,PES_ECHO);
    pmessage (WS_MAIN,"Waiting for model selection");
}
```

```
/*
 * FUNCTION:    psm_init_menus
 *
 *      This function initializes all the menus (except for the model menu)
 *      used in PriSM.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_menus ()
{
    int    i;
    static Pchar text[9] = {"Light ##"};

    Pchoicerec help_rec;
    Pvalrec    model_complexity_rec;
    Pchoicerec reset_model_rec;
    Pchoicerec animate_model_rec;
    Pchoicerec image_readback_rec;
    Pchoicerec display_type_rec;
    Pchoicerec depth_cueing_rec;
    Pchoicerec shading_menu_rec;
    Pchoicerec multi_view_display_rec;
    Pchoicerec view_size_rec;
    Pchoicerec view_projection_rec;
    Pchoicerec dial_menu_rec;
    Pchoicerec view_object_rec;

    Pvalrec       light_source_rec;
    Pchoicerec edit_light_source_rec;
    Pchoicerec light_rec;

    static Plimit help_area                   = {0.01,0.74,0.04,0.26};
    static Plimit model_complexity_area       = {0.15,0.45,0.95,1.00};
    static Plimit reset_model_area            = {0.15,0.30,0.91,0.95};
    static Plimit animate_model_area        = {0.15,0.30,0.87,0.91};
    static Plimit image_readback_area         = {0.15,0.30,0.83,0.87};
    static Plimit display_type_area           = {0.30,0.45,0.90,0.95};
    static Plimit depth_cueing_area           = {0.45,0.60,0.95,1.00};
    static Plimit shading_menu_area           = {0.465,0.585,0.85,0.95};
    static Plimit multi_view_display_area    = {0.60,0.75,0.95,1.00};
    static Plimit view_size_area            = {0.60,0.75,0.90,0.95};
    static Plimit view_projection_area        = {0.60,0.75,0.85,0.90};
    static Plimit dial_menu_area            = {0.80,1.00,1.00,1.00};
    static Plimit main_view_object_area       = {0.02,0.02,0.77,0.82};
```

```
static Plimit aux_view_object_area        = {0.00,0.00,0.76,0.84};
static Plimit light_source_area            = {0.00,0.46,0.92,1.00};
static Plimit edit_light_source_area     = {0.00,0.30,0.84,0.92};
static Plimit light_area[9]                = {0.00,0.00,0.00,0.00,
                                              0.46,0.64,0.92,1.00,
                                              0.64,0.82,0.92,1.00,
                                              0.82,1.00,0.92,1.00,
                                              0.46,0.64,0.84,0.92,
                                              0.64,0.82,0.84,0.92,
                                              0.82,1.00,0.84,0.92,
                                              0.55,0.73,0.76,0.84,
                                              0.73,0.91,0.76,0.84};

static Pchar *help_text[2] = {"ON","OFF"};
static Pchar *display_type_text[2] = {"Grayscale DB","True Color"};
static Pchar *shading_menu_text[3] = {"Wireframe","Flat","Smooth"};
static Pchar *panel_choice_text[2] = {"/panel_choice_off",
                                      "/panel_choice_on"};
static Pchar *view_projection_text[2] = {"Parallel","Perspective"};
static Pchar *dial_menu_text[7] = {"Rotate - Translate - Scale",
                                   "Model Surface Properties",
                                   "Animate - View Mapping",
                                   "View Orientation",
                                   "Depth Cue Parameters",
                                   "Model Component Colors",
                                   "Background Color"};

static Pchar *view_object_text[7] = {"Spatial Mechanism Model",
                                     "Light Sources",
                                     "RGB Color Cube",
                                     "RGB Slider Bars",
                                     "HSV Color Cone",
                                     "CIE Color Triangle",
                                     "None"};

static Pchar *edit_light_source_text[2] = {"Color","Orientation"};
static Pchar *light_text[5] = {"Ambient","Infinite","Point","Spot",
                               "None"};

help_rec.choicepet1_datarec.number = 2;
help_rec.choicepet1_datarec.data = "Help";
help_rec.choicepet1_datarec.strings = help_text;
pinitchoice (WS_MAIN,DEV_CHOICE_HELP,PCH_OK,0,PET_TEXT_CYCLE,
             &help_area,&help_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_HELP,PEVENT,PES_ECHO);

pinitchoice (WS_AUX,DEV_CHOICE_HELP,PCH_OK,0,PET_TEXT_CYCLE,
             &help_area,&help_rec);
psetchoicemode (WS_AUX,DEV_CHOICE_HELP,PEVENT,PES_ECHO);

model_complexity_rec.valpet1_datarec.low = 2;
model_complexity_rec.valpet1_datarec.high = 16;
model_complexity_rec.valpet1_datarec.data = "Model complexity";
pinitval (WS_MAIN,DEV_VALUATOR_MODEL_COMPLEXITY,4.0,1,
          &model_complexity_area,&model_complexity_rec);
psetvalmode (WS_MAIN,DEV_VALUATOR_MODEL_COMPLEXITY,PEVENT,PES_ECHO);

reset_model_rec.uchoicepet1_datarec.data = "Reset Model";
pinitchoice (WS_MAIN,DEV_CHOICE_RESET_MODEL,PCH_OK,0,PET_BUTTON,
             &reset_model_area,&reset_model_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_RESET_MODEL,PEVENT,PES_ECHO);

animate_model_rec.uchoicepet1_datarec.data = "Animate Model";
pinitchoice (WS_MAIN,DEV_CHOICE_ANIMATE_MODEL,PCH_OK,0,PET_BUTTON,
             &animate_model_area,&animate_model_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_ANIMATE_MODEL,PEVENT,PES_ECHO);

image_readback_rec.uchoicepet1_datarec.data = "PHIGS+ Image Readback";
pinitchoice (WS_MAIN,DEV_CHOICE_IMAGE_READBACK,PCH_OK,0,PET_BUTTON,
             &image_readback_area,&image_readback_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_IMAGE_READBACK,PEVENT,PES_ECHO);
```

```
display_type_rec.choicepet1_datarec.number = 2;
display_type_rec.choicepet1_datarec.data = "Display Type";
display_type_rec.choicepet1_datarec.strings = display_type_text;
pinitchoice (WS_MAIN,DEV_CHOICE_DISPLAY_TYPE,PCH_OK,CHOICE_TRUE_COLOR_DB,
             PET_TEXT_CYCLE,&display_type_area,&display_type_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_DISPLAY_TYPE,PEVENT,PES_ECHO);

depth_cueing_rec.choicepet1_datarec.number = 2;
depth_cueing_rec.choicepet1_datarec.data = "Depth Cueing";
depth_cueing_rec.choicepet1_datarec.strings = panel_choice_text;
pinitchoice (WS_MAIN,DEV_CHOICE_DEPTH_CUEING,PCH_OK,FALSE,PET_TEXT_CYCLE,
             &depth_cueing_area,&depth_cueing_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_DEPTH_CUEING,PEVENT,PES_ECHO);

shading_menu_rec.choicepet3_datarec.number = 3;
shading_menu_rec.choicepet3_datarec.data = "Shading";
shading_menu_rec.choicepet3_datarec.strings = shading_menu_text;
pinitchoice (WS_MAIN,DEV_CHOICE_SHADING_SELECTION,PCH_OK,
             CHOICE_SHADING_SMOOTH,PET_STATIC_MENU,&shading_menu_area,
             &shading_menu_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_SHADING_SELECTION,PEVENT,PES_ECHO);

multi_view_display_rec.choicepet1_datarec.number = 2;
multi_view_display_rec.choicepet1_datarec.data = "Multi-view Display";
multi_view_display_rec.choicepet1_datarec.strings = panel_choice_text;
pinitchoice (WS_MAIN,DEV_CHOICE_MULTI_VIEW_DISPLAY,PCH_OK,FALSE,
             PET_TEXT_CYCLE,&multi_view_display_area,
             &multi_view_display_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_MULTI_VIEW_DISPLAY,PEVENT,PES_ECHO);

view_size_rec.choicepet1_datarec.number = 2;
view_size_rec.choicepet1_datarec.data = "Full Window View";
view_size_rec.choicepet1_datarec.strings = panel_choice_text;
pinitchoice (WS_MAIN,DEV_CHOICE_VIEW_SIZE,PCH_OK,PART_VIEW,
             PET_TEXT_CYCLE,&view_size_area,&view_size_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_VIEW_SIZE,PEVENT,PES_ECHO);

view_projection_rec.choicepet1_datarec.number = 2;
view_projection_rec.choicepet1_datarec.data = "View Projection";
view_projection_rec.choicepet1_datarec.strings = view_projection_text;
pinitchoice (WS_MAIN,DEV_CHOICE_VIEW_PROJECTION,PCH_OK,PPARALLEL,
             PET_TEXT_CYCLE,&view_projection_area,&view_projection_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_VIEW_PROJECTION,PEVENT,PES_ECHO);

dial_menu_rec.choicepet3_datarec.number = 7;
dial_menu_rec.choicepet3_datarec.data = "Dial Box";
dial_menu_rec.choicepet3_datarec.strings = dial_menu_text;
pinitchoice (WS_MAIN,DEV_CHOICE_DIAL_SELECTION,PCH_OK,0,PET_STATIC_MENU,
             &dial_menu_area,&dial_menu_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_DIAL_SELECTION,PEVENT,PES_ECHO);

view_object_rec.choicepet1_datarec.number = 7;
view_object_rec.choicepet1_datarec.data = "Current View Selection";
view_object_rec.choicepet1_datarec.strings = view_object_text;
pinitchoice (WS_MAIN,DEV_CHOICE_MAIN_VIEW_OBJECT,PCH_OK,
             CHOICE_SPATIAL_MECHANISM_MODEL,PET_TEXT_CYCLE,
             &main_view_object_area,&view_object_rec);
psetchoicemode (WS_MAIN,DEV_CHOICE_MAIN_VIEW_OBJECT,PEVENT,PES_ECHO);

pinitchoice (WS_AUX,DEV_CHOICE_AUX_VIEW_OBJECT,PCH_OK,CHOICE_NONE,
             PET_TEXT_CYCLE,&aux_view_object_area,&view_object_rec);
psetchoicemode (WS_AUX,DEV_CHOICE_AUX_VIEW_OBJECT,PEVENT,PES_ECHO);

light_source_rec.valpet1_datarec.low = 0;
light_source_rec.valpet1_datarec.high = 8;
light_source_rec.valpet1_datarec.data = "Light source";
pinitval (WS_AUX,DEV_VALUATOR_LIGHT_SOURCE,0.0,1,
          &light_source_area,&light_source_rec);
psetvalmode (WS_AUX,DEV_VALUATOR_LIGHT_SOURCE,PEVENT,PES_ECHO);
```

```
    edit_light_source_rec.choicepet1_datarec.number = 2;
    edit_light_source_rec.choicepet1_datarec.data = "Edit Light Source";
    edit_light_source_rec.choicepet1_datarec.strings= edit_light_source_text;
    pinitchoice (WS_AUX,DEV_CHOICE_EDIT_LIGHT_SOURCE,PCH_OK,
                 CHOICE_LIGHT_COLOR,PET_TEXT_CYCLE,&edit_light_source_area,
                 &edit_light_source_rec);
    psetchoicemode (WS_AUX,DEV_CHOICE_EDIT_LIGHT_SOURCE,PEVENT,PES_ECHO);

    light_rec.choicepet1_datarec.number = 5;
    light_rec.choicepet1_datarec.strings = light_text;
    light_rec.choicepet1_datarec.data = text;
    text[7] = '1';
    pinitchoice (WS_AUX,DEV_CHOICE_LIGHT_1,PCH_OK,PLT_AMBIENT,
                 PET_TEXT_CYCLE,&light_area[1],&light_rec);
    psetchoicemode (WS_AUX,DEV_CHOICE_LIGHT_1,PEVENT,PES_ECHO);
    text[7] = '2';
    pinitchoice (WS_AUX,DEV_CHOICE_LIGHT_2,PCH_OK,PLT_INFINITE,
                 PET_TEXT_CYCLE,&light_area[2],&light_rec);
    psetchoicemode (WS_AUX,DEV_CHOICE_LIGHT_2,PEVENT,PES_ECHO);
    for (i=3; i<=8; i++) {
        sprintf (&text[7],"%1d",i);
        pinitchoice (WS_AUX,i,PCH_OK,PLT_NONE,PET_TEXT_CYCLE,
                     &light_area[i],&light_rec);
        psetchoicemode (WS_AUX,i,PEVENT,PES_ECHO);
    }
}
```

```
/*
 * FUNCTION:    psm_box
 *
 *      psm_box creates a 2-D box with a PHIGS fill area primitive.
 *
 * INPUT ARGUMENTS:
 *
 *      xmin:   minimum x coordinate
 *      xmax:   maximum x coordinate
 *      ymin:   minimum y coordinate
 *      ymax:   maximum y coordinate
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_box (xmin,xmax,ymin,ymax)
Pfloat xmin, xmax, ymin, ymax;
{
    Ppoint pts[4];

    pts[0].x = xmin; pts[0].y = ymin;
    pts[1].x = xmax; pts[1].y = ymin;
    pts[2].x = xmax; pts[2].y = ymax;
    pts[3].x = xmin; pts[3].y = ymax;
    pfillarea (4,pts);
}
```

```
/*
 * FUNCTION:    psm_init_view_dividers
 *
 *      psm_init_view_dividers creates the PHIGS structure containing the
 *      yellow dashed view dividers used in the multi-view display of the
 *      spatial mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_view_dividers ()
{
    Ppoint pts[2];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_OUTLINES);
    psetviewind (VIEW_DEFAULT);
    psetintstyle (PHOLLOW);
    psetintcolourind (CYAN);
    psm_box (0.01,0.74,0.27,1.00);
    psm_box (0.01,0.74,0.04,0.26);
    psm_box (0.75,0.99,0.04,1.00);
    pclosestruct ();
    ppoststruct (WS_MAIN,STRUCT_OUTLINES,2.0);

    popenstruct (STRUCT_VIEW_DIVIDERS);
    psetviewind (VIEW_LINES);
    psetlinetype (PLN_DASH);
    psetlinecolourind (YELLOW);
    pts[0].x = 0.50; pts[0].y = 0.00;
    pts[1].x = 0.50; pts[1].y = 1.00;
    ppolyline (2,pts);
    pts[0].x = 0.00; pts[0].y = 0.50;
    pts[1].x = 1.00; pts[1].y = 0.50;
    ppolyline (2,pts);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_init_pick
 *
 *      psm_init_pick initializes the middle mouse button as a pick input device
 *      to be used in picking links of the spatial mechanism model.  The
 *      mouse button is initialized in both the WS_MAIN and WS_AUX windows.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_pick ()
{
    Ppickpath path;
    Ppickpet0001 pick_rec;
    static Plimit3 echo_volume = {0.00,1.00,0.00,1.00,0.00,1.00};
    Pintlst inclusion, exclusion;
    Pint integers[1];

    path.depth = 0;
    pick_rec.aperture_left   = 5.0;
    pick_rec.aperture_right  = 5.0;
    pick_rec.aperture_top    = 5.0;
    pick_rec.aperture_bottom = 5.0;
    pick_rec.aperture_front  = 1.0;
    pick_rec.aperture_back   = 1.0;

    pinitpick3 (WS_MAIN,DEV_PICK_MODEL_COMPONENT,PP_OK,&path,PET_PICK,
                &echo_volume,&pick_rec,PBOTTOM_FIRST);
    pinitpick3 (WS_AUX,DEV_PICK_MODEL_COMPONENT,PP_OK,&path,PET_PICK,
                &echo_volume,&pick_rec,PBOTTOM_FIRST);
    psetpickmode (WS_MAIN,DEV_PICK_MODEL_COMPONENT,PEVENT,PES_ECHO);
    psetpickmode (WS_AUX,DEV_PICK_MODEL_COMPONENT,PEVENT,PES_ECHO);

    inclusion.number = 1;
    inclusion.integers = integers;
    exclusion.number = 0;
    exclusion.integers = integers;
    integers[0] = 1;
    psetpickfilter (WS_MAIN,DEV_PICK_MODEL_COMPONENT,&inclusion,&exclusion);
    psetpickfilter (WS_AUX,DEV_PICK_MODEL_COMPONENT,&inclusion,&exclusion);
}
```

```
/*
 * FUNCTION:    psm_init_locator
 *
 *      psm_init_locator initializes the left mouse button as a locator
 *      input device in all 3 windows.  In the WS_MAIN and WS_AUX windows,
 *      it is used to indicate locations in the rgb color bars and cie
 *      triangle.  In the WS_DIALS windows, it is used to indicate a location
 *      on one of the dials.  In addition, the mouse movement is initialized
 *      as a locator input device in the WS_DIALS window. This is used to
 *      determine the relative "rotation" of a dial.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_locator ()
{
    Ploc locator;
    static Plimit3 echo_volume = {0.00,1.00,0.00,1.00,0.00,1.00};
    Plocrec loc_rec;

    locator.view_index = VIEW_DEFAULT;
    pinitloc (WS_MAIN,DEV_LOCATOR_POSITION,&locator,PET_CURSOR,
              &echo_volume,&loc_rec);
    pinitloc (WS_AUX,DEV_LOCATOR_POSITION,&locator,PET_CURSOR,
              &echo_volume,&loc_rec);
    psetlocmode (WS_MAIN,DEV_LOCATOR_POSITION,PEVENT,PES_ECHO);
    psetlocmode (WS_AUX,DEV_LOCATOR_POSITION,PEVENT,PES_ECHO);

    pinitloc (WS_DIALS,DEV_LOCATOR_DIAL,&locator,PET_CURSOR,
              &echo_volume,&loc_rec);
    pinitloc (WS_DIALS,DEV_LOCATOR_DIAL_ANGLE,&locator,PET_CURSOR,
              &echo_volume,&loc_rec);
    psetlocmode (WS_DIALS,DEV_LOCATOR_DIAL,PEVENT,PES_ECHO);
}
```

Module: psm_input.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "model.h"
#include "lights.h"
#include "input.h"

extern PSM_model_rendering model;
extern PSM_light_sources lights;

float max (), min ();
```

```
struct {
    int main;
    int aux;
    int help;
    int model;
    int lights;
    int color;
} redraw;

struct {
    int dials;
    int dial_menu;
    int main_view;
    int aux_view;
    int light;
    int light_edit;
    int link;
} current;
```

```
/*
 * FUNCTION:    psm_get_input
 *
 *      psm_get_input is the main function for receiving and responding to
 *      interactive input.  Input is expected through menu choices (mouse),
 *      sliding valuators (mouse), picking model components (mouse),
 *      indicating location of color values (mouse), and turning valuators
 *      (dial box and simulated with mouse).
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_get_input ()
{
    int i, j, ndial, ncolor;
    char *s;

    Ppoint pts[4];

    Pchoice choice;
    Pevent event;
    Pfloat valuator;

    Pfloat dial_values[9], attr_values[9];

    Ppick pick;
    Ppickpath path;
    Ppickpathel pathel[1];
    static Plimit3 echo_volume = {0.00,1.00,0.00,1.00,0.00,1.00};
    Ploc locator;
    Plocrec loc_rec;
    static int animate = FALSE;
    static int help = TRUE;
    int action;

    Pint struct_id, view_index, view_location, link_id;
```

```
current.dials = DIALS_ROTATE_TRANSLATE_SCALE;
current.dial_menu = CHOICE_ROTATE_TRANSLATE_SCALE;
current.main_view = CHOICE_SPATIAL_MECHANISM_MODEL;
current.aux_view = CHOICE_NONE;
current.light = 0;
current.light_edit = CHOICE_LIGHT_COLOR;
current.link = MAXLINKS;

psm_reinit_dials (DIALS_ROTATE_TRANSLATE_SCALE,
                  (Pfloat *)&model.transformation);

pick.pick_path.pick_path = pathel;

while (TRUE) {

    redraw.main = redraw.aux = redraw.help = redraw.model =
    redraw.lights = redraw.color = FALSE;

    if (animate) {
        model.view_map.fractional_step += 1;
        model.view_map.step = model.view_map.fractional_step;
        redraw.model = psm_animate (&model.view_map.step);
        if (current.dials == DIALS_ANIMATE_VIEW_MAPPING)
            psm_update_dials (current.dials,(Pfloat *)&model.view_map);
        pawaitevent (0.0,&event);
    } else {
        pawaitevent (600.0,&event);
    }
    if (DEBUG) printf ("psm_get_input: WS=%d, DEV=%d, CLASS=%d, ",
                       event.ws,event.dev,event.class);

    action = ACTION_NONE;

    if (event.class==PI_PICK) {
        pgetpick (1,&pick);
        if (DEBUG) printf ("STAT=%d, DEPTH=%d\n",pick.status,
                           pick.pick_path.depth);

        if (current.dials == DIALS_MODEL_COMPONENT_COLORS) {
            if (pick.pick_path.depth >= 1) {
                if (DEBUG) printf ("STRUCT_ID=%d, PICK_ID=%d, ELEM=%d\n",
                        pick.pick_path.pick_path[0].struct_id,
                        pick.pick_path.pick_path[0].pick_id,
                        pick.pick_path.pick_path[0].el_num);
                link_id = pick.pick_path.pick_path[0].pick_id;
            } else {
                link_id = MAXLINKS;
            }
            pmessage (WS_MAIN,model.link_names[link_id]);
            if (link_id != current.link) {
                current.link = link_id;
                action = ACTION_MODEL_COMPONENT_COLORS;
            }
        }

    } else if (event.class==PI_CHOICE) {
        pgetchoice (&choice);
        if (DEBUG) printf ("STAT=%d, CHOICE=%d\n",choice.status,
                           choice.choice);


        if (event.dev == DEV_CHOICE_HELP) {
            if (help) {
                punpoststruct (WS_MAIN,STRUCT_HELP_TEXT);
                help = FALSE;
            } else {
                help = TRUE;
                ppoststruct (WS_MAIN,STRUCT_HELP_TEXT,0.0);
            }
            redraw.main = TRUE;
```

```
    } else if (event.ws == WS_MAIN) {
        if      (event.dev == DEV_CHOICE_MODEL_SELECTION)
                        action = ACTION_MODEL_SELECTION;
        else if (event.dev == DEV_CHOICE_RESET_MODEL)
                        action = ACTION_RESET_MODEL;
        else if (event.dev == DEV_CHOICE_ANIMATE_MODEL)
                        action = ACTION_ANIMATE_MODEL;
        else if (event.dev == DEV_CHOICE_IMAGE_READBACK)
                        action = ACTION_IMAGE_READBACK;
        else if (event.dev == DEV_CHOICE_DISPLAY_TYPE)
                        action = ACTION_DISPLAY_TYPE;
        else if (event.dev == DEV_CHOICE_DEPTH_CUEING)
                        action = ACTION_DEPTH_CUEING;
        else if (event.dev == DEV_CHOICE_SHADING_SELECTION)
                        action = ACTION_SHADING_SELECTION;
        else if (event.dev == DEV_CHOICE_MULTI_VIEW_DISPLAY)
                        action = ACTION_MULTI_VIEW_DISPLAY;
        else if (event.dev == DEV_CHOICE_VIEW_SIZE)
                        action = ACTION_VIEW_SIZE;
        else if (event.dev == DEV_CHOICE_VIEW_PROJECTION)
                        action = ACTION_VIEW_PROJECTION;
        else if (event.dev == DEV_CHOICE_MAIN_VIEW_OBJECT)
                        action = ACTION_MAIN_VIEW_OBJECT;
        else if (event.dev == DEV_CHOICE_DIAL_SELECTION) {
            current.dial_menu = current.dials = choice.choice;
            action = psm_set_dial_action ();
        }

    } else if (event.ws == WS_AUX) {
        if (event.dev == DEV_CHOICE_EDIT_LIGHT_SOURCE) {
            current.light_edit = choice.choice;
            psm_set_current_dials ();
            action = psm_set_dial_action ();
        } else if (event.dev == DEV_CHOICE_AUX_VIEW_OBJECT) {
            action = ACTION_AUX_VIEW_OBJECT;
        } else if (event.dev >= DEV_CHOICE_LIGHT_1 &&
                   event.dev <= DEV_CHOICE_LIGHT_8) {
            action = ACTION_LIGHT_TYPE;
        }
    }

} else if (event.class == PI_LOCATOR) {

    pgetloc (&locator);
    if (DEBUG) printf ("locator=%d %f %f \n",locator.view_index,
                       locator.position.x,locator.position.y);
    action = ACTION_DIALS;
    if (event.ws == WS_DIALS)
        ndial = locator.view_index - VIEW_DIALS;

} else if (event.class==PI_VALUATOR) {

    pgetval (&valuator);
    if (DEBUG) printf ("value=%f\n",valuator);

    if (event.dev < 3) {
        if (event.dev == DEV_VALUATOR_MODEL_COMPLEXITY) {
            action = ACTION_MODEL_COMPLEXITY;
        } else if (event.dev == DEV_VALUATOR_LIGHT_SOURCE) {
            current.light = valuator;
            psm_set_current_dials ();
            action = psm_set_dial_action ();
        }
    } else {
        action = ACTION_DIALS;
    }
}

redraw.help = psm_display_help (action);
```

```
if (action == ACTION_LIGHT_TYPE) {

    i = event.dev;
    lights.source[i].type = choice.choice;
    redraw.model = psm_set_light_source (i);
    psm_set_current_dials ();
    action = psm_set_dial_action ();

} else if (action == ACTION_RESET_MODEL) {
    psm_reset_model ();
    redraw.main = TRUE;
    redraw.aux = TRUE;
    pmessage (WS_MAIN,"Model reset completed");
    action = psm_set_dial_action ();
}

switch (action) {

case ACTION_MODEL_COMPLEXITY:

    if (valuator != model.nsegments) {
        pmessage (WS_MAIN,"Model complexity changed");
        model.nsegments = valuator;
        psm_init_primitives ();
        redraw.model = TRUE;
    }
    break;

case ACTION_MODEL_SELECTION:

    if (choice.choice == num_models) return (1);
    animate = FALSE;
    current.link = MAXLINKS;
    psm_get_model (choice.choice);
    psm_reset_model_view ();
    psm_model_view_mapping ();
    psm_model_view_orientation ();
    psm_view_lights (WS_MAIN);
    psm_view_lights (WS_AUX);
    redraw.model = TRUE;
    break;

case ACTION_ANIMATE_MODEL:

    if (animate) {
        pmessage (WS_MAIN,"Animation stopped");
        animate = FALSE;
    } else if (current.main_view == CHOICE_SPATIAL_MECHANISM_MODEL ||
               current.main_view == CHOICE_LIGHT_SOURCES ||
               current.aux_view  == CHOICE_SPATIAL_MECHANISM_MODEL ||
               current.aux_view  == CHOICE_LIGHT_SOURCES) {
        pmessage (WS_MAIN,"Animation started");
        animate = TRUE;
    }
    break;

case ACTION_IMAGE_READBACK:

    psm_image_readback ();
    break;

case ACTION_DISPLAY_TYPE:

    if (choice.choice == CHOICE_TRUE_COLOR_DB)
        pmessage (WS_MAIN,"True Color");
    else
        pmessage (WS_MAIN,"Grayscale Double Buffered");
    psm_change_display (choice.choice);
    redraw.main = TRUE;
    redraw.aux = TRUE;
    break;
```

```
case ACTION_DEPTH_CUEING:

    if (choice.choice)
        pmessage (WS_MAIN,"Depth Cueing On");
    else
        pmessage (WS_MAIN,"Depth Cueing Off");
    model.depth_cue_on = choice.choice;
    redraw.model = psm_depth_cueing ();
    psm_view_lines ();
    break;

case ACTION_SHADING_SELECTION:

    if (choice.choice == model.shading_method) {
        break;

    } else if (choice.choice == CHOICE_WIREFRAME) {
        pmessage (WS_MAIN,"Wireframe - no shading selected");
        model.shading_method = CHOICE_WIREFRAME;
        psm_init_primitives ();

    } else if (choice.choice == CHOICE_SHADING_FLAT) {
        pmessage (WS_MAIN,"Flat shading selected");
        if (model.shading_method == CHOICE_WIREFRAME) {
            model.shading_method = CHOICE_SHADING_FLAT;
            psm_init_primitives ();
        } else
            model.shading_method = CHOICE_SHADING_FLAT;
        psm_shading_method (CHOICE_SHADING_FLAT);

    } else if (choice.choice == CHOICE_SHADING_SMOOTH) {
        pmessage (WS_MAIN,"Smooth shading selected");
        if (model.shading_method == CHOICE_WIREFRAME) {
            model.shading_method = CHOICE_SHADING_SMOOTH;
            psm_init_primitives ();
        } else
            model.shading_method = CHOICE_SHADING_SMOOTH;
        psm_shading_method (CHOICE_SHADING_SMOOTH);
    }

    redraw.model = TRUE;
    break;

case ACTION_MAIN_VIEW_OBJECT:
case ACTION_AUX_VIEW_OBJECT:

    if (event.ws == WS_MAIN) {
        struct_id = STRUCT_MAIN;
        view_index = VIEW_MAIN;
        view_location = model.view_location;
        current.main_view = choice.choice;
        redraw.main = TRUE;
    } else if (event.ws == WS_AUX) {
        struct_id = STRUCT_AUX;
        view_index = VIEW_AUX;
        view_location = FULL_VIEW;
        current.aux_view = choice.choice;
        redraw.aux = TRUE;
    }

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    if (event.ws == WS_MAIN)
        punpoststruct (WS_MAIN,STRUCT_VIEW_DIVIDERS);
```

```
        if (choice.choice == CHOICE_SPATIAL_MECHANISM_MODEL) {
            pmessage (WS_MAIN,"Spatial Mechanism Model");
            if (model.multi_view && event.ws == WS_MAIN) {
                ppoststruct (WS_MAIN,STRUCT_VIEW_DIVIDERS,2.0);
                for (i=0; i<4; i++) {
                    psetviewind (VIEW_MODEL_MAIN+i);
                    if (i==0) psetdcueind (MODEL_DEPTH_CUE);
                    pexecutestruct (STRUCT_MECHANISM);
                    if (i==0) psetdcueind (OFF);
                    psetviewind (VIEW_AXES_MAIN+i);
                    pexecutestruct (STRUCT_AXES);
                }
            } else {
                psetviewind (VIEW_AXES_MAIN);
                pexecutestruct (STRUCT_AXES);
                psetviewind (VIEW_MODEL_MAIN);
                psetdcueind (MODEL_DEPTH_CUE);
                pexecutestruct (STRUCT_MECHANISM);
            }

        } else if (choice.choice == CHOICE_LIGHT_SOURCES) {
            pmessage (WS_MAIN,"Light Sources");
            psm_view_lights (event.ws);
            psetviewind (VIEW_LIGHTS);
            pexecutestruct (STRUCT_LIGHT_SOURCES);
            pexecutestruct (STRUCT_MECHANISM);
            pexecutestruct (STRUCT_VIEW_LINES);

        } else if (choice.choice == CHOICE_RGB_COLOR_CUBE) {
            pmessage (WS_MAIN,"RGB Color Cube");
            psm_view_rgb_cube (event.ws,view_index,view_location);
            psetviewind (view_index);
            pexecutestruct (STRUCT_RGB_CUBE);

        } else if (choice.choice == CHOICE_RGB_SLIDER_BARS) {
            pmessage (WS_MAIN,"RGB Slider Bars");
            psm_view_rgb_bars (event.ws,view_index,view_location);
            psetviewind (view_index);
            pexecutestruct (STRUCT_RGB_BARS);

        } else if (choice.choice == CHOICE_HSV_COLOR_CONE) {
            pmessage (WS_MAIN,"HSV Color Cone");
            psm_view_hsv_cone (event.ws,view_index,view_location);
            psetviewind (view_index);
            pexecutestruct (STRUCT_HSV_CONE);

        } else if (choice.choice == CHOICE_CIE_COLOR_TRIANGLE) {
            pmessage (WS_MAIN,"CIE Color Triangle");
            psm_view_cie_triangle (event.ws,view_index,view_location);
            psetviewind (view_index);
            pexecutestruct (STRUCT_CIE_TRIANGLE);
        }

        pclosestruct ();
        if (current.main_view!=CHOICE_SPATIAL_MECHANISM_MODEL &&
            current.main_view!=CHOICE_LIGHT_SOURCES &&
            current.aux_view!=CHOICE_SPATIAL_MECHANISM_MODEL &&
            current.aux_view!=CHOICE_LIGHT_SOURCES)
            animate = FALSE;

        break;

    case ACTION_MULTI_VIEW_DISPLAY:

        if (choice.choice)
            pmessage (WS_MAIN,"Multi-view Display");
        else
            pmessage (WS_MAIN,"Single View Display");
        model.multi_view = choice.choice;
        psm_view_model (WS_MAIN);
        psm_view_axes (WS_MAIN);
```

```
        if (current.main_view == CHOICE_SPATIAL_MECHANISM_MODEL) {
            pseteditmode (PEDIT_INSERT);
            popenstruct (STRUCT_MAIN);
            pemptystruct (STRUCT_MAIN);

            if (model.multi_view) {
                ppoststruct (WS_MAIN,STRUCT_VIEW_DIVIDERS,2.0);
                for (i=0; i<4; i++) {
                    psetviewind (VIEW_MODEL_MAIN+i);
                    if (i==0) psetdcueind (MODEL_DEPTH_CUE);
                    pexecutestruct (STRUCT_MECHANISM);
                    if (i==0) psetdcueind (OFF);
                    psetviewind (VIEW_AXES_MAIN+i);
                    pexecutestruct (STRUCT_AXES);
                    pexecutestruct (STRUCT_AXES);
                }

            } else {
                punpoststruct (WS_MAIN,STRUCT_VIEW_DIVIDERS);
                psetviewind (VIEW_AXES_MAIN);
                pexecutestruct (STRUCT_AXES);
                psetviewind (VIEW_MODEL_MAIN);
                psetdcueind (MODEL_DEPTH_CUE);
                pexecutestruct (STRUCT_MECHANISM);
            }

            pclosestruct ();
            redraw.main = TRUE;
        }
        break;

    case ACTION_VIEW_SIZE:

        model.view_location = choice.choice;
        if (model.view_location == FULL_VIEW) {
            pmessage (WS_MAIN,"Full Window View");
            punpoststruct (WS_MAIN,STRUCT_COLOR_MODELS);
            punpoststruct (WS_MAIN,STRUCT_OUTLINES);
        } else {
            pmessage (WS_MAIN,"Part Window View");
            ppoststruct (WS_MAIN,STRUCT_COLOR_MODELS,0.0);
            ppoststruct (WS_MAIN,STRUCT_OUTLINES,2.0);
        }

        psm_view_dividers (WS_MAIN,model.view_location);
        psm_view_model (WS_MAIN);
        psm_view_axes (WS_MAIN);
        psm_view_lights (WS_MAIN);

        if (current.main_view == CHOICE_LIGHT_SOURCES)
            psm_view_lights (WS_MAIN);
        else if (current.main_view == CHOICE_RGB_COLOR_CUBE)
            psm_view_rgb_cube (WS_MAIN,VIEW_MAIN,model.view_location);
        else if (current.main_view == CHOICE_RGB_SLIDER_BARS)
            psm_view_rgb_bars (WS_MAIN,VIEW_MAIN,model.view_location);
        else if (current.main_view == CHOICE_HSV_COLOR_CONE)
            psm_view_hsv_cone (WS_MAIN,VIEW_MAIN,view_location);
        else if (current.main_view == CHOICE_CIE_COLOR_TRIANGLE)
            psm_view_cie_triangle (WS_MAIN,VIEW_MAIN,view_location);
        redraw.main = TRUE;
        break;

    case ACTION_VIEW_PROJECTION:
```

```
    if (choice.choice == PPERSPECTIVE)
        pmessage (WS_MAIN,"Perspective Projection");
    else
        pmessage (WS_MAIN,"Parallel Projection");
    model.view_projection = choice.choice;
    psm_view_lines ();
    psm_view_model (WS_MAIN);
    psm_view_axes (WS_MAIN);
    psm_view_model (WS_AUX);
    psm_view_axes (WS_AUX);
    redraw.model = TRUE;
    break;

case ACTION_ROTATE_TRANSLATE_SCALE:

    psm_reinit_dials (DIALS_ROTATE_TRANSLATE_SCALE,
                      (Pfloat *)&model.transformation);
    break;

case ACTION_MODEL_SURFACE_PROPERTIES:

    redraw.color = psm_update_color_models (model.surface.color);
    psm_reinit_dials (DIALS_MODEL_SURFACE_PROPERTIES,
                      (Pfloat *)&model.surface);
    break;

case ACTION_ANIMATE_VIEW_MAPPING:

    psm_reinit_dials (DIALS_ANIMATE_VIEW_MAPPING,
                        (Pfloat *)&model.view_map);
    break;

case ACTION_VIEW_ORIENTATION:

    psm_reinit_dials (DIALS_VIEW_ORIENTATION,
                        (Pfloat *)&model.view_orient);
    break;

case ACTION_DEPTH_CUE_PARAMETERS:

    redraw.color = psm_update_color_models (model.depth_cue.color);
    psm_reinit_dials (DIALS_DEPTH_CUE_PARAMETERS,
                        (Pfloat *)&model.depth_cue);
    break;

case ACTION_MODEL_COMPONENT_COLORS:

    if (current.link >= model.nlinks) current.link = MAXLINKS;
    redraw.color = psm_update_color_models
        (model.link_colors[current.link]);
    psm_reinit_dials (DIALS_MODEL_COMPONENT_COLORS,
         (Pfloat *)&model.link_colors[current.link]);
    break;

case ACTION_BACKGROUND_COLOR:

    redraw.color = psm_update_color_models (model.background);
    psm_reinit_dials (DIALS_BACKGROUND_COLOR,
                    (Pfloat *)&model.background);
    break;

case ACTION_LIGHT_COLOR:

    redraw.color = psm_update_color_models
                (lights.source[current.light].color);
    psm_reinit_dials (DIALS_LIGHT_COLOR,
                (Pfloat *)&lights.source[current.light].color);
    break;

case ACTION_LIGHT_ORIENTATION:
```

```
        psm_reinit_dials (current.dials,
                    (Pfloat *)&lights.source[current.light].orient);
        break;

    case ACTION_DIALS:

        switch (current.dials) {
        case DIALS_ROTATE_TRANSLATE_SCALE:

            redraw.help = psm_display_help
                        (ACTION_ROTATE_TRANSLATE_SCALE);
            psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&model.transformation);
            redraw.model = psm_global_transformation ();
            psm_update_dials (current.dials,
                            (Pfloat *)&model.transformation);
            break;

        case DIALS_MODEL_SURFACE_PROPERTIES:

            redraw.help = psm_display_help
                        (ACTION_MODEL_SURFACE_PROPERTIES);
            if (event.ws != WS_DIALS && event.class == PI_LOCATOR)
                psm_evaluate_color_locator (locator,&ncolor,
                                        (Pfloat *)&model.surface.color);
            else {
                psm_evaluate_dials (event,&ndial,current.dials,
                                (Pfloat *)&model.surface);
                ncolor = ndial - 6;
            }

            if (ncolor >= 0) {
                psm_update_color_values (ncolor,&model.surface.color);
                redraw.color = psm_update_color_models
                                        (model.surface.color);
            }
            redraw.model = psm_surface_properties ();
            psm_update_dials (current.dials,(Pfloat *)&model.surface);
            break;

        case DIALS_ANIMATE_VIEW_MAPPING:

            redraw.help = psm_display_help (ACTION_ANIMATE_VIEW_MAPPING);
            model.view_map.step = model.view_map.fractional_step;
            psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&model.view_map);
            model.view_map.fractional_step = model.view_map.step;
            redraw.model = psm_animate (&model.view_map.step);
            if (!redraw.model)
                redraw.model = psm_model_view_mapping ();

            psm_update_dials (current.dials,(Pfloat *)&model.view_map);
            break;

        case DIALS_VIEW_ORIENTATION:

            redraw.help = psm_display_help (ACTION_VIEW_ORIENTATION);
            psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&model.view_orient);
            redraw.model = psm_model_view_orientation ();
            psm_update_dials (current.dials,
                            (Pfloat *)&model.view_orient);
            break;
```

```
case DIALS_DEPTH_CUE_PARAMETERS:

    redraw.help = psm_display_help (ACTION_DEPTH_CUE_PARAMETERS);
    if (event.ws != WS_DIALS && event.class == PI_LOCATOR)
        psm_evaluate_color_locator (locator,&ncolor,
                            (Pfloat *)&model.depth_cue.color);
    else {
        psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&model.depth_cue);
        ncolor = ndial - 6;
    }

    if (ncolor >= 0) {
        psm_update_color_values (ncolor,&model.depth_cue.color);
        redraw.color = psm_update_color_models
                                (model.depth_cue.color);
    }
    redraw.model = psm_depth_cueing ();
    psm_update_dials (current.dials,(Pfloat *)&model.depth_cue);
    break;

case DIALS_MODEL_COMPONENT_COLORS:

    redraw.help = psm_display_help
                  (ACTION_MODEL_COMPONENT_COLORS);
    if (current.link >= model.nlinks) current.link = MAXLINKS;
    if (event.ws != WS_DIALS && event.class == PI_LOCATOR)
        psm_evaluate_color_locator (locator,&ndial,
                   (Pfloat *)&model.link_colors[current.link]);
    else
        psm_evaluate_dials (event,&ndial,current.dials,
                   (Pfloat *)&model.link_colors[current.link]);
    redraw.color = psm_update_color (ndial,
                       MODEL_COLOR_INDEX+current.link,
                       &model.link_colors[current.link]);
    redraw.model = redraw.color;

    if (redraw.color && current.link == MAXLINKS)
        for (i=0; i<MAXLINKS; i++) {
            model.link_colors[i] = model.link_colors[MAXLINKS];
            psetcolourrep (WS_MAIN,MODEL_COLOR_INDEX+i,
                           &model.link_colors[i].rgb);
            psetcolourrep (WS_AUX,MODEL_COLOR_INDEX+i,
                           &model.link_colors[i].rgb);
        }
    break;

case DIALS_BACKGROUND_COLOR:

    redraw.help = psm_display_help (ACTION_BACKGROUND_COLOR);
    if (event.ws != WS_DIALS && event.class == PI_LOCATOR)
        psm_evaluate_color_locator (locator,&ndial,
                                (Pfloat *)&model.background);
    else
        psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&model.background);
    redraw.color = psm_update_color (ndial,BACKGROUND,
                       &model.background);
    if (redraw.color) redraw.main = redraw.aux = TRUE;
    break;

case DIALS_LIGHT_COLOR:
```

```
                redraw.help = psm_display_help (ACTION_LIGHT_COLOR);
                if (event.ws != WS_DIALS && event.class == PI_LOCATOR)
                    psm_evaluate_color_locator (locator,&ndial,
                            (Pfloat *)&lights.source[current.light].color);
                else
                    psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&lights.source[current.light].color);
                redraw.color = psm_update_color (ndial,
                                LIGHT_COLOR_INDEX+current.light,
                                &lights.source[current.light].color);
                if (redraw.color) {
                    psm_light_source_rep (current.light);
                    redraw.model = TRUE;
                }
                break;

            case DIALS_INFINITE_LIGHT:
            case DIALS_POINT_LIGHT:
            case DIALS_SPOT_LIGHT:

                redraw.help = psm_display_help (ACTION_LIGHT_ORIENTATION);
                psm_evaluate_dials (event,&ndial,current.dials,
                            (Pfloat *)&lights.source[current.light].orient);
                redraw.model = psm_set_light_source (current.light);
                psm_update_dials (current.dials,
                            (Pfloat *)&lights.source[current.light].orient);
                break;

            default:

                pmessage (WS_MAIN,"***Invalid dials***");
            }

        case ACTION_NONE:

            break;

        default:

            pmessage (WS_MAIN,"***Not Implemented***");
        }

        psm_redraw_structures (help);

        for (i=3; i<12; i++)
            pflushevents (WS_MAIN,PI_VALUATOR,i);
        pflushevents (WS_DIALS,PI_LOCATOR,3);
    }
}
```

```
/*
 * FUNCTION:    psm_redraw_structures
 *
 *      psm_redraw_structures determines if the structures posted to WS_MAIN
 *      and WS_AUX should be redrawn based on which structures are displayed
 *      and which structures have been modified
 *
 * INPUT ARGUMENTS:
 *
 *      help:   indicates if help is currently posted (on)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_redraw_structures (help)
int help;
{
    if (redraw.help) {
        if (help) redraw.main = TRUE;
    }

    if (redraw.model) {
        if (current.main_view == CHOICE_SPATIAL_MECHANISM_MODEL ||
            current.main_view == CHOICE_LIGHT_SOURCES)
            redraw.main = TRUE;
        if (current.aux_view == CHOICE_SPATIAL_MECHANISM_MODEL ||
            current.aux_view == CHOICE_LIGHT_SOURCES)
            redraw.aux = TRUE;
    }

    if (redraw.lights) {
        if (current.main_view == CHOICE_LIGHT_SOURCES)
            redraw.main = TRUE;
        if (current.aux_view == CHOICE_LIGHT_SOURCES)
            redraw.aux = TRUE;
    }

    if (redraw.color) {
        if (model.view_location == PART_VIEW ||
            current.main_view == CHOICE_RGB_COLOR_CUBE ||
            current.main_view == CHOICE_RGB_SLIDER_BARS ||
            current.main_view == CHOICE_HSV_COLOR_CONE ||
            current.main_view == CHOICE_CIE_COLOR_TRIANGLE)
            redraw.main = TRUE;
        if (current.aux_view == CHOICE_RGB_COLOR_CUBE ||
            current.aux_view == CHOICE_RGB_SLIDER_BARS ||
            current.aux_view == CHOICE_HSV_COLOR_CONE ||
            current.aux_view == CHOICE_CIE_COLOR_TRIANGLE)
            redraw.aux = TRUE;
    }

    if (redraw.main) predrawallstruct (WS_MAIN,PALWAYS);
    if (redraw.aux) predrawallstruct (WS_AUX,PALWAYS);
}
```

```
/*
 * FUNCTION:    psm_set_dial_action
 *
 *      psm_set_dial_action translates the current setting of the dials into
 *      an action.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      the action constant to be acted upon
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_set_dial_action ()
{
    if          (current.dials == DIALS_ROTATE_TRANSLATE_SCALE)
        return (ACTION_ROTATE_TRANSLATE_SCALE);
    else if (current.dials == DIALS_MODEL_SURFACE_PROPERTIES)
        return (ACTION_MODEL_SURFACE_PROPERTIES);
    else if (current.dials == DIALS_ANIMATE_VIEW_MAPPING)
        return (ACTION_ANIMATE_VIEW_MAPPING);
    else if (current.dials == DIALS_VIEW_ORIENTATION)
        return (ACTION_VIEW_ORIENTATION);
    else if (current.dials == DIALS_DEPTH_CUE_PARAMETERS)
        return (ACTION_DEPTH_CUE_PARAMETERS);
    else if (current.dials == DIALS_MODEL_COMPONENT_COLORS)
        return (ACTION_MODEL_COMPONENT_COLORS);
    else if (current.dials == DIALS_BACKGROUND_COLOR)
        return (ACTION_BACKGROUND_COLOR);
    else if (current.dials == DIALS_LIGHT_COLOR)
        return (ACTION_LIGHT_COLOR);
    else if (current.dials == DIALS_INFINITE_LIGHT)
        return (ACTION_LIGHT_ORIENTATION);
    else if (current.dials == DIALS_POINT_LIGHT)
        return (ACTION_LIGHT_ORIENTATION);
    else if (current.dials == DIALS_SPOT_LIGHT)
        return (ACTION_LIGHT_ORIENTATION);
    return (ACTION_NONE);
}
```

```
/*
 * FUNCTION:    psm_set_current_dials
 *
 *      psm_set_current_dials determines which set of dial values should
 *      be displayed and active for modification.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_set_current_dials ()
{
    if (current.light > 0 && lights.source[current.light].on) {
        redraw.lights = psm_light_ray (ON);
        if (current.light_edit == CHOICE_LIGHT_COLOR) {
            current.dials = DIALS_LIGHT_COLOR;
        } else {
            if (lights.source[current.light].type ==
                        PLT_INFINITE)
                current.dials = DIALS_INFINITE_LIGHT;
            else if (lights.source[current.light].type ==
                        PLT_POINT)
                current.dials = DIALS_POINT_LIGHT;
            else if (lights.source[current.light].type ==
                        PLT_SPOT)
                current.dials = DIALS_SPOT_LIGHT;
            else
                current.dials = current.dial_menu;
        }

    } else {
        redraw.lights = psm_light_ray (OFF);
        current.dials = current.dial_menu;
    }
}
```

```
/*
 * FUNCTION:    psm_change_display
 *
 *      psm_change_display changes the workstation type of the WS_MAIN and
 *      WS_AUX windows to TRUE COLOR DOUBLE BUFFERED or GRAYSCALE (PSEUDO
 *      COLOR) DOUBLE BUFFERED.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_change_display (display_type)
Pint display_type;
{
    Puescapeinrec0010 workstation_type;
    Pescapeout out;

    workstation_type.ws_truecolour = display_type;
    workstation_type.ws_dbl_buf = TRUE;

    workstation_type.ws_id = WS_MAIN;
    pescape (puescape0010,&workstation_type,&out);
    workstation_type.ws_id = WS_AUX;
    pescape (puescape0010,&workstation_type,&out);
}
```

```
/*
 * FUNCTION:    psm_evaluate_color_locator
 *
 *      psm_evaluate_color_locator determines if a locator input falls within
 *      the rgb slider bars or cie color triangle structures.  If so, the
 *      color number that corresponds to the dials is set, along with the
 *      value indicated by the locator position.
 *
 * INPUT ARGUMENTS:
 *
 *      locator:    contains the view number and position of the locator
 *                  input
 *
 * OUTPUT ARGUMENTS:
 *
 *      ncolor:     color number that was changed
 *                 -1:  none
 *                  0:  red
 *                  1:  green
 *                  2:  blue
 *                  3:  hue
 *                  4:  saturation
 *                  5:  value
 *                  6:  x (CIE diagram)
 *                  7:  y (CIE diagram)
 *                  8:  luminance
 *      color_values:  array of color values corresponding to ncolor
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_evaluate_color_locator (locator,ncolor,color_values)
Ploc locator;
Pint *ncolor;
Pfloat color_values[9];
{

    *ncolor = -1;
    if (locator.view_index == VIEW_RGB_BARS ||
        (locator.view_index == VIEW_MAIN &&
         current.main_view == CHOICE_RGB_SLIDER_BARS) ||
        (locator.view_index == VIEW_AUX &&
         current.aux_view == CHOICE_RGB_SLIDER_BARS)) {
        if         (locator.position.y <= MAX_RED_Y &&
                   locator.position.y >= MIN_RED_Y)
            *ncolor = 0;
        else if    (locator.position.y <= MAX_GREEN_Y &&
                   locator.position.y >= MIN_GREEN_Y)
            *ncolor = 1;
        else if    (locator.position.y <= MAX_BLUE_Y &&
                   locator.position.y >= MIN_BLUE_Y)
            *ncolor = 2;
        if (*ncolor >= 0) {
            color_values[*ncolor] = locator.position.x;
        }
```

```
    } else if (locator.view_index == VIEW_CIE_TRIANGLE ||
               (locator.view_index == VIEW_MAIN &&
                current.main_view == CHOICE_CIE_COLOR_TRIANGLE) ||
               (locator.view_index == VIEW_AUX &&
                current.aux_view == CHOICE_CIE_COLOR_TRIANGLE)) {
        *ncolor = 7;
        color_values[6] = locator.position.x;
        color_values[7] = locator.position.y;
    }
}
```

```
/*
 * FUNCTION:    psm_update_color
 *
 *      psm_update_color calls functions to update color values, the color
 *      of an indexed color, and the dial window that shows the current
 *      values of the color.
 *
 * INPUT ARGUMENTS:
 *
 *      ncolor:     color number that was changed
 *      index:      color index of the color that was changed
 *
 * OUTPUT ARGUMENTS:
 *
 *      color:  contains updated values of the RGB, HSV, and CIE color
 *              models
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_update_color (ncolor,index,color)
Pint ncolor, index;
PSM_color_models *color;
{
    int redraw_color;

    if (ncolor < 0) return (0);

    psm_update_color_values (ncolor,color);
    psetcolourrep (WS_MAIN,index,color->rgb);
    psetcolourrep (WS_AUX,index,color->rgb);
    redraw_color = psm_update_color_models (*color);
    psm_update_dials (current.dials,(Pfloat *)color);
    return (redraw_color);
}

#define IMAGE_WIDTH 1000
#define IMAGE_HEIGHT 1000
#define IMAGE_SIZE IMAGE_WIDTH*IMAGE_HEIGHT
```

```
/*
 * FUNCTION:    psm_image_readback
 *
 *      psm_image_readback performs a copy screen function by reading the
 *      red, green, and blue data displayed on the workstation screen and
 *      converting it into grayscale bitmap data.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_image_readback ()
{
    int i;
    Puescapeinrec0001 screen_area;
    Puescapeoutrec0001 red_bitmap, green_bitmap, blue_bitmap;
    unsigned char *red_bitmap_data, *green_bitmap_data, *blue_bitmap_data;
    int value;
    FILE *fopen(), *fp;
    Pcobundl rgb;

    screen_area.origin.x = 0.00;
    screen_area.origin.y = 1.00;
    screen_area.origin.z = 0.05;
    screen_area.ws_id = WS_MAIN;
    screen_area.dim.x_dim = IMAGE_WIDTH;
    screen_area.dim.y_dim = IMAGE_HEIGHT;

    pmessage (WS_MAIN,"Reading red channel");
    screen_area.channel_number = PIMCHAN_RED;
    red_bitmap.array = (Punsigned *) malloc(IMAGE_SIZE);
    red_bitmap_data = (unsigned char *)red_bitmap.array;
    pescape (puescape0001,&screen_area,&red_bitmap);

/*
    pmessage (WS_MAIN,"Writing red data; please wait...");
    fp = fopen("bitmap.red","w");
    for (i=0; i<IMAGE_SIZE; i++) {
        fprintf (fp,"%c",red_bitmap_data[i]);
    }
    fclose (fp);
*/

    pmessage (WS_MAIN,"Reading green channel");
    screen_area.channel_number = PIMCHAN_GREEN;
    green_bitmap.array = (Punsigned *) malloc(IMAGE_SIZE);
    green_bitmap_data = (unsigned char *)green_bitmap.array;
    pescape (puescape0001,&screen_area,&green_bitmap);

/*
    pmessage (WS_MAIN,"Writing green data; please wait...");
    fp = fopen("bitmap.green","w");
    for (i=0; i<IMAGE_SIZE; i++) {
        fprintf (fp,"%c",green_bitmap_data[i]);
    }
    fclose (fp);
*/
```

```
    pmessage (WS_MAIN,"Reading blue channel");
    screen_area.channel_number = PIMCHAN_BLUE;
    blue_bitmap.array = (Punsigned *) malloc(IMAGE_SIZE);
    blue_bitmap_data = (unsigned char *)blue_bitmap.array;
    pescape (puescape0001,&screen_area,&blue_bitmap);

/*
    pmessage (WS_MAIN,"Writing blue data; please wait...");
    fp = fopen("bitmap.blue","w");
    for (i=0; i<IMAGE_SIZE; i++) {
        fprintf (fp,"%c",blue_bitmap_data[i]);
    }
    fclose (fp);
*/

    pmessage (WS_MAIN,"Writing grayscale data; please wait...");
    fp = fopen("bitmap.gray","w");
    for (i=0; i<IMAGE_SIZE; i++) {
        rgb.x = (Pfloat)red_bitmap_data[i];
        rgb.y = (Pfloat)green_bitmap_data[i];
        rgb.z = (Pfloat)blue_bitmap_data[i];
        value = (max(max(rgb.x,rgb.y),rgb.z) +
                 min(min(rgb.x,rgb.y),rgb.z))/2.0;
        fprintf (fp,"%c",(unsigned char)value);
    }
    fclose (fp);

    free ((Punsigned *)red_bitmap.array);
    free ((Punsigned *)green_bitmap.array);
    free ((Punsigned *)blue_bitmap.array);

    pmessage (WS_MAIN,"Image readback completed");
}
```

Module: psm_lights.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "model.h"
#include "lights.h"

extern PSM_model_rendering model;
extern PSM_light_sources lights;

float max (), min ();
```

```
/*
 * FUNCTION:    psm_init_lights
 *
 *      This function initializes the PHIGS structure containing the
 *      light models.  This first light source is set to ambient, the
 *      second is set to infinite, and light sources 3-8 are off.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_lights ()
{
    int i;
    static Pcobundl gray = {0.50,0.50,0.50};
    static Pcobundl white = {1.00,1.00,1.00};
    static Ppoint3 origin = {0.00,0.00,0.00};
    static Pvector3 init_direction = {-1.00,-1.00,-1.00};

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_LIGHT_SOURCES);
    psethlhsrid (ON);
    psetnormreorientmode (PNOFLIP);
    psetfaceprocmode (PDIST_NO,PCULL_BACK);
    psetintstyle (PSOLID);
    plabel (LABEL_SHADING_METHOD);
    psetintshademethod (PSH_COLOUR);
    psetintlightmethod (PLM_SPECULAR);
    plabel (LABEL_LIGHT_SOURCE_STATE);
    plabel (LABEL_LIGHT_SOURCE_STATE);

    psm_set_default_surface_properties ();
    for (i=1; i<=8; i++)
        pexecutestruct (STRUCT_LIGHT_SOURCES+i);
    pclosestruct ();

    for (i=1; i<=MAXLIGHTS; i++) {
        lights.source[i].on = FALSE;
        lights.source[i].type = PLT_NONE;
        lights.source[i].color.rgb = white;
        psm_rgb_to_hsv_cie (&lights.source[i].color);
        lights.source[i].orient.position  = origin;
        lights.source[i].orient.direction = init_direction;
        lights.source[i].orient.exponent  = 1.0;
        lights.source[i].orient.angle     = 30.0;
    }

    lights.source[1].type = PLT_AMBIENT;
    lights.source[1].color.rgb = gray;
    psm_rgb_to_hsv_cie (&lights.source[1].color);
    psm_set_light_source (1);

    lights.source[2].type = PLT_INFINITE;
    psm_set_light_source (2);
}
```

```
/*
 * FUNCTION:    psm_set_light_source
 *
 *      This function sets the light source for the given index to the
 *      indicated light source representation and PHIGS model.
 *
 * INPUT ARGUMENTS:
 *
 *      index:  index of the light source (1-8) to be set
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_set_light_source (index)
Pint index;
{
    static Pint last_type[MAXLIGHTS+1];
    static PSM_light_orientation last_orient[MAXLIGHTS+1];
    int redraw;

    Pint type;
    PSM_light_orientation orient;
    Pmatrix3 trans_matrix;
    Pvector3 rotation, shift;
    Pint ierr;
    Pfloat dx, dy, dz, len;

    redraw = FALSE;

    type = lights.source[index].type;
    if (last_type[index] != type) redraw = TRUE;
    psm_light_list (index);

    pseteditmode (PEDIT_INSERT);

    if (type == PLT_NONE) {
        popenstruct (STRUCT_LIGHT_SOURCES+index);
        pemptystruct (STRUCT_LIGHT_SOURCES+index);
        pclosestruct ();

    } else {
        orient = lights.source[index].orient;
        orient.direction.x = max (-1.0,orient.direction.x);
        orient.direction.x = min (1.0,orient.direction.x);
        orient.direction.y = max (-1.0,orient.direction.y);
        orient.direction.y = min (1.0,orient.direction.y);
        orient.direction.z = max (-1.0,orient.direction.z);
        orient.direction.z = min (1.0,orient.direction.z);
        orient.exponent    = max (0.0,orient.exponent);
        orient.angle       = max (0.0,orient.angle);
        orient.angle       = min (180.0,orient.angle);
        lights.source[index].orient = orient;
```

```
        if (last_orient[index].position.x  != orient.position.x ||
            last_orient[index].position.y  != orient.position.y ||
            last_orient[index].position.z  != orient.position.z ||
            last_orient[index].direction.x != orient.direction.x ||
            last_orient[index].direction.y != orient.direction.y ||
            last_orient[index].direction.z != orient.direction.z ||
            last_orient[index].exponent    != orient.exponent ||
            last_orient[index].angle       != orient.angle)
            redraw = TRUE;
        else if (!redraw)
            return (0);

        if (type == PLT_AMBIENT) {
            popenstruct (STRUCT_LIGHT_SOURCES+index);
            pemptystruct (STRUCT_LIGHT_SOURCES+index);
            pclosestruct ();

        } else {
            popenstruct (STRUCT_LIGHT_SOURCES+index);
            pemptystruct (STRUCT_LIGHT_SOURCES+index);
            psetlinecolourind (LIGHT_COLOR_INDEX+index);
            psetintcolourind (LIGHT_COLOR_INDEX+index);
            plabel (LABEL_LOCAL_TRANSFORMATION);

            dx = orient.direction.x;
            dy = orient.direction.y;
            dz = orient.direction.z;
            rotation.z = 0.00;
            len = sqrt (dy*dy + dz*dz);
            if (len > 0.00) {
                rotation.x = -acos(dz/len);
                if (dy != 0.00) rotation.x *= dy/fabs(dy);
            } else {
                rotation.x = 0.00;
            }

            dz = len;
            len = sqrt (dx*dx + dz*dz);
            if (len > 0.00) {
                rotation.y = acos(dz/len);
                if (dx != 0.00) rotation.y *= dx/fabs(dx);
            } else {
                rotation.y = 0.00;
            }

            shift.x = orient.position.x;
            shift.y = orient.position.y;
            shift.z = orient.position.z;
            psm_build_tran3 (rotation,shift,trans_matrix);
            psetlocaltran3 (trans_matrix,PREPLACE);

            if (type == PLT_INFINITE) {
                pexecutestruct (STRUCT_INFINITE_LIGHT);

            } else if (type == PLT_POINT) {
                pexecutestruct (STRUCT_POINT_LIGHT);

            } else if (type == PLT_SPOT) {
                pexecutestruct (STRUCT_SPOT_LIGHT);
            }
            pclosestruct ();
        }
        psm_light_source_rep (index);
    }

    last_type[index] = type;
    last_orient[index] = orient;
    return (redraw);
}
```

```
/*
 * FUNCTION:    psm_light_list
 *
 *      psm_light_list makes a new light source state list (which light
 *      sources are on) used in the lighting of the spatial mechanism,
 *      the axes set, and the light source models themselves.
 *
 * INPUT ARGUMENTS:
 *
 *      index:  index of the light source (1-8) that was turned on or off
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_light_list (index)
Pint index;
{
    int i, num_on, num_off, on;

    on = (lights.source[index].type != PLT_NONE);
    if (on != lights.source[index].on) {
        lights.source[index].on = on;
        num_on = num_off = 0;
        for (i=1; i<=8; i++) {
            if (lights.source[i].type != PLT_NONE)
                lights.on[num_on++] = i;
            else
                lights.off[num_off++] = i;
        }

        lights.onlst.number    = num_on;
        lights.onlst.integers  = lights.on;
        lights.offlst.number   = num_off;
        lights.offlst.integers = lights.off;

        pseteditmode (PEDIT_REPLACE);

        popenstruct (STRUCT_MECHANISM);
        psetelemptr (BEGINNING);
        psetelemptrlabel (LABEL_LIGHT_SOURCE_STATE);
        poffsetelemptr (1);
        psetlightsrcst (&lights.onlst,&lights.offlst);
        pclosestruct ();

        popenstruct (STRUCT_AXES);
        psetelemptr (BEGINNING);
        psetelemptrlabel (LABEL_LIGHT_SOURCE_STATE);
        poffsetelemptr (1);
        psetlightsrcst (&lights.onlst,&lights.offlst);
        pclosestruct ();

        popenstruct (STRUCT_LIGHT_SOURCES);
        psetelemptr (BEGINNING);
        psetelemptrlabel (LABEL_LIGHT_SOURCE_STATE);
        poffsetelemptr (1);
        psetlightsrcst (&lights.onlst,&lights.offlst);
        pclosestruct ();
    }
}
```

```
/*
 * FUNCTION:    psm_light_source_rep
 *
 *      psm_light_source_rep set the PHIGS representation of the light
 *      source.
 *
 * INPUT ARGUMENTS:
 *
 *      index:  light source index (1-8) of the light source representation
 *              to be set.
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_light_source_rep (index)
Pint index;
{
    Pint type;
    Pcobundl rgb;
    Plightsrcrec rep;

    type = lights.source[index].type;

    rgb = lights.source[index].color.rgb;
    if (type == PLT_AMBIENT) {
        rep.ambient.colour.colour_model = PCM_RGB;
        rep.ambient.colour.colour = rgb;

    } else if (type == PLT_INFINITE) {
        rep.infinite.colour.colour_model = PCM_RGB;
        rep.infinite.colour.colour = rgb;
        rep.infinite.direction = lights.source[index].orient.direction;

    } else if (type == PLT_POINT) {
        rep.point.colour.colour_model = PCM_RGB;
        rep.point.colour.colour = rgb;
        rep.point.position = lights.source[index].orient.position;

    } else if (type == PLT_SPOT) {
        rep.spot.colour.colour_model = PCM_RGB;
        rep.spot.colour.colour = rgb;
        rep.spot.position = lights.source[index].orient.position;
        rep.spot.direction = lights.source[index].orient.direction;
        rep.spot.exponent = lights.source[index].orient.exponent;
        rep.spot.angle = lights.source[index].orient.angle*PI/180.0;

    } else {
        return;
    }

    psetcolourrep (WS_MAIN,LIGHT_COLOR_INDEX+index,&rgb);
    psetcolourrep (WS_AUX,LIGHT_COLOR_INDEX+index,&rgb);
    psetlightsrcrep (WS_MAIN,index,type,&rep);
    psetlightsrcrep (WS_AUX,index,type,&rep);
}
```

```
/*
 * FUNCTION:    psm_light_ray
 *
 *      psm_light_ray creates the PHIGS structure that contains a line
 *      used to represent the direction of directional light sources.
 *
 * INPUT ARGUMENTS:
 *
 *      on:  true when the "light ray" line is to be inserted into the
 *           structure
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_light_ray (on)
int on;
{
    static Ppoint3 pts[2] = {0.00,0.00,0.00, 0.00,0.00,999.0};
    static int light_ray = OFF;

    if (light_ray == on) return (0);

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_LIGHT_RAY);
    pemptystruct (STRUCT_LIGHT_RAY);
    if (on) {
        psetlinetype (PLN_DOTDASH);
        ppolyline3 (2,pts);
    }
    pclosestruct ();
    light_ray = on;
    return (1);
}
```

Module: psm_model.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include <ctype.h>
#include "common.h"
#include "model.h"
#include "input.h"
#define MAXPOS 100

extern PSM_model_rendering model;

float max (), min ();
```

```
/*
 * FUNCTION:    psm_init_model
 *
 *      psm_init_model initializes the PHIGS structure containing the
 *      spatial mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_model ()
{
    static Pcobundl black = {0.00,0.00,0.00};

    popenstruct (STRUCT_MECHANISM);
    pemptystruct (STRUCT_MECHANISM);
    plabel (LABEL_GLOBAL_TRANSFORMATION);
    plabel (LABEL_GLOBAL_TRANSFORMATION);
    psethlhsrid (ON);
    psetnormreorientmode (PNOFLIP);
    psetfaceprocmode (PDIST_NO,PCULL_BACK);
    psetintstyle (PSOLID);
    plabel (LABEL_SHADING_METHOD);
    psetintshademethod (PSH_COLOUR);
    psetintlightmethod (PLM_SPECULAR);
    plabel (LABEL_LIGHT_SOURCE_STATE);
    plabel (LABEL_LIGHT_SOURCE_STATE);
    plabel (LABEL_SURFACE_PROPERTIES);
    plabel (LABEL_SURFACE_PROPERTIES);
    plabel (LABEL_STRUCT_MECHANISM);
    pexecutestruct (STRUCT_MECHANISM+1);
    pclosestruct ();

    model.nsteps = 0;
    model.link_names[MAXLINKS] = "Entire mechanism picked";
    model.nsegments = 4;
    model.depth_cue_on = FALSE;
    model.shading_method = CHOICE_SHADING_SMOOTH;
    model.multi_view = FALSE;
    model.view_location = PART_VIEW;
    model.view_projection = PPARALLEL;

    model.background.rgb = black;
    psm_rgb_to_hsv_cie (&model.background);
}
```

```
/*
 * FUNCTION:    psm_reset_model
 *
 *      psm_reset_model resets many of the data parameters used to display
 *      the spatial mechanism model.  This reset operation affects the
 *      following display characteristics:
 *          Rotate - Translate - Scale
 *          Model Surface Properties
 *          Animation step
 *          Depth Cue Parameters
 *          Model Component Colors
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_reset_model ()
{
    int i;
    static Pcobundl black = {0.00,0.00,0.00};
    static Pcobundl white = {1.00,1.00,1.00};
    static Pcobundl steel_blue = {0.20,0.20,1.00};
    static Pvector3 rotate = {0.00,0.00,0.00};
    static Pvector3 translate = {0.00,0.00,0.00};
    static Pvector3 scale = {1.00,1.00,1.00};

    model.transformation.rotate = rotate;
    model.transformation.translate = translate;
    model.transformation.scale = scale;
    psm_global_transformation ();

    model.surface.ambient = 1.0;
    model.surface.diffuse = 1.0;
    model.surface.specular = 1.0;
    model.surface.exponent = 1.0;
    model.surface.color.rgb = white;
    psm_rgb_to_hsv_cie (&model.surface.color);
    psm_surface_properties ();

    model.view_map.step = 1;
    model.view_map.fractional_step = 1;
    psm_animate (&model.view_map.step);

    model.depth_cue.planes[0] = 1.00;
    model.depth_cue.planes[1] = 0.00;
    model.depth_cue.scaling[0] = 1.00;
    model.depth_cue.scaling[1] = 0.00;
    model.depth_cue.color.rgb = black;
    psm_rgb_to_hsv_cie (&model.depth_cue.color);
    psm_depth_cueing ();

    for (i=0; i<=MAXLINKS; i++) {
        model.link_colors[i].rgb = steel_blue;
        psm_rgb_to_hsv_cie (&model.link_colors[i]);
        psetcolourrep (WS_MAIN,MODEL_COLOR_INDEX+i,&steel_blue);
        psetcolourrep (WS_AUX,MODEL_COLOR_INDEX+i,&steel_blue);
    }
```

```
    psm_reset_model_view ();
    psm_model_view_mapping ();
    psm_model_view_orientation ();
    psm_view_lights (WS_MAIN);
    psm_view_lights (WS_AUX);
}
```

```
/*
 * FUNCTION:    psm_reset_model_view
 *
 *      This function resets the view of the spatial mechanism model to
 *      the original orientation.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_reset_model_view ()
{
    static Pvector3 vpn = {1.00,1.00,1.00};
    static Pvector3 vup = {-0.50,1.00,-0.50};

    model.view_map.zoom_model = 1.00;
    model.view_map.zoom_lights = 1.00;
    model.view_map.prp.x = model.view_map.prp.y = 0.00;
    model.view_map.prp.z = model.view_edge_len;
    model.view_map.front_plane = model.view_edge_len/2.0;
    model.view_map.view_plane = 0.00;
    model.view_map.back_plane = -model.view_edge_len/2.0;
    model.view_orient.vrp = model.center;
    model.view_orient.vpn = vpn;
    model.view_orient.vup = vup;
}
```

```
Pfloat max_radius;
Ppoint3 max_bound, min_bound;

typedef struct {
    Pint      id;
    Pvector3 pos;
} Link_position;

int npos;
Link_position link_pos[MAXPOS];
int links[MAXLINKS];

typedef struct {
    int count;
    char *name;
} PSM_input_elements;
```

```
PSM_input_elements input[12] = {0, "INVALID",
                                   9, "external revolute joint",
                                   9, "internal revolute joint",
                                0, "connecting element",
                                0, "mechanism link",
                                9, "external prismatic joint",
                                9, "internal prismatic joint",
                                9, "external cylindric joint",
                                9, "internal cylindric joint",
                                8, "external spheric joint",
                                4, "internal spheric joint",
                                   11, "pillow block"};
```

```
/*
 * FUNCTION:    psm_get_model
 *
 *      psm_get_model retrieves a selected model.  A new model center is
 *      computed along with the relative travel of some dial functions
 *      that are based on the model geometry.
 *
 * INPUT ARGUMENTS:
 *
 *      nmodel: the number (or index) of the model requested
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *       1 if unsuccessful
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_get_model (nmodel)
int nmodel;
{
    char filename[MAXLEN];
    int i, lenname;

    struct_id_free = STRUCT_ID_FREE;

    max_bound.x = max_bound.y = max_bound.z = 0.00;
    min_bound = max_bound;
    max_radius = 0.00;

    lenname = strlen (model_names[nmodel]);
    strncpy (filename,model_names[nmodel],lenname);
    strcpy (&filename[lenname],".mod");
    if (psm_read_model(filename) != 0) return (1);
    strcpy (&filename[lenname],".pos");
    if (psm_read_positions(filename) != 0) return (1);

    model.center.x = (min_bound.x+max_bound.x)/2.0;
    model.center.y = (min_bound.y+max_bound.y)/2.0;
    model.center.z = (min_bound.z+max_bound.z)/2.0;
    model.view_edge_len = 1.5 * max (max (max_bound.x-min_bound.x,
                          max_bound.y-min_bound.y), max_bound.z-min_bound.z);

    for (i=0; i<3; i++) {
        psm_update_travel (DIALS_ROTATE_TRANSLATE_SCALE,i+3,
                           model.view_edge_len);
        psm_update_travel (DIALS_VIEW_ORIENTATION,i,model.view_edge_len);
        psm_update_travel (DIALS_INFINITE_LIGHT,i,model.view_edge_len);
        psm_update_travel (DIALS_POINT_LIGHT,i,model.view_edge_len);
        psm_update_travel (DIALS_SPOT_LIGHT,i,model.view_edge_len);
    }
```

```
    for (i=3; i<9; i++)
        psm_update_travel (DIALS_ANIMATE_VIEW_MAPPING,i,model.view_edge_len);

    if (DEBUG) {
        printf ("psm_get_model:\n");
        printf ("\tmin_bound = %f %f %f\n",
                min_bound.x,min_bound.y,min_bound.z);
        printf ("\tmax_bound = %f %f %f\n",
                max_bound.x,max_bound.y,max_bound.z);
        printf ("\tmodel.center = %f %f %f\n",
                model.center.x,model.center.y,model.center.z);
        printf ("\tview_edge_len = %f\n",model.view_edge_len);
        for (i=0; i<npos; i++)
            printf ("\tlink_pos %d = %f %f %f\n",
                    link_pos[i].id,link_pos[i].pos.x,
                    link_pos[i].pos.y,link_pos[i].pos.z);
        printf ("\tmax_radius = %f\n",max_radius);
    }

    psm_light_sources (max_radius);
    psm_global_transformation ();
    psm_surface_properties ();
}
```

```
/*
 * FUNCTION:    psm_read_model
 *
 *      This function reads the model data from the requested file.  The
 *      data is checked for basic integrity.
 *
 * INPUT ARGUMENTS:
 *
 *      filename:  complete filename (including extension) of the model file
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 if successful; otherwise non-zero
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_model (filename)
char *filename;
{
    FILE *fp, *fopen();
    char line[MAXLEN], descr[MAXLEN], *p;
    int len, lenname, i, j, k;
    int type, id, step, nline;
    Pfloat Ro, Ri, Len, Ly, Lz, Z, Z1, Z2;
    Pvector3 Rotation, Position;
    Pmatrix3 trans_matrix;
    Ppoint3 pt3;
    Pint ierr;

    float max (); min ();
```

```
    if ((fp=fopen(filename,"r")) == NULL) {
        psm_error_msg (filename,0,"CANNOT OPEN THE MODEL FILE");
        return (1);
    } else if (fgets (descr,MAXLEN,fp) == NULL) {
        psm_error_msg (filename,0,"CANNOT READ THE MODEL FILE");
        return (1);
    } else {
        pmessage (WS_MAIN,descr);
    }

    model.nlinks = 0;
    npos = 0;
    nline = 1;

    while (fgets (line,MAXLEN,fp)) {
        nline++;
        sscanf (line,"%d %d",&type,&id);
        if (id < 0) {
            psm_error_msg (filename,nline,"Invalid ID (negative)");
            fclose (fp);
            return (1);
        }
        len = strlen(line) - 1;
        i = 0;
        while (i<len && (isdigit(line[i]) || isspace(line[i])))
            i++;
        len -= i;
        strncpy (descr,&line[i],len);
        descr[len] = '\0';
        if (DEBUG) printf ("psm_read_model: type = %d, id = %d, %s\n",
                           type,id,descr);

        nline++;
        switch (type) {

        case EXTERNAL_REVOLUTE_JOINT:
        case INTERNAL_REVOLUTE_JOINT:
        case EXTERNAL_PRISMATIC_JOINT:
        case INTERNAL_PRISMATIC_JOINT:
        case EXTERNAL_CYLINDRIC_JOINT:
        case INTERNAL_CYLINDRIC_JOINT:
        case EXTERNAL_SPHERIC_JOINT:
        case INTERNAL_SPHERIC_JOINT:
        case PILLOW_BLOCK:
            psm_read_joint (type,id,fp,filename,nline,&ierr);
            break;

        case CONNECTING_ELEMENT:
            psm_read_connecting_element (id,fp,filename,nline,&ierr);
            break;

        case MECHANISM_LINK:
            psm_read_mechanism_link (id,fp,filename,descr,nline,&ierr);
            break;

        default:
            psm_error_msg (filename,nline,"Invalid element type");
            fclose (fp);
            return (1);
        }
        if (ierr) {
            fclose (fp);
            return (ierr);
        }
    }
    fclose (fp);
}
```

```
/*
 * FUNCTION:    psm_read_joint
 *
 *      This function reads the joint data from the model file for the given
 *      joint type and creates a PHIGS structure based on the data read.
 *
 * INPUT ARGUMENTS:
 *
 *      type:   joint type (EXTERNAL_REVOLUTE_JOINT for example)
 *      id:     identifier for the joint (already read from the model file)
 *      fp:     file pointer that is currently positioned to read the data
 *              for the given joint type
 *      filename:  complete name of the model file
 *      nline:  current line number being read in the model file
 *
 * OUTPUT ARGUMENTS:
 *
 *      ierr:   non-zero on error; 0 otherwise
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_joint (type,id,fp,filename,nline,ierr)
int type, id;
FILE *fp;
char *filename;
int nline, *ierr;
{
    char line[MAXLEN], errmsg[MAXLEN];
    Pfloat Ro, Ri, Len, Z1, Z2, Z, Ly, Lz;
    Pvector3 Rotation, Position;
    int count;

    *ierr = 0;
    if (fgets (line,MAXLEN,fp)) {
        switch (type) {
        case EXTERNAL_REVOLUTE_JOINT:
        case EXTERNAL_CYLINDRIC_JOINT:
        case EXTERNAL_PRISMATIC_JOINT:
            count = sscanf (line,"%f %f %f %f %f %f %f %f %f",
                            &Ro,&Ri,&Len,&Rotation.z,&Rotation.y,&Rotation.x,
                            &Position.x,&Position.y,&Position.z);
            break;

        case INTERNAL_REVOLUTE_JOINT:
        case INTERNAL_CYLINDRIC_JOINT:
        case INTERNAL_PRISMATIC_JOINT:
            count = sscanf (line,"%f %f %f %f %f %f %f %f %f",
                            &Ro,&Z1,&Z2,&Rotation.z,&Rotation.y,&Rotation.x,
                            &Position.x,&Position.y,&Position.z);
            break;

        case EXTERNAL_SPHERIC_JOINT:
            count = sscanf (line,"%f %f %f %f %f %f %f %f",
                            &Ro,&Ri,&Rotation.z,&Rotation.y,&Rotation.x,
                            &Position.x,&Position.y,&Position.z);
            break;

        case INTERNAL_SPHERIC_JOINT:
            count = sscanf (line,"%f %f %f %f",
                            &Ro,&Position.x,&Position.y,&Position.z);
            break;
```

```
        case PILLOW_BLOCK:
            count = sscanf (line,"%f %f %f %f %f %f %f %f %f %f %f",
                            &Ro,&Ri,&Z,&Ly,&Lz,
                            &Rotation.z,&Rotation.y,&Rotation.x,
                            &Position.x,&Position.y,&Position.z);
        }

        if (count == input[type].count) {
            if (DEBUG) psm_print_joint (type,Ro,Ri,Len,Z1,Z2,Z,Ly,Lz,
                                        Rotation,Position);
            psm_convert_to_radians (&Rotation);

            if (type == EXTERNAL_REVOLUTE_JOINT) {
                psm_external_revolute_joint (STRUCT_ELEMENTS+id,Ro,Ri,Len,
                                             Rotation,Position);

            } else if (type == INTERNAL_REVOLUTE_JOINT) {
                if (*ierr = psm_record_link_pos (id,filename,nline,Rotation,
                                                 Position,Z1,Z2)) return;
                psm_internal_revolute_joint (STRUCT_ELEMENTS+id,Ro,Z1,Z2,
                                             Rotation,Position);

            } else if (type == EXTERNAL_PRISMATIC_JOINT) {
                psm_external_prismatic_joint (STRUCT_ELEMENTS+id,Ro,Ri,Len,
                                              Rotation,Position);

            } else if (type == INTERNAL_PRISMATIC_JOINT) {
                if (*ierr = psm_record_link_pos (id,filename,nline,Rotation,
                                                 Position,Z1,Z2)) return;
                psm_internal_prismatic_joint (STRUCT_ELEMENTS+id,Ro,Z1,Z2,
                                              Rotation,Position);

            } else if (type == EXTERNAL_CYLINDRIC_JOINT) {
                psm_external_cylindric_joint (STRUCT_ELEMENTS+id,Ro,Ri,Len,
                                              Rotation,Position);

            } else if (type == INTERNAL_CYLINDRIC_JOINT) {
                if (*ierr = psm_record_link_pos (id,filename,nline,Rotation,
                                                 Position,Z1,Z2)) return;
                psm_internal_cylindric_joint (STRUCT_ELEMENTS+id,Ro,Z1,Z2,
                                              Rotation,Position);

            } else if (type == EXTERNAL_SPHERIC_JOINT) {
                psm_external_spheric_joint (STRUCT_ELEMENTS+id,Ro,Ri,
                                            Rotation,Position);

            } else if (type == INTERNAL_SPHERIC_JOINT) {
                psm_internal_spheric_joint (STRUCT_ELEMENTS+id,Ro,Position);

            } else if (type == PILLOW_BLOCK) {
                psm_pillow_block (STRUCT_ELEMENTS+id,Ro,Ri,Z,Ly,Lz,
                                  Rotation,Position);

            } else {
                psm_error_msg (filename,nline,
                     "Invalid joint type");
                *ierr = 1;
            }
            if (Ro > max_radius) max_radius = Ro;
            return;
        }
    }
    strcpy (errmsg,"Incomplete ");
    strcat (errmsg,input[type].name);
    psm_error_msg (filename,nline,errmsg);
    *ierr = 1;
}
```

```
/*
 * FUNCTION:    psm_convert_to_radians
 *
 *      psm_convert_to_radians converts an XYZ rotation in degrees to the
 *      corresponding rotation in radians.
 *
 * INPUT ARGUMENTS:
 *
 *      Rotation:  XYZ rotation in degrees
 *
 * OUTPUT ARGUMENTS:
 *
 *      Rotation:  XYZ rotation in radians
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_convert_to_radians (Rotation)
Pvector3 *Rotation;
{
    Rotation->x *= PI/180.0;
    Rotation->y *= PI/180.0;
    Rotation->z *= PI/180.0;
}
```

```
/*
 * FUNCTION:    psm_print_joint
 *
 *      This function prints (for debugging purposes) the joint data
 *      read while parsing the model data file.
 *
 * INPUT ARGUMENTS:
 *
 *      type:   joint type
 *      Ro:     outer radius
 *      Ri:     inner radius
 *      Len:    length of joint
 *      Z1:     first endpoint along the Z-axis
 *      Z2:     second endpoint along the Z-axis
 *      Z:      offset along the Z-axis
 *      Ly:     length dimension along the Y-axis
 *      Lz:     length dimension along the Z-axis
 *      Rotation:  XYZ rotation around the origin
 *      Position:  XYZ position (offset from the origin)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_print_joint (type,Ro,Ri,Len,Z1,Z2,Z,Ly,Lz,Rotation,Position)
int type;
Pfloat Ro, Ri, Len, Z1, Z2, Z, Ly, Lz;
Pvector3 Rotation, Position;
{
    printf ("psm_print_joint: %s\n",input[type].name);
```

```
    switch (type) {

    case EXTERNAL_REVOLUTE_JOINT:
    case EXTERNAL_CYLINDRIC_JOINT:
    case EXTERNAL_PRISMATIC_JOINT:
        printf ("\tRo = %f, Ri = %f, Len = %f\n",Ro,Ri,Len);
        printf ("\tRotation = %f %f %f\n",
                Rotation.z,Rotation.y,Rotation.x);
        printf ("\tPosition = %f %f %f\n",
                Position.x,Position.y,Position.z);
        break;

    case INTERNAL_REVOLUTE_JOINT:
    case INTERNAL_CYLINDRIC_JOINT:
    case INTERNAL_PRISMATIC_JOINT:
        printf ("\tRo = %f, Z1 = %f, Z2 = %f\n",Ro,Z1,Z2);
        printf ("\tRotation = %f %f %f\n",
                Rotation.z,Rotation.y,Rotation.x);
        printf ("\tPosition = %f %f %f\n",
                Position.x,Position.y,Position.z);
        break;

    case EXTERNAL_SPHERIC_JOINT:
        printf ("\tRo = %f, Ri = %f\n",Ro,Ri);
        printf ("\tRotation = %f %f %f\n",
                Rotation.z,Rotation.y,Rotation.x);
        printf ("\tPosition = %f %f %f\n",
                Position.x,Position.y,Position.z);
        break;

    case INTERNAL_SPHERIC_JOINT:
        printf ("\tRo = %f\n",Ro);
        printf ("\tPosition = %f %f %f\n",
                Position.x,Position.y,Position.z);
        break;

    case PILLOW_BLOCK:
        printf ("\tRo = %f, Ri = %f, Z = %f, Ly = %f, Lz = %f\n",
                Ro,Ri,Z,Ly,Lz);
        printf ("\tRotation = %f %f %f\n",
                Rotation.z,Rotation.y,Rotation.x);
        printf ("\tPosition = %f %f %f\n",
                Position.x,Position.y,Position.z);
    }
}
```

```
/*
 * FUNCTION:    psm_record_link_pos
 *
 *      psm_record_link_pos records the position of the endpoints of
 *      internal revolute, prismatic, and cylindric joints in the link
 *      position data array.  This information is later used to determine
 *      the approximate bounding volume for the spatial mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      id:     identifier of joint
 *      filename:  complete name of the model file
 *      nline:  current line number of the model file
 *      Rotation:  XYZ rotation around the origin
 *      Position:  XYZ position (offset from the origin)
 *      Z1:     first endpoint along the Z-axis
 *      Z2:     second endpoint along the Z-axis
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 if maximum link positions is exceeded
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_record_link_pos (id,filename,nline,Rotation,Position,Z1,Z2)
int id;
char *filename;
int nline;
Pvector3 Rotation, Position;
Pfloat Z1, Z2;
{
    int ierr;
    Ppoint3 pt3;
    Pmatrix3 trans_matrix;

    psm_build_tran3 (Rotation,Position,trans_matrix);
    link_pos[npos].id = id;
    pt3.x = pt3.y = 0.00; pt3.z = Z1;
    ptranpt3 (&pt3,trans_matrix,&ierr,&link_pos[npos++].pos);
    if (ierr) printf ("psm_record_link_pos: trans pt error = %d\n",ierr);
    if (npos >= MAXPOS) {
        psm_error_msg (filename,nline,"Maximum link positions exceeded");
        return (1);
    }

    link_pos[npos].id = id;
    pt3.x = pt3.y = 0.00; pt3.z = Z2;
    ptranpt3 (&pt3,trans_matrix,&ierr,&link_pos[npos++].pos);
    if (ierr) printf ("psm_record_link_pos: trans pt error = %d\n",ierr);
    if (npos >= MAXPOS) {
        psm_error_msg (filename,nline,"Maximum link positions exceeded");
        return (1);
    }
}
```

```
/*
 * FUNCTION:    psm_read_connecting_element
 *
 *      This function reads and parses the data for a connecting element
 *      and then creates a PHIGS structure based on the data read.
 *
 * INPUT ARGUMENTS:
 *
 *      id:     identifier for the connecting element (already read from the
 *              model file)
 *      fp:     file pointer that is currently positioned to read the data
 *              for the connecting element
 *      filename:  complete name of the model file
 *      nline:  current line number being read in the model file
 *
 * OUTPUT ARGUMENTS:
 *
 *      ierr:   non-zero on error; 0 otherwise
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_connecting_element (id,fp,filename,nline,ierr)
int id;
FILE *fp;
char *filename;
int nline, *ierr;
{
    char line[MAXLEN];
    Pfloat Ro;
    int i, Num;
    Pvector3 Position, Positions[10];

    *ierr = 0;
    fscanf (fp,"%f %d",&Ro,&Num);
    if (DEBUG) {
        printf ("psm_read_connecting_elements:\n");
        printf ("\tRo = %f, Num = %d\n",Ro,Num);
    }
    if (Num < 0 || Num > 10) {
        *ierr = 1;
        return;
    }

    for (i=0; i<Num; i++) {
        fscanf (fp,"%f %f %f",&Positions[i].x,
                &Positions[i].y,&Positions[i].z);
        if (DEBUG) printf ("\tlink positions = %f %f %f\n",
                           Positions[i].x,Positions[i].y,Positions[i].z);
        link_pos[npos].id = id;
        link_pos[npos++].pos = Positions[i];
        if (npos >= MAXPOS) {
            psm_error_msg (filename,nline,
                           "Maximum link positions exceeded");
            *ierr = 1;
            return;
        }
    }

    fgets (line,MAXLEN,fp);
    psm_connecting_link (STRUCT_ELEMENTS+id,Ro,Num,Positions);
    if (Ro > max_radius) max_radius = Ro;
}
```

```
/*
 * FUNCTION:    psm_read_mechanism_link
 *
 *      This function reads and parses the data for a mechanism link
 *      and then creates a PHIGS structure based on the data read.
 *
 * INPUT ARGUMENTS:
 *
 *      id:     identifier for the mechanism link (already read from the
 *              model file)
 *      fp:     file pointer that is currently positioned to read the data
 *              for the given joint type
 *      filename:  complete name of the model file
 *      descr:  user defined description of the mechanism link as read from
 *              the model file
 *      nline:  current line number being read in the model file
 *
 * OUTPUT ARGUMENTS:
 *
 *      ierr:   non-zero on error; 0 otherwise
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_mechanism_link (id,fp,filename,descr,nline,ierr)
int id;
FILE *fp;
char *filename, *descr;
int nline, *ierr;
{
    char line[MAXLEN], *p;
    Pfloat Ro;
    int i, j, len, Num, link_id;
    Pvector3 Position;
    static Pint integers[1] = {1};
    static Pintlst inclusion = {1,integers};

    *ierr = 0;
    if (model.nlinks >= MAXLINKS) {
        psm_error_msg (filename,nline,"Exceeded maximum link limit");
        *ierr = 1;
        return;
    }

    links[model.nlinks] = id;
    len = strlen (descr);
    p = (char *) malloc(len+8);
    strcpy (p,descr);
    strcat (p," picked");
    model.link_names[model.nlinks] = p;
    fscanf (fp,"%d",&Num);
    if (DEBUG) {
        printf ("psm_read_mechanism_link:\n");
        printf ("\tNum = %d\n",Num);
    }
    if (Num < 0 || Num > 10) {
        *ierr = 1;
        return;
    }
```

```
    link_id = STRUCT_ELEMENTS+id;
    popenstruct (link_id);
    pemptystruct (link_id);
    psetindivcsf (PCO_POLYLINE,PINDEXED);
    psetindivcsf (PCO_INTERIOR,PINDEXED);
    psetlinecolourind (MODEL_COLOR_INDEX+model.nlinks);
    psetintcolourind (MODEL_COLOR_INDEX+model.nlinks);
    paddnameset (&inclusion);
    psetpickid (model.nlinks);

    for (i=0; i<Num; i++) {
        fscanf (fp,"%d",&id);
        if (DEBUG) printf ("\tlink id = %d\n",id);
        pexecutestruct (STRUCT_ELEMENTS+id);
        for (j=0; j<npos; j++)
            if (link_pos[j].id==id)
                    link_pos[j].id = links[model.nlinks];
    }
    fgets (line,MAXLEN,fp);
    pclosestruct ();
    model.nlinks++;
}
```

```
/*
 * FUNCTION:    psm_read_positions
 *
 *      This function reads the model position (animation) data from the
 *      requested file.  The data is checked for basic integrity.  At each
 *      position, the set of link positions that represents the framework
 *      of the mechanism are transformed to the indicated position.  These
 *      link positions are then used to compute the bounding volume of the
 *      mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      filename:  complete filename (including extension) of the model file
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 on error; 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_read_positions (filename)
char *filename;
{
    FILE *fp, *fopen();
    char line[MAXLEN], descr[MAXLEN];
    int lenname, i, j, k;
    int type, id, Num, step, current_step, nline;
    int valid_link;
    Pvector3 Rotation, Position;
    Pmatrix3 trans_matrix;
    Ppoint3 pt3;
    Pint ierr;

    float max (); min ();
```

```
if ((fp=fopen(filename,"r")) == NULL) {
    psm_error_msg (filename,0,"CANNOT OPEN THE POSITION FILE");
    return (1);
} else if (fgets (descr,MAXLEN,fp) == NULL) {
    psm_error_msg (filename,0,"CANNOT READ THE POSITION FILE");
    return (1);
}

if (fgets (line,MAXLEN,fp) == NULL) {
    psm_error_msg (filename,0,"THE POSITION FILE IS TOO SHORT");
    fclose (fp);
    return (1);
} else {
    sscanf (line,"%d",&model.nsteps);
    if (DEBUG) printf ("psm_read_model: nsteps = %d\n",model.nsteps);
    psm_update_travel (DIALS_ANIMATE_VIEW_MAPPING,0,
                       (Pfloat)model.nsteps);
}

nline = 2;
current_step = 0;
while (fgets (line,MAXLEN,fp) != NULL) {
    nline++;
    if (8 == sscanf (line,"%d %d %f %f %f %f %f %f",&step,&id,
                     &Rotation.z,&Rotation.y,&Rotation.x,
                     &Position.x,&Position.y,&Position.z)) {
        if (DEBUG) {
            printf ("psm_read_positions:\n");
            printf ("\tstep = %d, link id = %d\n",step,id);
            printf ("\tRotation = %f %f %f\n",
                    Rotation.z,Rotation.y,Rotation.x);
            printf ("\tPosition = %f %f %f\n",
                         Position.x,Position.y,Position.z);
        }

        if (id < 0) {
            psm_error_msg (filename,nline,"Invalid ID (negative)");
            if (current_step != 0) pclosestruct ();
            fclose (fp);
            return (1);
        } else if (step <= 0 || step > model.nsteps) {
            psm_error_msg (filename,nline,"Invalid step number");
            if (current_step != 0) pclosestruct ();
            fclose (fp);
            return (1);
        }

        valid_link = FALSE;
        for (i=0; i<model.nlinks; i++)
            if (id == links[i]) valid_link = TRUE;
        if (!valid_link) {
            psm_error_msg (filename,nline,"Invalid link id");
            if (current_step != 0) pclosestruct ();
            fclose (fp);
            return (1);
        }

        if (step != current_step) {
            if (current_step != 0) pclosestruct ();
            current_step = step;
            popenstruct (STRUCT_MECHANISM+step);
            pemptystruct (STRUCT_MECHANISM+step);
        }

        psm_convert_to_radians (&Rotation);
        psm_build_tran3 (Rotation,Position,trans_matrix);
        psetlocaltran3 (trans_matrix,PREPLACE);
        pexecutestruct (STRUCT_ELEMENTS+id);
```

```
            for (i=0; i<npos; i++)
                if (link_pos[i].id == id) {
                    ptranpt3 (&link_pos[i].pos,trans_matrix,&ierr,&pt3);
                    if (ierr) printf
                        ("psm_read_model: transform pt error = %d\n",ierr);
                    if (pt3.x < min_bound.x) min_bound.x = pt3.x;
                    if (pt3.x > max_bound.x) max_bound.x = pt3.x;
                    if (pt3.y < min_bound.y) min_bound.y = pt3.y;
                    if (pt3.y > max_bound.y) max_bound.y = pt3.y;
                    if (pt3.z < min_bound.z) min_bound.z = pt3.z;
                    if (pt3.z > max_bound.z) max_bound.z = pt3.z;
                }

        } else {
            psm_error_msg (filename,nline,"Incomplete link position data");
            fclose (fp);
            return (1);
        }
    }
    pclosestruct ();
    fclose (fp);
}
```

```
/*
 * FUNCTION:    psm_error_msg
 *
 *      psm_error_msg displays the given error message data on the PHIGS
 *      message window of the main workstation window.
 *
 * INPUT ARGUMENTS:
 *
 *      filename:  name of the file where the error was detected
 *      nline:  line number of the file where the error was detected
 *      errmsg: message text describing the error detected
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_error_msg (filename,nline,errmsg)
char *filename;
int nline;
char *errmsg;
{
    int lenname, lenmsg;
    char text[100], text_nline[13];

    lenname = strlen (filename);
    lenname = (lenname < 20) ? lenname : 20;
    strcpy (text,"*** ");
    strncat (text,filename,lenname);
```

```
    lenmsg = strlen (errmsg);
    lenmsg = (lenmsg < (80-lenname)) ? lenmsg : (80-lenname);
    if (nline > 0) {
        sprintf (text_nline,", line%4d: ",nline);
        strcat (text,text_nline);
    } else {
        strcat (text,": ");
    }
    strncat (text,errmsg,lenmsg);
    strcat (text," ***");
    pmessage (WS_MAIN,text);
}
```

```
/*
 * FUNCTION:    psm_global_transformation
 *
 *      psm_global_transformation computes the global transformation for
 *      the spatial mechanism model based on the rotation, translation,
 *      and scaling data for the model.  The PHIGS structure for the model
 *      is updated if the transformation has changed.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 if no change in the global transformation; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_global_transformation ()
{
    Pvector3 rotate, translate, scale;
    static Pvector3 last_rotate, last_translate, last_scale;
    Pmatrix3 trans_matrix;
    Pint ierr;

    rotate    = model.transformation.rotate;
    translate = model.transformation.translate;
    scale     = model.transformation.scale;

    if (rotate.x < 0.0) rotate.x   = fmod(rotate.x,360.0) + 360.0;
    if (rotate.x > 360.0) rotate.x = fmod(rotate.x,360.0);
    if (rotate.y < 0.0) rotate.y   = fmod(rotate.y,360.0) + 360.0;
    if (rotate.y > 360.0) rotate.y = fmod(rotate.y,360.0);
    if (rotate.z < 0.0) rotate.z   = fmod(rotate.z,360.0) + 360.0;
    if (rotate.z > 360.0) rotate.z = fmod(rotate.z,360.0);
    if (scale.x < 0.0) scale.x     = 0.00;
    if (scale.y < 0.0) scale.y     = 0.00;
    if (scale.z < 0.0) scale.z     = 0.00;

    if (last_rotate.x == rotate.x &&
        last_rotate.y == rotate.y &&
        last_rotate.z == rotate.z &&
        last_translate.x == translate.x &&
        last_translate.y == translate.y &&
        last_translate.z == translate.z &&
        last_scale.x == scale.x &&
        last_scale.y == scale.y &&
        last_scale.z == scale.z) return (0);
```

```
    pbuildtran3 (&model.center,translate,rotate.x*PI/180.0,
                 rotate.y*PI/180.0,rotate.z*PI/180.0,scale,
                 &ierr,trans_matrix);
    if (ierr) printf
                ("psm_global_trans: build transformation error = %d\n",ierr);

    model.transformation.rotate    = rotate;
    model.transformation.translate = translate;
    model.transformation.scale     = scale;

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_MECHANISM);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_GLOBAL_TRANSFORMATION);
    poffsetelemptr (1);
    psetglobaltran3 (trans_matrix);
    pclosestruct ();

    last_rotate = rotate;
    last_translate = translate;
    last_scale = scale;
    return (1);
}
```

```
/*
 * FUNCTION:    psm_set_default_surface_properties
 *
 *      This function sets the default surface properties for the PHIGS
 *      model of the spatial mechanism.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_set_default_surface_properties ()
{
    Psurfprop surfprop;
    static Pcobundl white = {1.00,1.00,1.00};

    surfprop.ambient = 1.00;
    surfprop.diffuse = 1.00;
    surfprop.specular = 1.00;
    surfprop.colour.colour_model = PCM_RGB;
    surfprop.colour.colour = white;
    surfprop.exponent = 1.00;
    surfprop.transparency = 0.00;
    psetsurfprop (&surfprop);
}
```

```
/*
 * FUNCTION:    psm_surface_properties
 *
 *      psm_surface_properties updates the surface property data in the
 *      PHIGS structure for the mechanism model if the surface property
 *      data has changed since the last call to this function.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 if no change in the surface properties; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_surface_properties ()
{
    Psurfprop surfprop;
    static Psurfprop last_surfprop;

    if (model.surface.ambient < 0.0) model.surface.ambient   = 0.0;
    if (model.surface.ambient > 1.0) model.surface.ambient   = 1.0;
    if (model.surface.diffuse < 0.0) model.surface.diffuse   = 0.0;
    if (model.surface.diffuse > 1.0) model.surface.diffuse   = 1.0;
    if (model.surface.specular < 0.0) model.surface.specular = 0.0;
    if (model.surface.specular > 1.0) model.surface.specular = 1.0;
    if (model.surface.exponent < 0.0) model.surface.exponent = 0.0;

    surfprop.ambient             = model.surface.ambient;
    surfprop.diffuse             = model.surface.diffuse;
    surfprop.specular            = model.surface.specular;
    surfprop.colour.colour_model = PCM_RGB;
    surfprop.colour.colour       = model.surface.color.rgb;
    surfprop.exponent            = model.surface.exponent;
    surfprop.transparency        = 0.00;  /*  Not implemented  */

    if (last_surfprop.ambient == surfprop.ambient &&
        last_surfprop.diffuse == surfprop.diffuse &&
        last_surfprop.specular == surfprop.specular &&
        last_surfprop.exponent == surfprop.exponent &&
        last_surfprop.colour.colour.x == surfprop.colour.colour.x &&
        last_surfprop.colour.colour.y == surfprop.colour.colour.y &&
        last_surfprop.colour.colour.z == surfprop.colour.colour.z)
        return (0);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_MECHANISM);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_SURFACE_PROPERTIES);
    poffsetelemptr (1);
    psetsurfprop (&surfprop);
    pclosestruct ();

    last_surfprop = surfprop;
    return (1);
}
```

```
/*
 * FUNCTION:    psm_animate
 *
 *      psm_animate is used to simulate movement of the spatial mechanism
 *      model by displaying the model at different positions as defined in
 *      the position file for the model.
 *
 * INPUT ARGUMENTS:
 *
 *      step:   the step (position number) of the spatial mechanism model
 *              to position the model
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 if no change in the step; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_animate (step)
Pfloat *step;
{
    int i, j, k;
    Pint istep;
    static Pint last_step = 1;

    istep = *step - 1;
    if (istep >= model.nsteps)
        istep = istep % model.nsteps;
    else if (*step < 1.0)
        istep = istep % model.nsteps + model.nsteps - 1;
    istep += 1;
    *step = istep;

    if (istep == last_step) return (0);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_MECHANISM);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_STRUCT_MECHANISM);
    poffsetelemptr (1);
    pexecutestruct (STRUCT_MECHANISM+istep);
    pclosestruct ();

    last_step = istep;
    return (1);
}
```

```
/*
 * FUNCTION:    psm_shading_method
 *
 *      psm_shading_method set the method for shading the spatial mechanism
 *      model.
 *
 * INPUT ARGUMENTS:
 *
 *      method: shading method
 *              PSH_NONE:   none, flat, or constant shading
 *              PSH_COLOUR: Gouraud, bilinear interpolated, or smooth shading
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_shading_method (method)
Pint method;
{
    if (method != PSH_NONE && method != PSH_COLOUR)
        return (1);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_MECHANISM);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_SHADING_METHOD);
    poffsetelemptr (1);
    psetintshademethod (method);
    pclosestruct ();
    popenstruct (STRUCT_AXES);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_SHADING_METHOD);
    poffsetelemptr (1);
    psetintshademethod (method);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_depth_cueing
 *
 *      psm_depth_cueing updates the depth cueing data in the PHIGS
 *      structure for the mechanism model if the depth cueing data has
 *      changed since the last call to this function.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 if no change in the depth cueing parameters; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
int psm_depth_cueing ()
{
    int i;
    Pdcuebundl dcue_rep;
    static Pdcuebundl last_dcue_rep;
    Pfloat planes[2];

    if (model.depth_cue_on)
        dcue_rep.mode = PALLOWED;
    else
        dcue_rep.mode = PSUPPRESSED;

    for (i=0; i<2; i++) {
        planes[i] = max (0.0,model.depth_cue.planes[i]);
        planes[i] = min (1.0,planes[i]);
        model.depth_cue.scaling[i] = max (0.0,model.depth_cue.scaling[i]);
        model.depth_cue.scaling[i] = min (1.0,model.depth_cue.scaling[i]);
        dcue_rep.scaling[i] = model.depth_cue.scaling[i];
    }
    dcue_rep.refplanes[0] = model.depth_cue.planes[0] =
                                        max (planes[0],planes[1]);
    dcue_rep.refplanes[1] = model.depth_cue.planes[1] =
                                        min (planes[0],planes[1]);
    dcue_rep.colour.colour_model = PCM_RGB;
    dcue_rep.colour.colour = model.depth_cue.color.rgb;

    if (last_dcue_rep.refplanes[0] != dcue_rep.refplanes[0] ||
        last_dcue_rep.refplanes[1] != dcue_rep.refplanes[1]) {
        if (model.depth_cue_on) psm_view_lines ();
    } else if (last_dcue_rep.mode == dcue_rep.mode &&
        last_dcue_rep.scaling[0] == dcue_rep.scaling[0] &&
        last_dcue_rep.scaling[1] == dcue_rep.scaling[1] &&
        last_dcue_rep.colour.colour.x == dcue_rep.colour.colour.x &&
        last_dcue_rep.colour.colour.y == dcue_rep.colour.colour.y &&
        last_dcue_rep.colour.colour.z == dcue_rep.colour.colour.z)
        return (0);

    psetdcuerep (WS_MAIN,MODEL_DEPTH_CUE,&dcue_rep);
    psetdcuerep (WS_AUX,MODEL_DEPTH_CUE,&dcue_rep);

    last_dcue_rep = dcue_rep;
    return (1);
}
```

Module: psm_objects.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "model.h"
#include "lights.h"
#include "input.h"

extern PSM_model_rendering model;
extern PSM_light_sources lights;
```

```
/*
 * FUNCTION:    psm_build_tran3
 *
 *      psm_build_tran3 builds a 3-D transformation matrix in the following
 *      order:
 *          Z rotation
 *          Y rotation
 *          X rotation
 *          XYZ translation
 *
 * INPUT ARGUMENTS:
 *
 *      rotation:  XYZ rotation "vector"
 *      shift:     XYZ shift (translation) vector
 *
 * OUTPUT ARGUMENTS:
 *
 *      trans_matrix:  transformation matrix generated
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_build_tran3 (rotation,shift,trans_matrix)
Pvector3 rotation, shift;
Pmatrix3 trans_matrix;
{
    Pint ierr;
    Pmatrix3 rotx_matrix, roty_matrix, rotz_matrix, shift_matrix;

    protatez (rotation.z,&ierr,rotz_matrix);
    if (ierr) printf ("psm_build_tran3: rotate z error #%d\n",ierr);
    protatey (rotation.y,&ierr,roty_matrix);
    if (ierr) printf ("psm_build_tran3: rotate y error #%d\n",ierr);
    protatex (rotation.x,&ierr,rotx_matrix);
    if (ierr) printf ("psm_build_tran3: rotate x error #%d\n",ierr);
    ptranslate3 (&shift,&ierr,shift_matrix);
    if (ierr) printf ("psm_build_tran3: translate 3 error #%d\n",ierr);

    pcomposematrix3 (roty_matrix,rotz_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_build_tran3: compose mtx error #%d\n",ierr);
    pcomposematrix3 (rotx_matrix,trans_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_build_tran3: compose mtx error #%d\n",ierr);
    pcomposematrix3 (shift_matrix,trans_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_build_tran3: compose mtx error #%d\n",ierr);
}
```

```
/*
 * FUNCTION:    psm_axes
 *
 *      psm_axes creates the PHIGS structure containing the red, green, and
 *      and blue axes set used to show the current orientation of the
 *      spatial mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_axes ()
{
    Pint ierr;
    Pmatrix3 rot_matrix;
    Pvector3 Positions[4];
    Pvector3 Rotation, Position;

    static Ppoint3 origin = {0.00,0.00,0.00};
    static Pvector3 scale = {1.00,1.00,1.00};
    Pvector3 shift, rotation;
    Pfloat x_angle, y_angle, z_angle;
    Pmatrix3 trans_matrix;

    psm_object_cylinder (STRUCT_AXIS_CYLINDER,0.1,-0.4,0.8,TRUE,FALSE);
    psm_object_cone (STRUCT_AXIS_CONE,0.2,0.8,1.2);

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_AXIS_ARROW);
    pexecutestruct (STRUCT_AXIS_CYLINDER);
    pexecutestruct (STRUCT_AXIS_CONE);
    pclosestruct ();

    popenstruct (STRUCT_AXES);

    psethlhsrid (ON);
    psetnormreorientmode (PNOFLIP);
    psetfaceprocmode (PDIST_NO,PCULL_BACK);
    psetintstyle (PSOLID);
    plabel (LABEL_SHADING_METHOD);
    psetintshademethod (PSH_COLOUR);
    psetintlightmethod (PLM_SPECULAR);
    plabel (LABEL_LIGHT_SOURCE_STATE);
    plabel (LABEL_LIGHT_SOURCE_STATE);
    psm_set_default_surface_properties ();

    psetintcolourind (RED);
    psetlinecolourind (RED);
    protatey (PI/2.0,&ierr,rot_matrix);
    if (ierr) printf ("psm_axes: rotate y error #%d\n",ierr);
    psetlocaltran3 (rot_matrix,PREPLACE);
    pexecutestruct (STRUCT_AXIS_ARROW);
```

```
    psetintcolourind (GREEN);
    psetlinecolourind (GREEN);
    protatex (-PI/2.0,&ierr,rot_matrix);
    if (ierr) printf ("psm_axes: rotate x error #%d\n",ierr);
    psetlocaltran3 (rot_matrix,PREPLACE);
    pexecutestruct (STRUCT_AXIS_ARROW);

    psetintcolourind (BLUE);
    psetlinecolourind (BLUE);
    protatex (0.0,&ierr,rot_matrix);
    if (ierr) printf ("psm_axes: rotate x error #%d\n",ierr);
    psetlocaltran3 (rot_matrix,PREPLACE);
    pexecutestruct (STRUCT_AXIS_ARROW);

    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_init_primitives
 *
 *      psm_init_primitives initializes the six graphics primitives used
 *      to model the spatial mechanism, axes set, and the light sources.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_primitives ()
{
    psm_unit_cylinder ();
    psm_unit_cone ();
    psm_unit_prism ();
    psm_unit_half_sphere ();
    psm_unit_circle ();
    psm_unit_frustum ();
}
```

```
/*
 * FUNCTION:    psm_unit_cylinder
 *
 *      psm_unit_cylinder generates the PHIGS structure containing a
 *      wireframe or polygonal unit cylinder with its local axis along
 *      the Z-axis and end faces at Z=0 and Z=1.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_cylinder ()
{
    int i;
    Pfloat theta;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    Pptnorm3 ptnorm[4];
    Ppoint3 pt3[4];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_CYLINDER);
    pemptystruct (STRUCT_UNIT_CYLINDER);

    ptnorm[2].point.x = ptnorm[3].point.x = 1.00;
    ptnorm[2].point.y = ptnorm[3].point.y = 0.00;
    ptnorm[2].point.z = 0.00; ptnorm[3].point.z = 1.00;
    ptnorm[2].normal.x = 1.00;
    ptnorm[2].normal.y = 0.00;
    ptnorm[2].normal.z = 0.00;
    ptnorm[3].normal = ptnorm[2].normal;
    vdata.ptnorm = ptnorm;
    for (i=1; i<=4*model.nsegments; i++) {
        theta = PI*i/(2.0*model.nsegments);
        ptnorm[0] = ptnorm[3];
        ptnorm[1] = ptnorm[2];
        ptnorm[2].point.x = ptnorm[2].normal.x = cos(theta);
        ptnorm[2].point.y = ptnorm[2].normal.y = sin(theta);
        ptnorm[3] = ptnorm[2];
        ptnorm[3].point.z = 1.00;
        if (model.shading_method == CHOICE_WIREFRAME) {
            pt3[0] = ptnorm[1].point;
            pt3[1] = ptnorm[2].point;
            pt3[2] = ptnorm[3].point;
            pt3[3] = ptnorm[0].point;
            ppolyline3 (4,pt3);
        } else {
            pfillarea3data (PFA_NONE,PVERT_NORMAL,PCM_RGB,&fadata,4,&vdata);
        }
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_unit_cone
 *
 *      psm_unit_cone generates the PHIGS structure containing a
 *      wireframe or polygonal unit cone with its local axis along
 *      the Z-axis, base at Z=0, and point at Z=1.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_cone ()
{
    int i;
    Pfloat theta;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    Pptnorm3 ptnorm[3];
    Ppoint3 pt3[3];
    Ppoint3 base[64];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_CONE);
    pemptystruct (STRUCT_UNIT_CONE);

    ptnorm[1].point.x = 1.00;
    ptnorm[1].point.y = 0.00;
    ptnorm[1].point.z = 0.00;
    ptnorm[2].point.x = 0.00;
    ptnorm[2].point.y = 0.00;
    ptnorm[2].point.z = 1.00;
    ptnorm[1].normal.x = 1.00;
    ptnorm[1].normal.y = 0.00;
    ptnorm[1].normal.z = 1.00;
    ptnorm[2].normal = ptnorm[1].normal;
    pt3[2] = ptnorm[2].point;
    vdata.ptnorm = ptnorm;

    for (i=1; i<=4*model.nsegments; i++) {
        theta = PI*i/(2.0*model.nsegments);
        ptnorm[0] = ptnorm[1];
        ptnorm[1].point.x = ptnorm[1].normal.x = cos(theta);
        ptnorm[1].point.y = ptnorm[1].normal.y = sin(theta);
        theta = PI*((float)i-0.5)/(2.0*model.nsegments);
        ptnorm[2].normal.x = cos(theta);
        ptnorm[2].normal.y = sin(theta);
        fadata.gnormal = ptnorm[2].normal;

        if (model.shading_method == CHOICE_WIREFRAME) {
            pt3[0] = ptnorm[0].point;
            pt3[1] = ptnorm[1].point;
            ppolyline3 (3,pt3);
        } else {
            pfillarea3data (PFA_NORMAL,PVERT_NORMAL,PCM_RGB,
                            &fadata,3,&vdata);
            base[i-1] = ptnorm[1].point;
        }
```

```
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_unit_prism
 *
 *      psm_unit_prism generates the PHIGS structure containing a
 *      wireframe or polygonal unit prism with its local axis along
 *      the Z-axis and end faces at Z=0 and Z=1.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_prism ()
{
    int i;
    Pfloat theta;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    Ppoint3 pt3[4], side1[4], side2[4];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_PRISM);
    pemptystruct (STRUCT_UNIT_PRISM);

    theta = -PI/4.0;
    pt3[1].x = cos(theta);
    pt3[1].y = sin(theta);
    pt3[1].z = 0.00;
    pt3[2].x = cos(theta);
    pt3[2].y = sin(theta);
    pt3[2].z = 1.00;
    vdata.pt = pt3;

    for (i=0; i<4; i++) {
        theta = PI*i/2.0 + PI/4.0;
        pt3[0] = pt3[1];
        pt3[3] = pt3[2];
        pt3[1].x = pt3[2].x = cos(theta);
        pt3[1].y = pt3[2].y = sin(theta);
        if (model.shading_method == CHOICE_WIREFRAME) {
            ppolyline3 (4,pt3);
        } else {
            theta = PI*i/2.0;
            fadata.gnormal.x = cos(theta);
            fadata.gnormal.y = sin(theta);
            fadata.gnormal.z = 0.00;
            pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_RGB,&fadata,4,&vdata);
            side1[i] = pt3[0];
            side2[i] = pt3[3];
        }
    }
```

```
    if (model.shading_method != CHOICE_WIREFRAME) {
        fadata.gnormal.x = 0.00;
        fadata.gnormal.y = 0.00;
        fadata.gnormal.z = -1.00;
        vdata.pt = side1;
        pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_RGB,&fadata,4,&vdata);
        fadata.gnormal.z = 1.00;
        vdata.pt = side2;
        pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_RGB,&fadata,4,&vdata);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_unit_half_sphere
 *
 *      psm_unit_half_sphere generates the PHIGS structure containing a
 *      wireframe or polygonal unit half sphere with its center at the
 *      origin, its base in the XY plane, and its dome on the positive
 *      Z side.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_half_sphere ()
{
    int i, j, ierr;
    Pfloat theta0, theta1, phi0, phi1;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    Pptnorm3 ptnorm[4];
    Ppoint3 pt3[4];
    Pmatrix3 rotation_matrix;

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_1_8TH_SPHERE);
    pemptystruct (STRUCT_UNIT_1_8TH_SPHERE);

    for (i=1; i<=model.nsegments; i++) {
        phi0 = PI*i/(2.0*model.nsegments);
        phi1 = PI*(i-1)/(2.0*model.nsegments);
        theta0 = 0.00;
        theta1 = PI/(2.0*(model.nsegments-i+1));
        ptnorm[1].point.x = ptnorm[1].normal.x = cos(phi1);
        ptnorm[1].point.y = ptnorm[1].normal.y = 0.00;
        ptnorm[1].point.z = ptnorm[1].normal.z = sin(phi1);
        ptnorm[2].point.x = ptnorm[2].normal.x = cos(phi0);
        ptnorm[2].point.y = ptnorm[2].normal.y = 0.00;
        ptnorm[2].point.z = ptnorm[2].normal.z = sin(phi0);
        ptnorm[3].point.x = ptnorm[3].normal.x = cos(phi1)*cos(theta1);
        ptnorm[3].point.y = ptnorm[3].normal.y = cos(phi1)*sin(theta1);
        ptnorm[3].point.z = ptnorm[3].normal.z = sin(phi1);
```

```
        if (model.shading_method == CHOICE_WIREFRAME) {
            pt3[0] = ptnorm[2].point;
            pt3[1] = ptnorm[3].point;
            pt3[2] = ptnorm[1].point;
            ppolyline3 (3,pt3);
        } else {
            vdata.ptnorm = &ptnorm[1];
            pfillarea3data (PFA_NONE,PVERT_NORMAL,PCM_RGB,&fadata,3,&vdata);
        }

        for (j=1; j<=model.nsegments-i; j++) {
            ptnorm[0] = ptnorm[2];
            ptnorm[1] = ptnorm[3];
            theta0 = PI*j/(2.0*(model.nsegments-i));
            theta1 = PI*(j+1)/(2.0*(model.nsegments-i+1));
            ptnorm[2].point.x = ptnorm[2].normal.x = cos(phi0)*cos(theta0);
            ptnorm[2].point.y = ptnorm[2].normal.y = cos(phi0)*sin(theta0);
            ptnorm[2].point.z = ptnorm[2].normal.z = sin(phi0);
            ptnorm[3].point.x = ptnorm[3].normal.x = cos(phi1)*cos(theta1);
            ptnorm[3].point.y = ptnorm[3].normal.y = cos(phi1)*sin(theta1);
            ptnorm[3].point.z = ptnorm[3].normal.z = sin(phi1);

            if (model.shading_method == CHOICE_WIREFRAME) {
                pt3[0] = ptnorm[3].point;
                pt3[1] = ptnorm[1].point;
                pt3[2] = ptnorm[2].point;
                pt3[3] = ptnorm[3].point;
                ppolyline3 (4,pt3);
            } else {
                vdata.ptnorm = ptnorm;
                pfillarea3data (PFA_NONE,PVERT_NORMAL,PCM_RGB,&fadata,3,&vdata);
                vdata.ptnorm = &ptnorm[1];
                pfillarea3data (PFA_NONE,PVERT_NORMAL,PCM_RGB,&fadata,3,&vdata);
            }
        }
    }
    pclosestruct ();

    popenstruct (STRUCT_UNIT_HALF_SPHERE);
    pemptystruct (STRUCT_UNIT_HALF_SPHERE);

    for (i=0; i<4; i++) {
        protatez (PI*i/2.0,&ierr,rotation_matrix);
        if (ierr) printf ("psm_unit_half_sphere: rotate z error #%d\n",ierr);
        psetlocaltran3 (rotation_matrix,PREPLACE);
        pexecutestruct (STRUCT_UNIT_1_8TH_SPHERE);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_unit_circle
 *
 *      psm_unit_circle generates the PHIGS structure containing a
 *      wireframe or polygonal unit circle in the XY plane with its
 *      center point at the origin, and its normal in the positive Z
 *      direction.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_circle ()
{
    int i;
    Pfloat theta;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    Ppoint3 circle[64];
    Ppoint3 pt3[2];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_CIRCLE);
    pemptystruct (STRUCT_UNIT_CIRCLE);

    for (i=0; i<4*model.nsegments; i++) {
        theta = PI*i/(2.0*model.nsegments);
        circle[i].x = cos(theta);
        circle[i].y = sin(theta);
        circle[i].z = 0.00;
    }

    if (model.shading_method == CHOICE_WIREFRAME) {
        vdata.pt = pt3;
        pt3[0] = circle[0];
        pt3[1] = circle[2*model.nsegments];
        ppolyline3 (2,pt3);
        pt3[0] = circle[model.nsegments];
        pt3[1] = circle[3*model.nsegments];
        ppolyline3 (2,pt3);

    } else {
        fadata.gnormal.x = 0.00;
        fadata.gnormal.y = 0.00;
        fadata.gnormal.z = 1.00;
        vdata.pt = circle;
        pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,
                        4*model.nsegments,&vdata);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_unit_frustum
 *
 *      psm_unit_frustum generates the PHIGS structure containing a
 *      wireframe or polygonal unit frustum with its local axis along
 *      the Y-axis, its large base at Y=-1, and its smaller top at Y=0.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_unit_frustum ()
{
    int i;
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    static Ppoint3 corners[8] = {1.00,-1.00,1.00,   1.00,-1.00,-1.00,
                                -1.00,-1.00,-1.00, -1.00,-1.00,1.00,
                                 -0.50,0.00,0.50,      -0.50,0.00,-0.50,
                                 0.50,0.00,-0.50,      0.50,0.00,0.50};
    Ppoint3 pt3[4];

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_UNIT_FRUSTUM);
    pemptystruct (STRUCT_UNIT_FRUSTUM);

    vdata.pt = pt3;
    for (i=0; i<4; i++) {
        pt3[0] = corners[i];
        pt3[1] = corners[(i+1)%4];
        pt3[2] = corners[7-(i+1)%4];
        pt3[3] = corners[7-i];
        if (model.shading_method == CHOICE_WIREFRAME) {
            ppolyline3 (4,pt3);
        } else {
            fadata.gnormal.x = cos(PI*i/2.0);
            fadata.gnormal.y = 0.50;
            fadata.gnormal.z = -sin(PI*i/2.0);
            pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,4,&vdata);
        }
    }

    if (model.shading_method != CHOICE_WIREFRAME) {
        vdata.pt = corners;
        fadata.gnormal.x = 0.00;
        fadata.gnormal.y = -1.00;
        fadata.gnormal.z = 0.00;
        pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,4,&vdata);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_object_cylinder
 *
 *      psm_object_cylinder generates a PHIGS structure containing a
 *      unit cylinder that has been scaled and positioned with its local
 *      axis along the Z-axis.  Depending on the value of endcap1 and
 *      endcap2, the structure can contain unit circles that are scaled
 *      and positioned at the base and top of the cylinder.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      radius:    radius of the cylinder
 *      z1:        base face of the cylinder along the Z-axis
 *      z2:        top face of the cylinder along the Z-axis
 *      endcap1:   true if cylinder has a closed base
 *      endcap2:   true if cylinder has a closed top
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 on error (z1=z2); 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_object_cylinder (struct_id,radius,z1,z2,endcap1,endcap2)
Pint struct_id;
Pfloat radius, z1, z2;
Pint endcap1, endcap2;
{
    Pfloat ftemp;
    int itemp;
    static Ppoint3 origin = {0.00,0.00,0.00};
    Pvector3 shift, scale;
    Pfloat x_angle, y_angle, z_angle;
    Pint ierr;
    Pmatrix3 trans_matrix;

    if (z1 == z2) {
        return (1);
    } else if (z1 > z2) {
        ftemp = z1;
        z1 = z2;
        z2 = ftemp;
        itemp = endcap1;
        endcap1 = endcap2;
        endcap2 = itemp;
    }

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);

    scale.x = scale.y = radius; scale.z = z2-z1;
    x_angle = y_angle = z_angle = 0.00;
    shift.x = shift.y = 0.00; shift.z = z1;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_cylinder: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_CYLINDER);
```

```
    if (endcap1) {
        y_angle = PI;
        pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                     trans_matrix);
        if (ierr) printf
            ("psm_object_cylinder: build trans error #%d\n",ierr);
        psetlocaltran3 (trans_matrix,PREPLACE);
        pexecutestruct (STRUCT_UNIT_CIRCLE);
    }

    if (endcap2) {
        y_angle = 0.00;
        shift.z = z2;
        pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                     trans_matrix);
        if (ierr) printf
            ("psm_object_cylinder: build trans error #%d\n",ierr);
        psetlocaltran3 (trans_matrix,PREPLACE);
        pexecutestruct (STRUCT_UNIT_CIRCLE);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_object_cone
 *
 *      psm_object_cone generates a PHIGS structure containing a
 *      unit cone that has been scaled and positioned with its local
 *      axis along the Z-axis.  Also, this structure will contain a unit
 *      circle scaled and positioned at the base of the cone.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      radius:    radius of the base of the cone
 *      z1:        base face of the cone along the Z-axis
 *      z2:        top of the cone along the Z-axis
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 on error (z1=z2); 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_object_cone (struct_id,radius,z1,z2)
Pint struct_id;
Pfloat radius, z1, z2;
{
    static Ppoint3 origin = {0.00,0.00,0.00};
    Pvector3 shift, scale;
    Pfloat x_angle, y_angle, z_angle;
    Pint ierr;
    Pmatrix3 trans_matrix;
    Pfloat ftemp;
```

```
    x_angle = y_angle = z_angle = 0.00;
    shift.x = 0.00; shift.y = 0.00; shift.z = z1;
    if (z1 == z2) {
        return (1);
    } else if (z1 > z2) {
        y_angle = PI;
        ftemp = z1;
        z1 = z2;
        z2 = ftemp;
    }

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);

    scale.x = radius; scale.y = radius; scale.z = z2-z1;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_cone: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_CONE);

    y_angle += PI;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_cone: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_CIRCLE);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_object_prism
 *
 *      psm_unit_prism generates a PHIGS structure containing a
 *      unit prism that has been scaled and positioned with its local
 *      axis along the Z-axis.  Z1 (the base) must be less than Z2.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      diagonal:  distance from the centerline of the prism to an edge
 *      z1:        base face of the prism along the Z-axis
 *      z2:        top face of the prism along the Z-axis
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 on error (z1>=z2); 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_object_prism (struct_id,diagonal,z1,z2)
Pint struct_id;
Pfloat diagonal, z1, z2;
{
    static Ppoint3 origin = {0.00,0.00,0.00};
    Pvector3 shift, scale;
    static Pfloat x_angle = 0.00;
    static Pfloat y_angle = 0.00;
    static Pfloat z_angle = 0.00;
    Pint ierr;
    Pmatrix3 trans_matrix;
    Pfloat ftemp;

    if (z1 = z2) {
        return (1);
    } else of (z1 > z2) {
        ftemp = z1;
        z1 = z2;
        z2 = ftemp;
    }

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);

    scale.x = diagonal; scale.y = diagonal; scale.z = z2-z1;
    shift.x = 0.00; shift.y = 0.00; shift.z = z1-0.00;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_prism: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_PRISM);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_object_half_sphere
 *
 *      psm_unit_half_sphere generates a PHIGS structure containing a
 *      unit half sphere that has been scaled, rotated, and positioned, with
 *      its center along the Z-axis.  Before the rotation, the dome of
 *      the half sphere is on the positive Z side relative to the center.
 *      parallel to the XY plane
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      radius:    radius of the half sphere
 *      z1:        center of the half sphere along the Z-axis
 *      y_rot:     rotation of the half sphere around the Y-axis
 *      endcap:    true if the half sphere has a closed base
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      0 for no error
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_object_half_sphere (struct_id,radius,z1,y_rot,endcap)
Pint struct_id;
Pfloat radius, z1, y_rot;
Pint endcap;
{
    static Ppoint3 origin = {0.00,0.00,0.00};
    Pvector3 shift, scale;
    static Pfloat x_angle = 0.00;
    Pfloat y_angle;
    static Pfloat z_angle = 0.00;
    Pint ierr;
    Pmatrix3 trans_matrix;

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);

    scale.x = radius; scale.y = radius; scale.z = radius;
    y_angle = y_rot;
    shift.x = 0.00; shift.y = 0.00; shift.z = z1;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_sphere: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_HALF_SPHERE);

    if (endcap) {
        y_angle += PI;
        pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                     trans_matrix);
        if (ierr) printf
                ("psm_object_half_sphere: build trans error #%d\n",ierr);
        psetlocaltran3 (trans_matrix,PREPLACE);
        pexecutestruct (STRUCT_UNIT_CIRCLE);
    }
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_object_frustum
 *
 *      psm_unit_frustum generates a PHIGS structure containing a
 *      unit frustum that has been scaled and positioned along the Z-axis.
 *      The large base is originally at Y=-1, and the smaller top is
 *      originally at Y=0.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      z:         center of the top face along the Z-axis
 *      lx:        length (scale) in the X direction
 *      ly:        length (scale) in the Y direction
 *      lz:        length (scale) in the Z direction
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      1 on error (lx, ly, or lz <= 0.0); 0 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_object_frustum (struct_id,z,lx,ly,lz)
Pint struct_id;
Pfloat z, lx, ly, lz;
{
    static Ppoint3 origin = {0.00,0.00,0.00};
    Pvector3 shift, scale;
    static Pfloat x_angle = 0.00;
    static Pfloat y_angle = 0.00;
    static Pfloat z_angle = 0.00;
    Pint ierr;
    Pmatrix3 trans_matrix;

    if (lx<=0.0 || ly<=0.0 || lz<=0.0) return (1);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);

    scale.x = lx; scale.y = ly; scale.z = lz;
    shift.x = 0.00; shift.y = 0.00; shift.z = z;
    pbuildtran3 (&origin,&shift,x_angle,y_angle,z_angle,&scale,&ierr,
                 trans_matrix);
    if (ierr) printf ("psm_object_frustum: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_UNIT_FRUSTUM);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_external_revolute_joint
 *
 *      psm_external_revolute_joint generates a PHIGS structure containing
 *      two cylinders (centered at the origin with their local axes along the
 *      Z-axis) that have the specified radii, length, rotation about the
 *      origin, and position.  The smaller (inner) cylinder will have a
 *      radius greater than Ri but less than Ro.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius of the joint
 *      Ri:        inner radius of the joint (assumed Ri<Ro)
 *      Len:       length of the joint
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_external_revolute_joint (struct_id,Ro,Ri,Len,Rotation,Position)
Pint struct_id;
Pfloat Ro, Ri, Len;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pfloat leno;
    Pint id_cyl_out, id_cyl_in;
```

```
    id_cyl_out = struct_id_free++;
    leno = Len-(Ro-Ri);
    psm_object_cylinder (id_cyl_out,Ro,-leno/2.0,leno/2.0,TRUE,TRUE);
    id_cyl_in = struct_id_free++;
    psm_object_cylinder (id_cyl_in,(Ro+Ri)/2.0,-Len/2.0,Len/2.0,TRUE,TRUE);

    psm_build_tran3 (Rotation,Position,trans_matrix);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_cyl_out);
    pexecutestruct (id_cyl_in);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_internal_revolute_joint
 *
 *      psm_internal_revolute_joint generates a PHIGS structure containing
 *      one cylinder (with a half sphere at Z1 and an end face at Z2) that
 *      has the specified radii, length, rotation about the origin, and
 *      position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        radius of the joint
 *      Z1:        end of the joint with the half sphere
 *      Z2:        other end of the joint with the circular face
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_internal_revolute_joint (struct_id,Ro,Z1,Z2,Rotation,Position)
Pint struct_id;
Pfloat Ro, Z1, Z2;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_cyl, id_sph;
    Pfloat rotate;

    id_cyl = struct_id_free++;
    psm_object_cylinder (id_cyl,Ro,Z1,Z2,FALSE,TRUE);
    id_sph = struct_id_free++;
    if (Z1 > Z2)
        rotate = 0.00;
    else
        rotate = PI;
    psm_object_half_sphere (id_sph,Ro,Z1,rotate,FALSE);

    psm_build_tran3 (Rotation,Position,trans_matrix);
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_cyl);
    pexecutestruct (id_sph);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_external_cylindric_joint
 *
 *      psm_external_cylindric_joint generates a PHIGS structure containing
 *      one cylinder (centered at the origin with its local axes along the
 *      Z-axis) that has the specified radius, length, rotation about the
 *      origin, and position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius of the joint
 *      Ri:        inner radius of the joint (not used)
 *      Len:       length of the joint
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_external_cylindric_joint (struct_id,Ro,Ri,Len,Rotation,Position)
Pint struct_id;
Pfloat Ro, Ri, Len;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_cyl;

    id_cyl = struct_id_free++;
    psm_object_cylinder (id_cyl,Ro,-Len/2.0,Len/2.0,TRUE,TRUE);

    psm_build_tran3 (Rotation,Position,trans_matrix);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_cyl);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_internal_cylindric_joint
 *
 *      psm_internal_cylindric_joint generates a PHIGS structure containing
 *      one cylinder (with a half sphere at Z1 and an end face at Z2) that
 *      has the specified radii, length, rotation about the origin, and
 *      position.  The internal cylindric joint is identical to an internal
 *      revolute joint, so for simplicity this function calls
 *      psm_internal_revolute_joint.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        radius of the joint
 *      Z1:        end of the joint with the half sphere
 *      Z2:        other end of the joint with the circular face
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_internal_cylindric_joint (struct_id,Ro,Z1,Z2,Rotation,Position)
Pint struct_id;
Pfloat Ro, Z1, Z2;
Pvector3 Rotation, Position;
{
    psm_internal_revolute_joint (struct_id,Ro,Z1,Z2,Rotation,Position);
}
```

```
/*
 * FUNCTION:    psm_external_prismatic_joint
 *
 *      psm_external_prismatic_joint generates a PHIGS structure containing
 *      one prism (centered at the origin with its local axes along the
 *      Z-axis) that has the specified radius (diagonal), length, rotation
 *      about the origin, and position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius (diagonal) of the joint
 *      Ri:        inner radius (diagonal) of the joint (not used)
 *      Len:       length of the joint
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_external_prismatic_joint (struct_id,Ro,Ri,Len,Rotation,Position)
Pint struct_id;
Pfloat Ro, Ri, Len;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_pri;

    id_pri = struct_id_free++;
    psm_object_prism (id_pri,Ro,-Len/2.0,Len/2.0);

    psm_build_tran3 (Rotation,Position,trans_matrix);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_pri);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_internal_prismatic_joint
 *
 *      psm_internal_prismatic_joint generates a PHIGS structure containing
 *      one prism (with a sphere at Z1) that has the specified radius
 *      (diagonal of the prism), length, rotation about the origin, and
 *      position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        radius (diagonal) of the joint
 *      Z1:        end of the joint with the sphere (two half spheres)
 *      Z2:        other end of the joint
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_internal_prismatic_joint (struct_id,Ro,Z1,Z2,Rotation,Position)
Pint struct_id;
Pfloat Ro, Z1, Z2;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_pri, id_sph1, id_sph2;

    id_pri = struct_id_free++;
    psm_object_prism (id_pri,Ro,Z1,Z2);
    id_sph1 = struct_id_free++;
    psm_object_half_sphere (id_sph1,Ro,Z1,0.00,FALSE);
    id_sph2 = struct_id_free++;
    psm_object_half_sphere (id_sph2,Ro,Z1,PI,FALSE);

    psm_build_tran3 (Rotation,Position,trans_matrix);
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_pri);
    pexecutestruct (id_sph1);
    pexecutestruct (id_sph2);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_external_spheric_joint
 *
 *      psm_external_spheric_joint generates a PHIGS structure containing
 *      a half sphere (centered at the origin with the dome of the half
 *      sphere on the negative Z side of the center) that has the specified
 *      radius, rotation about the origin, and position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius of the joint
 *      Ri:        inner radius of the joint (not used)
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_external_spheric_joint (struct_id,Ro,Ri,Rotation,Position)
Pint struct_id;
Pfloat Ro, Ri;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_sph;

    id_sph = struct_id_free++;
    psm_object_half_sphere (id_sph,Ro,0.00,PI,TRUE);

    psm_build_tran3 (Rotation,Position,trans_matrix);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_sph);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_internal_spheric_joint
 *
 *      psm_internal_spheric_joint generates a PHIGS structure containing
 *      two half spheres (centered at the origin) that have the specified
 *      radius and position.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius of the joint
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_internal_spheric_joint (struct_id,Ro,Position)
Pint struct_id;
Pfloat Ro;
Pvector3 Position;
{
    static Pvector3 Rotation = {0.00,0.00,0.00};
    Pmatrix3 trans_matrix;
    Pint id_sph1, id_sph2;

    id_sph1 = struct_id_free++;
    psm_object_half_sphere (id_sph1,Ro,0.00,0.00,FALSE);
    id_sph2 = struct_id_free++;
    psm_object_half_sphere (id_sph2,Ro,0.00,PI,FALSE);

    psm_build_tran3 (Rotation,Position,trans_matrix);

    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_sph1);
    pexecutestruct (id_sph2);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_connecting_link
 *
 *      psm_connecting_link generates a PHIGS structure containing
 *      multiple cylinders and half spheres (with the given radius) which
 *      connect the point-to-point positions specified.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        radius of the cylinders and half spheres
 *      Num:       number of positions connected
 *      Positions: XYZ positions that the link connects
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_connecting_link (struct_id,Ro,Num,Positions)
Pint struct_id;
Pfloat Ro;
Pint Num;
Pvector3 *Positions;
{
    int i;
    Pvector3 Rotation;
    Pmatrix3 trans_matrix;
    Pint id_cyl, id_sph;
    Pfloat dx, dy, dz, len;

    for (i=0; i<Num-1; i++) {
        dx = Positions[i+1].x - Positions[i].x;
        dy = Positions[i+1].y - Positions[i].y;
        dz = Positions[i+1].z - Positions[i].z;
        len = sqrt(dx*dx + dy*dy + dz*dz);
        id_cyl = struct_id_free++;
        psm_object_cylinder (id_cyl,Ro,0.00,len,FALSE,FALSE);
        if (i<(Num-2)) {
            id_sph = struct_id_free++;
            psm_object_half_sphere (id_sph,Ro,len,0.00,FALSE);
        }

        Rotation.z = 0.00;
        len = sqrt(dy*dy + dz*dz);
        if (len > 0.00) {
            Rotation.x = -acos(dz/len);
            if (dy != 0.00) Rotation.x *= dy/fabs(dy);
        } else
            Rotation.x = 0.00;
        dz = len;
        len = sqrt(dx*dx + dz*dz);
        if (len > 0.00) {
            Rotation.y = acos(dz/len);
            if (dx != 0.00) Rotation.y *= dx/fabs(dx);
        } else
            Rotation.y = 0.00;
        psm_build_tran3 (Rotation,Positions[i],trans_matrix);
```

```
        pseteditmode (PEDIT_INSERT);
        popenstruct (struct_id);
        if (i==0) pemptystruct (struct_id);
        psetlocaltran3 (trans_matrix,PREPLACE);
        pexecutestruct (id_cyl);
        if (i<(Num-2)) pexecutestruct (id_sph);
        pclosestruct ();
    }
}
```

```
/*
 * FUNCTION:    psm_pillow_block
 *
 *      psm_pillow_block generates a PHIGS structure containing a frustum,
 *      and two cylinders (centered at the origin with their local axes
 *      along the Z-axis) that have the specified radii, initial Z offset,
 *      length, rotation about the origin, and position.  The smaller (inner)
 *      cylinder will have a radius greater than Ri but less than Ro.
 *
 * INPUT ARGUMENTS:
 *
 *      struct_id: PHIGS id for the structure created
 *      Ro:        outer radius of the cylindric section
 *      Ri:        inner radius of the cylindric (assumed Ri<Ro)
 *      Z:         center point of the cylinders along the Z-axis (before
 *                 the rotation is applied)
 *      Ly:        length (height) of the frustum in the Y direction
 *      Lz:        length of the joint (cylinders and frustum) in the
 *                 Z direction
 *      Rotation:  XYZ rotation about the origin (applied in Z, Y, X order)
 *      Position:  XYZ translation
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_pillow_block (struct_id,Ro,Ri,Z,Ly,Lz,Rotation,Position)
Pint struct_id;
Pfloat Ro, Ri, Z, Ly, Lz;
Pvector3 Rotation, Position;
{
    Pmatrix3 trans_matrix;
    Pint id_fru, id_cyl1, id_cyl2;
    Pfloat Z1, Z2;

    id_fru = struct_id_free++;
    psm_object_frustum (id_fru,Z,2.0*Ro,Ly,Lz);
    id_cyl1 = struct_id_free++;
    Z1 = Z - Lz/2.0;
    Z2 = Z + Lz/2.0;
    psm_object_cylinder (id_cyl1,Ro,Z1,Z2,TRUE,TRUE);
    id_cyl2 = struct_id_free++;
    Z1 = Z - Lz*3.0/4.0;
    Z2 = Z + Lz*3.0/4.0;
    psm_object_cylinder (id_cyl2,(Ro+Ri)/2.0,Z1,Z2,TRUE,TRUE);

    psm_build_tran3 (Rotation,Position,trans_matrix);
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (struct_id);
    pemptystruct (struct_id);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_fru);
    pexecutestruct (id_cyl1);
    pexecutestruct (id_cyl2);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_light_sources
 *
 *      psm_light_sources creates the base models for the infinite, point,
 *      and spot light sources.
 *
 * INPUT ARGUMENTS:
 *
 *      scale:  single combined XYZ scale
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_light_sources (scale)
Pfloat scale;
{
    psm_infinite_light (scale);
    psm_point_light (scale);
    psm_spot_light (scale);
}
```

```
/*
 * FUNCTION:    psm_infinite_light
 *
 *      psm_infinite_light generates a PHIGS structure containing
 *      a cylinder and a cone with the center (where the base of the cone
 *      and cylinder meet) at the origin and the resulting arrow pointing
 *      along the Z-axis in the positive direction.
 *
 * INPUT ARGUMENTS:
 *
 *      scale:  single combined XYZ scale
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_infinite_light (scale)
Pfloat scale;
{
    Pvector3 scale_vector;
    Pmatrix3 trans_matrix;
    Pint ierr;
    Pint id_cyl, id_cone;

    id_cyl = struct_id_free++;
    psm_object_cylinder (id_cyl,0.5,-8.0,0.0,TRUE,FALSE);
    id_cone = struct_id_free++;
    psm_object_cone (id_cone,1.0,0.0,4.0);

    scale_vector.x = scale_vector.y = scale_vector.z = scale;
    pscale3 (&scale_vector,&ierr,trans_matrix);
    if (ierr) printf ("psm_infinite_light: scale error #%d\n",ierr);

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_INFINITE_LIGHT);
    pemptystruct (STRUCT_INFINITE_LIGHT);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_cyl);
    pexecutestruct (id_cone);
    pexecutestruct (STRUCT_LIGHT_RAY);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_point_light
 *
 *      psm_point_light generates a PHIGS structure containing
 *      two half spheres with the center at the origin.
 *
 * INPUT ARGUMENTS:
 *
 *      scale:  single combined XYZ scale
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_point_light (scale)
Pfloat scale;
{
    Pvector3 scale_vector;
    Pmatrix3 trans_matrix;
    Pint ierr;
    Pint id_sph1, id_sph2;

    id_sph1 = struct_id_free++;
    psm_object_half_sphere (id_sph1,1.0,0.0,0.0,FALSE);
    id_sph2 = struct_id_free++;
    psm_object_half_sphere (id_sph2,1.0,0.0,PI,FALSE);

    scale_vector.x = scale_vector.y = scale_vector.z = scale;
    pscale3 (&scale_vector,&ierr,trans_matrix);
    if (ierr) printf ("psm_spot_light: scale error #%d\n",ierr);
```

```
    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_POINT_LIGHT);
    pemptystruct (STRUCT_POINT_LIGHT);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_sph1);
    pexecutestruct (id_sph2);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_spot_light
 *
 *      psm_spot_light generates a PHIGS structure containing
 *      a cone with the base at the origin and the top on the negative
 *      Z-axis.
 *
 * INPUT ARGUMENTS:
 *
 *      scale:  single combined XYZ scale
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_spot_light (scale)
Pfloat scale;
{
    Pvector3 scale_vector;
    Pmatrix3 trans_matrix;
    Pint ierr;
    Pint id_cone;

    id_cone = struct_id_free++;
    psm_object_cone (id_cone,1.0,0.0,-4.0);

    scale_vector.x = scale_vector.y = scale_vector.z = scale;
    pscale3 (&scale_vector,&ierr,trans_matrix);
    if (ierr) printf ("psm_spot_light: scale error #%d\n",ierr);

    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_SPOT_LIGHT);
    pemptystruct (STRUCT_SPOT_LIGHT);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (id_cone);
    pexecutestruct (STRUCT_LIGHT_RAY);
    pclosestruct ();
}
```

Module: psm_views.c

```
#include <stdio.h>
#include <phigs.h>
#include <math.h>
#include "common.h"
#include "chrom_coord.h"
#include "model.h"

extern PSM_model_rendering model;

float max (), min ();

/*
 * FUNCTION:    psm_init_views
 *
 *      psm_init_views initializes views for the color models, view
 *      divider lines (for multi-view display), and the dials.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_views ()
{
    psm_view_rgb_cube (WS_MAIN,VIEW_RGB_CUBE,SIDE_VIEW);
    psm_view_rgb_bars (WS_MAIN,VIEW_RGB_BARS,SIDE_VIEW);
    psm_view_hsv_cone (WS_MAIN,VIEW_HSV_CONE,SIDE_VIEW);
    psm_view_cie_triangle (WS_MAIN,VIEW_CIE_TRIANGLE,SIDE_VIEW);
    psm_view_dividers (WS_MAIN,PART_VIEW);
    psm_view_dials (WS_DIALS);
}

Pint ierr;
Pviewrep  vrep;
Pviewrep3 vrep3;
Pviewmapping  mapping;
Pviewmapping3 mapping3;
```

```
/*
 * FUNCTION:    psm_view_rgb_cube
 *
 *      psm_view_rgb_cube creates a view for the rgb color cube.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *      view_index: PHIGS view index number
 *      location:   location of the view (PART_VIEW, FULL_VIEW, or SIDE_VIEW)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_rgb_cube (ws_id,view_index,location)
Pint ws_id, view_index, location;
{
    static Ppoint3  vrp =          { 0.00, 0.00, 0.00};
    static Pvector3 vpn =          { 1.00, 1.00, 1.00};
    static Pvector3 vup =          {-1.00, 1.00,-1.00};
    static Plimit   window =      {-0.95, 0.95,-1.00, 0.90};
    static Plimit3  viewport[3] = { 0.01, 0.74, 0.27, 1.00, 0.00, 1.00,
                                    0.00, 1.00, 0.00, 1.00, 0.00, 1.00,
                                    0.75, 0.99, 0.76, 1.00, 0.00, 1.00};
    static Ppoint3  prp =          { 0.00, 0.00, 8.00};

    pevalvieworientationmatrix3 (&vrp,&vpn,&vup,&ierr,
                                 vrep3.orientation_matrix);
    if (ierr) printf ("psm_view_rgb_cube: view orient error #%d\n",ierr);
    mapping3.window = window;
    mapping3.viewport = viewport[location];
    mapping3.proj = PPERSPECTIVE;
    mapping3.prp = prp;
    mapping3.view_plane =  0.00;
    mapping3.back_plane = -1.00;
    mapping3.front_plane = 2.00;
    pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
    if (ierr) printf ("psm_view_rgb_cube: view mapping error #%d\n",ierr);
    vrep3.clip_limit = viewport[location];
    vrep3.clip_xy =    PCLIP;
    vrep3.clip_back =  PCLIP;
    vrep3.clip_front = PCLIP;
    psetviewrep3 (ws_id,view_index,&vrep3);
}
```

```
/*
 * FUNCTION:    psm_view_rgb_bars
 *
 *      psm_view_rgb_bars creates a view for the rgb color bars.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *      view_index: PHIGS view index number
 *      location:   location of the view (PART_VIEW, FULL_VIEW, or SIDE_VIEW)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_rgb_bars (ws_id,view_index,location)
Pint ws_id, view_index, location;
{
    static Ppoint  vrp =          { 0.00, 0.00};
    static Pvector vup =          { 0.00, 1.00};
    static Plimit  window =         {-0.05, 1.05,-0.15, 0.95};
    static Plimit  viewport[3] =  { 0.01, 0.74, 0.27, 1.00,
                                    0.00, 1.00, 0.00, 1.00,
                                    0.75, 0.99, 0.52, 0.76};

    pevalvieworientationmatrix (&vrp,&vup,&ierr,
                                vrep.orientation_matrix);
    if (ierr) printf ("psm_view_rgb_bars: view orient error #%d\n",ierr);
    mapping.window = window;
    mapping.viewport = viewport[location];
    pevalviewmappingmatrix (&mapping,&ierr,vrep.mapping_matrix);
    if (ierr) printf ("psm_view_rgb_bars: view mapping error #%d\n",ierr);
    vrep.clip_limit = viewport[location];
    vrep.clip_xy =    PCLIP;
    psetviewrep (ws_id,view_index,&vrep);
    psetviewtraninputpri (ws_id,view_index,VIEW_DEFAULT,PHIGHER);
}
```

```
/*
 * FUNCTION:    psm_view_hsv_cone
 *
 *      psm_view_hsv_cone creates a view for the hsv color cone.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *      view_index: PHIGS view index number
 *      location:   location of the view (PART_VIEW, FULL_VIEW, or SIDE_VIEW)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_view_hsv_cone (ws_id,view_index,location)
Pint ws_id, view_index, location;
{
    static Ppoint3  vrp =          { 0.00, 0.00, 0.00};
    static Pvector3 vpn =          { 0.00, 1.00, 1.50};
    static Pvector3 vup =          { 0.00, 1.50,-1.00};
    static Plimit   window =      {-1.50, 1.50,-1.60, 2.05};
    static Plimit3  viewport[3] = { 0.01, 0.74, 0.27, 1.00, 0.00, 1.00,
                                    0.00, 1.00, 0.00, 1.00, 0.00, 1.00,
                                    0.75, 0.99, 0.28, 0.52, 0.00, 1.00};
    static Ppoint3  prp =          { 0.00, 0.00, 8.00};

    pevalvieworientationmatrix3 (&vrp,&vpn,&vup,&ierr,
                                 vrep3.orientation_matrix);
    if (ierr) printf ("psm_view_hsv_cone: view orient error #%d\n",ierr);
    mapping3.window = window;
    mapping3.viewport = viewport[location];
    mapping3.proj = PPERSPECTIVE;
    mapping3.prp = prp;
    mapping3.view_plane =  0.00;
    mapping3.back_plane = -1.50;
    mapping3.front_plane = 1.50;
    pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
    if (ierr) printf ("psm_view_hsv_cone: view mapping error #%d\n",ierr);
    vrep3.clip_limit = viewport[location];
    vrep3.clip_xy =    PCLIP;
    vrep3.clip_back =  PCLIP;
    vrep3.clip_front = PCLIP;
    psetviewrep3 (ws_id,view_index,&vrep3);
}
```

```
/*
 * FUNCTION:    psm_view_cie_triangle
 *
 *      psm_view_cie_triangle creates a view for the cie color triangle.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *      view_index: PHIGS view index number
 *      location:   location of the view (PART_VIEW, FULL_VIEW, or SIDE_VIEW)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_cie_triangle (ws_id,view_index,location)
Pint ws_id, view_index, location;
{
    static Ppoint  vrp =          { 0.00, 0.00};
    static Pvector vup =          { 0.00, 1.00};
    static Plimit  viewport[3] =  { 0.01, 0.74, 0.27, 1.00,
                                    0.00, 1.00, 0.00, 1.00,
                                    0.75, 0.99, 0.04, 0.28};
    Plimit window;
    Pfloat xdel, ydel, edge;

    float max ();
```

```
    pevalvieworientationmatrix (&vrp,&vup,&ierr,
                                vrep.orientation_matrix);
    if (ierr) printf ("psm_view_cie_triangle: view orient error #%d\n",ierr);
    xdel = XR - XB;
    ydel = YG - YB;
    edge = max (xdel,ydel) * 1.10;
    window.xmin = XB - (edge-xdel)/2.0;
    window.xmax = XR + (edge-xdel)/2.0;
    window.ymin = YB - (edge-ydel)/2.0;
    window.ymax = YG + (edge-ydel)/2.0;
    mapping.window = window;
    mapping.viewport = viewport[location];
    pevalviewmappingmatrix (&mapping,&ierr,vrep.mapping_matrix);
    if (ierr) printf ("psm_view_cie_triangle: view map error #%d\n",ierr);
    vrep.clip_limit = viewport[location];
    vrep.clip_xy =    PCLIP;
    psetviewrep (ws_id,view_index,&vrep);
    psetviewtraninputpri (ws_id,view_index,VIEW_DEFAULT,PHIGHER);
}
```

```
/*
 * FUNCTION:    psm_view_axes
 *
 *      psm_view_axes creates view(s) for the XYZ axes set.  The location
 *      of the view(s) depends on the workstation window.  WS_MAIN may be
 *      set to multi-view; therefore, this function would create 4 axes
 *      views.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representations are assigned to
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_axes (ws_id)
Pint ws_id;
{
    int i;
    static Ppoint3  vrp =          { 0.00, 0.00, 0.00};
    static Pvector3 vpn[4] =      { 1.00, 1.00, 1.00,
                                    0.00, 1.00, 0.00,
                                    0.00, 0.00, 1.00,
                                    1.00, 0.00, 0.00};
    static Pvector3 vup[4] =      {-1.00, 1.00,-1.00,
                                    0.00, 0.00,-1.00,
                                    0.00, 1.00, 0.00,
                                    0.00, 1.00, 0.00};
    static Plimit   window =      {-1.50, 1.50,-1.50, 1.50};
    static Ppoint   viewref[4][2]={ 0.74,0.635, 1.00, 0.50,
                                   0.375,0.635, 0.50, 0.50,
                                   0.375, 0.27, 0.50, 0.00,
                                    0.74, 0.27, 1.00, 0.00};
    static Ppoint3  prp =          { 0.00, 0.00, 4.00};
    Plimit3 viewport;
    Pint multi_view, location, projection;
```

```
    if (ws_id == WS_AUX) {
        multi_view = FALSE;
        location = FULL_VIEW;
    } else {
        multi_view = model.multi_view;
        location = model.view_location;
    }
    projection = model.view_projection;

    mapping3.window = window;
    mapping3.proj = projection;
    mapping3.prp = prp;
    mapping3.view_plane =  0.00;
    mapping3.back_plane = -1.50;
    mapping3.front_plane = 1.50;
    vrep3.clip_xy =    PCLIP;
    vrep3.clip_back =  PCLIP;
    vrep3.clip_front = PCLIP;

    if (multi_view) {
        pevalvieworientationmatrix3 (&vrp,&model.view_orient.vpn,
                                     &model.view_orient.vup,&ierr,
                                     vrep3.orientation_matrix);
        if (ierr) printf ("psm_view_axes: view orient error #%d\n",ierr);
        viewport.xmin = viewref[0][location].x - 0.1;
        viewport.xmax = viewref[0][location].x;
        viewport.ymin = viewref[0][location].y;
        viewport.ymax = viewref[0][location].y + 0.1;
        viewport.zmin = 0.00;
        viewport.zmax = 1.00;
        mapping3.viewport = viewport;
        pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
        if (ierr) printf ("psm_view_axes: view map error #%d\n",ierr);
        vrep3.clip_limit = viewport;
        psetviewrep3 (ws_id,VIEW_AXES_MAIN,&vrep3);
        for (i=1; i<=3; i++) {
            pevalvieworientationmatrix3 (&vrp,&vpn[i],&vup[i],&ierr,
                                         vrep3.orientation_matrix);
            if (ierr) printf ("psm_view_axes: view orient error #%d\n",ierr);
            viewport.xmin = viewref[i][location].x - 0.1;
            viewport.xmax = viewref[i][location].x;
            viewport.ymin = viewref[i][location].y;
            viewport.ymax = viewref[i][location].y + 0.1;
            viewport.zmin = 0.00;
            viewport.zmax = 1.00;
            mapping3.viewport = viewport;
            pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
            if (ierr) printf ("psm_view_axes: view map error #%d\n",ierr);
            vrep3.clip_limit = viewport;
            psetviewrep3 (ws_id,VIEW_AXES_MAIN+i,&vrep3);
        }
    } else {
        pevalvieworientationmatrix3 (&vrp,&model.view_orient.vpn,
                                     &model.view_orient.vup,&ierr,
                                     vrep3.orientation_matrix);
        if (ierr) printf ("psm_view_axes: view orient error #%d\n",ierr);
        viewport.xmin = viewref[3][location].x - 0.1;
        viewport.xmax = viewref[3][location].x;
        viewport.ymin = viewref[3][location].y;
        viewport.ymax = viewref[3][location].y + 0.1;
        viewport.zmin = 0.00;
        viewport.zmax = 1.00;
        mapping3.viewport = viewport;
        pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
        if (ierr) printf ("psm_view_axes: view mapping error #%d\n",ierr);
        vrep3.clip_limit = viewport;
        psetviewrep3 (ws_id,VIEW_AXES_MAIN,&vrep3);
    }
}
```

```
/*
 * FUNCTION:    psm_view_lights
 *
 *      psm_view_lights creates a view for the light source and viewing
 *      volume display.  The location of the view depends on the workstation
 *      window.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_lights (ws_id)
Pint ws_id;
{
    int i;
    static Pvector3 vpn =         { 0.00, 0.00, 1.00};
    static Pvector3 vup =         { 0.00, 1.00, 0.00};
    static Ppoint viewref[2] =  { 0.74, 0.27, 1.00, 0.00};
    static Pfloat viewedge[2] = { 0.73,1.00};
    Plimit3 viewport;
    Pint location;

    if (ws_id == WS_AUX)
        location = FULL_VIEW;
    else
        location = model.view_location;

    mapping3.window.xmin = mapping3.window.ymin =
                    -model.view_edge_len/model.view_map.zoom_lights;
    mapping3.window.xmax = mapping3.window.ymax =
                     model.view_edge_len/model.view_map.zoom_lights;
    mapping3.proj = PPARALLEL;
    mapping3.prp.x = 0.00;
    mapping3.prp.y = 0.00;
    mapping3.prp.z = model.view_edge_len*4.0;
    mapping3.view_plane =  0.00;
    mapping3.back_plane = -model.view_edge_len*2.0;
    mapping3.front_plane = model.view_edge_len*2.0;
    vrep3.clip_xy =    PCLIP;
    vrep3.clip_back =  PCLIP;
    vrep3.clip_front = PCLIP;

    pevalvieworientationmatrix3 (&model.center,&vpn,&vup,&ierr,
                                 vrep3.orientation_matrix);
    if (ierr) printf ("psm_view_lights: view orient error #%d\n",ierr);
    viewport.xmin = viewref[location].x - viewedge[location];
    viewport.xmax = viewref[location].x;
    viewport.ymin = viewref[location].y;
    viewport.ymax = viewref[location].y + viewedge[location];
    viewport.zmin = 0.00;
    viewport.zmax = 1.00;
    mapping3.viewport = viewport;
    pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
    if (ierr) printf ("psm_view_lights: view mapping error #%d\n",ierr);
    vrep3.clip_limit = viewport;
    psetviewrep3 (ws_id,VIEW_LIGHTS,&vrep3);
}
```

```
/*
 * FUNCTION:    psm_view_model
 *
 *      psm_view_model creates view(s) for the spatial mechanism model.
 *      The location of the view(s) depends on the workstation window.
 *      WS_MAIN may be set to multi-view; therefore, this function would
 *      create 4 model view.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representations are assigned to
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_model (ws_id)
Pint ws_id;
{
    int i;
    static Pvector3 vpn[4] =      { 1.00, 1.00, 1.00,
                                    0.00, 1.00, 0.00,
                                    0.00, 0.00, 1.00,
                                    1.00, 0.00, 0.00};
    static Pvector3 vup[4] =      {-1.00, 1.00,-1.00,
                                    0.00, 0.00,-1.00,
                                    0.00, 1.00, 0.00,
                                    0.00, 1.00, 0.00};
    static Ppoint   viewref[4][2]={ 0.74,0.635, 1.00, 0.50,
                                   0.375,0.635, 0.50, 0.50,
                                   0.375, 0.27, 0.50, 0.00,
                                    0.74, 0.27, 1.00, 0.00};
    static Pfloat   viewedge[2] = { 0.365,0.50};
    Plimit3 viewport;
    Pint multi_view, location, projection;

    if (ws_id == WS_AUX) {
        multi_view = FALSE;
        location = FULL_VIEW;
    } else {
        multi_view = model.multi_view;
        location = model.view_location;
    }
    projection = model.view_projection;

    mapping3.window.xmin = mapping3.window.ymin =
                        -model.view_edge_len/(2.0*model.view_map.zoom_model);
    mapping3.window.xmax = mapping3.window.ymax =
                         model.view_edge_len/(2.0*model.view_map.zoom_model);
    mapping3.proj = projection;
    mapping3.prp = model.view_map.prp;
    mapping3.front_plane = model.view_map.front_plane;
    mapping3.view_plane = model.view_map.view_plane;
    mapping3.back_plane = model.view_map.back_plane;
    vrep3.clip_xy =    PCLIP;
    vrep3.clip_front = PCLIP;
    vrep3.clip_back =  PCLIP;
```

```
    if (multi_view) {
        pevalvieworientationmatrix3 (&model.view_orient.vrp,
                                     &model.view_orient.vpn,
                                     &model.view_orient.vup,
                                     &ierr,vrep3.orientation_matrix);
        if (ierr) printf ("psm_view_model: view orient err #%d\n",ierr);
        viewport.xmin = viewref[0][location].x - viewedge[location];
        viewport.xmax = viewref[0][location].x;
        viewport.ymin = viewref[0][location].y;
        viewport.ymax = viewref[0][location].y + viewedge[location];
        viewport.zmin = 0.00;
        viewport.zmax = 1.00;
        mapping3.viewport = viewport;
        pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
        if (ierr) printf ("psm_view_model: view map error #%d\n",ierr);
        vrep3.clip_limit = viewport;
        psetviewrep3 (ws_id,VIEW_MODEL_MAIN,&vrep3);

        for (i=1; i<=3; i++) {
            pevalvieworientationmatrix3 (&model.view_orient.vrp,
                                         &vpn[i],&vup[i],&ierr,
                                         vrep3.orientation_matrix);
            if (ierr) printf ("psm_view_model: view orient err #%d\n",ierr);
            viewport.xmin = viewref[i][location].x - viewedge[location];
            viewport.xmax = viewref[i][location].x;
            viewport.ymin = viewref[i][location].y;
            viewport.ymax = viewref[i][location].y + viewedge[location];
            mapping3.viewport = viewport;
            pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
            if (ierr) printf ("psm_view_model: view map error #%d\n",ierr);
            vrep3.clip_limit = viewport;
            psetviewrep3 (ws_id,VIEW_MODEL_MAIN+i,&vrep3);
        }

    } else {
        pevalvieworientationmatrix3 (&model.view_orient.vrp,
                                     &model.view_orient.vpn,
                                     &model.view_orient.vup,
                                     &ierr,vrep3.orientation_matrix);
        if (ierr) printf ("psm_view_model: view orient error #%d\n",ierr);
        viewport.xmin = viewref[3][location].x - 2.0*viewedge[location];
        viewport.xmax = viewref[3][location].x;
        viewport.ymin = viewref[3][location].y;
        viewport.ymax = viewref[3][location].y + 2.0*viewedge[location];
        viewport.zmin = 0.00;
        viewport.zmax = 1.00;
        mapping3.viewport = viewport;
        pevalviewmappingmatrix3 (&mapping3,&ierr,vrep3.mapping_matrix);
        if (ierr) printf ("psm_view_model: view mapping error #%d\n",ierr);
        vrep3.clip_limit = viewport;
        psetviewrep3 (ws_id,VIEW_MODEL_MAIN,&vrep3);
    }
}
```

```
/*
 * FUNCTION:    psm_view_dividers
 *
 *      psm_view_dividers creates a view for the multi-view dividing lines.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *      location:   location of the view (PART_VIEW or FULL_VIEW)
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_dividers (ws_id,location)
Pint ws_id, location;
{
    static Ppoint  vrp =            { 0.00, 0.00};
    static Pvector vup =            { 0.00, 1.00};
    static Plimit  window =           { 0.00, 1.00, 0.00, 1.00};
    static Plimit  viewport[2] =  { 0.01, 0.74, 0.27, 1.00,
                                    0.00, 1.00, 0.00, 1.00};

    pevalvieworientationmatrix (&vrp,&vup,&ierr,vrep.orientation_matrix);
    if (ierr) printf ("psm_view_dividers: view orient error #%d\n",ierr);
    mapping.window = window;
    mapping.viewport = viewport[location];
    pevalviewmappingmatrix (&mapping,&ierr,vrep.mapping_matrix);
    if (ierr) printf ("psm_view_dividers: view mapping error #%d\n",ierr);
    vrep.clip_limit = viewport[location];
    vrep.clip_xy =    PCLIP;
    psetviewrep (ws_id,VIEW_LINES,&vrep);
}
```

```
/*
 * FUNCTION:    psm_view_dials
 *
 *      psm_view_dials creates a view for the simulated dials display.
 *
 * INPUT ARGUMENTS:
 *
 *      ws_id:      workstation id the view representation is assigned to
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_view_dials (ws_id)
Pint ws_id;
{
    int i;
```

```
    static Ppoint  vrp =      { 0.00, 0.00};
    static Pvector vup =      { 0.00, 1.00};
    static Plimit  window =     {-0.50, 0.50,-0.60, 0.40};
    static Pfloat  xmin =        0.035;
    static Pfloat  ymax =        0.98;
    static Pfloat  xdel =        0.31;
    static Pfloat  ydel =        0.31;
    Plimit viewport;

    pevalvieworientationmatrix (&vrp,&vup,&ierr,vrep.orientation_matrix);
    if (ierr) printf ("psm_view_dials: view orient error #%d\n",ierr);
    mapping.window = window;
    vrep.clip_xy =    PCLIP;

    for (i=0; i<9; i++) {
        viewport.xmin = xmin + xdel*(i%3);
        viewport.xmax = viewport.xmin + xdel;
        viewport.ymax = ymax - ydel*(i/3);
        viewport.ymin = viewport.ymax - ydel;
        mapping.viewport = viewport;
        pevalviewmappingmatrix (&mapping,&ierr,vrep.mapping_matrix);
        if (ierr) printf ("psm_view_dials: view mapping error #%d\n",ierr);
        vrep.clip_limit = viewport;
        psetviewrep (ws_id,VIEW_DIALS+i,&vrep);
    psetviewtraninputpri (ws_id,VIEW_DIALS+i,VIEW_DEFAULT,PHIGHER);
    }
}
```

```
/*
 * FUNCTION:    psm_model_view_mapping
 *
 *      psm_model_view_mapping updates the view mapping data (viewing
 *      volume) for viewing the mechanism model if the view
 *      mapping data has changed since to last call to this function.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *       0 if no change in the view mapping parameters; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_model_view_mapping ()
{
    int redraw;
    static Pfloat last_zoom_model;
    static Pfloat last_zoom_lights;
    static Ppoint3 last_prp;
    static Pfloat last_front_plane;
    static Pfloat last_view_plane;
    static Pfloat last_back_plane;
    Pfloat zoom_model, zoom_lights, front_plane, view_plane, back_plane;
    Ppoint3 prp;

    zoom_model = model.view_map.zoom_model;
    zoom_lights = model.view_map.zoom_lights;
    prp = model.view_map.prp;
    front_plane = model.view_map.front_plane;
    view_plane = model.view_map.view_plane;
    back_plane = model.view_map.back_plane;
```

```
    zoom_model = max (0.05,zoom_model);
    zoom_lights = max (0.05,zoom_lights);

    if (last_prp.z != prp.z) {
        front_plane = min (prp.z-0.01,front_plane);
        view_plane =  min (front_plane-0.01,view_plane);
        back_plane =  min (view_plane-0.01,back_plane);
    } else if (last_front_plane != front_plane) {
        prp.z =       max (front_plane+0.01,prp.z);
        view_plane =  min (front_plane-0.01,view_plane);
        back_plane =  min (view_plane-0.01,back_plane);
    } else if (last_view_plane != view_plane) {
        front_plane = max (view_plane+0.01,front_plane);
        prp.z =       max (front_plane+0.01,prp.z);
        back_plane =  min (view_plane-0.01,back_plane);
    } else if (last_back_plane != back_plane) {
        view_plane =  max (back_plane+0.01,view_plane);
        front_plane = max (view_plane+0.01,front_plane);
        prp.z =       max (front_plane+0.01,prp.z);
    }

    model.view_map.zoom_model = zoom_model;
    model.view_map.zoom_lights = zoom_lights;
    model.view_map.prp = prp;
    model.view_map.front_plane = front_plane;
    model.view_map.view_plane = view_plane;
    model.view_map.back_plane = back_plane;

    redraw = FALSE;
    if (last_zoom_lights != zoom_lights) {
        redraw = TRUE;
        psm_view_lights (WS_MAIN);
        psm_view_lights (WS_AUX);
    }

    if (last_zoom_model != zoom_model ||
        last_prp.x != prp.x ||
        last_prp.y != prp.y ||
        last_prp.z != prp.z ||
        last_front_plane != front_plane ||
        last_view_plane != view_plane ||
        last_back_plane != back_plane) {
        redraw = TRUE;
        psm_view_lines ();
        psm_view_model (WS_MAIN);
        psm_view_model (WS_AUX);
    }

    last_zoom_model = zoom_model;
    last_zoom_lights = zoom_lights;
    last_prp = prp;
    last_front_plane = front_plane;
    last_view_plane = view_plane;
    last_back_plane = back_plane;

    return (redraw);
}
```

```
/*
 * FUNCTION:    psm_model_view_orientation
 *
 *      psm_model_view_orientation updates the view orientation data
 *      for viewing the mechanism model it updates the global orientation of
 *      the PHIGS structure depicting the viewing volume if the view
 *      orientation data has changed since to last call to this function.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *       0 if no change in the view orientation parameters; 1 otherwise
 *
 * PROGRAMMER:  David E. Montgomery
 */

int psm_model_view_orientation ()
{
    static Ppoint3 last_vrp;
    static Pvector3 last_vpn;
    static Pvector3 last_vup;
    Ppoint3 vrp;
    Pvector3 vpn, vup;
    Pvector3 rotation, shift;
    Pmatrix3 trans_matrix;
    Pmatrix3 rotx_matrix, roty_matrix, rotz_matrix, shift_matrix;
    Pfloat dx, dy, dz, len;
    Ppoint3 temp_vpn, temp_vup;
    Pint ierr;

    vrp = model.view_orient.vrp;
    vpn = model.view_orient.vpn;
    vup = model.view_orient.vup;

    vpn.x = max (-1.0,vpn.x);
    vpn.x = min (1.0, vpn.x);
    vpn.y = max (-1.0,vpn.y);
    vpn.y = min (1.0, vpn.y);
    vpn.z = max (-1.0,vpn.z);
    vpn.z = min (1.0, vpn.z);
    vup.x = max (-1.0,vup.x);
    vup.x = min (1.0, vup.x);
    vup.y = max (-1.0,vup.y);
    vup.y = min (1.0, vup.y);
    vup.z = max (-1.0,vup.z);
    vup.z = min (1.0, vup.z);

    model.view_orient.vrp = vrp;
    model.view_orient.vpn = vpn;
    model.view_orient.vup = vup;

    if (last_vup.x != vup.x ||
        last_vup.y != vup.y ||
        last_vup.z != vup.z) {

        dx = vup.x; dy = vup.y; dz = vup.z;
        temp_vpn.x = vpn.x; temp_vpn.y = vpn.y; temp_vpn.z = vpn.z;
```

```
    len = sqrt(dx*dx + dy*dy);
    if (len > 0.00) {
        rotation.z = -acos(dy/len);
        if (dx != 0.00) rotation.z *= dx/fabs(dx);
    } else
        rotation.z = 0.00;
    protatez (-rotation.z,&ierr,rotz_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot z #%d\n",ierr);
    ptranpt3 (&temp_vpn,rotz_matrix,&ierr,&temp_vpn);
    if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);

    dy = len;
    len = sqrt(dy*dy + dz*dz);
    if (len > 0.00) {
        rotation.x = acos(dy/len);
        if (dz != 0.00) rotation.x *= dz/fabs(dz);
    } else
        rotation.x = 0.00;
    protatex (-rotation.x,&ierr,rotx_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot x #%d\n",ierr);
    ptranpt3 (&temp_vpn,rotx_matrix,&ierr,&temp_vpn);
    if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);

    dx = temp_vpn.x; dz = temp_vpn.z;
    len = sqrt(dx*dx + dz*dz);
    if (len > 0.00) {
        rotation.y = acos(dz/len);
        if (dy != 0.00) rotation.y *= dx/fabs(dx);
    } else
        rotation.y = 0.00;

    protatey (rotation.y,&ierr,roty_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot y #%d\n",ierr);
    protatex (rotation.x,&ierr,rotx_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot x #%d\n",ierr);
    protatez (rotation.z,&ierr,rotz_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot z #%d\n",ierr);

    pcomposematrix3 (rotx_matrix,roty_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_model_view_orientation: compose #%d\n",ierr);
    pcomposematrix3 (rotz_matrix,trans_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_model_view_orientation: compose #%d\n",ierr);

    temp_vpn.x = 0.00; temp_vpn.y = 0.00; temp_vpn.z = 1.00;
    ptranpt3 (&temp_vpn,trans_matrix,&ierr,&temp_vpn);
    if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);
    vpn.x = temp_vpn.x; vpn.y = temp_vpn.y; vpn.z = temp_vpn.z;
    model.view_orient.vpn = vpn;

} else if (last_vpn.x != vpn.x ||
           last_vpn.y != vpn.y ||
           last_vpn.z != vpn.z ||
           last_vrp.x != vrp.x ||
           last_vrp.y != vrp.y ||
           last_vrp.z != vrp.z) {

    dx = vpn.x; dy = vpn.y; dz = vpn.z;
    temp_vup.x = vup.x; temp_vup.y = vup.y; temp_vup.z = vup.z;

    len = sqrt(dx*dx + dz*dz);
    if (len > 0.00) {
        rotation.y = acos(dz/len);
        if (dx != 0.00) rotation.y *= dx/fabs(dx);
    } else
        rotation.y = 0.00;
    protatey (-rotation.y,&ierr,roty_matrix);
    if (ierr) printf ("psm_model_view_orientation: rot y #%d\n",ierr);
    ptranpt3 (&temp_vup,roty_matrix,&ierr,&temp_vup);
    if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);
```

```
        dz = len;
        len = sqrt(dy*dy + dz*dz);
        if (len > 0.00) {
            rotation.x = -acos(dz/len);
            if (dz != 0.00) rotation.x *= dy/fabs(dy);
        } else
            rotation.x = 0.00;
        protatex (-rotation.x,&ierr,rotx_matrix);
        if (ierr) printf ("psm_model_view_orientation: rot x #%d\n",ierr);
        ptranpt3 (&temp_vup,rotx_matrix,&ierr,&temp_vup);
        if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);

        dx = temp_vup.x; dy = temp_vup.y;
        len = sqrt(dx*dx + dy*dy);
        if (len > 0.00) {
            rotation.z = -acos(dy/len);
            if (dy != 0.00) rotation.z *= dx/fabs(dx);
        } else
            rotation.z = 0.00;

        protatez (rotation.z,&ierr,rotz_matrix);
        if (ierr) printf ("psm_model_view_orientation: rot z #%d\n",ierr);
        protatex (rotation.x,&ierr,rotx_matrix);
        if (ierr) printf ("psm_model_view_orientation: rot x #%d\n",ierr);
        protatey (rotation.y,&ierr,roty_matrix);
        if (ierr) printf ("psm_model_view_orientation: rot y #%d\n",ierr);

        pcomposematrix3 (rotx_matrix,rotz_matrix,&ierr,trans_matrix);
        if (ierr) printf ("psm_model_view_orientation: compose #%d\n",ierr);
        pcomposematrix3 (roty_matrix,trans_matrix,&ierr,trans_matrix);
        if (ierr) printf ("psm_model_view_orientation: compose #%d\n",ierr);

        temp_vup.x = 0.00; temp_vup.y = 1.00; temp_vup.z = 0.00;
        ptranpt3 (&temp_vup,trans_matrix,&ierr,&temp_vup);
        if (ierr) printf ("psm_model_view_orientation: tranpt #%d\n",ierr);
        vup.x = temp_vup.x; vup.y = temp_vup.y; vup.z = temp_vup.z;
        model.view_orient.vup = vup;

    } else
        return (0);

    shift.x = vrp.x; shift.y = vrp.y; shift.z = vrp.z;
    ptranslate3 (&shift,&ierr,shift_matrix);
    if (ierr) printf ("psm_model_view_orientation: translate3 #%d\n",ierr);
    pcomposematrix3 (shift_matrix,trans_matrix,&ierr,trans_matrix);
    if (ierr) printf ("psm_model_view_orientation: compose #%d\n",ierr);

    pseteditmode (PEDIT_REPLACE);
    popenstruct (STRUCT_VIEW_LINES);
    psetelemptr (BEGINNING);
    psetelemptrlabel (LABEL_GLOBAL_TRANSFORMATION);
    poffsetelemptr (1);
    psetglobaltran3 (trans_matrix);
    pclosestruct ();

    psm_view_model (WS_MAIN);
    psm_view_model (WS_AUX);
    psm_view_axes (WS_MAIN);
    psm_view_axes (WS_AUX);

    last_vrp = vrp; last_vpn = vpn; last_vup = vup;
    return (1);
}
```

```
/*
 * FUNCTION:    psm_init_view_lines
 *
 *      psm_init_view_lines initializes the PHIGS structure containing the
 *      view lines (viewing volume representation).
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_init_view_lines ()
{
    pseteditmode (PEDIT_INSERT);
    popenstruct (STRUCT_VIEW_LINES);
    pemptystruct (STRUCT_VIEW_LINES);
    plabel (LABEL_GLOBAL_TRANSFORMATION);
    plabel (LABEL_GLOBAL_TRANSFORMATION);
    psethlhsrid (ON);
    psetnormreorientmode (PNOFLIP);
    psetfaceprocmode (PDIST_NO,PCULL_BACK);
    psetintstyle (PSOLID);
    plabel (LABEL_VIEW_LINES);
    plabel (LABEL_VIEW_LINES);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_view_lines
 *
 *      psm_view_lines creates all the lines and fill areas in the PHIGS
 *      structure that represent the viewing volume for the spatial
 *      mechanism model.
 *
 * INPUT ARGUMENTS:
 *
 *      NONE
 *
 * OUTPUT ARGUMENTS:
 *
 *      NONE
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */
```

```
psm_view_lines ()
{
    int i;
    Pvector3 slope[5];
    Pfloat len, const;
    Ppoint3 front[5], view[5], back[5], reference[5],
            dc_front[5], dc_back[5], side[4];
    Ppoint3 pt3[4][2];
    Pfasdata3 fadata;
    Pfasvdata3 vdata;
    static Ppoint3 origin = {0.00,0.00,0.00};
    static Pfloat angle = 0.00;
    Pvector3 shift, scale;
    Pmatrix3 trans_matrix;
    Pint ierr;
    static Pcolour dark_blue =        {PCM_RGB, 0.00,0.00,0.70};
    static Pcolour blue =            {PCM_RGB, 0.00,0.00,1.00};
    static Pcolour light_blue =       {PCM_RGB, 0.00,0.50,1.00};
    static Pcolour dark_magenta =    {PCM_RGB, 0.70,0.00,0.70};
    static Pcolour magenta =        {PCM_RGB, 1.00,0.00,1.00};
    static Pcolour light_magenta =    {PCM_RGB, 1.00,0.50,1.00};

    len = model.view_edge_len/(2.0*model.view_map.zoom_model);
    view[0].x = view[0].y = view[1].y = view[3].x =
                            view[4].x = view[4].y = len;
    view[1].x = view[2].x = view[2].y = view[3].y = -len;
    for (i=0; i<=4; i++) {
        view[i].z = model.view_map.view_plane;
        if (model.view_projection == PPERSPECTIVE) {
            slope[i].x = model.view_map.prp.x - view[i].x;
            slope[i].y = model.view_map.prp.y - view[i].y;
        } else {
            slope[i].x = model.view_map.prp.x;
            slope[i].y = model.view_map.prp.y;
        }
        slope[i].z = model.view_map.prp.z - view[i].z;
    }

    for (i=0; i<=4; i++) {
        const = (model.view_map.front_plane - view[i].z)/slope[i].z;
        front[i].x = slope[i].x*const + view[i].x;
        front[i].y = slope[i].y*const + view[i].y;
        front[i].z = model.view_map.front_plane;
        const = (model.view_map.back_plane - view[i].z)/slope[i].z;
        back[i].x = slope[i].x*const + view[i].x;
        back[i].y = slope[i].y*const + view[i].y;
        back[i].z = model.view_map.back_plane;

        if (model.view_projection == PPARALLEL) {
            const = (model.view_map.prp.z - view[i].z)/slope[i].z;
            reference[i].x = slope[i].x*const + view[i].x;
            reference[i].y = slope[i].y*const + view[i].y;
            reference[i].z = model.view_map.prp.z;
        }

        dc_front[i].z = (front[i].z-back[i].z)*model.depth_cue.planes[0]
                        + back[i].z;
        const = (dc_front[i].z - view[i].z)/slope[i].z;
        dc_front[i].x = slope[i].x*const + view[i].x;
        dc_front[i].y = slope[i].y*const + view[i].y;

        dc_back[i].z = (front[i].z-back[i].z)*model.depth_cue.planes[1]
                        + back[i].z;
        const = (dc_back[i].z - view[i].z)/slope[i].z;
        dc_back[i].x = slope[i].x*const + view[i].x;
        dc_back[i].y = slope[i].y*const + view[i].y;
    }
```

```
psetedítmode (PEDIT_INSERT);
popenstruct (STRUCT_VIEW_LINES);
psetelemptr (BEGINNING);
pdelelemslabels (LABEL_VIEW_LINES,LABEL_VIEW_LINES);

psetindivcsf (PCO_INTERIOR,PDIRECT);
psetindivcsf (PCO_POLYLINE,PDIRECT);

const = (model.view_map.prp.z - view[3].z)/slope[3].z;
side[2].x = slope[3].x*const + view[3].x;
side[2].y = slope[3].y*const + view[3].y;
side[2].z = model.view_map.prp.z;
side[3] = back[3];

for (i=0; i<4; i++) {
    side[0] = side[3];
    side[1] = side[2];
    const = (model.view_map.prp.z - view[i].z)/slope[i].z;
    side[2].x = slope[i].x*const + view[i].x;
    side[2].y = slope[i].y*const + view[i].y;
    side[2].z = model.view_map.prp.z;
    side[3] = back[i];

    if (i == 1)
        psetintcolour (&light_magenta);
    else if (i == 3)
        psetintcolour (&dark_magenta);
    else
        psetintcolour (&magenta);
    psm_normal_vector (side[1],side[0],side[3],&fadata.gnormal);
    vdata.pt = side;
    pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,4,&vdata);
    pt3[i][0] = side[2];
    pt3[i][1] = side[3];
}

psetintcolour (&light_blue);
fadata.gnormal.x = fadata.gnormal.y = 0.00;
fadata.gnormal.z = -1.00;
vdata.pt = front;
pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,4,&vdata);

psetintcolour (&dark_blue);
fadata.gnormal.z = 1.00;
vdata.pt = back;
pfillarea3data (PFA_NORMAL,PVERT_NONE,PCM_NONE,&fadata,4,&vdata);

psetlinecolour (&light_magenta);
for (i=0; i<2; i++)
    ppolyline3 (2,pt3[i]);
psetlinecolour (&dark_magenta);
for (i=2; i<4; i++)
    ppolyline3 (2,pt3[i]);
psetlinecolour (&light_blue);
ppolyline3 (5,front);
psetlinecolour (&blue);
ppolyline3 (5,view);
psetlinecolour (&dark_blue);
ppolyline3 (5,back);

psetindivcsf (PCO_INTERIOR,PINDEXED);
psetindivcsf (PCO_POLYLINE,PINDEXED);

if (model.view_projection == PPARALLEL) {
    psetlinecolourind (CYAN);
    ppolyline3 (5,reference);
}
```

```
    if (model.depth_cue_on) {
        psetlinetype (PLN_DOT);
        psetlinecolourind (YELLOW);
        ppolyline3 (5,dc_front);
        ppolyline3 (5,dc_back);
    }

    shift.x = model.view_map.prp.x;
    shift.y = model.view_map.prp.y;
    shift.z = model.view_map.prp.z;
    scale.x = scale.y = scale.z = model.view_edge_len/10.0;
    pbuildtran3 (&origin,&shift,angle,angle,angle,&scale,&ierr,trans_matrix);
    if (ierr) printf ("psm_view_lines: build trans error #%d\n",ierr);
    psetlocaltran3 (trans_matrix,PREPLACE);
    pexecutestruct (STRUCT_AXES);
    pclosestruct ();
}
```

```
/*
 * FUNCTION:    psm_normal_vector
 *
 *      psm_normal_vector creates a normal vector to the three planar points
 *      by crossing the vector pt1-pt2 into pt3-pt2.
 *
 * INPUT ARGUMENTS:
 *
 *      pt1:     first planar point
 *      pt2:     second planar point
 *      pt3:     third planar point
 *
 * OUTPUT ARGUMENTS:
 *
 *      normal:  normal vector calculated
 *
 * RETURN VALUE:
 *
 *      NONE
 *
 * PROGRAMMER:  David E. Montgomery
 */

psm_normal_vector (pt1,pt2,pt3,normal)
Ppoint3 pt1, pt2, pt3;
Pvector3 *normal;
{
    Pvector3 a, b;

    a.x = pt1.x - pt2.x;
    a.y = pt1.y - pt2.y;
    a.z = pt1.z - pt2.z;

    b.x = pt3.x - pt2.x;
    b.y = pt3.y - pt2.y;
    b.z = pt3.z - pt2.z;

    normal->x = a.y*b.z - a.z*b.y;
    normal->y = -(a.x*b.z - a.z*b.x);
    normal->z = a.x*b.y - a.y*b.x;
}
```

Appendix D

UNIX Makefile for PriSM

```
# Makefile for main program

SOURCES= prism.c psm_init.c psm_input.c psm_color.c psm_views.c  \
         psm_dials.c psm_objects.c psm_model.c psm_help.c psm_lights.c
LIBS= -lphigs /usr/RNeWS/lib/libcps.a -lm
PROGRAM= prism

CFLAGS= -sun4
CPPFLAGS=
LDFLAGS=
LINTFLAGS= -abchx
OUTPUT_OPTION=

OBJECTS= $(SOURCES:.c=.o)
LINTFILES= $(SOURCES:.c=.ln)

LINK.c= $(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH)
LINT.c= lint $(LINTFLAGS) $(CPPFLAGS) $(TARGET_ARCH)
.SUFFIXES: .c .ln
.c.ln: $(LINT.c) $(OUTPUT_OPTION) -i $<

.KEEP_STATE:

$(PROGRAM): $(OBJECTS)
    $(LINK.c) -o $@ $(OBJECTS) $(LIBS)

lint: $(LINTFILES)

$(LINTFILES):
    $(LINT.c) $(SOURCES)

clean:
    rm -f $(PROGRAM) $(OBJECTS) $(LINTFILES)
```

Appendix E

Sample Model and Position Files

Model file: rssr.mod

```
RSSR Spatial Mechanism Design
   1   1  EXTERNAL REVOLUTE JOINT
   0.1000    0.0500    0.2400    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   9   2  EXTERNAL SPHERIC JOINT
   0.1200    0.1080    0.0000   29.8326   -0.0002    0.7500    0.0000    0.0000
   3   3  CONNECTING ELEMENT
   0.0500   4
   0.0000    0.0000    0.0000
   0.2000    0.0000    0.0000
   0.5432    0.0000   -0.3606
   0.7500    0.0000    0.0000
   4 101  RESE LINK      1
   3   1   2   3
   1   5  EXTERNAL REVOLUTE JOINT
   0.1000    0.0500    0.2400    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   9   6  EXTERNAL SPHERIC JOINT
   0.1200    0.1080    0.0000  -68.1148 -269.9999    3.9528    0.0000    0.0000
   3   7  CONNECTING ELEMENT
   0.0500   4
   0.0000    0.0000    0.0000
   3.1399    0.4545    0.0000
   4.3385    0.1549    0.0000
   3.9528    0.0000    0.0000
   4 102  RESE LINK      2
   3   5   6   7
  10   9  INTERNAL SPHERIC JOINT
   0.1080    0.0000    0.0000    0.0000
  10  10  INTERNAL SPHERIC JOINT
   0.1080    1.6007    0.0000    0.0000
   3  11  CONNECTING ELEMENT
   0.0500   4
   0.0000    0.0000    0.0000
   0.2400    0.0000    0.0000
   1.3607    0.0000    0.0000
   1.6007    0.0000    0.0000
   4 103  SISI LINK      3
   3   9  10  11
```

```
 11 130  PILLOW BLOCK 1
  0.1200    0.0500    0.2800    0.2400    0.1200    0.0000    0.0000    0.0000    0.0000
  0.0000    2.5000
  2 140  INTERNAL REVOLUTE JOINT
  0.0500    0.4000   -0.1600    0.0000    0.0000    0.0000    0.0000    0.0000    2.5000
 11 150  PILLOW BLOCK 2
  0.1200    0.0500    0.2800    0.2400    0.1200    0.0000    0.0000   90.0000    1.0000
  0.0000    0.0000
  2 160  INTERNAL REVOLUTE JOINT
  0.0500    0.4000   -0.1600    0.0000    0.0000   90.0000    1.0000    0.0000    0.0000
  4 170  GROUND ELEMENTS
  4 130 140 150 160
```

Position file: rssr.pos

```
RSSR Spatial Mechanism Design
 12 ANIMATION STEPS
    1  101      0.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    1  102    108.4349      0.0000     90.0000      1.0000      0.0000      0.0000
    1  103    128.6619      0.0000     90.0000      0.7500      0.0000      2.5000
    1  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    2  101     30.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    2  102    109.2019      0.0000     90.0000      1.0000      0.0000      0.0000
    2  103    126.3871      0.0000    106.9185      0.6495      0.3750      2.5000
    2  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    3  101     60.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    3  102    112.9769      0.0000     90.0000      1.0000      0.0000      0.0000
    3  103    124.9959      0.0000    119.6915      0.3750      0.6495      2.5000
    3  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    4  101     90.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    4  102    123.0889      0.0000     90.0000      1.0000      0.0000      0.0000
    4  103    136.3394      0.0000    132.7400      0.0000      0.7500      2.5000
    4  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    5  101    120.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    5  102    135.3329      0.0000     90.0000      1.0000      0.0000      0.0000
    5  103    153.8028      0.0000    156.8028     -0.3750      0.6495      2.5000
    5  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    6  101    150.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    6  102    143.9529      0.0000     90.0000      1.0000      0.0000      0.0000
    6  103    165.0461      0.0000   -155.2150     -0.6495      0.3750      2.5000
    6  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    7  101    180.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    7  102    146.9519      0.0000     90.0000      1.0000      0.0000      0.0000
    7  103    167.5947      0.0000    -90.0002     -0.7500      0.0000      2.5000
    7  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    8  101    210.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    8  102    143.9529      0.0000     90.0000      1.0000      0.0000      0.0000
    8  103    165.0460      0.0000    -24.7863     -0.6495     -0.3750      2.5000
    8  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
    9  101    240.0000      0.0000      0.0000      0.0000      0.0000      2.5000
    9  102    135.3329      0.0000     90.0000      1.0000      0.0000      0.0000
    9  103    153.8026      0.0000     23.1981     -0.3750     -0.6495      2.5000
    9  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
   10  101    270.0000      0.0000      0.0000      0.0000      0.0000      2.5000
   10  102    123.0889      0.0000     90.0000      1.0000      0.0000      0.0000
   10  103    136.3394      0.0000     47.2601      0.0000     -0.7500      2.5000
   10  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
   11  101    300.0000      0.0000      0.0000      0.0000      0.0000      2.5000
   11  102    112.9769      0.0000     90.0000      1.0000      0.0000      0.0000
   11  103    124.9959      0.0000     60.3083      0.3750     -0.6495      2.5000
   11  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
   12  101    330.0000      0.0000      0.0000      0.0000      0.0000      2.5000
   12  102    109.2019      0.0000     90.0000      1.0000      0.0000      0.0000
   12  103    126.3871      0.0000     73.0814      0.6495     -0.3750      2.5000
   12  170      0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
```

Appendix F

Guide to Running PriSM

1. On the Sun 4/260 workstation (the monochrome monitor), logon to the system with the following (no password required):

   vtsun1 login: **prism**

2. Start the Raster Tech version of NeWS on the Sun4 workstation by entering:

   vtsun% **startnews**

3. At this point you will be using the Sun workstation keyboard and the Sony graphics monitor. While holding down the 3rd (right) mouse button, select:

   Applications = > Terminals = > Fixed Startup = > Console

4. Move the arrow cursor into the console window and type:

   vtsun% **prism**

5. Move the arrow to the Models menu, press and hold the 1st (left) mouse button, move the arrow to the desired selection, and release the mouse button.

6. Carefully read the help messages to understand how to operate PriSM. The help text can be toggled on and off by clicking the 3rd mouse button while the arrow is in the main window.

7. To exit the program, select EXIT PriSM on the Models menu list.

8. To stop NeWS, hold down the 3rd mouse button while the arrow is pointing to the background pattern and select:

   Exit NeWS = > Yes, really!

9. To logout type:

   vtsun% **logout**

If the Sun workstation locks-up and needs to be rebooted, use the following sequence:

L1–A B <CR> (press and hold L1, press A, release L1 and A, type B and the RETURN key)

Vita

David Eric Montgomery was born on January 16, 1963 in Oak Ridge, Tennessee. He graduated from Oak Ridge High School in 1981 and enrolled at VPI&SU in the fall of that same year. He participated in cross country and track through-out high school and his first two years of college where he received All-Metro honors in cross country. In the spring of 1985, David graduated with Summa Cum Laude honors from the department of Mechanical Engineering and with a minor in Computer Science.

After one quarter of graduate school in fall 1985, David accepted a full-time position at Litton Poly-Scientific in Blacksburg as a software specialist. He continued graduate studies while working at Poly-Scientific until he completed his Masters Degree in April 1990.

David and his wife Karen, also a VPI&SU graduate, became the proud parents of their first child, Eric Josiah, on July 17, 1988. As Christians, the Montgomerys are active members of Dayspring Christian Community in Blacksburg.