MIDAS: MOLECULAR INTERACTIVE DISPLAY AND SIMULATION

by

THOMAS EVAN FERRIN

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

MEDICAL INFORMATION SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

San Francisco

. . . . . .

Date University Librarian

Degree Conferred: JAN 04 1987 . . . . . .

I dedicate this thesis to the two people
who have always stood behind me and
encouraged me to pursue all of my dreams,
my parents, Elizabeth and Harold Ferrin.

ACKNOWLEDGEMENTS

I will begin by acknowledging two people who have been very influential during my graduate studies: my thesis adviser and mentor Professor Robert Langridge, and my friend and preceptor Professor Irwin Kuntz. Their excitement over my work, their numerous helpful suggestions, and their continued support over the handful of years it required to complete my work are sincerely appreciated. I also would like to acknowledge the staff of the UCSF Computer Graphics Laboratory, particularly Conrad Huang, Laurie Jarvis and Greg Couch, whose programming work contributed to a working and usable system.

The fact that the MIDAS system has been a very successful project that has been used by dozens of scientists for their personal research can perhaps be attributed most to the discussions this author had with many early users of the system. I thank these numerous unnamed individuals for the many good ideas that they shared with me and for their tolerance in suffering through the early development stages of a new system when things were far less stable than desired.

Chapters 2 and 3 of this thesis have been submitted for publication in the Journal of Molecular Graphics and will appear, respectively, as follows:

T.E. Ferrin *et al.*, Molecular Interactive Display and Simulation (MIDAS),

and

T.E. Ferrin *et al.*, The MIDAS Database System.

Lastly, I acknowledge my direct support from the National Institutes of Health, Division of Research Resources, Grant RR-1081.

ABSTRACT

The problem of intelligently and comprehensively depicting three-dimensional macromolecular models such as proteins and nucleic acids is addressed using interactive color computer graphics techniques. Programs written in the C programming language for storage, access, display and interaction with macromolecular models are described and illustrated. The system provides a general method for visual studies of protein-protein, protein-nucleic acid, and protein-drug interactions.

TABLE OF CONTENTS

Chapter 1	Introduction	1
	The Need for New Software	3
Chapter 2	Historical Perspective	6
	Design Goals	7
	Data Structures	8
	MIDAS Display Program and Data Flow	11
	Model Display and Manipulation	11
	MIDAS Command Language	13
	Model Surfaces	21
	Structural Modifications	22
	Coordinate Recovery	22
	MIDAS Command Language Grammar	24
Chapter 3	Background	32
	Nature of Macromolecular Data	33
	Design Goals	35
	Database Description	36
	File Organization	37
	Subroutine Descriptions	40
	Example Subroutine Usage	55
	Miscellaneous Details	57
	Data Structure Definitions	60
Chapter 4	Conclusions	63
	Future Work	64
	References	74
	Appendix: Program Listings	77

LIST OF TABLES

Table 2-1: MIDAS Special Characters for Atom Selection 16

Table 2-2: Sample Specifications for Model Components 17

Table 2-3: MIDAS Interactive Commands 19

Table 2-4: MIDAS Editor Commands 20

Table 3-1: MDBS Subroutine Summary 43

Table 3-2: Mseekr Mode Options 45

Table 4-1: Publications Derived from MIDAS 66

Table 4-2: Sites Licensed to Use MIDAS 72

LIST OF FIGURES

Figure 2-1: MIDAS Database Construction 9

Figure 2-2: MIDAS Display Program Overview 12

Figure 3-1: MDBS Logical Database View 36

Figure 3-2: MDBS Physical Database View 38

Chapter 1

Introduction

Many of the problems of modern biology are concerned with the detailed relationship between biological function and molecular structure, and especially those relationships which deal with proteins and nucleic acids. Since it is the three dimensional structure of these molecules that determine their unique biological activities, visualization of these structures contributes greatly to understanding the rationale of their conformation and function. Researchers thus often used physical models to depict three dimensional structure. Unfortunately, 3D models of proteins and nucleic acids can consume weeks or even months of time to construct, and even then have numerous disadvantages: 1) they are physically large and therefore awkward to manipulate, 2) because of their large size they necessitate the incorporation of extraneous physical support such as rods and wires to counteract the force of gravity, 3) they are so complex internally as to make it difficult to adjust molecular distances and bond angles within the interior regions of the structure, and 4) their physical nature makes it virtually impossible to accurately synchronize the experimental "hand driven" modifications of the physical model with the molecular coordinates stored within the computer.

The introduction of computer generated images in the early 1960's was thus an important advance in the field of molecular modeling. At first these images were staticly generated by "hardcopy" plotting devices. These black and white two dimensional line drawing renditions of three dimensional objects clearly lacked realism, and each 2D view often took minutes or even hours to produce. In 1964, however, at Project MAC at the Massachusetts Institute of Technology an *interactive* graphics display was first used for depicting molecular structures by Levinthal and Langridge. [LANG65, LEVIN65, LEVIN66] These early pioneers must be given ample credit for realizing the potential benefits of the application of interactive computer graphics to the field of

molecular modeling. These researchers often attempted to write complex computer application programs even though the knowledge base of computer science was still in its infancy. Still, the basic ideas introduced then of interacting in real-time with the image displayed on the cathode ray tube (CRT) screen through use of a "light pen" and "joystick" remain today.

The progress in both biochemistry and computer science during the late 1960's and early 1970's was substantial. While the structure of only three proteins had been worked out by 1965 (myoglobin, hemoglobin and lysozyme), there are presently about 250 proteins whose structures have been published. Computer programs capable of interactively displaying protein and nucleic acid structures became more widely available by the establishment of such national facilities as the Princeton University Computer Graphics Laboratory in 1970.

These early computer programs and graphics displays, however, were limited in the size of molecular structures which they could accurately depict and yet still retain a reasonable level of interaction with the scientist user. The graphics displays were large and expensive, and often required a large dedicated computer such as a DEC System 10 to drive the display. In addition, the large size of the biological structures of most interest to the research community were often several times the capacity of the graphics display subsystem. An imbalance was present. Protein structure determination had progressed at a faster rate than computer science and interactive graphics display technology, and chemists were faced with the problem of depicting important biological structures which were too large to conveniently manipulate either with a physical model or a virtual model within the computer.

In the mid 1970's hardware technology caught up with the needs of the biochemist. The DEC PDP-11/70 was the first "high performance" minicomputer commercially available,[1] and this machine coupled with the newly announced high performance interactive three dimensional

[1] The DEC PDP-11/70, introduced in 1975, offered a main memory configuration of up to 4 megabytes as well as a high speed I/O bus. Typical configurations sold for around $75,000.

graphics display subsystem available from Evans and Sutherland (E&S), the Picture System 2, created an environment ripe for molecular modeling research applications. The PDP-11/70 and Picture System 2 combination was further enhanced with the announcement by E&S in July 1979 of the availability of a color CRT display; this enabled another dimension of information to be depicted on the CRT screen.

Thus, fifteen years after the first application of interactive graphics to molecular modeling, computer technology finally was sufficiently advanced for widespread application. The only component of the system glaringly lacking was the necessary user friendly and interactive software to manage the available hardware.

The Need for New Software

The previous discussion illustrates many of the problems facing the interactive computer graphics "molecular modeling" user in the late 1970's. The typical user could not get by with only a knowledge of biochemistry or physical chemistry, but almost inevitably had to know how to program computers as well. The previous generation of interactive graphics programs, capable of adequately handling small molecules, did not have the capacity to store and manipulate the then newly refined macromolecules. Attempts to modify earlier generation programs for display of large protein and nucleic acid structures met with limited success. [FERR77] For example, the CAAPS display program developed by Langridge and Harbison [LANG75] was specialized for a dedicated DEC System 10 and a by then obsolete E&S LDS-1 graphics display. The major problems of adapting the older display software to the new hardware typically involved the inability to scale algorithms beyond their original designs. The per process address space of the PDP-11/70 was limited to 64K bytes,[2] however the size of known conformations for macromolecules was increasing rapidly and it was no longer possible to hold all relevant information in main

[2] If the special "separate I&D" address space mapping of the PDP-1170 hardware was enabled, the per process space could be increased to 128 K bytes. This created other programming problems, however. [FERR80]

memory. Thus there was a crucial need for a compact data representation that still allowed very fast access in order to minimize command execution delays for the user. In addition, a new program command language was needed that provided a syntactically compact way of specifying small regions of interest within a large structure. These specialized regions of interest (i.e. binding sites) often involve a relatively small region of the overall structure, but are the focus of attention for the chemist, since key chemical activity usually involves these sites. Graphics display software must not only be efficient in displaying the overall molecular structure, but must also be efficient at manipulating a relatively small geometric group within the overall three dimensional structure of the molecule.

Efficient access and convenient manipulation of large biomolecular structures were by necessity key design goals for any newly developed software. Clearly, both the advancing needs of the scientist user and the newly available advanced display technology dictated either a redesign of some existing software system or committing to an entirely new software design. Given the apparent rapid advances in the field of computer science and the eagerness of this young scientist, the only rational decision at the time appeared to be a completely new and originally designed system for modeling macromolecular structures. The new system needed to address not only the deficiences of previous interactive modeling systems, but also the important needs of a then rapidly growing scientist user community. Specifically, the new system needed to be able to model the *interactions* of macromolecules with one another and the interactions of these molecules with smaller structures such as drugs. This requirement of modeling interactions among multiple macromolecules was a new and yet crucial requirement.

In addition to the above requirements, advances in timeshared operating systems for minicomputers [RITC74] made a timesharing-based host system both desirable and feasible.

In summary, the rapidly increasing database of refined macromolecular structures, along with the advent of advancing scientist user requirements, hardware graphics display and computer

technology, and new ideas derived from the field of computer science combined to provide a ripe environment for a new generation of molecular modeling software.

Chapter 2

Abstract

The Molecular Interactive Display and Simulation (MIDAS) System is designed for the display and manipulation of large macromolecules such as proteins and nucleic acids. Several ancillary programs allow for such features as computing the surface of a molecule, the selection of an active site region within a molecule, and computation of electrostatic charge potentials. At the core of MIDAS is a hierarchical database system, designed specifically for macromolecules and both compact in its storage requirements and fast in its data access.

Historical Perspective

MIDAS is the most recent in a series of interactive molecular graphics systems whose direct lineage extends back to the first developments in interactive molecular graphics at Project MAC, MIT, in 1964. [LANG65, LEVI65, LEVI66] National Institutes of Health (NIH) support began with the formation of the Computer Graphics Laboratory at Princeton University in 1969[1] and resulted in a number of pioneering developments including CAAPS (Computer Aided Analysis of Protein Structure). [LANG75]

In 1976 this NIH research resource moved to U.C.S.F. A new graphics package [FERR80] was designed to operate under the UNIX[2] [RITC74] operating system and a new molecular graphics system designed (MMS), initially in collaboration with the Chemistry Department at U.C. San Diego. This evolved at UCSF into a system, MIDS, which was able to accommodate the new developments in color displays. [LANG81] The system was used by numerous visitors to the U.C.S.F. Computer Graphics Laboratory.

[1] The Princeton University Computer Graphics Laboratory was established by a grant in 1970 from the Division of Research Resources of the National Institutes of Health (RR-578); funded also was provided by Princeton University.

[2] UNIX is a registered trademark of AT&T Bell Laboratories.

In 1980 we decided to redesign the system completely, making use of the lessons learned over the previous 15 years. The result of this effort, MIDAS, emphasizes highly interactive display and manipulation, with a data structure designed for very fast access to and manipulation of large and complex molecules such as protein and nucleic acids. [FERR84, LANG84, PATT84]

MIDAS was originally developed on the UNIX operating system for use with an Evans and Sutherland (E&S) Picture System (PS) 2 display and recently converted for use on other operating systems (VMS and UNIX System V) and other graphics display engines (E&S PS330 and PS350, MPS, and Silicon Graphics IRIS).

Design Goals

MIDAS was designed primarily for modeling proteins and nucleic acids and their interactions with each other and with small molecules such as drugs. Macromolecules are large compared to typical drug molecules: the average number of atoms in protein and nucleic acid structures as found in the Brookhaven Protein Data Bank [BERN77] are 1500 and 1000, respectively, and these are only a small subset of the huge family of these molecules found in nature, the size of which can range up into millions of atoms. The average amount of data associated with each atom is roughly 20 bytes (atom name, {x,y,z} coordinates, some physical properties, and graphics information). Thus, the memory requirement for studying an average sized two molecule interaction is roughly 50 kilobytes, just for the raw data. Since MIDAS was originally developed on a Digital Equipment (DEC) PDP-11/70 computer which had an inherent limited address space, an important aspect of our design goal was therefore to build a modeling system which processes a large amount of data using little main memory.[3] Equally important, however, was that this aspect

[3] In the late seventies the cost of main memory was about a thousand times greater per byte than the cost of magnetic disks. Most machines were used as centralized time sharing machines whose resources were typically shared among many users. Major technological advances in the past half-decade have influenced computing systems significantly. Microprocessors have become powerful enough to allow users to have their own personal work stations. In the past ten years there has been a four-fold decrease in the cost per byte of disk storage. In this same time period the cost per byte of main memory has dropped by a factor of a hundred! These new technological influences would obviously have a major bearing on any design decisions for a new modeling system that was being developed today.

of the design did not impact performance; the system must provide real-time interactive performance with minimal CPU utilization, and it must not use excessive amounts of disk space for storing the molecular databases.

MIDAS incorporates into one package the capabilities found through the years to contribute to interpretability of complex objects. These include real time display interaction though various input devices and a coherent command syntax, stereoscopic image generation, the use of color and representation of molecular surfaces. Hardware capabilities such as depth cueing, perspective, clipping and real-time stereo are incorporated. Emphasis is placed on the interactive selection, manipulation and docking of drugs and receptors.

In short, the program was required to be both efficient enough to operate within the space limitations of the available computer system and of sufficiently high performance to provide a truly interactive tool for scientist users.

Data Structures

In designing MIDAS, we first needed a data structure which could store the large amounts of data associated with a molecular model efficiently and provide fast access for a modeling program. The proteins and nucleic acid biomolecules for which MIDAS is intended are built from smaller component molecules which are chained linearly into large structures. The diversity of these biomolecules is due largely to sequence variations of only 20 common amino acids in proteins and 4 common nucleotides in nucleic acids. This redundancy in the structure of component residues of large biomolecules provides an efficiency in nature which is useful in data storage as well. Residue structural data such as atom names, bonding pattern, and linkage atoms, need only be specified once, and then applied to all like residues within a model. Only the actual coordinate data (3-D position in a Cartesian coordinate system) is explicitly required for each atom.

The MIDAS database structure [FERR86a] thus consists of three binary files for each molecular model (see figure 1). A 'data' file contains the coordinate data, a 'template' file contains the

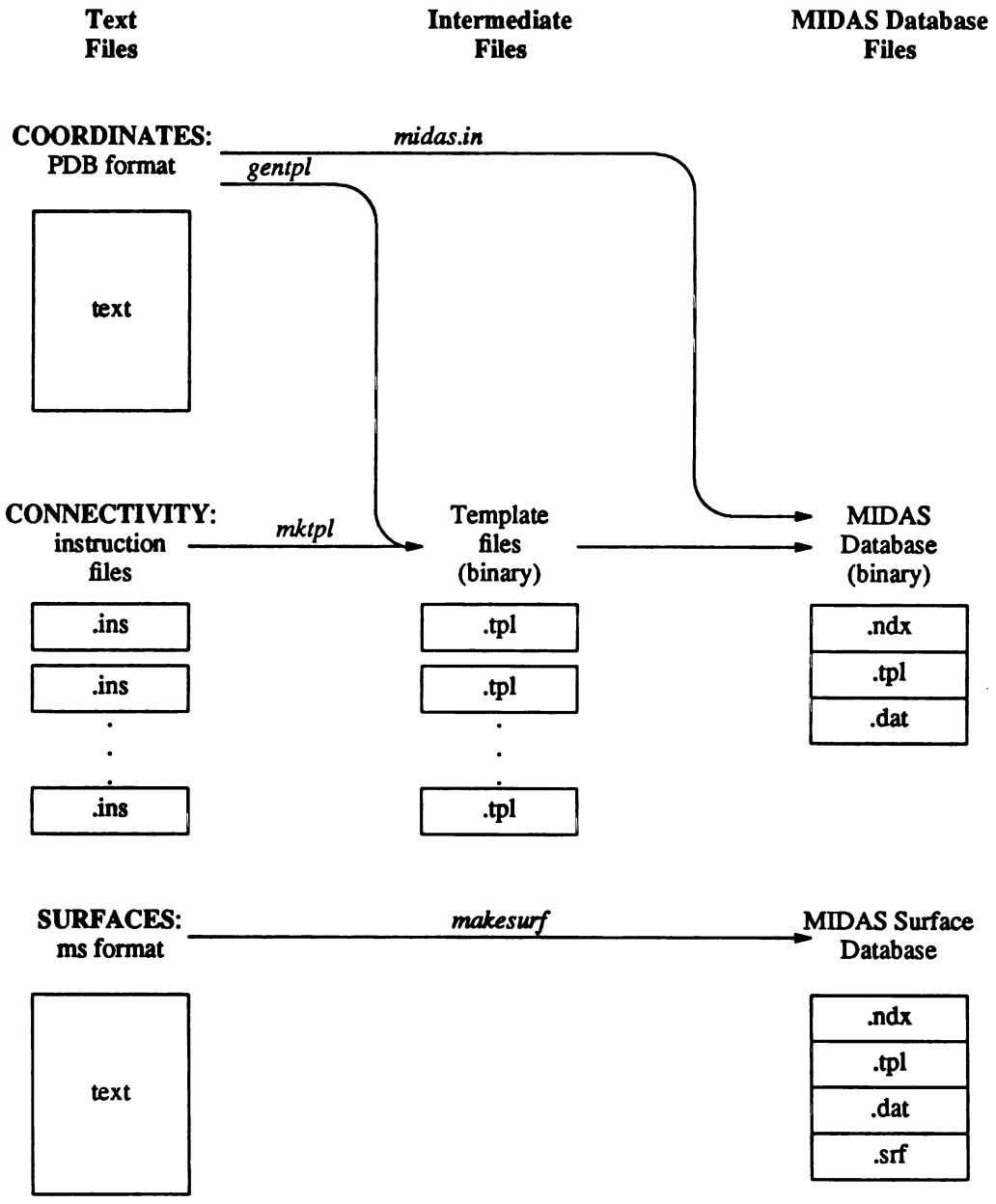

Figure 2-1: MIDAS Database Construction

Construction of a MIDAS database using the *midas.in* utility program requires two types of input: (1) coordinate data in Protein Data Bank format and (2) connectivity data in the form of a binary template for each residue appearing in the database. These templates are generated from either connectivity instruction files using the *mktpl* program or from Protein Data Bank coordinate files using the *gentpl* utility. A MIDAS solvent accessible surface database is converted from a ms format file using the *makesurf* utility.

residue structural information, and an 'index' file relates the two. The binary format of the database provides for fast access by the MIDAS display program while the elimination of redundant structural information significantly reduces the MIDAS main memory requirement during program execution.

The utility program *midas.in* builds a MIDAS database (set of three files) from crystallographic data; the user is required to provide the residue sequence and the atom coordinate data as well as the more general residue structural data. The latter is termed a "template" since it provides the connectivity information for all atoms in a residue.

Residue templates are generally prepared in advance. Since most models contain residues found commonly in nature, a library of residue templates is made available on the system. If the user's input data conforms to the established Brookhaven Protein Data Bank conventions,[4] the corresponding MIDAS database is readily constructed in a few minutes.

Should the input data contain residues not found in the library, the *midas.in* program will try to construct an appropriate template. This is done by calculating the distances between atoms of the residue and comparing these distances to a standard set of atom radii to determine the bonding pattern. Alternatively, users may construct templates by hand using a series of DRAW and MOVE instructions which are used by another utility program, *maketpl*, to generate the template.

To summarize, the templates contain the name and connectivities of each atom in the residue, the residue name, and identify the atoms which link the residue to the next residue and previous residue in the overall sequence of residues which form the molecule. The Protein Data Bank input data file provides the sequence of residues and the atomic coordinates for each atom in the residues. These files are processed by *midas.in* to create three binary files making up a MIDAS

[4] See the document "Protein Data Bank Atomic Coordinate Entry Description", distributed by the Brookhaven Protein Data Bank.

database which is accessed by the MIDAS display program.

MIDAS Display Program and Data Flow

The MIDAS display program can be configured to display an arbitrary number of molecular models (databases) simultaneously, although in practice a dozen simultaneously displayed molecules provides sufficient capacity for studying nearly all problems of current interest while still maintaining a comprehensible display. Even with a dozen simultaneously displayed models MIDAS still provides real-time user interaction with the displayed objects (molecules).

MIDAS is divided into three program modules. One handles the real-time interaction with the user including polling the input devices and sending instructions back to the display device. A second module modifies and maintains the disk based MIDAS databases. This includes managing structural data as well as color and temperature factor data. A third module parses commands and manages the graphics objects. Objects are regenerated only when necessary. Thus, a newly generated object returned from the database editor is merged with the unchanged stored objects and then sent as a single display list to the display device. Figure 2 summarizes the data flow among the various modules.

Model Display and Manipulation

The MIDAS display program represents molecules as wire models. Atoms are points which are joined by line segments to represent bonds and these points can be selectively labeled for identification. The user sees on the display screen a two dimensional representation of a three dimensional object enhanced by perspective and depth cueing and, optionally, stereoscopic viewing.

The MIDAS display program provides the user with both global and local interactive object manipulation capabilities.

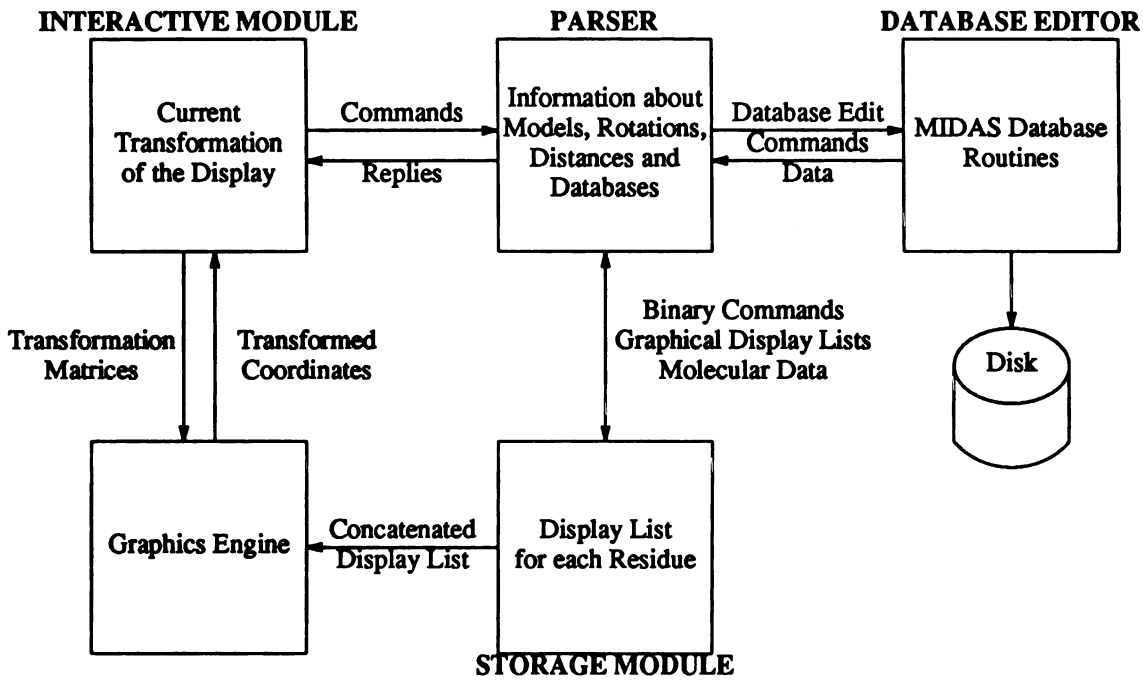

Figure 2-2: MIDAS Display Program Overview

The *interactive module* receives user input from the interactive devices (joysticks, knobs, keyboard, etc.). Interactive commands are processed by the interactive module and transformation matrices sent back to the graphics engine. Editor commands are sent on to the *parser*. If the command requires that the MIDAS database be edited, the command is sent to the *database editor* which returns new object data for the modified objects. The parser then sends either the new object in the form of a binary display list or a binary command (if the database editor was not needed) to the *storage module*. The storage module sends a concatenated display list of all the objects to the graphics display engine. The parser and storage modules are actually a single process, but have been distinguished here for clarity.

The global object manipulation capabilities include translation of one or more objects in the X, Y and Z directions, rotation of the objects about the X, Y, Z laboratory axes centered at the common center of mass, and scaling of the screen image. When more than one model is displayed, any subset of models may be selected for independent translation and rotation about the recalculated center of mass. The center of rotation may be set to any atom. Clipping planes in the X, Y, and Z Cartesian coordinate system provide viewing access to otherwise obscured parts of the molecule.

All global manipulations are controlled in real-time by the user through knobs, joysticks, switches, and a two dimensional data tablet. The nearly immediate response of the display to user input via these devices provides the user with a strong sense of a "real" object. With the addition of stereoscopic viewing via a synchronized display and shutter device, the realism becomes so great that the user may be tempted to pick up the "molecule" and examine it in hand.

MIDAS emphasizes dynamic local object manipulation. Atomic bonds and labels are selectively displayed by the user. Isolating the area of interest in large biochemical systems is key to simplifying an otherwise unwieldy amount of graphical data.

Bond rotations about any non-ring bond are performed interactively, usually by assigning the rotation to a knob. This bond rotation capability coupled with the interactive ability to monitor interatomic distances and angles provides a powerful tool for quantitatively modeling molecular interactions. Selective display of molecular surfaces and partial molecular surfaces further enhances study of molecular interaction. Both bonds and surfaces may be colored independently, and surfaces are rotated and translated with the associated structure. The specifics of these modeling techniques are discussed in the following section with the associated MIDAS command.

MIDAS Command Language

In addition to device input, MIDAS also accepts command text.[5] The MIDAS command language is based on a LALR grammar [AHO79] and is formally described in appendix 2-1.

Key to the MIDAS command language is the syntax used to address molecular objects. The molecular models displayed by MIDAS are referenced by a hierarchical syntax designed to allow

[5] A complete description of the MIDAS Command Language is given in MIDAS User's Manual. [JARV85] Early in the development of MIDAS the advantages of a command language versus a menu based system were considered, and a command language based on typed commands was chosen. While a typed command language provides greater flexibility and speed, and can be implemented more easily, a menu based system is currently being developed.

selection of only the portions of the model of interest. The hierarchical scheme is as follows:

- **Models:** Models are the most general (highest) level of the hierarchy. As currently configured, up to twelve molecular models may be displayed at a time (although there is no fundamental limit to the number), and each model has both an associated name and number by which it is referenced. A model is usually single molecule, but it is not so restricted. For example, the four amino acid chains and heme group in hemoglobin may be included together in a single model.

- **Residues:** Within each model, component residue(s) are named and uniquely numbered. Residue numbers may be negative or positive (to conform with biochemical literature) and are usually sequential within a model.

- **Atoms:** Within each residue, individual atoms are referenced by a unique atom name.

The syntax developed for accessing these components of molecular models uses the symbols "#", ":", "@" as identifiers for models, residues and atoms, respectively. These identifiers, followed by the associated component name or number, are grouped into expressions which evaluate to selected portions of models. Conventions for the syntax are as follows:

(1) Identifier/components are written in order from model to residue to atom and evaluated so that a given atom identifier refers to the most recent (i.e. previous) residue and model identifier in the string. When an identifier of a higher hierarchical level follows one of lower level, intervening identifiers of lower value are ignored and the most recent higher level identifier is associated with the named component(s). For example, #0:3@ca:8@na refers to the CA atom of residue 3 on model 0 and the NA atom of residue 8 on model 0.

(2) A residue or model identifier lacking a lower level specification implies "all". E.g., #0 implies all atoms of all residues in model 0.

(3) An identifier lacking a higher level specification implies "all". E.g. #0@CA indicates all alpha carbon atoms in model 0 (all residues implied), and @CA alone indicates all alpha carbon atoms in all residues in all displayed models.

(4) Components of the same hierarchical level may be grouped under the same identifier using the "-" character to specify a range and "," to delimit an unordered list. E.g. :21-24,35 indicates residues 21 through 24, inclusive, and residue 35. MIDAS does not guarantee the order of evaluation in such cases.

(5) Additional special characters including wild card matching characters are listed in Table 2-1. Expression examples are given in Table 2-2.

The hierarchical syntax described above provides flexible and explicit access to component parts of molecular objects, allowing the user to be very general or very specific in a minimal number of key strokes. In molecular models the "area of interest" is usually not a contiguous set of residues, e.g. in proteins the active site usually includes contributions from non-sequential residues in the folded amino acid chain. In this respect, the syntax, while explicit, may be quite tedious; thus MIDAS provides an aliasing facility for frequently typed strings, a utility program, *irs*, for active site atom selection, and the notion of "sessions", where a work session may be saved so that the selection process need not be repeated when the MIDAS program is later restarted. As an example of the aliasing facility, consider the command **"alias AS #0:145:248:270"**, which is a typical specification for the active site within a model (in this case carboxypeptidase). The string "AS" can now be used in place of the explicit atom identifier expression in any other MIDAS command.

Realizing that typing in atom specifications is a disadvantage for poor typists, we included in MIDAS a system of "atom picking" using the data tablet and cursor puck to ease the tedium of typing in strings of residue and atom names. "Atom picking" allows users to point to the

Table 2-1: MIDAS Special Characters for Atom Selection

Symbol	Function	Usage
#	model number	# *model_number* where *model_number* is an integer
:	residue	: *residue* where *residue* is a residue name, residue sequence number or range of residues
@	atom name	@ *atom_name* where *atom_name* is an atom name or range of atoms
-	range	specifies a range of atoms such as @CB-* (beta carbon to the last atom), a range of residues such as :35-66 (residues 35 through 66) or a range of colors such as red-blue (shades of red, magenta, and blue)
*	wild card match	matches whole atom or residue names. For example, #0:*@CA selects the alpha carbon atoms of all residues
?	single character wild card	used for atom and residue *names* only. For example :G?? selects all three letter residue names beginning with "G"
%	every *nth* residue or atom	For example, :%5 selects every fifth residue in the sequence
b>	temperature factor	b> *temp_factor* selects all atoms with temperature factors greater than *temp_factor*. b< *temp_factor* selects all atoms with temperature factors less than *temp_factor*. For example, b>20 b<25 selects all atoms with temperature factors greater than 20 and less than 25.
e>	electrostatic potential	e> *potential* selects all atoms with electrostatic potentials greater than *potential*. e< *potential* selects all atoms with electrostatic potentials less than *potential*. For example, e>10 e<20 selects all atoms with electrostatic potentials between 10 and 20 kcal/mole.

Table 2-2: Sample Specifications for Model Components

Expression	Meaning
#0	model 0 (all atoms, all residues implied)
#0:4-10,15	residues 4 through 10 inclusive and 15 of model 0 (all atoms implied)
@CA	all carbon alpha atoms on all models, all residues (implied)
#1:12-20@CA:14@N	alpha carbon atoms in residues 12 through 20 and the nitrogen atom on residue 14. All apply to model 1.

atom of interest, and MIDAS returns the appropriate character string specification. The user types a MIDAS command substituting a "+" where the atom specification is to appear. This activates the data tablet and cursor such that a cross hair appearing on the display may be moved to the desired atom. Pressing the button on the cursor puck signals the selection, and MIDAS substitutes the full hierarchical atom name for the first occurring "+" in the typed command. Multiple "+" symbols may appear in the command line; each is sequentially assigned a complete atom specification. The command may be edited after substitution using the normal MIDAS keyboard editing characters; e.g. ^w (control-w) to erase a word. As a degenerate case, a "+" by itself without an accompanying MIDAS command still activates the atom picking facility and thus allows a user to temporarily identify a particular residue or atom, but then to ignore this information simply by typing ^u (control-u) to erase the line.

MIDAS commands are divided into two categories: *interactive commands* which change the view of the displayed objects and *editor commands* which modify the displayed object. These two command types correspond to the primary program module in which they are carried out. Editor commands are typically "molecule specific" (e.g. **addaa** for adding a new amino acid

onto the end of a molecule), while the more general interactive commands could just as easily be applied to any computer generated object shown on the display screen. Editor commands require invoking the MIDAS database editor module to modify the database and are thus slower to execute than interactive commands, which require only that an instruction be sent to the display system. The editor and interactive commands are summarized in Tables 2-3 and 2-4, respectively.

The action of most MIDAS commands may be reversed by preceding the command with the tilde character "~". This is essentially an "undo" for the commands.

The command language duplicates all the capabilities of the input devices with interactive keyboard commands: **move** translates objects, **turn** rotates objects, **scale** applies a numerical scaling factor, **clip** adjusts the clipping planes, and **select** mimics the switches for activating models. These and other interactive commands essentially change the "view" of the model(s) on the screen. The duplication in the MIDAS command language of the functionality of interactive devices is especially advantageous for constructing command "scripts". Sequences of commands are prepared as part of an ordinary text file using a standard text editor and can then be "read" into the program with the **source** command just as if the commands had been interactively typed on the graphics display keyboard. This feature is often used during filming sessions and for giving demonstrations to visitors.

A molecular model displayed in its entirety contains more information than the user either needs nor wants. It is important to be able to simplify the picture to emphasize the areas of interest or generalize the structure. The **display** command allows selective display of atoms, residues, and models. Thus, a large protein may be simplified by displaying only the backbone (i.e. eliminating the side chains). Such a simplied picture clarifies the structure allowing the viewer to easily distinguish gross secondary and tertiary structural features such as beta pleated sheets and alpha helices. The command **chain** "chains" together only the displayed atoms, ignoring intervening atoms. In this case the lines connecting the atoms are "virtual" bonds that serve to

Table 2-3: MIDAS Interactive Commands

Interactive Commands	
Command	Function
alias	set command alias
align	align two atoms along the z-axis
angle	calculate the angle between three atoms
assign	assign knobs and joysticks
cd	change current working directory
charsz	change text character size
clear	equivalent to unset
clip	move clipping planes
cofr	change center of rotation
copy	send image to electrostatic plotter
cpk	produce a space-filling model on raster terminal
fix	make bond rotations permanent
freeze	reset knobs (avoids picture drift)
help	get information on MIDAS commands
intensity	set intensity of hither and yon clipping planes
match	superimpose two models
move	translate selected models
push/pop	push and pop images on the picture stack
record	record all executed MIDAS commands in a file
repeat	fetch previous command line
reset	reset all models to original orientations
rock	rock a structure about the x, y or z axis
roll	roll a structure or bond rotation about the x, y, or z axis
run	execute a shell command and send output to MIDAS
save	save a MIDAS session
savepos	save model orientation
scale	apply scaling factor to all models
section	change sectioning of the display
select	select models for move, rock, roll, turn commands
set/unset	set options
sleep	temporarily suspend all activity
source	read and execute a command file
speed	set the control speed of knobs and joysticks
stop	terminate the current MIDAS session
system	execute a UNIX (VMS) shell command
thickness	change thickness of the displayed section
vmstat	collect performance information
transform	apply a transformation matrix to each selected molecule
translate	translate molecules in molecular coordinate system
turn	turn a structure about the x, y, or z axis
wait	interrupt processing until model has stopped moving
window	display the entire molecule on the screen

Table 2-4: MIDAS Editor Commands

Editor Commands	
Command	Function
addaa	add amino acid to the end of molecule
addgrp	add new group to a residue
brotat	"backwards" bond rotation
chain	chain specified atoms together
color	color bonds, labels and surfaces
delete	delete a group from a residue
display	display specified molecules, residues, atoms
distance	display atom distances
getcrd	return x, y, z coordinates for an atom
label	label atoms and residues
link	join two residue chains
menu	display menu of available molecular structures
open	open a MIDAS database for display
read	read a command file containing only editor commands
reverse	reverse the direction of a rotation
rlabel	enable residue labeling
(f)rotat	activate a bond rotation
setcom	set parameters for picture system objects
show	display specified atoms deleting all others
surface	display model surface
swapaa	exchange an amino acid for another
swapna	exchange a nucleotide for another
vdw	display van der Waals surface
vdwopt	set van der Waals surface options

distinguish gross structural features. For example, the alpha carbon atoms of a protein may be chained (ignoring amino nitrogen and carboxyl carbon) to produce a generalized view of the protein.

Having simplified a model structure, it is useful to carefully select details of the molecular structure for display. The portion of interest is often the active site of a protein. As more detail is added back to the model, color becomes indispensable for enabling the eye to differentiate among the various displayed structures. Protein and substrate may be colored differently and residues

involved in molecular interactions highlighted by color. Bonds, labels and surfaces (described in the next section) may be colored independently. This is of particular use for docking two surfaced models where the surfaces are very close to one another and possibly touch. The ability for the eye to distinguish between the two surfaces is crucial. Color also provides an alternative mechanism for atom identification: bonds may be color coded according to atom type, for example.

Model Surfaces

A key feature of MIDAS is the ability to display molecular surfaces. These are represented as dots about the wire model and characterize contours, pockets, and internal cavities. Studying molecule interactions is heavily dependent on accurately characterizing the molecular surfaces in the area of contact.

There are two types of molecular surfaces supported under MIDAS: van der Waal surfaces and solvent accessible surfaces. Van der Waals surfaces [BASH83] are fast to compute and provide "interactive" surfaces which accommodate bond rotations readily. MIDAS assigns a van der Waals radius to each atom according to its atom type, creates the surface and clips the areas of overlap to create a Corey-Pauling-Koltun (CPK) type model. Note that the surface of *each* atom may be represented, not just those atoms on the exterior surface of the molecule. These surfaces may be selected for display on an atom by atom basis. Surfaces for atoms internal to the exterior molecular surface of the model may be eliminated from the display using the aforementioned *irs* utility program (Interior atom Removal and site Selection).

Solvent accessible surfaces are calculated off-line using an algorithm first proposed by Lee and Richards [LEE71, RICH77] and implemented at UCSF by Connolly. [LANG81, CONN81, CONN83] The surface is calculated by rolling a theoretical water "probe" around the van der Waals surface of the molecule and using the contact reentry points to determine the surface. The resulting surface is smooth and free of "seams" between atoms. The program output, a *ms* format[6] file, is

preprocessed prior to display to create a MIDAS database surface file which is accessed by the MIDAS display program. This is a molecular surface and may not be selectively displayed on an atom by atom basis except at the level of the initial calculation. A major distinction between the van der Waals and solvent accessible surfaces is the "tearing" of the latter when a bond rotation is performed.

Surfaces are useful for visualizing contact points of intermolecular interactions. A utility program, *esp*, which calculates the electrostatic potential of a solvent accessible surface and stores the information in a MIDAS surface database allows the MIDAS display program to selectively color surface regions by electrostatic charge. Visualization of the electrostatic pattern at the molecular surface provides additional information for docking.

Structural Modifications

In general, MIDAS is not designed for making major structural changes to displayed models. For small changes, however, MIDAS has the following capabilities:

(1) Amino acid residues may be added using the *addaa* command and nucleotides added using the *addna* command.

(2) One residue may be substituted for another using the commands *swapaa* (for an amino acid) and *swapna* (for a nucleotide).

(3) The *addgrp* command enables the user to add a small chemical group from an on-line library of groups to an existing residue.

Coordinate Recovery

Coordinate recovery is of particular importance in docking problems so that visually docked models may be subjected to further computational analysis, such as energy minimization

[6] The *ms* program was written at UCSF by Michael Connolly. The program is available from the Quantum Chemistry Program Exchange (QCPE) at Indiana University.

calculations. MIDAS provides coordinate recovery via (1) a MIDAS command, **getcrd**, which returns the coordinates of a named atom to the display screen and (2) writing out the entire MIDAS session (i.e. all databases and surfaces) with the MIDAS **save** command and converting binary atom coordinate data to a text file format using the *midas.out* utility program. The latter method requires that bond rotations be "fixed" in the new positions using the MIDAS command **fix**. A saved MIDAS session may be restarted, thus making it convenient for users to break long modeling sessions into several shorter time periods.

Appendix 2-1: MIDAS Command Language Grammar

The MIDAS command language forms a nearly context free grammar[7] and hence can be described by a Backus-Naur Form (BNF) syntactic specification. [AHO79] This approach has resulted in several advantages; most importantly was that at an early stage it was possible to generate a prototype of the language interpreter by utilizing the UNIX compiler-compiler tool YACC. [JOHN75] This allowed formal analysis of the initial language in order to spot potential ambiguities and conflicts in the parser production rules. It also allowed us to get a "feel" of how the language was to be used without the necessity of implementing the complete parser. [FREE77,WASS82]

Because of the desirability to provide comprehensive expression error analysis and informative error messages, however, the final implementation of the MIDAS language parser relies not on YACC, but on a recursive-descent (top-down) parsing strategy. This method allows for the detection of syntactic errors in the input command string as soon as it is possible to do so based on a left-to-right scan of the input, normally as soon as the lexical analyzer returns the next input token. Formally speaking, then, the MIDAS command language is a LL(1) grammar, since the parser scans its input left-to-right, constructing a leftmost derivation in reverse, and examining no more than one of the next input symbols on the command line.

The BNF specification of the language follows. Notational conventions are: [] indicates exactly zero or one occurrence, {} indicates zero or more occurrences, and {}+ one or more occurrences. Capitalized words are terminal symbols (tokens) while lower case words are nonterminal symbols (syntactic variables). The nonterminal symbol "command-list" is the start symbol.

---

[7] The language is not strictly context free because arguments to some MIDAS commands are UNIX filename specifications which can themselves be arbitrary strings. However, because the MIDAS syntax analyzer is implemented by recursive descent techniques, this minor amount of context sensitivity can be localized to the lexical analysis portion of the language interpreter and hence the language grammar can be precisely described by formal BNF notation.

```
command_list  :   { command eoc }+
              ;

eoc           :   ';'     /* end-of-command */
              |   '\n'
              |   EOF
              ;


command       :   ALIAS [ WORD REMAINDER ]
              |   ASSIGN [ assign_args ]
              |   ROCK movement
              |   SET [ set_args ]
              |   UNIT [ unit_args ]
              |   COPY [ copy_args ]
              |   CPK [ cpk_args ]
              |   HELP [ WORD ]
              |   REDRAW [ selectors ] atoms
              |   VDWOPT vdwopt_args
              |   COLOR color [ ',' selectors ] atoms
              |   OPEN [ open_args ]
              |   ADD_GROUP add_group_args atoms
              |   DELETE_GROUP delete_group_args atoms
              |   compute_op atoms
              |   file_op string_list
              |   WAIT [ NUMBER [ time_unit ] ]
              |   number_op1 [ number_list ]
              |   number_op2 number_list
              |   word_op word_list
              |   movement
              |   monitor_op NUMBER atoms
              |   single_op
              |   unix_op REMAINDER
              |   db_edit_op atoms
              |   add_residue add_residue_args atoms
              |   swap_residue residue_type atoms
              |   '~' tilde_command
              |   '!' REMAINDER
              ;

tilde_command :   ALIAS word_list
              |   COFR
              |   ASSIGN physical_device
              |   SELECT number_list
              |   SAVEPOS word_list
              |   SET set_args
              |   SETCOM number_list
              |   OPEN number_list
              |   logical_device
              |   monitor_op number_list
              |   db_edit_op atoms
```

```
                ;

compute_op      :       ALIGN
                |       COFR
                |       MATCH
                |       GETCRD
                ;

word_list       :       { WORD }+
                ;

number_list     :       NUMBER { ',' NUMBER }
                ;

string_list     :       { STRING }+
                ;

assign_args     :       FKEY NUMBER TO SELECT number_list
                |       physical_device TO logical_devices
                ;

physical_device :       device_type device_index device_subindex
                ;

device_type     :       JOYSTICK
                |       DIAL
                |       TABLET
                ;

device_index    :       NUMBER
                ;

device_subindex :       X
                |       Y
                |       Z
                ;

logical_devices :       { logical_device }+
                ;

logical_device  :       XROT
                |       YROT
                |       ZROT
                |       BONDROT
                |       XTRAN
                |       YTRAN
                |       ZTRAN
                |       SCALE
                |       SECTION
                |       THICKNESS
                |       CLIP plane
```

```
                ;

plane           :       LEFT
                |       RIGHT
                |       TOP
                |       BOTTOM
                |       HITHER
                |       YON
                ;

file_op         :       CD
                |       RECORD
                |       SAVE
                |       SOURCE
                |       READ
                ;

number_op1      :       CHARSZ
                |       SELECT
                |       SPEED
                |       INTENSITY
                ;

number_op2      :       FIX
                |       REVERSE
                |       FIXREVERSE
                |       SETCOM
                ;

movement        :       logical_device [ delta ] [ duration ] [ wait_period ]
                ;

delta           :       NUMBER [ spatial_unit ]
                ;

duration        :       FOR NUMBER [ time_unit ]
                |       FOREVER
                ;

wait_period     :       AFTER NUMBER [ time_unit ]
                ;

word_op         :       SAVEPOS
                |       RESET
                ;

monitor_op      :       ANGLE
                |       DISTANCE
                |       ROTATE
                |       FROTATE
                |       BROTATE
```

```
                ;

single_op       :       FREEZE
                |       PUSH
                |       POP
                |       REPEAT
                |       WINDOW
                |       VMSTAT
                |       STOP
                ;

unix_op         :       RUN
                |       SYSTEM
                ;

unit_args       :       TIME [ time_unit ]
                |       DISTANCE [ spatial_unit ]
                |       ANGLE [ spatial_unit ]
                ;

set_args        :       { set_arg }
                ;

set_arg         :       CONFIG
                |       HALFBOND
                |       INDEPENDENT
                |       LABELS
                |       LIGHTS
                |       REASSIGN
                |       ORTHO
                |       PAIR
                |       SINGLE
                |       STEREO
                |       TEXT
                |       VERBOSE
                ;

time_unit       :       SECOND
                |       MINUTE
                |       FRAME
                ;

spatial_unit    |       DEGREE
                |       RADIAN
                |       ANGSTROM
                |       PS
                ;

atoms           :       { model residue atom tfrange esrange }
                ;
```

```
model         :   [ '#' word_ranges [ count ] ]
              ;

residue       :   [ ':' word_ranges [ count ] ]
              ;

atom          :   [ '@' word_ranges [ count ] ]
              ;

word_ranges   :   WORD [ '—' WORD ]
              ;

count         :   '%' NUMBER
              ;

tfrange       :   { tfspec }
              ;

tfspec        :   TF '>' NUMBER
              |   TF '<' NUMBER
              ;

esrange       :   { esspec }
              ;

esspec        :   ESP '>' NUMBER
              |   ESP '<' NUMBER
              ;

db_edit_op    :   DISPLAY
              |   CHAIN
              |   LABEL
              |   SHOW
              |   RLABEL
              |   VDW
              |   SURFACE
              |   LINK
              ;

selectors     :   selector { ',' selector }
              ;

selector      :   BOND
              |   LABEL
              |   SURFACE
              |   VDW
              ;

vdwopt_args   :   { vdwopt_arg }
              ;
```

```
vdwopt_arg      :   RADII STRING
                |   DENSITY NUMBER
                |   NUC
                |   PROT
                ;

color_selectors :   color [ ',' selectors ]
                ;

color           :   NUMBER
                |   RED
                |   GREEN
                |   BLUE
                |   CYAN
                |   MAGENTA
                |   YELLOW
                |   WHITE
                ;

open_args       :   [ selector ] NUMBER STRING
                ;

add_group_args:     group_type ',' NUMBER ',' NUMBER [ ',' NUMBER
                    [ ',' WORD ] ]
                ;

group_type      :   WORD
                ;

add_residue     :   ADDAA
                |   ADDNA
                ;

add_residue_args :  residue_type ',' NUMBER ',' NUMBER
                    [ ',' NUMBER ]
                ;

swap_residue    :   SWAPAA
                |   SWAPNA
                ;

residue_type    :   WORD
                ;

copy_args       :   { copy_arg }
                ;

copy_arg        :   BOX
                |   DATE
                ;
```

```
cpk_args        :       PLOT
                ;

help_args       :       WORD
                ;
```

Chapter 3

Abstract

The Molecular Interactive Display And Simulation (MIDAS) Database System is a hierarchical database specifically designed for complex macromolecular models such as proteins and nucleic acids. Each molecular model consists of one or more molecules made of up a linear sequence of smaller units which we loosely refer to as "residues". Each of these residue units, in turn, is composed of one or more even smaller units called "atoms". The complete model may simply consist of a single component molecule, as in the case of a water molecule, or it may be a long chain of more complex residue components such as in the case of amino acid and nucleic acid sequences. Complex functional groups such as heme and NADH may also be specified as single subunit components, and these groups can then be incorporated into a larger model so as to form a single complex.

The various model component types are defined as arbitrary graphs of atoms with defined starting and ending points. The component type thus defines the connectivity of atoms for that component, as well as the linkage atoms to adjacent model components. Molecular "data" is stored in the "leaves" of the database hierarchy and is therefore directly associated with the atoms of a particular residue component, the component having been specified by type and position in the sequence of residues making up the chain. Individual atom data, however, is not restricted to a specific format or quantity, thereby allowing both flexibility and future extensions to easily be made to the database.

Background

As described in chapter 2, the MIDAS (Molecular Interactive Display and Simulation) system was designed for real-time modeling of proteins and nucleic acids and their interactions with

each other and with small molecules such as drugs. [FERR86b] The design of a real-time molecular display system requires an underlying data structure which efficiently stores the large amounts of data associated with a macromolecular model and provides sufficiently fast access to support the real-time needs of the associated display program. Previous molecular modeling programs, for example CAAPS, [LANG75] have combined the database access primitives within the the basic program structure of the display program. This lack of program modularity is only a symptom, however, of a more fundamental deficiency within these early generation molecular modeling programs with regard to their database design. While existing relational database systems provided a convenient program interface, until recently these database management systems were limiting because of their relatively slow access time and large disk space requirements. Lack of an adequately performing database system for representing macromolecular data was the major motivation for our undertaking of our own database system design.

The Nature of Macromolecular Data

The currently available macromolecular structure data[1] is largely the result of crystallographic studies of proteins and nucleic acids. These protein and nucleic acid biomolecules are built from smaller component molecules called residues and bases, respectively, and are chained linearly into large structures. (Although classically the term "residue" applies only to the amino acid subunit components of protein molecules, we use the term throughout this paper to indicate the generic repeating subunit component present in both proteins and nucleic acids.) The residues are typically less than 100 atoms in size and have a single linkage atom to the next residue in the linear sequence and a single linkage atom to the previous residue in the sequence. Additionally, in the case of proteins, there may be a specialized linkage between two non-adjacent residues in the sequence via a disulfide bond. A complex molecular model may consist of one or more

[1] Published molecular structure data is archived and distributed by the Brookhaven Protein Data Bank.

chains of residues of varying length. Some chains may consist of only a single residue. For example, the hemoglobin molecule consists of four amino acid chains and four single residue heme groups, totaling 578 residues and 4556 (non-hydrogen) atoms.

The task of managing molecular models of this complexity from within a real-time modeling program would be nearly overwhelming even with present day existing hardware[2] were it not that the diversity of biomolecules is due largely to residue sequence variations of only 20 common amino acids in proteins and 4 common nucleotides in nucleic acids. This redundancy in the structure of component residues of large biomolecules provides an efficiency in nature which can be used to advantage in our database design. Residue structure data such a atom names, bonding pattern, and linkage atoms need only be specified once, and then applied to all like residues within a model. Only the actual coordinate data (3-D position in a Cartesian coordinate system) is explicitly required for each atom. Using the example of hemoglobin again, there are only 20 unique residues among the total 578 residues. (The amino acid isoleucine does not appear in hemoglobin. Thus, there are 19 unique amino acids and the heme "residue").

The nature of the macromolecular structure data thus lends itself to a hierarchical modeling approach. Each molecular model consists of one or more sequences of residues. Each residue, in turn, consists of atoms which are connected in a specific bonding pattern. Associated with each atom is the position of that atom in a Cartesian coordinate system. This hierarchy is summarized below:

- **Model:** consists of one or more chains of residues, each chain consisting of one or more residues.
- **Residue:** consists of one or more atoms. All atoms within a given residue are connected to form an arbitrary graph.

[2] The MIDAS database system was originally developed on a DEC PDP-11/70 computer for use with an Evans and Sutherland Picture System 2 graphics display.

- **Atoms:** form the leaves of the hierarchy. Data associated with the individual atoms usually consists of atomic coordinates, but molecular surfaces or other arbitrary data may be stored as well. Each atom in the entire model can be uniquely described by a model number, residue sequence number, and atom name.

Conceptually this data model can be viewed as a "tree" of nodes (i.e. a hierarchy), where each leaf in the node contains coordinate data (see figure 1). For attributes such as color, each leaf node can logically be considered as being subdivided into separate nodes for atomic bonds, labels and surfaces.

Design Goals

As described in the previous chapter, a major goal in the design of MIDAS was to build a modeling system which processes the large amounts of data associated with macromolecules using little main memory, gives real-time performance with minimal CPU utilization, and does not use excessive amounts of disk space for storing the molecular database. Clearly, the database subsystem is a critical component in achieving this goal. Other important goals related to the database subsystem include the desirability of storing both coordinate and non-coordinate data (e.g. atom and residue names), the ability to modify existing database records as well as to add entirely new records to an existing database (to facilitate, for example, the replacement of one amino acid residue with another), the ability to do generalized "search" operations for data within a database, and, lastly, to provide a means of extending the types of data stored within the database so as to be able to include the results derived from new ideas and discoveries. This last goal is particularly demanding, since it likely includes new types of data records and relationships which were not originally considered by the database designer; another way of stating this goal is to say that it is desirable to maintain an "open" database architecture.

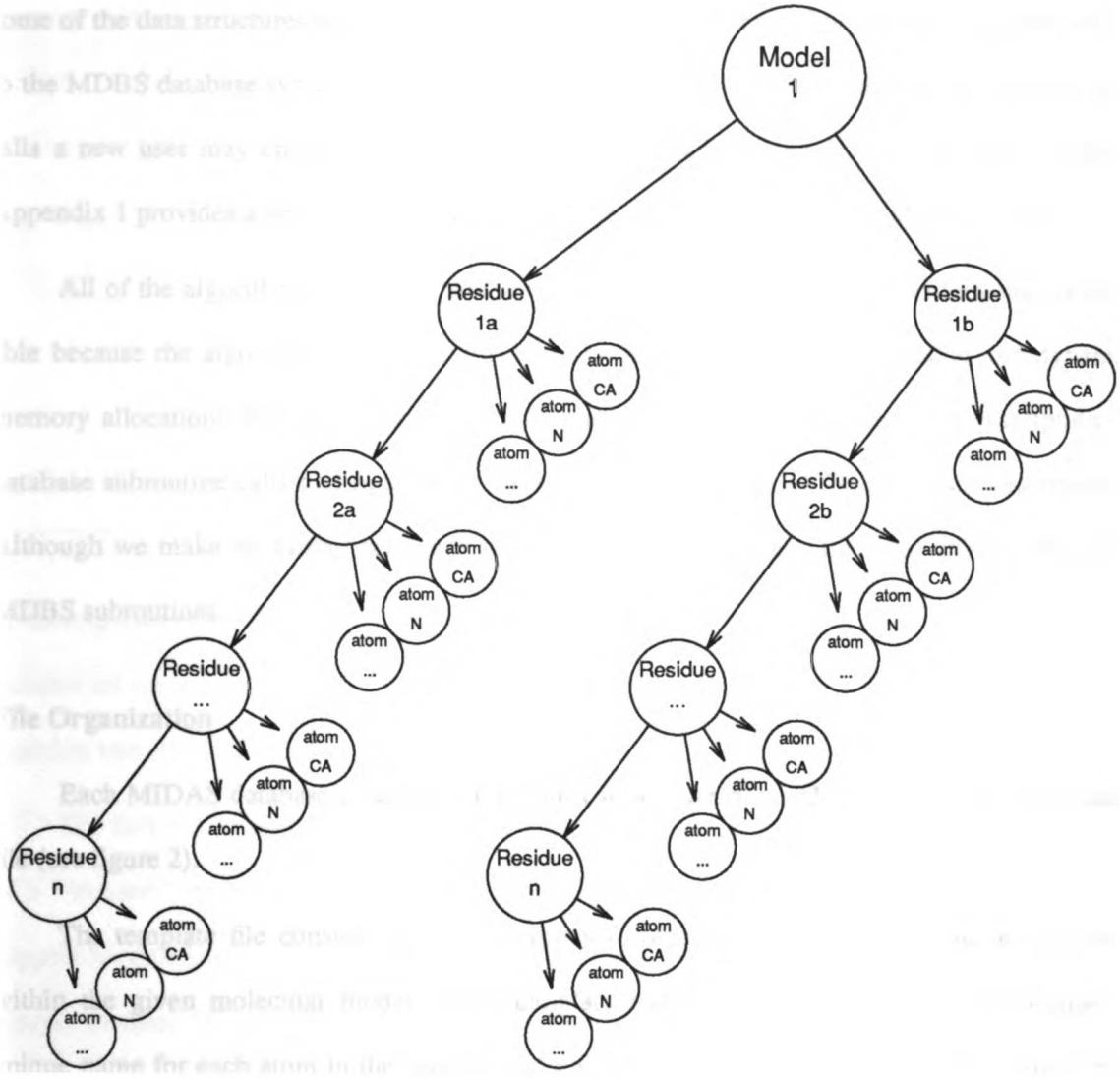

Figure 3-1: Conceptual Organization of an MDBS Hierarchical Database

Database Description

We first describe the general file organization and design philosophy of the MIDAS Database System (MDBS). Next we provide a detailed description of each of the user-level entry points into the MDBS package. Following this we give two brief examples of programs which

use the MDBS routines for creating a MIDAS database and then displaying the resulting model on a simple graphics display system. A brief section follows these examples which describes some of the data structures unique to the MIDAS display program and how these structures relate to the MDBS database system. Also included in this section is a list of some of the common pitfalls a new user may encounter when using the MDBS subroutines for the first time. Finally Appendix 1 provides a detailed description of the internal structure of the database records.

All of the algorithms described here are programmed in C. [KERN78] This language is suitable because the algorithms make extensive use of structures, pointers, recursion and dynamic memory allocation. Familiarity with the C language is assumed in the detailed description of database subroutine calls and in some of the more technical parts of the algorithm descriptions. Although we make no further reference to it, a Fortran-77 language interface also exists to the MDBS subroutines.

File Organization

Each MIDAS database consists of three disk files: a *template* file, an *index* file, and a *data* file (see figure 2).

The template file consists of distinct entries for each different type of residue contained within the given molecular model. Each template record contains the residue type name, a unique name for each atom in the residue, and a description of the connectivity of atoms within the residue, including which atoms are the first and last in the template (i.e. those atoms which connect to the previous and next residues in the polypeptide chain, respectively). As will be discussed below, the observant reader will note that template records are still "pinned" [ULLM80] by this data design since an "index value" still points to particular records in the template file. This restriction does not prevent residues within existing databases from being substituted, or in some other way fundamentally modified, however, since the only possible consequence to the template file is the necessity of creating a new residue record at the end of the file. The only disadvantage

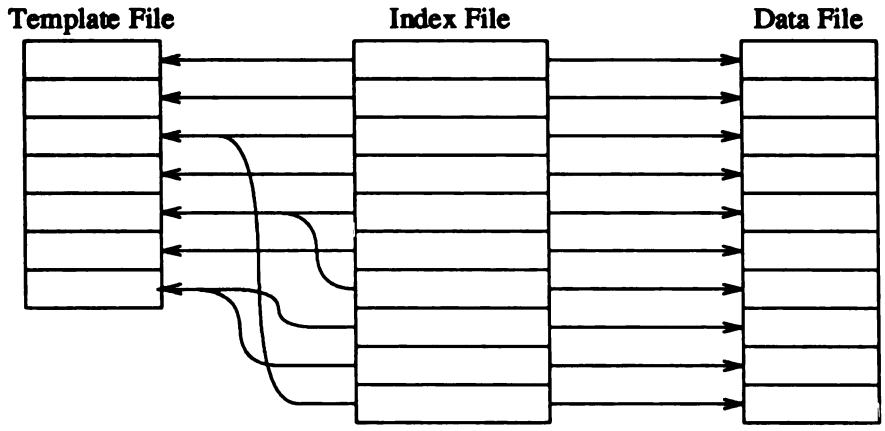

Figure 3-2: MDBS Physical Database Organization

of having pinned records in the template file is that the file cannot be kept sorted. Since the number of records within the file is always relatively small (range 15 - 30), efficient in-core searches can always be made.

The fact that template files consist of only pinned records also does not prevent the concept of a "master" template file, provided the underlying operating system provides the necessary support for multiple links to the same data file (i.e. multiple directory entries referring to the same filesystem data). This would potentially allow for a single template file to be used for all protein and nucleic acid databases, since the total number of entries would only be of order 30 (20 amino acids plus 4 nucleotides plus a few special cases). Although the UNIX implementation of the MDBS subroutines use the shared template feature when saving away modified databases, this feature of the database design is not considered crucial to its success. The chief reason for this is that the data storage requirements for template files is already so minimal that the potential additional space savings is not significant. For example, the total storage requirements for all of our present 266 protein and nucleic acid data structures consists of 17.5Mbytes of disk space. Of this

total, 0.7Mb (4%) is used for storage of template files, 1.4Mb (8%) is used for storage of index files (see below), and the remaining 15.4Mb (88%) is used for storage of data files. We note in passing that the equivalent Protein Data Bank standard ASCII data representation requires 42.1Mb of data storage; thus MDBS represents a 42% space savings for ASCII versus MDBS data representation formats.

This significant space savings is also complemented by the far better data access times available with when using a random access technique such as that provided by a "dense index". [ULLM80] In MDBS, the index file contains a table giving the ordering of residues within a molecule. This file thus serves as an index into both the template file (which describes the structure of each residue) and the data file (which contains the coordinate or other data associated with each residue). References to records in the template file are made via an index to the particular residue structure in the file, while references to the records in the data file are via a direct pointer (equivalent to "file offset").

The existence of both the template file (contains the residue name) and the index file (contains the residue sequence number) provides for fast searches of the database. Searching by residue type proceeds linearly through a small in-core table (i.e. the template file), while searching by sequence number is done via an in-core binary search [KNUT72] of a somewhat larger table[3] (i.e. the index file); thus a database "seek" operation always completes very quickly and without any accesses to disk.

The data file contains only user-defined data associated with each residue. No structure is imposed on this data other than that it must be of fixed length records and that the record size must be known. Users may store arbitrary data in this file. Because all data records are referenced with a direct pointer, all data can be accessed within a single disk head seek. This provides

[3]This can be done because the alpha-numeric character strings representing sequence numbers are presorted lexicographically in the index file.

for optimum data access and, if fact, makes MDBS viable in today's typically rich virtual memory environment. The fact that access to a particular virtual memory page on secondary disk storage also takes a single disk head seek as well means that the two access methods are roughly equivalent. The difference is that MDBS makes all the disk access decisions directly, while with virtual memory the disk access decisions are made by the operating system. Since disk access time is at least three orders of magnitude greater than main memory access time it makes little difference who makes the decision.

Records in the data file appear in random order and thus disk blocks are completely filled. This is most important for MDBS data files, since they comprise approximately 90% of the disk storage requirements for any particular database (see above). New records are always added at the end of the data file, and deletions cause "holes" to appear in the file. Since deletions are an infrequent operation this deficiency is not considered significant.

More specific information on the internal structure of MDBS records can be found in Appendix 1. For most purposes, however, the subroutine package described here is sufficient to manage the database files, and potential users need not be concerned with the details of the internal file structure.

Subroutine Descriptions

The MDBS subroutine package provides a powerful and efficient means of accessing MIDAS databases, however in order to effectively utilize the package it is necessary to understand how stored information is accessed. In general, there are three places in which information may be held:

(1) On disk: The MDBS disk files described above define the database and provide long term storage.

(2) In internal buffers: Storage buffers in main memory are allocated and managed by the MDBS subroutine package when the database files are accessed.

(3) In user defined buffers: MDBS subroutines return information requested by the user into space allocated by the user.

This tri-level data storage scheme allows a program to minimize the number of disk accesses for a given task. For example, searching the database for a specific residue determines if that residue exists, and this operation can be performed without actually reading any of the coordinate data associated with the residue. Likewise, it is possible to retrieve information such as the residue sequence number and type, and the number of atoms contained within the residue, but then only access the atom coordinate information if it is explicitly needed. As an example, consider the steps for updating the coordinates for a single atom in a database record:

(1) Open the database for reading and writing,

(2) Locate the desired information, usually by seeking to a specific residue,

(3) Read all the atom data associated with the residue into main memory,

(4) Retrieve specific atom data from the internal MDBS buffer into user buffer space,

(5) Modify atom data and copy it back to the internal buffer,

(6) Write the residue data from main memory to disk when all modifications for a specific residue are complete,

(7) Close the database when all residue changes have been made.

Thus, database changes can be made on a convenient atom-by-atom basis, utilizing only enough user defined buffer space to hold a single atom's worth of data at any one time. This atom-by-atom access technique incurs little performance penality because disk accesses are internally buffered on a residue-by-residue basis.

The file */usr/include/mdbs.h* must be included in programs which utilize the MDBS subroutine package. If the programmer/user is accessing a database used by the MIDAS display program, the file */usr/include/midas.h* must be included as well. This latter file contains the structure definitions for the atom coordinate data and graphics display status and color data.

The routines available for manipulating databases can be divided into 4 categories as shown in table 3-1. The following section describes each of these routines in detail:

1. Mopen

Calling protocol:

```
mopen(database, mode)
char *database, *mode;
```

where *database* is the name of the database to be opened, and *mode* is the mode with which it should be opened.

Mopen is used to open a database in a specified mode. The available modes are :

read	"r"	open an existing database for reading only
read/write	"rw"	open an existing database for reading and writing
write	"w"	create and open a database for writing only

Note that opening a database in "read" or "read/write" mode implies that the database already exists. Conversely, opening a database in "write" mode implies that the database does not exist and should be created. Creating a database which already exists truncates the original database. If an existing database is opened in "rw" mode (thereby indicating the user intends to make changes), a temporary copy of the data file (*database*.dat) is made so that the original database will not be lost if the user subsequently decides not to save away changes after all. See the description of *msave* for additional information on saving modified databases.

Usually a database is accessed on a residue-by-residue basis. This means that data for an entire residue is stored in an internal buffer while updates and additions are made. The entire

Table 3-1: MDBS Subroutine Summary

(1) Routines for opening and closing the database files:

mopen	Open a new or existing database
mclose	Close a database
msave	Save the current (and presumably modified) database

(2) Routines to locate residue information:

mseekr	Seek to a given residue
mrconn	Determine connectivity of two residues

(3) Routines for passing information between disk and main memory:

mreadr	Read residue information into memory
mreada	Read atom information into memory
mwrtr	Write residue information
mwrite	Write atom and residue information
mwrta	Mark current residue as complete and advance to next residue
mflush	Force information to be written out to disk

(4) Routines for passing information between internal buffers and user buffers:

mseeka	Find an atom index
matom	Find an atom name
mlink	Return the index of the linkage (last) atom of a residue
mchief	Return the index of the chief (first) atom of a residue
mdatptr	Return a pointer to the atom data currently in main memory
mgeta	Copy atom information to a user buffer
mputa	Copy atom information from a user buffer
mtrav	Traverse a residue connectivity graph
maconn	Determine connectivity of two atoms
mchain	Determine whether an atom is a mainchain or sidechain atom
mamap	Return indices of all atoms connected to a given atom

In addition, there are three string comparison routines which recognize MIDAS "wildcard" characters:

amatch	Atom name string comparison
smatch	Sequence number string comparison
tmatch	Atom type string comparison

residue is written to the disk file upon completion of the updates. If large amounts of data are associated with each atom in a residue, however, the entire residue may not fit into main memory on a small computer. In this case the database may be accessed on an atom-by-atom basis, designated "slow" mode.

In "slow" mode, each change to the database is written out to disk immediately instead of being stored in an internal buffer. As the name implies, this is a much slower method of access and should be employed only when the size of the atom data prohibits access in the default "residue-by-residue" mode. "Slow" mode is invoked by adding "s" to any of the above access modes. For example, "rws" indicates read/write in "slow" mode.

In summary, the expected mode specification is a string of characters.

To specify read mode, the string should include "r".
To specify write mode, the string should include "w".
To specify slow mode, the string should include "s".

Thus, a call such as

```
mopen("throx", "rs");
```

means to open the database *throx* for reading in slow mode.

Mopen returns a small positive integer identifying the particular database "stream" through which the database can be referred to in subsequent operations. If *mopen* is unable to open the named database, a value of -1 is returned.

2. Mseekr

Calling protocol:

```
mseekr(stream, residue, mode)
char *residue;
```

where *stream* is the integer database identifier returned by *mopen, residue* is a character string containing the name of a residue to be matched, and *mode* is an integer bit mask whose value is defined below.

Mseekr sets the position for the next input or output operation on *stream* by searching for a particular residue in the database. The search is merely a locating process and does not involve retrieving any information; it does, however, verify that a residue meeting the user's search cri-

teria has been found. *Mseekr* returns the number of atoms contained in the given residue or -1 if the search fails.

The octal mask bits set in *mode* determine how the search is done:

Table 3-2: Mseekr Option Modes

Bit	Set	Reset
01	*residue* = type name	*residue* = sequence number
02	Search backward	Search forward
04	Search from logical start of model	Search from current position
010	Start of model = last residue	Start of model = first residue

Note that the starting position of the search is only significant if bit 04 is set and if *residue* is a type, since sequence numbers are unique within a database. If *residue* is a type, then the search proceeds to the logical end of database but does not wrap around. If bit 01 is set, then '*' may be used to match any residue type and '?' may be used to match any single character embedded in a string. To position *stream* at one end of the database, seek from the desired end for residue type '*'. Searching by residue type proceeds linearly through a small in-core table, while searching by sequence number is done via an in-core binary search of a somewhat larger table (this can be done because the alpha-numeric character strings representing sequence numbers are presorted lexicographically in the index file); thus the *mseekr* operation always completes very quickly and without any accesses to disk.

Thus, to search for the next occurrence of "TYR" in the forward direction, the call is

```
mseekr(stream, "TYR", 01);
```

whereas, to search for the first occurrence of "TYR" from the "back end" of the molecule, the call is

```
mseekr(stream, "TYR", 017);
```

Note that this call merely locates the desired residue and does not retrieve any data. Data is retrieved using the *mreadr* and *mreada* routines.

3. Amatch, Smatch and Tmatch

Calling protocol:

```
amatch(str1, str2)
smatch(str1, str2)
tmatch(str1, str2)
char *str1, *str2;
```

where *str1* and *str2* are character strings to be compared.

Amatch performs a string comparison of two atom names, *str1* and *str2*. *Smatch* performs a string comparison of two residue sequence numbers, *str1* and *str2*. *Tmatch* performs a string comparison of two atom types, *str1* and *str2*. The routines return zero if the strings match, non-zero if there is no match. The wildcard characters "*" (entire string match) and "?" (single character match) are recognized. Note that the "*" wildcard stands alone and may not be used to match partial strings.

4. Mseeka

Calling protocol:

```
mseeka(stream, atomname)
char *atomname;
```

where *atomname* is the name of the atom to be matched.

Once the desired residue is located, *mseeka* may be used to find the index of a particular atom within the residue. This function is useful when only the name of the desired atom is known. For example, when creating a new database it may be necessary to know the index of an atom in order to save the associated coordinate information with the *mputa* subroutine. *Mseeka* returns the specified atom index as an integer value. A -1 return value indicates that the atom does not exist.

5. Mchief and Mlink

Calling protocol:

```
mchief(stream)
mlink(stream)
```

where *stream* is the integer database identifier returned by a previous call to *mopen*. *Mchief* returns the index of the "first" atom in the current residue. This is the defined starting point (atom) for any residue, and it is this atom that forms a bond with the previous residue in a sequence of multiple residues.

Similarly, *mlink* returns the index of the "last" atom in the current residue. This is the linkage atom to the next residue in the sequence, if one exists.

6. Maconn and Mrconn

Calling protocol:

```
maconn(stream, index1, index2)
mrconn(stream, seq_num1, seq_num2)
char *seq_num1, *seq_num2;
```

where *index1* and *index2* are the indices of two atoms in the current residue denoted by *stream* and *seq_num1* and *seq_num2* are residue sequence numbers.

Maconn and *mrconn* are used to determine atom connectivity and residue connectivity respectively. *Maconn* returns:

0	if the atoms are not connected
1	if *index1* precedes *index2*
2	if *index2* precedes *index1*
3	if *index1* and *index2* are the same atom
-1	if either atom index is invalid.

Similarly, *mrconn* returns:

0	if the residues are not connected

1 if *seq_num1* precedes *seq_num2*
2 if *seq_num2* precedes *seq_num1*
3 if the residues are one in the same
-1 if either sequence number is invalid.

7. Mchain

Calling protocol:

```
mchain(stream, index)
```

where *index* is an atom index in the current residue denoted by *stream*.

Mchain determines whether an atom is part of the mainchain of a molecule or, alternatively, part of a sidechain. *Mchain* returns:

0 if the atom is part of a sidechain
1 if the atom is part of the mainchain
-1 if an error occurs.

8. Mamap

Calling protocol:

```
mamap(stream, index, indices)
int *indices;
```

where *index* is the atom index of an atom in the current residue denoted by *stream* and *indices* is a pointer to an array of integers.

Mamap returns the number of connections to atom *index* within the residue template and places the indices of these atom connections in the array pointed to by *indices*. Note that the indices are returned only for those atoms which reside in the template of the current residue. All connecting atoms are returned regardless of whether there is data associated with the atom (this distinction is further elaborated upon in the next section). The array which receives the index values should be declared by

```
int  indices[MAXTO];
```

where *MAXTO* is defined in */usr/include/mdbs.h*.

9. Mreadr

Calling protocol:

```
mreadr(stream, resseq, restype)
char *resseq, *restype;
```

where *stream* indicates a previously opened database and *resseq* and *restype* are character arrays allocated by the user.

Mreadr retrieves the sequence name, type, and number of atoms of the current residue. The name and type are stored in the user supplied character arrays and the return value of *mreadr* is the number of atoms in the residue. This call is useful for reading through a molecule sequentially (perhaps to determine the residue sequence), but without reading the associated atom data.

10. Mreada

Calling protocol:

```
mreada(stream)
```

Mreada reads the atom data (typically atom coordinates, but potentially a molecular surface description or any other user defined information) for the current residue from disk into an internal buffer. Note that the database *stream* must have been previously positioned to the appropriate residue. This call makes the atom data available to the user through subsequent calls to *mgeta* or *mdatptr*. Note that *mreadr* and *mreada* have distinct functions thus enabling the user to *selectively* read data and avoid the expense of unnecessary disk accesses.

Mreada does nothing if the database was opened in "slow mode", since no internal buffering is done in this case.

11. Matom

Calling protocol:

```
matom(stream, index, atom_name)
char *atom_name;
```

where *index* is the index of the atom whose name is to be returned, and *atom_name* is the user buffer where the name is to be stored.

Matom retrieves the name of the atom specified by *index* for the residue denoted by *stream*. As an example, if a user has made a call to *mlink* to find the index of the linkage atom, a call to *matom* then retrieves the name of that atom.

12. Mdatptr

Calling protocol:

```
char *mdatptr(stream)
```

Mdatptr returns a pointer to the internal MDBS buffer which contains all the atom data for the residue denoted by *stream*. Note that the atom data must already have been read into main memory by a previous call to *mreada*. The *mdatptr* routine can be used to directly access atom data stored within an internal MDBS buffer, thereby avoiding the overhead associated with copying the information into user allocated buffer space. *Mdatptr* returns a pointer of type *char* because, in general, the datum size is not fixed. For MIDAS coordinate databases however, the pointer returned will be of the type *struct *atom_def* as defined in *midas.h*. See appendix 3-1 for the definition of the atom data structure used by MIDAS.

MDBS atom entries are stored alphabetically by atom name. Thus, if a pointer returned by *mdatptr* is incremented, the atoms will be referenced in alphabetical order.

Mdatptr returns a NULL if the database was opened in "slow mode", since an internal buffer does not exist in this case.

13. Mgeta

Calling protocol:

```
mgeta(stream, index, buffer)
char *buffer;
```

where *index* is the index of the atom whose information is to be copied, and *buffer* is where the information will be stored.

Mgeta copies the information associated with atom *index* into the user supplied buffer. The appropriate residue must have been read into main memory using *mreada* before the call to *mgeta*. If the database is being accessed in "slow" mode, the appropriate residue need only have been located (with a *mseekr* call, for example).

The user must ensure that the buffer provided is large enough to accommodate the data returned. For MIDAS coordinate databases, the space necessary is given by the C expression *sizeof (struct atom_def)*, however for user defined data structures the buffer size will most likely be different.

14. Mtrav

Calling protocol:

```
mtrav(stream, visit, again)
int visit(), again();
```

where *visit* and *again* are functions to be called when each atom is visited and "returned to".

Mtrav is used to "traverse" through a residue. Since residues are defined as graphs with distinct starting and ending points, it is possible to trace a path through the atoms of the residue. Each time an atom is first reached via a new path, *visit* is called. Subsequent occurrences during traversal of this same atom as part of the same path will incur calls to *again*.

The user functions *visit* and *again* are called with the following arguments:

visit(stream, index, ischief, islinkage, nson, firstime);
again(stream, index, ischief, islinkage, nson);

where *index* is the index of the current atom. *Ischief* is an integer which is non-zero if the current atom is the first atom of the residue. *Islinkage* is non-zero if the current atom is the last atom of the residue. *Nson* is the number of atoms which are connected to the current atom and are yet to be visited. *Firstime* is non-zero if the atom is being visited for the very first time (it is possible for an atom to be visited twice via different paths if it is part of a ring).

Mtrav is most useful for generating instructions for drawing a residue. To do this, *visit* would generate a *"lineto"* drawing command, and *again* would generate a *"moveto"* drawing command. (See the example at the end of this section).

15. Mwrtr

Calling protocol:

mwrtr(stream, resseq, restype, natom, atomsize)
char *resseq, *restype;

Mwrtr writes the residue information of the current residue on *stream* and returns the number of atoms in the residue. The residue sequence name and type name should be provided in *resseq* and *restype* respectively. *Natom* is the number of atoms in the residue. If *natom* is negative, then the number of atoms is taken to be the number of atoms defined in the residue template. *Mwrtr* can be thus be used to determine the number of atoms in a residue. *Atomsize* is the size of an atom datum in bytes.

This routine must be called before *mputa* if the residue is being modified or created for the first time.

16. Mputa

Calling protocol:

mputa(stream, index, buffer)
char *buffer;

Mputa copies the data for atom *index* from the user space *buffer* into the current residue on *stream.*

17. Mwrite

Calling protocol:

mwrite(stream, resseq, restype, atoms, natom, atomsize)
char *resseq, *restype;
char *atoms;

Mwrite combines the functionality of *mwrtr* and *mputa*; it writes a residue whose sequence and type are *resseq* and *restype* respectively. The associated atom data are found at memory locations beginning at *atoms*, and there are *natoms* pieces of data each of size *atomsize*. If *stream* is at the end of the database or *resseq* does not match the current residue sequence number, then the residue is appended to the end of the database. Otherwise, the current residue on *stream* will be replaced by the new residue.

18. Mwrta

Calling protocol:

mwrta(stream)

Mwrta sets a flag to indicate all data for the current residue specified by *stream* has been stored into an internal MDBS buffer and that processing for the current residue is complete. The next call to *mreadr* then returns the residue information for the next residue in sequence. Making this call before a call to *msave* insures that updating of the temporary database file is complete. This routine need not be called if *stream* is open in "slow mode", since in this case the data was already written to disk by a previous call to *mputa*.

19. Mflush

Calling protocol:

```
mflush(stream)
```

Mflush forces any data in the internal buffer associated with *stream* to be written out to disk. This routine is not normally called explicitly, but is instead used internally by the MDBS database subroutines; it's description is included here only for completeness. *Mflush* differs from *mwrta* in that the former always writes data to disk immediately, while the latter only writes out data when absolutely necessary.

20. Msave

Calling protocol:

```
msave(stream, newdb)
char *newdb;
```

were *newdb* is the name of the database into which the possibly changed data will be stored.

Msave copies the temporary database associated with *stream* to a new database named *newdb*. The temporary database must be open for both read and write access. If the new database name is omitted, then the original database is updated. The temporary database continues to remain open after the procedure call. Note that in order for the new database to contain all modifications, a call to *mwrta* must be made before *msave* is called. *Msave* will attempt to share template files between the original and new databases if possible. (See the section on file organization for additional details.)

Msave is useful for saving modified data without disturbing the parent database. Since all database changes are made initially to a temporary disk file rather than the original disk file, *msave* functions the same for both default and "slow" access modes.

21. Mclose

Calling protocol:

mclose(stream)

Mclose is called to close an open database. If the database was opened for writing only, then the internal MDBS data buffers are flushed so that the database will reflect all updates. If the database was opened for both read and write access, changes are **not** implicitly saved; to save changes in this case a call must first be made to the *msave* routine. *Mclose* also frees any previously allocated internal data buffers.

Examples

Two simple examples demonstrate the ease with which the above described subroutines may be used. In these examples, some error checking code and initialization routines have been omitted in order to retain clarity.

The first example involves converting a "Kraut" format database[4] into a MIDAS format database. In this example, *readin()* is a routine which reads in the data on one "card" image in a Kraut file and returns the type of card that was found. *Doinit()* and *done()* perform ancillary user processing for initialization and program clean up prior to exiting.

[4]The Kraut database format was originated by Professor Joe Kraut at the University of California, San Diego. It's use in recent years has declined in favor of the more popular Protein Data Bank format.

```
#include <mdbs.h>

char    *midasfile, *krautfile;

main(argc, argv)
int     argc;
char    *argv[];
{
        int     db, index, natom, firstres;
        char    resseq[RES_SEQ_SIZE+1], restype[RES_TYPE_SIZE+1];
        char    atname[AT_NAME_SIZE+1];
        int     coord[3];       /* space for atom coordinates */

        doinit(argc, argv);     /* user initialization */
        db = mopen(midasfile, "w");     /* open database stream  */
        firstres = 0;
        while (type = readin(resseq, restype, atname, coord)) {     /* read record */
                if (type == RESIDUE) {          /* readin returned a RESIDUE record */
                        if (!firstres) {        /* if this is not the first residue... */
                                mwrta(db);      /* write out previous residue atom data */
                                firstres = 1;
                        }
                        /* now store residue information for new residue type */
                        natom = mwrtr(db, resseq, restype, -1, sizeof(int)*3);
                }
                else if (type == ATOM) {        /* readin returned an ATOM record */
                        index = mseeka(db, atname);     /* find the index of the atom */
                        mputa(db, index, coord);        /* store the data in core */
                }
                else if (type == END) {         /* readin returned END of file */
                        mwrta(db);              /* write out last residue to disk */
                        break;
                }
        }
        mclose(db);     /* close the database */
        done();         /* cleanup and exit gracefully */
}
```

The second example shows how the database created in the previous example can be used to draw a picture of the molecule on a simple graphics display system. Again, *doinit()* and *done()* perform ancillary processing, while *moveto()* and *lineto()* are subroutine calls to a graphics library package and are use to position the CRT beam and to draw a line respectively.

```
#include <mdbs.h>

char    *midasfile;

main(argc, argv)
int     argc;
char    *argv[];
{
        int     db, natom;
        char    resseq[RES_SEQ_SIZE+1], restype[RES_TYPE_SIZE+1];
        char    atname[AT_NAME_SIZE+1];
        int     visit(), again();

        doinit(argc, argv);             /* user initialization */
        db = mopen(midasfile, "r"); /* open an existing database for reading */
        /* retrieve residue sequence number and type */
        while ((natom = mreadr(db, resseq, restype)) > 0) {
                mreada(db);             /* read the atom coordinates into core */
                mtrav(db, visit, again);    /* traverse the residue */
        }
        mclose(db);
        done();
}

visit(db, index, ischief, islink, nsons, firstime)
int     db, index, ischief, islink, nsons, firstime;
{
        int     coord[3];
        static int firstatom = 1;

        mgeta(db, index, (char *)coord);            /* retrieve coordinate data */
        if (firstatom) {
                moveto(coord[0], coord[1], coord[2]);       /* PS2 library call */
                firstatom = 0;
        } else
                lineto(coord[0], coord[1], coord[2]);       /* PS2 library call */
}

again(db, index, ischief, islink, nsons)
int     db, index, ischief, islink, nsons;
{
        int     coord[3];

        mgeta(db, index, (char *)coord);            /* retrieve coordinate data */
        moveto(coord[0], coord[1], coord[2]);       /* PS2 library call */
}
```

Miscellaneous Details

As described in an earlier section, the MDBS subroutines assume nothing about the format nor datum size of the user's data stored in the leaves of the database hierarchy. When preparing molecular models for display by the MIDAS display program, however, a specific atom data structure definition must be used. These data definitions can be found in the file */usr/include/midas.h*, and this definition file must be included in all programs which are preparing

data for display by MIDAS. For typical atom coordinate data, the structure definition is as follows:

```
struct atom_def {
        float   x,y,z;          /* atom coordinates */
        float   tempfac;        /* temperature factor */
        short   status;         /* atom status (e.g. visible) */
        char    color,scolor,lcolor;    /* bond, surface & label color */
        char    pad1;           /* padding for longword alignment */
        short   pad2;
}
```

Status is a bit array of atom status information. Most of the atom status information is managed internally by the MIDAS display program and need not be referenced during database operations outside the scope of the MIDAS. There are two exceptions, however. First, the EXISTBIT indicates that data actually exists for the given atom (i.e. that the coordinate data is valid), in contrast to the data simple being a "place holder" in the database. Second, the STARTBIT indicates that the given atom starts a new chain (i.e. the bit is set for the first atom of the first residue in a residue sequence). These status bits are defined in */usr/include/midas.h* and may be set by:

```
buffer.status |= STARTBIT;
buffer.status |= EXISTBIT;
```

before calling *mwrta*.

Separate color information for atom bonds, surfaces and labels may optionally be specified by the user when a database is first created. Color value definitions are also defined in */usr/include/midas.h*.

Several years of experience has shown that MDBS subroutines are easy to use and provide a "natural" way of referencing macromolecular models, particularly proteins and amino acids. Nevertheless, new users of the package often experience some difficulty when first trying to use the MDBS routines. Common errors include:

(1) If the database has not been opened in "slow mode" (as is the usual case), then data is not actually read in until there has been a call to *mreada*. Thus, calls to *mgeta* without a

call to *mreada* will return data which corresponds to the previously read residue (if any exists).

(2) If the database has not been opened in "slow mode", then data is not marked for output until a call to *mwrta* has been made.

(3) When traversing a residue using *mtrav*, an atom may be visited twice if it is part of a ring. The *firstime* variable indicates whether the visit is actually the very first one.

(4) In order to traverse residues using *mtrav*, all atoms must be present, whether there is valid data associated with them or not. This means the user may have to keep a status word along with the data in order to differentiate between real data and space-fillers (this is the function of the EXISTBIT noted above).

(5) When a database is opened for both reading and writing, the updates made to that database are not automatically saved on closing. A call must be made to *msave* in order to save the changes.

(6) *Mclose* must be called in order for temporary files to deleted. Programs that abort abnormally without calling *mclose* will leave temporary files on disk (this causes no serious consequences).

Appendix 3-1: Data Structure Definitions

The MIDAS database system is organized conceptually as a hierarchical database. Each structure model consists of a linear sequence of residues; each residue, in turn, consists of a connected graph of atoms. Each atom forms the leaf of the hierarchy and has data associated with it. Usually this data consists of atomic coordinates, but molecular surfaces or arbitrary user data may be stored as well.

The physical database for each model is arranged into three disk files: the *template* file (with suffix *.tpl*), the *index* file (suffix *.ndx*), and the *data* file (suffix *.dat*). An optional fourth file contains solvent accessible surface data (suffix *.srf*). Each file contains a leading header of the following format:

```
struct header_def {
        short   magic;
        short   rescnt;
        short   tplcnt;
        long_t  filesize;
};
```

The magic number field of this header structure is different for each of the files; this serves as both a file type identifier and version specifier. For any specific molecule, the other three fields are always the same in each file. *Tplcnt* is the number of templates used in the model and *rescnt* is the number of residues in the model. (This latter field is not actually used for template files because they can be shared by several databases.) *Filesize* is the size of the data file in bytes. None of these fields are critical for accessing data in the file, but are used for consistency checks.

The *template* file supplies the connectivity for each type of residue used in the model. Each template entry consists of a header and an atom list. The header has the following format:

```
struct tpl_def {
        short   natom;
        char    restype[RES_TYPE_SIZE];
        char    chief;
        char    linkage;
};
```

where *natom* is the number of atoms in the residue and *restype* is the ASCII name of the residue type (6 characters maximum). *Chief* and *linkage* are indices of the first atom and "linkage" atom of the residue. The atom list consists of atom entries sorted in lexicographical order by atom name. Each atom entry has the format:

```
struct connec_def {
        char    cstat;
        char    tocnt;
        char    to[MAXTO];
        char    atname[AT_NAME_SIZE];
};
```

where *cstat* contains flags to indicate whether this atom is part of the molecule's "mainchain" backbone or whether the atom is part of a closed ring structure and hence cannot be part of a bond rotation. *Tocnt* is the number of atoms which are connected to the current atom (6 maximum) and *to* is an array of indices to these connected atoms. *Atname* is the ASCII name of the atom (up to 6 characters). The entire list of atoms makes up an unidirectional graph which when traversed in a depth-first search will provide an optimum graphical traversal; i.e., if the traversal were followed with a pen, the resulting trace would be a minimal traversal which still maintains proper connectivity.

The *index* file specifies how residues are connected within a complete molecule. In addition to the standard header, there is a second header of the following form:

```
struct reshdr_def {
        short   firstres;
        short   lastres;
        short   atomsize;
};
```

where *firstres* and *lastres* are indices to the first and last residues of the model. *Atomsize* is the size in bytes of the atom entries in the data file (24 bytes for each atom when using the standard MIDAS display format shown below). This second header is followed by the residue entries with the following structure:

```
struct residue_def {
        short   type;
        char    seq[RES_SEQ_SIZE];
        short   atomcnt;
        short   resprev
        short   resnext;
        long_t  offset;
};
```

where *type* is an index to the particular residue structure in the template file for this residue and *seq* is the ASCII sequence name of the residue (6 characters maximum). *Resprev* and *resnext* are the indices to the residues preceding and following the current residue respectively. *Offset* is the address in the data file where the data for the atoms associated with the current residue are stored.

The *data* file consists of a list of atom entries. All atom entries for the same residue must be stored contiguously and be sorted by atom name. The datum size must be the same as that specified in the index file. Note that the number of atoms in the data file need not match the number of atoms in the template file, even if the residue type is the same. The data actually stored in this file is arbitrary. Thus extensions to the standard information usually supplied for molecules such as atom coordinates, temperature factors, etc. can easily be made. The standard information supplied for all Protein Data Bank molecules has the following format:

```
struct  atom_def        {
        float   x, y, z;                /* atom coordinates */
        float   tempfac;                /* temperature factor */
        short   status;                 /* atom status (e.g. "visible") */
        char    color, scolor, lcolor;  /* bond, surface & label color */
        char    pad1;                   /* padding for longword alignment */
        short   pad2;
};
```

The MIDAS database access routines are designed to be data independent; i.e. they assume no particular format for the data stored in the data file. Thus the same utility routines can be used for different types of data and application programs can then be made consistent throughout the system.

Chapter 4

Conclusions

The molecular modeling programs we have developed for studying macromolecular interactions have been of considerable benefit to our scientist user community beginning with our first "in-house" release of the MIDAS software in April 1982. As of this writing nearly 80 publications have resulted from work utilizing MIDAS (see table 4-1).

Beginning in the summer of 1983 we expanded our efforts at dissemination of the MIDAS software by converting the software to run under Digital's "VMS"† operating system in conjunction with an Evans & Sutherland Multi-Picture System (MPS) graphics display located at the European Molecular Biology Laboratory (EMBL) in Heidelberg, Germany. Although this initial work of "porting" the MIDAS system required considerable effort (approximately six man-months of work spread over a two year period), subsequent efforts have involved much less work and have been much more satisfactory. The MIDAS system currently runs on 2 different operating systems (UNIX and VMS) using four different graphics display systems (PS2, MPS, Silicon Graphics IRIS, and the Evans & Sutherland PS300). As of this writing MIDAS has been licensed for use at over 40 sites, with several other sites who have license approval pending (see table 4-2).

Taken together, the large number of scientific publications resulting from the use of MIDAS coupled with the significant number of sites other than UCSF which are licensed to run MIDAS serve as evidence of the overwhelming success of the MIDAS project, not just in terms of the successful implementation of the software, but also in terms of the acceptance MIDAS has gained within the scientist user community.

---

† VMS is a trademark of Digital Equipment Corp.

Future Work

MIDAS was originally designed for studying the interactions of macromolecules whose three dimensional conformation was well known. Several years of scientific advances in the fields of protein engineering, protein structure prediction, molecular dynamics, human interface design, relational database systems, and computer architecture have now intervened since the basic ideas and design choices were first made for MIDAS. Although MIDAS still performs adequately for today's molecular interaction studies, the new demands on the system made by recent scientific advances mean that we are rapidly approaching the limits of the present software. In order to meet tomorrow's needs, new software must be designed today. Already we are involved in discussions with our users so that we can be fully aware of what their anticipated needs will be over the next few years. We have been carefully tracking the continual advances in computer science and computer architecture. We are convinced that significant improvements cannot be made within the framework of the present MIDAS system and that the best approach to take at this time is to design a completely new interactive modeling system. Of course this does not mean that we will ignore of the ideas that have made the present system a success, but rather that we will take advantage of the related scientific advancements that have been made in recent years. For example, the hierarchical database model we based our work on was well understood when we did our original system design, in contrast to the then relatively new relational database model. In recent years, however, much work has been done on relational database systems, including relational systems specifically designed for molecular modeling applications, [MORF80] so that a new system design done today should take advantage of the flexibility afforded by a relational database model. Similarly, during the time of our original design, protein structure engineering frequently took the form of making only relatively simple changes to known structures, for example adjusting bond and torsional angles or substituting single amino acid residues or nucleotides. Advances in genetic engineering and protein folding now mean that scientists must rationally deal with much larger structural units, including such secondary structure components as complete alpha

helices and beta sheets.

Our new system will address these and other issues in as flexible, yet efficient, way as possible, given the constraints of present and near term future computing and graphics display environments. Using a rational design approach, there is good reason to expect that our next generation of molecular modeling software will be just as successful as the MIDAS system is today.

Table 4-1: Publications Derived from Research Work Utilizing MIDAS

(1) I.D. Kuntz, J. Blaney, S. Oatley, R. Langridge and T.E. Ferrin, "A Geometric Approach to Macromolecule-Ligand Interactions," *J. Mol. Biol.* **161** 269-288 (1982).

(2) R.N. Smith, C. Hansch, K.H. Kim, B. Omiya, G. Fukumura, C.D. Selassie, P.Y.C. Jow, J.M. Blaney and R. Langridge, "The Use of Crystallography, Graphics and Quantitative Structure-Activity Relationships in the Analysis of the Papain Hydrolysis of X-Phenyl Hippurates,q *Arch. Biochem. Biophys.* **215** 319-328 (1982).

(3) J.M. Blaney, E.C. Jorgensen, M.L. Connolly, T.E. Ferrin, R. Langridge, S.J. Oatley, J.M. Burridge and C.C.F. Blake, "Computer Graphics in Drug Design: Molecular Modeling of Thyroid Hormone-Prealbumin Interactions" *J. Med. Chem.* **25**, 785-790 (1982).

(4) C. Hansch, R. Li, J.M. Blaney and R. Langridge, "Comparison of the Inhibition of *Escherichia coli* and *Lactobacillus casei* Dihydrofolate Reductase by 2,4-Diamino-5-(Substituted-benzyl) pyrimidines: Quantitative Structure-Activity Relationships, X-ray Crystallography, and Computer Graphics in Structure-Activity Analysis," *J. Med. Chem.* **25** 777-784 (1982).

(5) R. Li, C. Hansch, D. Matthews, J.M. Blaney, R. Langridge, T.J. Delcamp, S.S. Susten and J.H. Freisheim, "A Comparison by QSAR, Crystallography, and Computer Graphics of the Inhibition of Various Dihydrofolate Reductases by 5-(X-Benzyl)-2,4-diamino-pyrimidines," *Quant. Struct.-Act. Relat.* **1**, 1-7 (1982).

(6) S. Sprang, R. Fletterick, M. Stern, D. Yang, N. Madsen, and J. Sturtevant, "Analysis of an Allosteric Binding Site: The Nucleoside Inhibitor Site of Phosphorylase a, *Biochem.* **21T**, 2036-2048 (1982).

(7) S.G. Withers, N.B. Madsen, R.J. Fletterick and S.R. Sprang, "The Catalytic Site of Phosphorylase: Structural Changes During Activation and Mechanistic Implications," *Biochemistry* **21**, 5372-5382 (1982).

(8) S.R. Sprang, E.J. Goldsmith and R.J. Fletterick, "The Catalytic Site of Glycogen Phosphorylase: Structure of the T-State and Specificity for a,D-Glucose" *Biochemistry* **21**, 5364-5371 (1982).

(9) R.J. Fletterick and S.R. Sprang, "Glycogen Phosphorylase Structure and Function," *Acc. Chem. Res.* **15**, 361-369 (1982).

(10) T. Minaga, J. McLick, N. Pattabiraman and E. Kun, "Steric Inhibition of Phenylboronate Complex Formation of 2′-5′-Phosphoribosyl)-5′-AMP," *J. Biol. Chem.* **257**, 11942-11945 (1982).

(11) R. Somack, T.A. Andrea and E.C. Jorgensen, "Thyroid Hormone Binding to Human Serum Prealbumin and Rat Liver Nuclear Receptor: Kinetics, Contribution of the Hormone Phenolic Hydroxyl Group, and Accommodation of Hormone Side-Chain Bulk," *Biochem.* **21**, 163-170 (1982).

(12) C.S. Craik, S. Sprang, R. Fletterick and W.J. Rutter, "Intron-exon splice junctions map at protein surfaces," *Nature* **299**, 180-182 (1982).

(13) R. Tilton and I.D. Kuntz, "Nuclear Magnetic Resonance Studies of ^{129}Xe with Myoglobin and Hemoglobin," *Biochemistry* **21**, 6850-6857 (1982).

(14) R. Fletterick and E. Goldsmith, "Oligosaccharide Conformation and Protein Saccharide Interactions in Solution," *Pure & Appl. Chem.* **55**, 577-588 (1983).

(15) J.A. Tainer, E.D. Getzoff, J.S. Richardson and D.C. Richardson, "Structure and Mechanism of copper, zinc superoxide dismutase", Nature **306**, 284-287 (1983).

(16) E.D. Getzoff, J.A. Tainer, P.K. Weiner, P.A. Kollman, J.S. Richardson and D.C. Richardson, "Electrostatic Recognition between superoxide and copper, zinc superoxide dismutase," Nature **306**, 287-290 (1983).

(17) P.A. Bash, N. Pattabiraman, C. Huang, T.E. Ferrin and R. Langridge, "Van der Waals Surfaces in Molecular Modelling: An Implementation Using Real-Time Computer Graphics" Science, **222**, 1325-1327 (1983).

(18) C. Hansch, B.A. Hathaway, Z. Guo, S.W. Dietrich, J.M. Blaney, R. Langridge, K.W. Volz and B.T. Kaufmann, "Crystallography, Quantitative Structure-Activity Relationships and Molecular Graphics in a Comparative Analysis of the Inhibition of Dihydrofolate Reductase from Chicken Liver and *Lactobacillus casei* by 4,6-Diamino-1,2-dihydro-2,2-dimethyl-1-(substituted- phenyl0-s-triazines. *J. Med. Chem.* **27**, 129-143 (1984).

(19) E. Canova-Davis and L. Waskell, "The Identification of the Heat-stable Microsomal Protein Required for Methoxyflurane Metabolism as Cytochrome b_5, *J. Biol. Chem.*, **259** 2541-2546 (1984).

(20) Jones, Katherine A. and Tjian, Robert, "Essential contact residues within SB40 large T antigen binding sites I and II identified by alkylation-interference, Cell, 1984 (in press). *J. Muscle Res. & Cell Motility* **4**, 671 (1983).

(21) K.R. Ely, J.N. Herron and A.B. Edmundson, "Three-Dimensional Structure of the orthorhombic form of the Mcg Bence-Jones dimer," *Prog. Immuno.* V, 61-66 (1983).

(22) S. Ramaprasad, R.D. Johnson and G.N. LaMar, "Paramagnetic Relaxation Determination of Peak Assignment and the Orientation of Ile-99 FG5 in Metcyanomyoglobin," *JACS* **106**, 5330 (1984).

(23) A.B. Edmundson, K.R. Ely and J.N. Herron, "A search for site-filling ligands in the Mcg Bence-Jones dimer: crystal binding studies of fluorescent compounds" *Mol. Immunology* **21**, 561-576 (1984).

(24) D. Gidoni, W.S. Dynan and R. Tjian, "Multiple specific contacts between a mammalian transcription factor and its cognate promoters" *Nature* **312**, 409 (1984).

(25) A.D. Sherry, J. Keepers, T.L. James and J. Teherani, "Methyl Motions in ^{13}C-Methylated Concanavalin as Studied by ^{13}C Magnetic Resonance Relaxation Techniques," *Biochemistry* **23**, 3181 (1984).

(26) R.F. Tilton, Jr., I.D. Kuntz, Jr., and G.A. Petsko, "Cavities in Proteins: Structure of a Metmyoglobin-Xenon Complex Solved to 1.9 " *Biochemistry* **23**, 2849 (1984).

(27) P.G. Schmidt and I.D. Kuntz, "Distance Measurements in Spin-Labeled Lysozyme," *Biochemistry* **23**, 4261 (1984).

(28) P. Rosevear, S. Sellin, B. Mannervik, I.D. Kuntz, and A.S. Mildvan, "NMR and Computer Modeling Studies of Glutathione Derivatives at the Active Site of Glyoxalase I," *J. Biol. Chem.* **259**, 1436 (1984).

(29) C. Hansch and J.M. Blaney in "Drug Design: Fact or Fantasy?" G. Jolles and K.R.H. Woodridge, eds. Academic Press, p 185 (1984).

(30) A. Carotti, C. Hansch, M.M. Mueller and J.M. Blaney, "Actinidin Hydrolysis of Substituted-Phenyl Hippurates: A Quantitative Structure-Activity Relationship and Graphics with Hydrolysis by Papain," *J. Med. Chem.* **27**, 1401 (1984).

(31) J.M. Blaney, C. Hansch, C. Silipo and A. Vittoria, "Structure-Activity Relationships of Dihydrofolate Reductase Inhibitors," *Chem. Rev.* **84**, 333 (1984).

(32) H.K. Schachman, C.D. Pauza, M. Navre, M.J. Karels, L. Wu and Y.R. Yang, "Location of amino acid alterations in mutants of aspartate transcarbamoylase: Structural aspects of interallelic complementation," *PNAS, USA* **82**, 115 (1984).

(33) P.V. Hornbeck and A.C. Wilson, "Local Effects of Amino Acid Substitutions on the Active Site Region of Lysozyme: A Comparison of Physical and Immunological Results," *Biochemistry* **23**, 998 (1984).

(34) N. Pattabiraman, R. Langridge and P.A. Kollman, "An Iterative Approach to Placing Counterions around DNA," *J. Biomolec. Struct. & Dynam.* **1**, 1525 (1984).

(35) V.S. Anathanarayanan, S.K. Brahmachari and N. Pattabiraman, "Proline-containing Beta turns in Peptides and Proteins: Analysis of Structural Data on Globular Proteins," *Arch. Biochem. Biophys.* **232**, 482 (1984).

(36) D.K. Srivastava, S.A. Bernhard, R. Langridge and J.A. McClarin, "Molecular Basis for the Transfer of Nicotinamide Adenine Dinucleotide among Dehydrogenases," *Biochemistry* **24**, 629-635 (1985).

(37) A.B. Edmundson and K.R. Ely, "The Mcg light chain: multiple conformations derived from a single amino acid sequence." In *Ann. Inst. Pasteur/Immunol.*, 136C, F.A. Saul and R.J. Poljak, eds., Paris, pp.259-294 (1985).

(38) A.B. Edmundson, K.R. Ely, J.N. Herron and A.L. Gibson, "Probing the binding sites in crystals of immunoglobulins." In *Investigation and Exploitation of Antibody Combining Sites*, E. Reid, G.M.W. Cook, and D.J. Morre, Eds., Plenum Publishing Corp., New York, pp. 33-50 (1985).

(39) A.B. Edmundson and K.R. Ely, "Binding of N-formylated chemotactic peptides in crystals of the Mcg light chain dimer: similarities with neutrophil receptors," *Mol. Immun.* **22**, 463-475 (1985).

(40) G.L.Seibel, U.C.Singh and P.A.Kollman: "A molecular dynamics simulation of double-helical B-DNA including counterions and water". *Biophysics* **82**, 6537-6540, (1985).

(41) U. Christensen, S. Ishida, S-i Ishii, Y. Mitsui, Y. Iitaka, J. McClarin, and R. Langridge, "Interactions of *Streptomyces* Subtilisin Inhibitor with *Streptomyces griseus* Proteases A and B. Enzyme Kinetic and Computer Simulation Studies", *J. Biochem.* **98**, 1263-1274 (1985).

(42) T. P. Lybrand and P.A.Kollman: "Molecular Mechanical Calculations on the Interaction of Ethidium Cation with Double-Helical DNA". *Biopolymers*, **24**, 1863-1879 (1985).

(43) S.N. Rao and P.A. Kollman; "On the Role of Uniform and Mixed Sugar Puckers in DNA Double-Helical Structures" *J. Am. Chem. Soc.* **107**, 6 (1985)

(44) P. A. Kollman: "Theoretical Chemistry Applied to Complex Molecular Interactions: Computer Graphics, Distance Geometry, Molecular Mechanics and Quantum Mechanics." *Acc. Chem. Res.*, **18**, 105-111 (1985)

(45) A. Carotti, C. Hansch, M.M. Mueller, and J.M. Blaney: Hydrolysis of Substituted Phenyl Hippurates: A Quantitative Structure-Activity Relationship and Graphics Comparison with Hydrolysis by Papain. *J. Med. Chem.* **28**, 261 (1985).

(46) T. Klein, D. Kneller, C. Huang, T. Ferrin, and R. Langridge. "Computer Graphics and Artificial Intelligence in Drug Design and Protein Engineering". *Journal of Molecular Graphics* **3**, 111-113 (1985).

(47) C. Hansch, J. McClarin, T. Klein and R. Langridge: "A Quantitative Structure-Activity Relationship and Molecular Graphics Study of Carbonic Anhydrase Inhibitors". *Mol. Pharm.* **27**, 493-498 (1985).

(48) T. Klein, D. Kneller, C. Huang, T. Ferrin and R. Langridge: "Computer Graphics and Artificial Intelligence in Drug Design and Protein Engineering" *J. Mol. Graphics* **3**, 111-113 (1985).

(49) M. S. Broido, T.L.James, G. Zon and J.W.Keepers: "Investigation of the Solution Structure of a DNA Octamer [d-(GCAATTCCg]2 Using Two-Dimensional Nuclear Overhauser Effect Spectroscopy." *Eur. J. Biochem.*, **150**, 117-128 (1985).

(50) N.R.Shine and T.L.James: "Interactions of Diastereometric Tripeptides of Lysyl-5-Fluro-Tryptophyl-Lysine with DNA. I. Optical and 19F NMR Studies of the Native DNA Complexes." *Biochem.* **24**, 4333-4341 (1985).

(51) N.R.Shine, E. Bubienko and T.L.James: "Interactions of Diastereometric Tripeptides of Lysyl-5-Fluoro-Tryptophyl-Lysine with DNA. II. Optical, 19F NMR and Strand Cleavage Studies of the Apurinic DNA Complexes." *Biochem.* **24**, 4341-4345 (1985).

(52) E.Berman, S.C.Brown, T.L.James and R.H.Shafer: "NMR Studies of Chromomycin A3 Interaction with DNA". *Biochem.* (1985).

(53) T.L.James, G.B.Young, M.S.Broido, J.W.Keepers, N.Jamin and G. Zon: "Quantitative Internuclear Distances via 2D NMR Spectra: A Test Case and a DNA Octamer Duplex. In *Biomolecular Structure and Interactions*. V. Sasisekharan and K.R.K. Easwaran, eds. (1985).

(54) P.H. Howard, J. Shaw and A.J. Otsuka: Nucleotide sequence of the gene encoding the biotin operon repressor and biotin holoenzyme synthetase functions of *Gene* **35**, 321-331 (1985).

(55) G.M. Ehrenfeld, L.O. Rodrigues, S. M. Hecht, C. Chang, V.J. Basus and N.J. Oppenheimer, "Copper(Ig-Bleomycin: Structurally Unique Complex that Mediates Oxidative DNA Strand Scission," *Biochemistry* **24**, 81-92 (1985).

(56) S. Manogaran, P.G. Schmidt, N.J. Oppenheimer and I.D. Kuntz, "Two-dimensional ^{1}NMR of Three Spin-labeled Derivatives of Bovine Pancreatic Trypsin Inhibitor" *Biochemistry* (1985).

(57) N. Jamin, T.L. James, and G. Zon: Two-dimensional nuclear Overhauser enhancement investigation of the solution structure and dynamics of the DNA octamer $[d(GGTATACC)]_2$. *Eur. J. Biochem.* **152**, 157-166 (1985).

(58) N. Pattabiraman and R. Langridge, "Flexibility of 3',5' deoxyribonucleoside diphosphates", *Journal of Biomolecular Structure and Dynamics* **2**, 683 - 692 (1985).

(59) N. Pattabiraman, M. Levitt, T. E. Ferrin and R. Langridge, "Molecular Modeling and Drug design: Real-time Energy Calculation and Minimization" *J. Comp. Chem* **6**, 432-436 (1985).

(60) P. R. Ortiz de Montellano and C. E. Catalano: "Epoxidation of Styrene by Hemoglobin and Myoglobin" *J. Biol. Chem.* **16**, 9265-9271 (1985).

(61) T. Klein, C. Huang, T. Ferrin, R. Langridge, and C. Hansch. "Computer Assisted Drug Receptor Mapping Analysis" in *Artificial Intelligence in Chemistry*. T.H. Pierce and B.A. Hohne, eds. ACS Symposium Series #306. 147-158 (1986).

(62) C. Hansch, T. Klein, J. McClarin, R. Langridge and N.W. Cornell: "A QSAR and Molecular Graphics Analysis of Hydrophobic Effects in the Interactions of Inhibitors with Alcohol Dehydrogenase". J. Med. Chem. **29**, 615-620 (1986).

(63) C. D. Selassie, Z. Fang, R. Li, C. Hansch, T. Klein, R. Langridge, and B. Kaufman. "Inhibition of Chicken Liver Dihydrofolate Reductase by 5-(substituted benzyl)-2,4-diamino pyrimidines. A QSAR and Graphics Analysis. J. Med. Chem. **29**, 621-626

(1986).

(64) J. Caldwell and P. A. Kollman: "A Molecular Mechanical Study of Netropsin-DNA Interactions" *Biopolymers* **25**, 249-266 (1986).

(65) S. Weiner, G. Seibel and P. A. Kollman: "The Nature of Enzyme Catalysis in Trypsin". *Proc. Nat. Acad. Sci. USA*, **88**, 649-653 (1986).

(66) S. R. Rao and P. A. Kollman: "Hydrogen Bonding Preference in 2,6 Diamino Purine-uracil (Thymine) and 8-Methyl Adenine-uracil (Thymine) Complexes". *Biopolymers* **25**, 267-280 (1986).

(67) M. Recanatini, T. Klein, C. Yang, J. McClarin, R. Langridge, and C. Hansch. "Quantitative Structure-Activity Relationships and Molecular Graphics in Ligand Receptor Interactions. Amidine Inhibition of Trypsin". *Molecular Pharmacology* **29** 436-446 (1986).

(68) W.A. Remers, S.N. Rao, U.C. Singh, and P.A. Kollman: "Conformations of Complexes between Mitomycin and Nucleotides. 2. Application of the Model to Mitomycin C Derivatives. Extension to Covalent Binding with Adenine". *J. Med. Chem.* **29**, 1256-1263 (1986).

(69) U. C. Singh, N. Pattabiraman, R. Langridge and P. A. Kollman, "Molecular Mechanical Studies of d(CGTACG)$_2$ and its complex with Triostin A with the middle AT base pairs in either Hoogsteen and or Watson-Crick Pairing" *Proc. Natl. Acad. Sci.*, USA (1986)

(70) A.B. Edmundson and K.R. Ely, "Determination of the three-dimensional structures of immunoglobulins" Handbook of Experimental Immunology, 4th edition, Blackwell Scientific Publications, Inc. Edinburgh, Scotland, in press (1986).

(71) **C. Hansch, "The use of QSAR and Molecular Graphics in the Study of Enzyme-ligand Interactions". *Chemistry* (Japan) (in press).

(72) **C. Hansch and T. Klein, "Molecular Graphics and QSAR in the Study of Enzyme-Ligand Interactions. On the Definition of Bioreceptors". *Accts. of Chem. Res.* (in press).

(73) **N. Pattabiraman, S.N. Rao, K. Scott, R. Langridge and P.A. Kollman, "Molecular Mechanical Simulations of Left- and Right-handed B-DNA", *Biopolymers* (in press).

(74) **R. S. Bhatnagar, K. R. Sorensen, N. Pattabiraman, R. Langridge, R. D. MacElroy, V. Renugopalakrishnan and S-G. Huang "Does Proline Contribute to the Stability of the Triple Helix?" *Science* (submitted).

(75) **B. G. Feuerstein, N. Pattabiraman and L.J.Marton.: "Spermine-DNA interactions: a theoretical study". *Proc. Natl. Acad. Sci., USA* (submitted).

(76) **L. Morgenstern, M. Recanatini, T. Klein, W. Steinmertz, C.Z. Yang, R. Langridge and C. Hansch, "Chymotrypsin Hydrolysis of X-Phenyl Hippurates. A QSAR and Molecular Graphics Analysis." *JACS* (submitted).

(77) **N. Pattabiraman, L. A. LaPlanche and R. Langridge, "Three-Dimensional Isopotential Energy Surfaces: Allowed Conformations for Protonated and Unprotonated Lidocaine" *Molecular Pharmacology* (submitted).

(78) **P. C. Babbitt, G. L. Kenyon, I. D. Kuntz, F. E. Cohen, J. D. Baxter, P. A. Benfield, J. D. Buskin, W. Gilbert, S. D. Hauschka, H. P. Hossle, C. P. Ordahl, M. L. Pearson, J. C. Perriard, L. Pickering, S. Putney, B. L. West and R. A. Ziven; "Comparisons of nine creatine kinase primary structures: implications for structure-activity relationships". *J. Biol. Chem.* (submitted).

(79) **T. Lybrand, S. Brown, R. Shafer and P. A. Kollman: "Computer Modeling of Actinomycin D Interactions with Double Helical DNA". *J. Molec. Biol.* (submitted).

Table 4-2: Sites Currently Licensed To Use MIDAS

(1) Cancer Research Laboratory of the Abbott Northwestern Hospital
(2) Columbia University
(3) European Molecular Biology Laboratory (EMBL)
(4) Institute of Cancer Research
(5) Los Alamos National Laboratory, GENBANK Group
(6) Los Alamos National Laboratory, Life Sciences Division
(7) Max Planck Institute
(8) National Cancer Institute
(9) National Institute of Environmental Health and Safety (NIEHS)
(10) Purdue University, Chemistry Dept.
(11) Research Institute for Advanced Computer Science (RIACS)
(12) Rutgers University
(13) The Salk Institute
(14) Smith-Kline-French Laboratories
(15) Stanford University, Computer Science Dept.
(16) Stanford University, Physics Dept.
(17) State University of New York at Syracuse
(18) Syracuse University
(19) University of Bordeaux I, France
(20) University Colorado
(21) University Illinois
(22) University Utah, Bioengineering Dept.
(23) University of Alabama
(24) University of Alberta
(25) University of Arizona, Chemistry Dept.
(26) University of Arizona, School of Pharmacy
(27) University of California, San Diego
(28) University of California, Santa Cruz
(29) University of Houston
(30) University of Michigan, Biophysics Dept.
(31) University of Michigan, School of Pharmacy
(32) University of Minnesota
(33) University of Naples
(34) University of North Carolina
(35) University of Saskatchewan
(36) University of Tromso, Norway

(37) University of Utah, Chemistry Dept.
(38) University of Virginia
(39) University of Washington
(40) University of Wisconsin
(41) Wright State University
(42) **California Institute of Technology

(** indicates license agreement pending.)

REFERENCES

AHO79 *Principles of Compiler Design*, A.V. Aho and J.D. Ullman, (Addison-Wesly Publishing Co., 1979).

BASH83 P.A. Bash *et al*, "Van der Waals Surfaces in Molecular Modeling: Implementation with Real-Time Computer Graphics", *Science* **222**, 1325 (1983).

BERN77 F.C. Bernstein, "The Protein Data Bank: a Computer Archive", *J. Mol. Biol.* **112**, 535-542 (1977).

BOND72 P.J. Bond, "Interactive Computer Graphics and Macromolecular Structures", *Computer Graphics* **6**, 13-26 (1972).

CONN81 M.L. Connolly, "Protein Surfaces and Interiors", Ph.D. Dissertation, (University of California, 1981).

CONN83 M.L. Connolly, "Solvent-Accessible Surfaces of Proteins and Nucleic Acids", *Science* **221**, 709 (1983).

FELD73 R.J. Feldmann, C.R.T. Bacon and J.S. Cohen, "Versatile Interactive Graphics Display System for Molecular Modelling by Computer", *Nature* **244**, 113-115 (1973).

FELD76 R.J. Feldmann, "The Design of Computing Systems for Molecular Modeling", *Annual Review of Biophys & Bioeng* **5**, 477-510 (1976).

FERR77 T. Ferrin *et al*, "MMS User Instruction Manual", (Computer Graphics Laboratory, University of California, San Francisco, September 1977).

FERR80 T.E. Ferrin and R. Langridge, "Interactive Computer Graphics with the UNIX Time-Sharing System", *Computer Graphics* **13**, 320-331 (1980).

FERR84 T.E. Ferrin *et al*, "Molecular Interactive Display and Simulation: MIDAS", *J. Mol. Gr.* **2**, p 55, (1984).

FERR86a T. Ferrin *et al*, "The MIDAS Database System", (submitted).

FERR86b T. Ferrin *et al*, "Molecular Interactive Display and Simulation (MIDAS)", (submitted).

FLET86 "Computer Graphics and Molecular Modeling", edited by R. Fletterick and M. Zoller, *Current Communications in Molecular Biology*, (Cold Spring Harbor Laboratory, 1986).

FREE77 "Tutorial on Software Design Techniques", edited by P. Freeman and A.I. Wasserman, (IEEE Computer Society, Long Beach, CA, 1977).

JARV85 L. Jarvis *et al*, "UCSF MIDAS User's Manual", (Computer Graphics Laboratory, University of California, San Francisco, November 1985).

JOHN65 C.K. Johnson, "ORTEP", Technical Report #3794, (Oak Ridge National Laboratory, 1965).

JOHN75 S.C. Johnson, "YACC - Yet Another Compiler Compiler", Computing Science Technical Report #32, (Bell Laboratories, Murray Hill, NJ, 1975).

KERN78 *The C Programming Language*, B.W. Kernighan and D.M. Ritchie, (Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1978).

KNUT72 *The Art of Computer Programming*, Vol. 3, "Searching and Sorting", D. Knuth, (Addison-Wesley Publishing Co., Reading, Mass, 1972).

LANG65 R. Langridge and A.W. MacEwan, in *Proceedings, IBM Scientific Computing Symposium on Computer Aided Experimentation* (IBM, Yorktown Heights, N.Y., 1965), p.305.

LANG75 R. Langridge, "Interactive Three-Dimensional Computer Graphics in Molecular Biology", *Computers in Life Science Research*, edited by William Siler and Donald A.B. Lindberg, pp 53-59 (1975).

LANG81 R. Langridge *et al*, "Real-Time Color Graphics in Studies of Molecular Interactions", *Science* **211**, 661-666 (1981).

LANG84 R. Langridge and T.E. Ferrin, "The Future of Molecular Graphics", *J. Mol. Gr.* **2**, p 56, (1984).

LEE77 B. Lee and F.M. Richards, *J. Mol. Biol.* **55**, 379-400 (1971).

LESK77 A.M. Lesk, "Macromolecular Marionettes", *Comput. Biol. Med.* **7**, 113-129 (1977).

LEVI65 C. Levinthal, in *Proceedings, IBM Scientific Computing Symposium on Computer Aided Experimentation* (IBM, Yorktown Heights, N.Y., 1965), p. 315

LEVI66 C. Levinthal, "Molecular Model-Building by Computer", *Sci. Am.* **214**, 42-52 (June 1966).

MORF80 A.J. Morffew, S.J.P. Todd and M.J. Snelgrove, "The Use of a Relational Data Base for Holding Molecule Data in a Molecular Graphics System", *Computers and Chemistry*, **4**, pg 149 (1980).

PATT84 N. Pattabiraman *et al*, "Real Time Energy Calculation and Minimization in Interactive Three Dimensional Computer Graphics", *J. Mol. Gr.* **2**, p 59, (1984).

RICH77 F.M. Richards, *Annu. Rev. Biophys Bioeng.* **6**, 151-176 (1977).

RITC74 D.M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", *Commun. ACM* **17**, 7, pp 365-375, (1974).

SUTH70 I.E. Sutherland, "Computer Displays", *Scientific American* **222**, 56-81 (1970).

ULLM80 *Principles of Database Systems*, J.D. Ullman, (Computer Science Press, Potomac, MD 20854, 1980).

APPENDIX

Program Listings

1. MDBS (MIDAS Database Subroutines)
2. EDITOR (MIDAS Database Editor Module)
3. INTR (MIDAS Interactive Module)

/usr/src/local/midas/h/mdbs.h

```
/* $Header: mdbs.h,v 3.14 86/10/29 10:56:15 pett Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          26 May 1983
 * Release 2.2                          26 Jul 1983
 * Release 2.3                          26 Aug 1983
 */

/*-----------------------------------------------------------------*
 *      Header File  of defines and structures for
 *      Molecular Interactive Display And Simulation
 *-----------------------------------------------------------------*/

#ifdef m68000
#define  VAXLONGS       1
#endif
#ifdef vax
#define  VAXLONGS       1
#endif

#ifndef  vms
#include "grafeng.h"
#include "pathnames.h"
#include <sys/types.h>
#else
#include "midas$h:machdef.h"
#include "midas$h:pathnames.h"
#include <types.h>
typedef  int      ino_t;
typedef  int      dev_t;
#ifndef  void
#define  void     int
#endif
#define  exit     exit_     /* Indirect exit() routine for VMS exit codes */
#endif   vms

/*----------BASIC DEFINITIONS-----------------------------------------*/

#define  MAXATOM 100             /* Maximum number of atoms per residue */
#define  MAXRES 500              /* Maximum number of residues per model */
#define  MAXTPL 20               /* Maximum number of templates per model */
#define  MAXSTREAM    12         /* Maximum number of MIDAS stream open */

#ifndef  vms
#define  MBASELEN     24         /* Length of base name + suffix */
#else    vms
#define  MBASELEN     30         /* Length of base name + suffix */
#define  ALWAYSCOPY               /* No file links in vms */
#define  unlink(a)    delete(a)          /* Same semantics, different spelling */
#endif   vms

#undef   TRUE
#define  TRUE         1
#define  FALSE        0
#define  NIL          0
#define  MAINCHAIN    01         /* Template node type */

/*
 * Changing any of the following definitions will result in the
 * invalidation of the entire data base.  Act accordingly.
 */
#define  AT_NAME_SIZE    4       /* Number of characters in atom names */
#define  RES_TYPE_SIZE   6       /* Number of characters in residue names */
#define  RES_SEQ_SIZE    6       /* Number of characters in residue seq */
#define  MAXTO           6       /* Number of connected atoms */

/*
 * Magic number definition
 */
#define  NDX_MAGIC       (short) 0167351   /* Index file 'in' */
#define  DAT_MAGIC       (short) 0162341   /* Data file 'da' */
#define  TPL_MAGIC       (short) 0172360   /* Template file 'tp' */

#ifdef m68000
typedef long       long_t;

# define           GETLONG(l, r)     (l = (long) r)
# define           STORELONG(l, r)   (l = (long_t) r)
# else
/*
 * Structure of longs stored on disk.  This structure is
 * only needed because padding of structures which contain
 * a member of type 'long' is unpredictable.
 */
struct   long_t_def        {
         short    l_hi;
         short    l_lo;
};
typedef  struct   long_t_def        long_t;
```

/usr/src/local/midas/h/mdbs.h

```
/*
 * Structures for maintaining compatibility between the PDP 11 and the VAX
 */
union   long_def        {       /* Longs are stored in VAX format */
        long    l_lword;
#ifdef  VAXLONGS
        long_t  l_sword;
#endif
#ifdef  pdp11
        struct  {
                short   s_lo;
                short   s_hi;
        }       l_sword;
#endif
};

#ifdef  VAXLONGS
#define GETLONG(l,r)    (_longconv.l_sword = (r), (l) = _longconv.l_lword)
#define STORELONG(l,r)  (_longconv.l_lword = (r), (l) = _longconv.l_sword)
#endif
#ifdef  pdp11
#define GETLONG(l,r)    (_longconv.l_sword.s_lo = (r).l_lo,\
                         _longconv.l_sword.s_hi = (r).l_hi,\
                         (l) = _longconv.l_lword)
#define STORELONG(l,r)  (_longconv.l_lword = (r),\
                         (l).l_lo = _longconv.l_sword.s_lo,\
                         (l).l_hi = _longconv.l_sword.s_hi)
#endif
#endif  m68000

/*
 * Residue structure definition
 */
struct  tpl_def         {
        short   natom;                  /* Residue size */
        char    restype[RES_TYPE_SIZE]; /* Residue type */
        char    chief;                  /* Index of first atom */
        char    linkage;                /* Atom linking to next res
};

/*
 * Residue in database definition
 */
struct  residue_def     {
        short   type;                   /* Residue type */
        char    seq[RES_SEQ_SIZE];      /* Residue sequence */
        short   atomcnt;                /* Number of atoms */
        short   resprev;                /* Previous residue */
        short   resnext;                /* Next residue */
        long_t  offset;                 /* File address of data */
};

/*
 * Atom connectivity structure definition
 */
struct  connec_def      {
        char    cstat;                  /* Status bits */
        char    tocnt;                  /* Number of connected atom */
        char    to[MAXTO];              /* Connected atoms indices */
        char    atname[AT_NAME_SIZE];   /* Atom name */
};

/*
 * Index file header definition
 */
struct  header_def      {
        short   magic;                  /* Magic number */
        short   rescnt;                 /* Number of residues */
        short   tplcnt;                 /* Number of templates */
        long_t  filesize;               /* Length of data file */
};

/*
 * Index file second header definition
 */
struct reshdr_def {
        short   firstres;               /* Index of first residue */
        short   lastres;                /* Index of last residue */
        short   atomsize;               /* Size of each atom entry */
};

/*
 * The convention of data is as follows :
 *      All upper case characters = constant
 *      All lower case characters = variable
 *      First character upper case = Macro-defined variable (see below)
 */

#define ACCESS          03
#define MFREE           00              /* Modes of database ... */
#define READ            01              /*      operation        */
#define WRITE           02
#define READWRITE       03
#define DIRECTION       04              /* 0=forward, 1=backward */
```

/usr/src/local/midas/h/mdbs.h

```
#define ADVANCE         030             /* Need to go to next
#define ADV_IF_R        010             /*       residue ?
#define ADV_IF_A        020
#define DAT_IN_CORE     040             /* Current atom data in co
#define MFLUSH          0100            /* Stream need flushing? *
#define TPLCHG          0200            /* Header(s) changed ?
#define SLOWMODE        0400            /* Do I/O by atom ? */

/* Define macro variables so that code will be easier to read */

#define Header      stream->header
#define Reshdr      stream->reshdr
#define Residue     stream->residue
#define Tpl         stream->tpl
#define Connec      stream->connec
#define Fd          stream->fd
#define Inode       stream->inode
#define Dev         stream->device
#define Floc        stream->floc
#define Mode        stream->mode
#define Maxatm      stream->maxatm
#define Curseq      stream->curres
#define Data        stream->data
#define Basename        stream->basename
#define Tmpfile     stream->tmpfile

#define Curres      Residue[Curseq]
#define Curtpl      Tpl[Curres.type]
#define Curcon      Connec[Curres.type]

#define NEXTR(z) ((z) == -1) ? -1 : \
                 ((Mode & DIRECTION) ? Residue[z].resprev : Residue[z].resnext)

/*
 * Database access stream definition
 */
struct  stream_def      {
        struct  header_def      header;         /* Database header */
        struct  reshdr_def      reshdr;         /* Index file second heade
        struct  residue_def     *residue;       /* Index */
        struct  tpl_def         *tpl;           /* Pointer to tpl headers */
        struct  connec_def      **connec;       /* Pointer to tpl data */
        int     fd;                             /* File descriptor of data */
        ino_t   inode;                          /* Inode of data file */
        dev_t   device;                         /* Dev of data file */
        long    floc;                           /* Location in datafile */
        int     mode;                           /* Database mode */
        int     maxatm;                         /* Largest residue in db */
        int     curres;                         /* Index to current residue */
        char    *data;                          /* Pointer to atom data */
        char    *basename;                      /* Basename of database */
        char    *tmpfile;                       /* Tempfile of this stream */
};

typedef struct  stream_def      *MDBS;

char    *mdatptr();
char    *malloc(), *realloc(), *gettemp();
int     getfile();
long    bseek(), btell(), lseek();

extern  struct  stream_def      _mstrm[MAXSTREAM];      /* Streams */
extern  union   long_def        _longconv;              /* Conversion union */
extern  int     err_midas;                              /* Midas error number */
extern  char    *midas_errlist[];                       /* Error messages */

/* Error codes from the database subroutines themselves
 * (as opposed to system call failures) */

#define EM_ILLMODE      (-1)
#define EM_TOOBIG       (-2)
#define EM_NOTEMP       (-3)
#define EM_SINVAL       (-4)
#define EM_NOPERM       (-5)
#define EM_OUTOFRANGE   (-6)
#define EM_TOOMANY      (-7)
#define EM_OUTOFMEMORY  (-8)
#define EM_MISMATCH     (-9)
#define EM_EMPTY        (-10)
#define EM_ILLOP        (-11)
#define EM_ATEOF        (-12)
```

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mopen.c,v 3.5 85/04/15 12:27:19 arnol
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          27 Jul 1983
 */

#include <stdio.h>
#ifndef  vms
#include <mdbs.h>
#include <sys/stat.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Mopen opens the database.  The index and template files are loaded into
 * core if the mode involves reading the database.  The datafile is opened
 * but otherwise untouched.
 */
#ifndef  f77
mopen(fname, type)
char     *fname, *type;
#else
mopen_(fname, type, len)
char     *fname, *type;
long     len;
#endif
{
        register                MDBS    stream;
        int             stno;
        int             mode, onbits;
        int             slowmode;
        int             n;
        int             fd;
        char            buf[512];

        /* Find an unused stream */
        for (stream = &_mstrm[0]; stream < &_mstrm[MAXSTREAM]; stream++)
                if ((Mode & ACCESS) == MFREE)
                        break;
        if (stream >= &_mstrm[MAXSTREAM]) {
                err_midas = EM_TOOMANY;
#ifdef   MDEBUG
                merror("mopen");
#endif
                return(-1);
        }
        stno = stream - &_mstrm[0];

        /* Find out how the database should be opened */
        if (*type == 'r') {
                if (*++type == 'w')
                        mode = READWRITE;
                else {
                        mode = READ;
                        type--;                 /* Look at this character again */
                }
                onbits = mode;
        } else if (*type == 'w') {
                mode = WRITE;
                if (*++type == 'r')
                        onbits = READWRITE;
                else {
                        onbits = WRITE;
                        type--;                 /* Look at this character again */
                }
        } else {
                err_midas = EM_ILLMODE;
#ifdef   MDEBUG
                merror("mopen");
#endif
                return(-1);
        }
        if (*(++type) == 's')
                slowmode = TRUE;
        else
                slowmode = FALSE;

        /* Here's where we do the work */
#ifdef   f77
        termin_(fname, (int) len);
#endif

        switch (mode) {

        case READ :

                /* Load the database and open up the date file */
                if (!mload(stno, fname)) {
#ifdef   MDEBUG
```

```
                        merror("mopen");
#endif
                        return(-1);
                }
                Fd = getfile(fname, DAT_MAGIC, "r");

                /* Restore the file name */
                if (Fd == -1) {
#ifdef  MDEBUG
                        merror("mopen(data file)");
#endif
                        return(-1);
                }
                brewind(Fd);

                /* Allocate space for data */
                if (!slowmode) {
                        Data = (char *) malloc((unsigned)
                                Maxatm*Reshdr.atomsize);
                        if (Data == NIL) {
                                Mode |= READ;
#ifdef  f77
                                mclose_(&stno);
#else
                                mclose(stno);
#endif
                                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                                merror("mopen(data)");
#endif
                                return(-1);
                        }
                }
                Curseq = Reshdr.firstres;
                break;

        case READWRITE :

                /* Note that is a database is opened for read/write,
                 * then the database must already exist.  That is,
                 * the user cannot create a database and then look at
                 * it.  It's a deficiency which I'm not sure is worth
                 * fixing. */
                if (!mload(stno, fname)) {
#ifdef  MDEBUG
                        merror("mopen");
#endif
                        return(-1);
                }
                fd = getfile(fname, DAT_MAGIC, "r");
                if (fd == -1) {
#ifdef  MDEBUG
                        merror("mopen(data file)");
#endif
                        return(-1);
                }

                /* Copy the datafile into a temporary file */
                Tmpfile = (char *) malloc((unsigned)
                        MBASELEN * sizeof (char));
                strcpy(Tmpfile, gettemp(MDATA));
                Fd = getfile(Tmpfile, DAT_MAGIC, "w+");
                if (Fd == -1) {
#ifdef  MDEBUG
                        merror("mopen(data file)");
#endif
                        return(-1);
                }
                while ((n = bread(buf, sizeof (char), 512, fd)) > 0)
                        bwrite(buf, sizeof (char), n, Fd);
                bclose(fd);
#ifdef  vax
                bclose(Fd);
                Fd = getfile(Tmpfile, DAT_MAGIC, "r+");
                /* Can't just rewind it if we want to use vio.
                 * Got to open the file as mode "r+" instead. */
#else
                brewind(Fd);
#endif

                /* Allocate space for data */
                if (!slowmode) {
                        Data = (char *) malloc((unsigned)
                                Maxatm * Reshdr.atomsize);
                        if (Data == NIL) {
                                Mode |= READ;
#ifdef  f77
                                mclose_(&stno);
#else
                                mclose(stno);
#endif
                                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                                merror("mopen(data)");
```

```
#endif
                return(-1);
        }
    }
    Curseq = Reshdr.firstres;

    break;

case WRITE :

    /* Here we only set the variables */
    Header.magic = NDX_MAGIC;
    Header.rescnt = 0;
    Header.tplcnt = 0;
    STORELONG(Header.filesize, sizeof (struct header_def));
    Reshdr.firstres = -1;
    Reshdr.lastres = -1;
    Reshdr.atomsize = 0;
    Tpl = NIL;
    Connec = NIL;
    Residue = NIL;
    Data = NIL;
    Inode = 0177777;
    Dev = 0177777;
    Maxatm = 0;
    Curseq = -1;
    if (onbits == READWRITE)
        Fd = getfile(fname, DAT_MAGIC, "w+");
    else
        Fd = getfile(fname, DAT_MAGIC, "w");
    if (Fd == -1) {
#ifdef  MDEBUG
        merror("mopen(data file)");
#endif
        return(-1);
    }
    break;
}

/* Keep track of where we're from */
Basename = (char *) malloc((unsigned)
        (strlen(fname)+1) * sizeof (char));
if (Basename == NIL) {
    Mode |= READ;
#ifdef  f77
    mclose_(&stno);
#else
    mclose(stno);
#endif
    err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
    merror("mopen(filename)");
#endif
    return(-1);
}
strcpy(Basename, fname);

/* Set the mode */
Mode |= onbits;
if (slowmode)
    Mode |= SLOWMODE;
Floc = 0L;
return(stno);
}

/*
 * Mload loads an existing database into core.  It returns TRUE if
 * loading was successful and FALSE otherwise.
 */
mload(stno, fname)
int     stno;
char    *fname;
{
    register        MDBS    stream;
    register        i;
    int     input;
    int     n, maxatom;
    int     success;
#ifndef vms
    struct  stat    buf;
#endif

    /* Read the index file */
    stream = &_mstrm[stno];
    input = getfile(fname, NDX_MAGIC, "r");
    if (input == -1) {
#ifdef  MDEBUG
        merror("mload(index file)");
#endif
        return(FALSE);
    }
    bread((char *) &Header, sizeof (struct header_def), 1, input);
    bread((char *) &Reshdr, sizeof (struct reshdr_def), 1, input);
    Residue = (struct residue_def *) malloc((unsigned) Header.rescnt *
```

```
                sizeof (struct residue_def));
    if (Residue == NIL) {
                bclose(input);
                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                merror("mload(residue)");
#endif
                return(FALSE);
        }
        n = bread((char *) Residue, sizeof (struct residue_def),
                (int) Header.rescnt, input);
        bclose(input);
        if (n != Header.rescnt) {
                free((char *) Residue);
                err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                merror("mload");
#endif
                return(FALSE);
        }

        /* Find the largest residue so we know how much space to save
         * for atom data */
        maxatom = 0;
        for (i = 0; i < Header.rescnt; i++) {
                if (Residue[i].atomcnt > maxatom)
                        maxatom = Residue[i].atomcnt;
        }
        Maxatm = maxatom;

        /* Now for the template file and the residue connectivities */
        input = getfile(fname, TPL_MAGIC, "r");
        if (input == -1) {
                free((char *) Residue);
#ifdef  MDEBUG
                merror("mload(template file)");
#endif
                return(FALSE);
        }

        /* Try to share the template variables (except in vms) */
#ifndef vms
        fstat(input, &buf);
        Inode = buf.st_ino;
        Dev = buf.st_dev;
        for (i = 0; i < MAXSTREAM; i++)
                if ((_mstrm[i].mode & ACCESS)
                        && buf.st_ino == _mstrm[i].inode) {
                        bclose(input);
                        Tpl = _mstrm[i].tpl;
                        Connec = _mstrm[i].connec;
                        Maxatm = _mstrm[i].maxatm;
                        return(TRUE);
                }
#endif

        /* General bookkeeping */
        Tpl = (struct tpl_def *) malloc((unsigned) Header.tplcnt *
                sizeof (struct tpl_def));
        if (Tpl == NIL) {
                bclose(input);
                free((char *) Residue);
                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                merror("mload(template)");
#endif
                return(FALSE);
        }
        Connec = (struct connec_def **) malloc((unsigned) Header.tplcnt *
                sizeof (struct connec_def *));
        if (Connec == NIL) {
                bclose(input);
                free((char *) Residue);
                free((char *) Tpl);
                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                merror("mload(conn list)");
#endif
                return(FALSE);
        }
        bseek(input, (long) sizeof (struct header_def), 0);
        success = TRUE;

        /* Read in template header and connectivity */
        for (i = 0; i < Header.tplcnt; i++) {
                n = bread((char *) &Tpl[i], sizeof (struct tpl_def), 1, input);
                if (n != 1) {
                        success = FALSE;
                        err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                        merror("mload(template header)");
#endif
                        break;
                }
```

```
                Connec[i] = (struct connec_def *)
                        malloc((unsigned)
                                Tpl[i].natom * sizeof (struct connec_def));
                if (Connec[i] == NIL) {
                        success = FALSE;
                        err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                        merror("mload(conn list)");
#endif
                        break;
                }
                n = bread((char *) Connec[i], sizeof (struct connec_def),
                        (int) Tpl[i].natom, input);
                if (n != Tpl[i].natom) {
                        success = FALSE;
                        err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                        merror("mload(connectivity)");
#endif
                        break;
                }
        }
        bclose(input);
        return(success);
}

/*
 * Getfile returns the file pointer for the given file
 */
int
getfile(fname, magic, mode)
char    *fname;
short   magic;                  /* MAGIC numbers are ALWAYS short */
char    *mode;
{
        char    fn[80];
        struct  header_def      header;
        register        fd;

        if (fname != NIL)
                strcpy(fn, fname);
        else {
                err_midas = EM_NOPERM;
#ifdef  MDEBUG
                merror("getfile(null filename)");
#endif
                return(-1);
        }
        switch (magic) {
                case NDX_MAGIC :
                        strcat(fn, MINDEX);
                        break;
                case DAT_MAGIC :
                        strcat(fn, MDATA);
                        break;
                case TPL_MAGIC :
                        strcat(fn, MTPL);
                        break;
        }
        fd = bopen(fn, mode);
        if (fd == -1 || *mode == 'w')
                return(fd);

        /* Check the header to make sure we got what we want */
        bread((char *) &header, sizeof header, 1, fd);
        if (header.magic != magic) {
                bclose(fd);
                err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                merror("getfile");
#endif
                return(-1);
        }
        brewind(fd);
        return(fd);
}
```

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mclose.c,v 3.4 84/12/06 13:52:48 amol
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          27 Jul 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Mclose closes the database.  The datafile is written out if necessary
 * and all associated variables are freed if they are not shared
 */
#ifndef  f77
mclose(stno)
int      stno;
#else
mclose_(fstno)
int      *fstno;
#endif
{
        register            MDBS    stream;
        register            i;
        int      found;
        char     fn[80];
        int      fd;
#ifdef   f77
        int      stno;

        stno = *fstno;
#endif
        if (stno < 0 || stno >= MAXSTREAM) {
                err_midas = EM_SINVAL;
#ifdef   MDEBUG
                merror("mclose");
#endif
                return(-1);
        }

        /* Find another stream which shares the template with this stream.
         * If there is one, then we do not delete the templates. */
        stream = &_mstrm[stno];
        if ((Mode & ACCESS) == MFREE) {
                err_midas = EM_NOPERM;
#ifdef   MDEBUG
                merror("mclose(file not open)");
#endif
                return(-1);
        }
        found = FALSE;
#ifndef  vms            /* No template sharing in vms */
        for (i = 0; i < MAXSTREAM; i++) {
                if (stno == i || (_mstrm[i].mode & ACCESS) == MFREE)
                        continue;
                if (_mstrm[i].inode == Inode) {
                        found = TRUE;
                        break;
                }
        }
#endif

        /* Clear out the stream and write out database if it was opened
         * for writing ONLY.  Note that mclose does NOT write out the
         * database if it were opened for readwrite. */
#ifdef   f77
        mflush_(fstno);
#else
        mflush(stno);
#endif
        if ((Mode & ACCESS) == WRITE || ((Mode & ACCESS) == READWRITE &&
                Inode == 0177777)) {

                /* Write out the data file */
                bseek(Fd, 0L, 0);
                Header.magic = DAT_MAGIC;
                bwrite((char *) &Header, sizeof (struct header_def), 1, Fd);
                fd = getfile(Basename, NDX_MAGIC, "w");
                if (fd == -1) {
                        strcpy(fn, Basename);
                        strcat(fn, MDATA);
                        unlink(fn);

                        /* Remove the temporary file */
                        if ((Mode & ACCESS) == READWRITE && Tmpfile != NIL) {
                                strcpy(fn, Tmpfile);
                                strcat(fn, MDATA);
                                unlink(fn);
```

```
                        free(Tmpfile);
                }
#ifdef  MDEBUG
                        merror("mclose(index file)");
#endif
                        return(-1);
                }

                /* Write out the index file */
                Header.magic = NDX_MAGIC;
                bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
                bwrite((char *) &Reshdr, sizeof (struct reshdr_def), 1, fd);
                bwrite((char *) Residue, sizeof (struct residue_def),
                        (int) Header.rescnt, fd);
                bclose(fd);

                /* Write out the template */
                fd = getfile(Basename, TPL_MAGIC, "w");
                if (fd == -1) {
                        /* We cannot create the template file so
                         * we get rid of the stuff we created before */
                        strcpy(fn, Basename);
                        strcat(fn, MINDEX);
                        unlink(fn);
                        strcpy(fn, Basename);
                        strcat(fn, MDATA);
                        unlink(fn);

                        /* Remove the temporary file */
                        if ((Mode & ACCESS) == READWRITE && Tmpfile != N
                                strcpy(fn, Tmpfile);
                                strcat(fn, MDATA);
                                unlink(fn);
                                free(Tmpfile);
                        }
#ifdef  MDEBUG
                        merror("mclose(template file)");
#endif
                        return(-1);
                }
                Header.magic = TPL_MAGIC;
                bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
                for (i = 0; i < Header.tplcnt; i++) {
                        bwrite((char *) &Tpl[i], sizeof (struct tpl_def),1, fd);
                        bwrite((char *) Connec[i], sizeof (struct connec_def),
                                (int) Tpl[i].natom, fd);
                }
                        bclose(fd);
                }
                bclose(Fd);

                /* Delete all the associated variables if they're not shared */
                if (!found) {
                        if (Connec != NIL) {
                                for (i = 0; i < Header.tplcnt; i++)
                                        if (Connec[i] != NIL)
                                                free((char *) Connec[i]);
                                free((char *) Connec);
                        }
                        if (Tpl != NIL)
                                free((char *) Tpl);
                }
                if (Residue != NIL)
                        free((char *) Residue);
                if (Data != NIL)
                        free(Data);
                if (Basename != NIL)
                        free(Basename);

                /* Remove the temporary file */
                if ((Mode & ACCESS) == READWRITE && Tmpfile != NIL) {
                        strcpy(fn, Tmpfile);
                        strcat(fn, MDATA);
                        unlink(fn);
                        free(Tmpfile);
                }

                Mode = 0;                       /* All bits clear = MFREE */
                return(0);                      /* All done */
}
```

mread.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mread.c,v 3.3 84/12/06 13:53:17 amol
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Mreadr copies the residue information into the user supplied variables
 */
#ifndef  f77
mreadr(stno, resseq, restype)
int      stno;
register         char    *resseq, *restype;
#else
mreadr_(fstno, resseq, restype, len1, len2)
int      *fstno;
register         char    *resseq, *restype;
long     len1, len2;
#endif
{
         register        MDBS    stream;
#ifdef   f77
         int     i, j;
         int     stno;

         stno = *fstno;
#endif
         stream = &_mstrm[stno];                 /* Set stream variable */
         if (!(Mode & READ)) {                   /* Check for permission */
                  err_midas = EM_NOPERM;
#ifdef   MDEBUG
                  merror("mreadr(read)");
#endif
                  return(-1);
         }
         if (Reshdr.atomsize <= 0) {             /* Is there anything there */
                  err_midas = EM_EMPTY;
#ifdef   MDEBUG
                  merror("mreadr");
#endif
                  return(-1);
         }
         if (Mode & ADV_IF_R) {                  /* Advance the stream if necessary */
#ifdef   f77
                  mflush_(fstno);
#else
                  mflush(stno);
#endif
                  Curseq = NEXTR(Curseq);
                  Mode &= ~(DAT_IN_CORE | ADVANCE);
         }
         if (Curseq == -1)                       /* Check for termination */
                  return(-1);
         strncpy(resseq, Curres.seq, RES_SEQ_SIZE);
         strncpy(restype, Curtpl.restype, RES_TYPE_SIZE);
#ifdef   f77
         fillsp_(resseq, (int) len1);
         fillsp_(restype, (int) len2);
#else
         resseq[RES_SEQ_SIZE] = '\0';
         restype[RES_TYPE_SIZE] = '\0';
#endif
         Mode |= ADV_IF_R;                       /* Set new mode */
         return(Curres.atomcnt);                 /* Done */
}

/*
 * Mreada checks whether the data for the current residue have been read.
 * If they have, then mreada advances the stream pointer.  It then reads
 * in the data for the residue and returns a pointer to the data.
 */
#ifndef  f77
mreada(stno)
int      stno;
#else
mreada_(fstno)
int      *fstno;
#endif
{
         register                MDBS    stream;
         int     nread;
         long    offset;
#ifdef   f77
         int     stno;
```

```
        stno = *fstno;
#endif

        stream = &_mstrm[stno];
        if (!(Mode & READ)) {                   /* Check permission */
                err_midas = EM_NOPERM;
#ifdef  MDEBUG
                merror("mreada(read)");
#endif
                return(-1);
        }
        if (Reshdr.atomsize <= 0) {             /* Check for empty database */
                err_midas = EM_EMPTY;
#ifdef  MDEBUG
                merror("mreada");
#endif
                return(-1);
        }
        if (Mode & SLOWMODE)                    /* Do nothing for slow mode */
                return(-1);
        if (Mode & ADV_IF_A) {                  /* Advance pointer */
#ifdef  f77
                mflush_(fstno);
#else
                mflush(stno);
#endif
                Curseq = NEXTR(Curseq);
                Mode &= ~(DAT_IN_CORE | ADVANCE);
        }
        if (Curseq == -1)                       /* Check for end */
                return(-1);
        if (Mode & DAT_IN_CORE)                 /* Check if we already have this */
                return(0);                      /* ... residue in core */

        GETLONG(offset, Curres.offset);
        if (Floc != offset)             /* Seek to correct location */
                bseek(Fd, offset, 0);
        nread = bread(Data, (int) Reshdr.atomsize, (int) Curres.atomcnt, Fd);
        Floc = btell(Fd);                       /* Set file offset */
        if (nread != Curres.atomcnt) {          /* Check for consistency */
                err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                merror("mreada(data file)");
#endif
                return(-1);
        }
        Mode |= (ADV_IF_A | DAT_IN_CORE);       /* Set new mode */
        return(0);                              /* Done */
}

/*
 * Mgeta loads the atomic data into the user supplied buffer
 */
#ifndef  f77
mgeta(stno, index, buffer)
int     stno, index;
char    *buffer;
#else
mgeta_(fstno, findex, buffer)
int     *fstno, *findex;
char    *buffer;
#endif
{
        register        MDBS    stream;
        int     nread;
        char    *dat;
        long    atloc, offset;
#ifdef  f77
        int     stno, index;

        stno = *fstno;
        index = *findex - 1;
#endif
        stream = &_mstrm[stno];
        if (!(Mode & READ)) {                           /* Permission ? */
                err_midas = EM_NOPERM;
#ifdef  MDEBUG
                merror("mgeta(read)");
#endif
                return(-1);
        }
        if (Curseq == -1)                               /* At a residue ? */
                return(-1);
        if (index < 0 || index >= Curres.atomcnt) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("mgeta");
#endif
                return(-1);
        }
        if (Mode & SLOWMODE) {
                atloc = index;
                GETLONG(offset, Curres.offset);
                atloc = offset + atloc * Reshdr.atomsize;
                if (Floc != atloc)                      /* Seek to correct location */
```

```
                bseek(Fd, atloc, 0);
        nread = bread(buffer, (int) Reshdr.atomsize, 1, Fd);
        Floc = btell(Fd);                       /* Set file offset */
        if (nread != 1) {                       /* Check for consistency *
                err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                merror("mreada(data file)");
#endif
                return(-1);
        }
    } else {
        dat = Data + index * Reshdr.atomsize;
        blkcpy(dat, buffer, Reshdr.atomsize);
    }
    return(0);
}

/*
 * Mdatptr returns the pointer to the current atomic data
 */
#ifndef f77
char    *
mdatptr(stno)
int     stno;
{
    register            MDBS    stream;

    stream = &_mstrm[stno];
    if (Mode & SLOWMODE) {
        err_midas = EM_ILLOP;
#ifdef  MDEBUG
        merror("mdatptr(slowmode)");
#endif
        return(NIL);
    }
    if (Curseq == -1)
        return(NIL);
    return(Data);
}
#endif

/*
 * Matom returns a pointer to the name of the atom which is in location
 * index of the current residue
 */
#ifndef f77
matom(stno, index, abuf)
register        stno;
register        index;
char    *abuf;
#else
matom_(fstno, findex, abuf, len)
int     *fstno, *findex;
char    *abuf;
long    len;
#endif
{
    register            MDBS    stream;
#ifdef  f77
    register            stno, index;

    stno = *fstno;
    index = *findex - 1;
#endif
    stream = &_mstrm[stno];
    if (Curseq == -1) {                         /* Check for end */
        err_midas = EM_ATEOF;
#ifdef  MDEBUG
        merror("matom");
#endif
        return(-1);
    }
    if (index < 0 || index >= Curres.atomcnt) {
        err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
        merror("matom");
#endif
        return(-1);
    }
    strncpy(abuf, Curcon[index].atname, AT_NAME_SIZE);
#ifdef f77
    fillsp_(abuf, (int) len);
#else
    abuf[AT_NAME_SIZE] = '\0';                  /* Null terminate the name */
#endif
    return(0);
}
```

msave.c

```
/* $Header: msave.c,v 3.5 86/03/10 11:43:29 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          27 Jul 1983
 */

#include <stdio.h>
#ifndef  vms
#include <mdbs.h>
#include <sys/stat.h>
extern int              errno;
#else
#include "MIDAS$H:mdbs.h"
extern noshare int              errno;
#endif

/*
 * Msave writes out the database into the given file.  The template file
 * will be shared unless new templates have been added.
 */
#ifndef  f77
msave(stno, file)
int     stno;
char    *file;
#else
msave_(fstno, file, len)
int     *fstno;
char    *file;
long    len;
#endif
{
        register                MDBS    stream;
        char    fn[80], alt[80], *suffix;
        char    buf[512];
        int     n;
        int     fd;
        int     samefile, samedev;
#ifndef  vms
        struct  stat    statb;
#endif
#ifdef   f77
        int     stno;
        stno = *fstno;
#endif
        if (stno < 0 || stno >= MAXSTREAM) {
                err_midas = EM_SINVAL;
#ifdef   MDEBUG
                merror("msave");
#endif
                return(-1);
        }
        stream = &_mstrm[stno];
        if ((Mode & ACCESS) != READWRITE) {             /* Can't save unless RW mode */
                err_midas = EM_NOPERM;
#ifdef   MDEBUG
                merror("msave(read/write)");
#endif
                return(-1);
        }

        /* If there is only one argument, then save into the original file */
        samefile = nargs() == 1;

        /* Copy the temporary file into the new data file */
        if (samefile) {
                strcpy(fn, Basename);
                suffix = &fn[strlen(fn)];
                samedev = TRUE;
        } else {
#ifdef   vms
                /* no link() on vms, hence can't share templates */
                strcpy(fn, file);
                suffix = &fn[strlen(fn)];
                samedev = FALSE;
#else    vms
#ifdef   f77
                termin_(file, (int) len);
#endif
                strcpy(fn, file);
                suffix = &fn[strlen(fn)];
                strcpy(suffix, MTPL);
                if (access(fn, 0) == 0) {
                        stat(fn, &statb);
                        if (statb.st_ino == inode)
                                samefile = TRUE;
                } else {
                        char    *cp, *fp, *rindex();

                        cp = rindex(fn, '/');
```

```
                        if (cp != NULL) {
                                *cp = '\0';
                                fp = fn;
                        } else
                                fp = ".";
                        n = stat(fp, &statb);
                        if (n == -1) {
                                err_midas = errno;
#ifdef  MDEBUG
                                merror("msave(template file)");
#endif
                                if (cp != NULL)
                                        *cp = '/';
                                return(-1);
                        }
                        if (cp != NULL)
                                *cp = '/';
                        samefile = FALSE;
                }
                samedev = statb.st_dev == Dev;
#endif  vms
        }
        strcpy(suffix, MDATA);
        fd = bopen(fn, "w");
        if (fd == -1) {
#ifdef  MDEBUG
                merror("msave(data file)");
#endif
                return(-1);
        }
#ifdef  f77
        mflush_(fstno);
#else
        mflush(stno);
#endif
        bseek(Fd, (long) sizeof (struct header_def), 0);
        Header.magic = DAT_MAGIC;
        bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
        while ((n = bread(buf, sizeof (char), 512, Fd)) > 0)
                bwrite(buf, sizeof (char), n, fd);
        bclose(fd);
        Floc = btell(Fd);

        /* Copy the template file if more templates have been added */
        if ((Mode & TPLCHG) || !samedev) {
                strcpy(suffix, MTPL);
                fd = bopen(fn, "w");
                if (fd == -1) {
#ifdef  MDEBUG
                        merror("msave(template file)");
#endif
                        strcpy(suffix, MDATA);
                        unlink(fn);
                        return(-1);
                }
                Header.magic = TPL_MAGIC;
                bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
                for (n = 0; n < Header.tplcnt; n++) {
                        bwrite((char *) &Tpl[n], sizeof (struct tpl_def),
                                1, fd);
                        bwrite((char *) Connec[n], sizeof (struct connec_def),
                                (int) Tpl[n].natom, fd);
                }
                bclose(fd);
        } else if (!samefile) {
#ifndef ALWAYSCOPY
                strcpy(suffix, MTPL);
                strcpy(alt, Basename);
                strcat(alt, MTPL);
                link(alt, fn);
#else
                fd = bopen(fn, "r+");
                if (fd == -1) {
#ifdef  MDEBUG
                        merror("msave(template file)");
#endif
                        unlink(fn);
                        strcpy(suffix, MDATA);
                        unlink(fn);
                        return(-1);
                }
                Header.magic = TPL_MAGIC;
                bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
                bclose(fd);
#endif
        }

        /* Write the new index file */
        strcpy(suffix, MINDEX);
        fd = bopen(fn, "w");
        if (fd == -1) {
                strcpy(suffix, MDATA);
                unlink(fn);
                if (!samefile) {
```

```
                strcpy(suffix, MTPL);
                unlink(fn);
        }
#ifdef  MDEBUG
        merror("msave(index file)");
#endif
        return(-1);
    }
    Header.magic = NDX_MAGIC;
    bwrite((char *) &Header, sizeof (struct header_def), 1, fd);
    bwrite((char *) &Reshdr, sizeof (struct reshdr_def), 1, fd);
    bwrite((char *) Residue, sizeof (struct residue_def),
            (int) Header.rescnt, fd);
    bclose(fd);                     /* Done */
    return(0);
}
```

mtrav.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mtrav.c,v 3.2 84/12/06 13:53:02 arnold
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

static   char    mark[MAXATOM];                  /* Mark of visit */

/*
 * Mtrav sets up the conditions required by msearch to traverse the residue
 */
#ifndef  f77
mtrav(stno, visit, again)
int      stno;
int      (*visit)(), (*again)();
#else
mtrav_(fstno, visit, again)
int      *fstno;
int      (*visit)(), (*again)();
#endif
{
         register        i;
         register        MDBS    stream;
#ifdef   f77
         int     stno;

         stno = *fstno;
#endif
         stream = &_mstrm[stno];
         if (Curseq == -1) {
#ifdef   MDEBUG
                  merror("Mtrav: At end of file.\n");
#endif
                  return(-1);
         }
         for (i = 0; i < MAXATOM; i++)                  /* Clear out the array */
                  mark[i] = FALSE;
         msearch(stno, Curtpl.chief, visit, again);     /* Search it */
         return(0);
}

/*
 * Msearch actually does the work of traversal.  It is a simple depth first
 * search of the residue.  Hopefully the residue is done correctly so that
 * this simple traversal will not run into trouble.
 */
msearch(stno, index, visit, again)
int      stno;
register         index;
int      (*visit)(), (*again)();
{
         register        i;
         register        MDBS    stream;
#ifdef   f77
         int     tindex, tchief, tlink, tocnt, tfirstime;
#endif

         /* Visit this node and mark it.  If we were here before, then
          * we will not pursue it any further.  */
         stream = &_mstrm[stno];
#ifdef   f77
         /* Make a temporary copy of everything for FORTRAN */
         tindex = index + 1;
         tchief = (index == Curtpl.chief);
         tlink = (index == Curtpl.linkage);
         tocnt = Curcon[index].tocnt;
         tfirstime = !mark[index];
         (*visit)(&stno, &tindex, &tchief, &tlink, &tocnt, &tfirstime);
#else
         (*visit)(stno, index, index == Curtpl.chief, index == Curtpl.linkage,
                  Curcon[index].tocnt, !mark[index]);
#endif
         if (mark[index])
                  return;
         mark[index] = TRUE;

         /* Visit each son and note our return */
         for (i = 0; i < Curcon[index].tocnt; i++) {
                  msearch(stno, Curcon[index].to[i], visit, again);
#ifdef   f77
                  tocnt = Curcon[index].tocnt - i - 1;
                  (*again)(&stno, &tindex, &tchief, &tlink, &tocnt);
#else
                  (*again)(stno, index, index == Curtpl.chief,
                           index == Curtpl.linkage, Curcon[index].tocnt - i - 1);
```

mwrite.c

```
/* $Header: mwrite.c,v 3.7 86/07/21 14:53:19 gregc Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          12 Jul 1983
 * Release 2.2                          27 Jul 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

static   int      noclear;

/*
 * Mwrite flushes the stream buffer if necessary and copies the
 * user information into the buffer.  Note that the new information
 * is NOT written out to disk immediately but is kept in case the
 * user repeatedly reference the same residue
 */
#ifndef  f77
mwrite(stno, resseq, restype, atoms, natom, atomsize)
int      stno;
char     *resseq, *restype;
char     *atoms;
int      natom, atomsize;
{
         register          MDBS     stream;

         stream = &_mstrm[stno];
         if (Mode & SLOWMODE) {
                  err_midas = EM_ILLOP;
#ifdef   MDEBUG
                  merror("mwrite(slowmode)");
#endif
                  return(-1);
         }
         noclear = TRUE;
         if ((natom = mwrtr(stno, resseq, restype, natom, atomsize)) == -1)
                  return(-1);
         if (atoms != Data)
                  blkcpy(atoms, Data, natom * atomsize);
         return(0);
}
#endif

/*
 * Mwrtr makes changes to the residue index if the residue is being
 * changed or added.
 */
#ifndef  f77
mwrtr(stno, resseq, restype, natom, atomsize)
int      stno;
char     *resseq, *restype;
int      natom, atomsize;
#else
mwrtr_(fstno, resseq, restype, fnatom, fatomsize, lens, lent)
int      *fstno;
char     *resseq, *restype;
int      *fnatom, *fatomsize;
long     lens, lent;
#endif
{
         int      i, j;
         int      match;
         int      last, comp;
         struct   residue_def       res;
         register          MDBS     stream;
#ifdef   f77
         int      stno, natom, atomsize;

         stno = *fstno;
         natom = *fnatom;
         atomsize = *fatomsize;
#endif
         stream = &_mstrm[stno];
         if (!(Mode & WRITE)) {                    /* Check for permission */
                  err_midas = EM_NOPERM;
#ifdef   MDEBUG
                  merror("mwrtr(write)");
#endif
                  return(-1);
         }
#ifdef   f77
         mflush_(fstno);
         termin_(resseq, (int) lens);
         termin_(restype, (int) lent);
#else
         mflush(stno);
```

```
#endif

        if (Curseq == -1)                       /* Are we at one end ? */
                match = -1;
        else                                    /* Are we replacing ? */
                match = smatch(Curres.seq, resseq);

        /* We are updating the current residue */
        if (match == 0) {
                if (Reshdr.atomsize != atomsize) {
                        err_midas = EM_ILLOP;
#ifdef  MDEBUG
                        merror("mwrtr(inconsistent atom size)");
#endif
                        return(-1);
                }
                i = dores(stno, restype, Curseq, natom, atomsize);
                if (i > 0 && !(Mode & SLOWMODE)) {
                        if (noclear)
                                noclear = FALSE;
                        else
                                blkclr(Data, i * atomsize);
                }
                return(i);
        }

        /* We are adding to the database */
        else {
                Curseq = -1;
                if (Reshdr.atomsize == 0)
                        Reshdr.atomsize = atomsize;
                else if (Reshdr.atomsize != atomsize) {
                        err_midas = EM_ILLOP;
#ifdef  MDEBUG
                        merror("mwrtr(inconsistent atom size)");
#endif
                        return(-1);
                }

                /* Get the extra space we need for the new residue */
                if (Residue == NIL)
                        Residue = (struct residue_def *)
                                malloc((unsigned) sizeof (struct residue_def
                else
                        Residue = (struct residue_def *)
                                realloc((char *) Residue,
                                (unsigned) (Header.rescnt + 1) *
                                sizeof (struct residue_def));
                if (Residue == NIL) {
                        err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                        merror("mwrtr(residue)");
#endif
                        return(-1);
                }

                /* Locate where in the list it should go */
                if (mberch(stno, resseq, &last, &comp) != -1) {
                        err_midas = EM_ILLOP;
#ifdef  MDEBUG
                        merror("mwrtr(existing residue)");
#endif
                        return(-1);
                }
                if (comp < 0)
                        last++;
                i = dores(stno, restype, Header.rescnt, natom, atomsize);

                /* Take appropriate action now that we have either added
                 * the new residue or failed */
                if (i < 0) {                    /* Blew it! */
                        Residue = (struct residue_def *)
                                realloc((char *) Residue, (unsigned)
                                Header.rescnt * sizeof (struct residue_def));
                        return(-1);
                } else {                        /* More work to do on success */

                        /* Fix up all the pointers */
                        if (Reshdr.firstres >= last)
                                Reshdr.firstres++;
                        if (Reshdr.lastres >= last)
                                Reshdr.lastres++;
                        for (j = 0; j < Header.rescnt; j++) {
                                if (Residue[j].resnext >= last)
                                        Residue[j].resnext++;
                                if (Residue[j].resprev >= last)
                                        Residue[j].resprev++;
                        }

                        /* Shift the new residue into the correct place.
                         * Note that we do this only if we succeed. Thus,
                         * failures will not mess up our format */
                        res = Residue[Header.rescnt];
                        for (j = Header.rescnt; j > last; j--)
                                Residue[j] = Residue[j - 1];
```

```
                Residue[last] = res;

                /* Link the residue into the list */
                Residue[last].resnext = -1;
                Residue[last].resprev = Reshdr.lastres;
                if (Reshdr.lastres != -1)
                        Residue[Reshdr.lastres].resnext = last;
                strncpy(Residue[last].seq, resseq, RES_SEQ_SIZE);
                Curseq = last;
                if (Reshdr.firstres == -1)
                        Reshdr.firstres = last;
                Reshdr.lastres = last;
                Header.rescnt++;
                if (!(Mode & SLOWMODE)) {
                        if (noclear)
                                noclear = FALSE;
                        else
                                blkclr(Data, i * atomsize);
                }
                return(i);
        }
    }
}

/*
 * Dores tries to load a residue into core.  Either an old template will
 * be used or a new one is added into core.
 */
dores(stno, restype, index, natom, atomsize)
int     stno;
char    *restype;
int     index;
int     natom, atomsize;
{
        register        i;
        struct  connec_def      *mgettpl();
        struct  connec_def      **save;
        long    ressize, filesize;
        register        MDBS    stream;

        /* Try to find the template in core */
        stream = &_mstrm[stno];
        for (i = 0; i < Header.tplcnt; i++)
                if (tmatch(Tpl[i].restype, restype) == 0)
                        break;

        /* Didn't find it.  Must get it from standard directory and
         * load it into core */
        if (i >= Header.tplcnt) {
                if (Tpl == NIL)
                        Tpl = (struct tpl_def *) malloc((unsigned)
                                sizeof (struct tpl_def));
                else
                        Tpl = (struct tpl_def *)
                                realloc((char *) Tpl, (unsigned)
                                (Header.tplcnt + 1) * sizeof (struct tpl_def));
                if (Tpl == NIL) {
                        err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                        merror("dores(tpl header)");
#endif
                        return(-1);
                }
                save = Connec;
                if (Connec == NIL)
                        Connec = (struct connec_def **) malloc((unsigned)
                                sizeof (struct connec_def *));
                else
                        Connec = (struct connec_def **)
                                realloc((char *) Connec,
                                (unsigned) (Header.tplcnt + 1) *
                                sizeof (struct connec_def *));
                if (Connec == NIL) {
                        /* Release connectivities (Note that we are actually
                         * using data in an area that is already marked as
                         * free by the memory allocation system.  We KNOW??
                         * that the memory will not be over-written yet. */
                        if (save != NIL)
                                for (i = 0; i < Header.tplcnt; i++)
                                        if (save[i] != NIL)
                                                free((char *) save[i]);
                        err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                        merror("dores(conn list)");
#endif
                        return(-1);
                }
                Connec[Header.tplcnt] = mgettpl(restype, &Tpl[Header.tplcnt]);
                if (Connec[Header.tplcnt] == NIL) {
                        Tpl = (struct tpl_def *) realloc((char *) Tpl,
                                (unsigned) (Header.tplcnt) *
                                sizeof (struct tpl_def));
                        Connec = (struct connec_def **) realloc((char *) Connec,
                                (unsigned) (Header.tplcnt) *
```

```
                                        sizeof (struct connec_def *));
                                return(-1);
                        }
                        Mode |= TPLCHG;
                        i = Header.tplcnt++;
                }
                if (natom < 0)
                        natom = Tpl[i].natom;
                if (natom > Maxatm) {
                        Maxatm = natom;
                        if (!(Mode & SLOWMODE)) {
                                if (Data == NIL)
                                        Data = (char *) malloc((unsigned)
                                                natom * atomsize);
                                else
                                        Data = realloc(Data, (unsigned) natom *
                                                atomsize);
                                if (Data == NIL) {
                                        err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
                                        merror("dores(data)");
#endif
                                        return(-1);
                                }
                        }
                }
                /* We must extend the database if this a a new residue or
                 * the number of atoms in the current residue increased */
                if (Curseq == -1 || natom > Residue[index].atomcnt) {
                        GETLONG(filesize, Header.filesize);
                        bseek(Fd, filesize, 0);
                        filesize = Floc = btell(Fd);
                        STORELONG(Residue[index].offset, filesize);
                        ressize = natom;                /* Convert to long before mult */
                        ressize *= atomsize;
                        filesize = bseek(Fd, ressize, 1);
                        STORELONG(Header.filesize, filesize);
                        bseek(Fd, Floc, 0);
                }
                Residue[index].type = i;
                Residue[index].atomcnt = natom;
                return(natom);
}

/*
 * Mgettpl gets the template from the standard directory and returns
 * the connectivity of the residue.  It returns NIL on failure.
 */
struct  connec_def      *
mgettpl(restype, tpl)
char    *restype;
struct  tpl_def   *tpl;
{
        register int            fd;
        struct  header_def      header;
        struct  connec_def      *p;
        char    fn[80];
        static  char    modeldir[80];

        /* Find the template file */
        if (modeldir[0] == '\0')
                tplpath(modeldir, "r");
        strcpy(fn, modeldir);
        strcat(fn, restype);
        strcat(fn, TDATA);
        fd = bopen(fn, "r");
        if (fd == -1) {
                strcpy(fn, TPL_PATH);
                strcat(fn, restype);
                strcat(fn, TDATA);
                fd = bopen(fn, "r");
                if (fd == -1) {
#ifdef  MDEBUG
                        merror("mgettpl(template file)");
#endif
                        return(NIL);
                }
        }
        bread((char *) &header, sizeof header, 1, fd);
        if (header.magic != TPL_MAGIC) {
                bclose(fd);
                err_midas = EM_MISMATCH;
#ifdef  MDEBUG
                merror("mgettpl(template)");
#endif
                return(NIL);
        }
        bread((char *) tpl, sizeof (struct tpl_def), 1, fd);
        p = (struct connec_def *)
                malloc((unsigned) tpl->natom * sizeof (struct connec_def));
        if (p == NIL) {
                bclose(fd);
                err_midas = EM_OUTOFMEMORY;
#ifdef  MDEBUG
```

```
#endif                  merror("mgettpl(conn list)");

                        return(NIL);
        }
        bread((char *) p, sizeof (struct connec_def), (int) tpl->natom, fd);
        bclose(fd);
        return(p);
}

/*
 * Mputa copies the information in the user buffer into the MIDAS buffer
 */
#ifndef f77
mputa(stno, index, buffer)
int     stno;
int     index;
char    *buffer;
#else
mputa_(fstno, findex, buffer)
int     *fstno;
int     *findex;
char    *buffer;
#endif
{
        register                MDBS    stream;
        char    *p;
        long    atloc, offset;
#ifdef  f77
        int     stno, index;

        stno = *fstno;
        index = *findex - 1;
#endif
        stream = &_mstrm[stno];
        if (!(Mode & WRITE)) {                  /* Check for permission */
                err_midas = EM_NOPERM;
#ifdef  MDEBUG
                merror("mputa(write)");
#endif
                return(-1);
        }
        if (Curseq == -1) {
                err_midas = EM_ATEOF;
#ifdef  MDEBUG
                merror("mputa");
#endif
                return(-1);
        }
        if (index < 0 || index >= Curres.atomcnt) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("mputa");
#endif
                return(-1);
        }
#ifdef  f77
        mflush_(fstno);
#else
        mflush(stno);
#endif
        if (Mode & SLOWMODE) {
                atloc = index;
                GETLONG(offset, Curres.offset);
                atloc = offset + atloc * Reshdr.atomsize;
                if (Floc != atloc)
                        beesk(Fd, atloc, 0);
                bwrite(buffer, (int) Reshdr.atomsize, 1, Fd);
                Floc = btell(Fd);
        } else {
                p = &Data[index * Reshdr.atomsize];
                if (p != buffer)
                        blkcpy(buffer, p, Reshdr.atomsize);
        }
        return(0);
}

/*
 * Mwrta sets the flush bit for the given stream
 */
#ifndef f77
mwrta(stno)
int     stno;
#else
mwrta_(fstno)
int     *fstno;
#endif
{
        register                MDBS    stream;
#ifdef  f77
        int     stno;

        stno = *fstno;
#endif
        stream = &_mstrm[stno];
```

```
        if (!(Mode & WRITE)) {
                err_midas = EM_NOPERM;
#ifdef  MDEBUG
                merror("mwrta(write)");
#endif
                return(-1);
        }
        if (Curseq == -1) {
                err_midas = EM_ATEOF;
#ifdef  MDEBUG
                merror("mwrta");
#endif
                return(-1);
        }
        if (Mode & SLOWMODE) {
                err_midas = EM_ILLOP;
#ifdef  MDEBUG
                merror("mwrta(slowmode)");
#endif
                return(-1);
        }
        Mode |= (MFLUSH | ADVANCE);
        return(0);
}

/*
 * Tplpath determines the file pathname for templates
 *
 * Algorithm is as follows:
 * First check for a MODELS environment
 * variable, if none is found then
 * look for a $HOME/models directory;
 * if still empty-handed then use the
 * current directory.
 */
tplpath(path, rwflag)
char *path, *rwflag;
{
        register char *cp;
#ifndef vms
        char *getenv();

        if (cp = getenv("MODELS")) {
                strcpy(path, cp);
                strcat(path, "/");
        } else if (cp = getenv("HOME")) {
                strcpy(path, cp);
                strcat(path, "/models/");
                if (access(path, 0) < 0)
                        strcpy(path, "./");
        } else
                strcpy(path, "./");
        if (strcmp(rwflag, "w")==0 && access(path, 02)) {
                err_midas = EM_NOPERM;
                return(-1);
        }
#else
        char *getlog();

        if (cp = getlog("USER$MODELS")) {
                strcpy(path, cp);
                cp = path + strlen(path) - 1;
                if (*cp != ']' && *cp != ':')
                        strcat(path, ":");
        } else if (cp = getlog("SYS$LOGIN")) {
                /* getenv("HOME") not used because of vms 3.7 bug */
                strcpy(path, cp);
                cp = path + strlen(path) - 1;
                strcat(path, "models.dir");
                if (access(path, 0) == 0)
                        strcpy(cp, ".models]");
                else
                        strcpy(path, "[]");
        } else
                strcpy(path, "[]");
        if (strcmp(rwflag, "w") == 0) {
                /*
                 * access() doesn't work when directories are specified
                 * by logical names in vms; determine write permission
                 * by trying to create a file.
                 */
                cp = &path[strlen(path)];
                strcat(path, "access.tst");
                if (close(creat(path, 0600)) < 0) {
                        err_midas = EM_NOPERM;
                        *cp = '\0';
                        return(-1);
                } else
                        delete(path);
                *cp = '\0';
        }
#endif
        return(0);
}
```

```
#ifdef	vms
/*
 * getlog - getenv() look-alike routine for VMS logical names
 *
 * Chris Carlson 1nov84
 */

#include <ssdef.h>
#include <descrip.h>

char *
getlog(log)
	char *log;
{
	struct dsc$descriptor_s log_d = {0, DSC$K_DTYPE_T, DSC$K_CLASS_S,
	static char trans[128];
	$DESCRIPTOR(trans_d, trans);
	short int translen;

	log_d.dsc$a_pointer = log;
	log_d.dsc$w_length = strlen(log);
	if (lib$sys_trnlog(&log_d, &translen, &trans_d, 0, 0, 0) == SS$_NORMAL) {
		do {
			log_d.dsc$a_pointer = trans;
			log_d.dsc$w_length = translen;
		} while (lib$sys_trnlog(&log_d, &translen, &trans_d, 0, 0, 0)
			== SS$_NORMAL);
		trans[translen] = '\0';
		return(trans);
	}
	return((char *)0);
}
#endif
```

misc.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/misc.c,v 3.3 84/12/06 13:52:52 arnold
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Mfirst returns true if the given residue sequence number corresponds
 * to the first residue of the stream
 */
#ifndef  f77
mfirst(stno, resseq)
int      stno;
char     *resseq;
#else
mfirst_(fstno, resseq, len)
int      *fstno;
char     *resseq;
long     len;
#endif
{
         register            MDBS    stream;

#ifdef   f77
         stream = &_mstrm[*fstno];
         termin_(resseq, (int) len);
#else
         stream = &_mstrm[stno];
#endif
         return(smatch(resseq, Residue[Reshdr.firstres].seq) == 0);
}

#ifndef  f77
mlink(stno)
int      stno;
#else
mlink_(fstno)
int      *fstno;
#endif
{
         register            MDBS    stream;

#ifdef   f77
         stream = &_mstrm[*fstno];
#else
         stream = &_mstrm[stno];
#endif
         return(Curtpl.linkage);
}

#ifndef  f77
mchief(stno)
int      stno;
#else
mchief_(fstno)
int      *fstno;
#endif
{
         register            MDBS    stream;

#ifdef   f77
         stream = &_mstrm[*fstno];
#else
         stream = &_mstrm[stno];
#endif
         return(Curtpl.chief);
}

#ifndef  f77
maconn(stno, ia, ib)
int      stno;
int      ia, ib;
#else
maconn_(fstno, fia, fib)
int      *fstno;
int      *fia, *fib;
#endif
{
         register            MDBS    stream;
         register            I;
#ifdef   f77
         int      ia, ib;

         stream = &_mstrm[*fstno];
         ia = *fia;
```

```
#else
        ib = *fib;
#endif
        stream = &_mstrm[stno];

        if (Curseq == -1)
                return(-1);
        if (ia > Curtpl.natom || ia < 0) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("maconn");
#endif
                return(-1);
        }
        if (ib > Curtpl.natom || ib < 0) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("maconn");
#endif
                return(-1);
        }
        if (ia == ib)
                return(3);
        for (i = 0; i < Curcon[ia].tocnt; i++)
                if (Curcon[ia].to[i] == ib)
                        return(1);
        for (i = 0; i < Curcon[ib].tocnt; i++)
                if (Curcon[ib].to[i] == ia)
                        return(2);
        return(0);
}

#ifndef f77
mrconn(stno, s1, s2)
int     stno;
char    *s1, *s2;
#else
mrconn_(fstno, s1, s2)
int     *fstno;
char    *s1, *s2;
#endif
{
        register        MDBS    stream;
        register        i;
        int     i1, i2, t1, t2;
#ifdef  f77
        int     stno;
        stno = *fstno;
#endif
        stream = &_mstrm[stno];
        i1 = mbsrch(stno, s1, &t1, &t2);
        i2 = mbsrch(stno, s2, &t1, &t2);
        if (i1 < 0)
                return(-1);
        if (i2 < 0)
                return(-1);
        if (i2 == i1)
                return(3);
        if (Residue[i1].resprev == i2)
                return(2);
        if (Residue[i1].resnext == i2)
                return(1);
        for (i = i1; i != -1; i = Residue[i].resprev)
                if (i == i2)
                        return(5);
        for (i = i1; i != -1; i = Residue[i].resnext)
                if (i == i2)
                        return(4);
        return(-1);
}

#ifndef f77
mchain(stno, index)
int     stno, index;
#else
mchain_(fstno, findex)
int     *fstno, *findex;
#endif
{
        register        MDBS    stream;
#ifdef  f77
        int     index;

        index = *findex - 1;
        stream = &_mstrm[*fstno];
#else
        stream = &_mstrm[stno];
#endif
        if (Curseq == -1)
                return(-1);
        if (index > Curtpl.natom || index < 0) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("mchain");
```

```
#endif
                return(-1);
        }
        return(Curcon[index].cstat & MAINCHAIN);
}

/*
 * Mamap returns the number of atoms connected to atom index
 * and the indices of those atoms in the given array
 */
#ifndef f77
mamap(stno, index, ibuf)
int     stno;
int     index;
#else
mamap_(fstno, findex, ibuf)
int     *fstno;
int     *findex;
#endif
int     ibuf[MAXTO];
{
        register        MDBS    stream;
        register        i, j;
        int     natom;
#ifdef  f77
        int     stno, index;

        stno = *fstno;
        index = *findex - 1;
#endif
        stream = &_mstrm[stno];
        if (Curseq == -1) {                     /* Check for end */
                err_midas = EM_ATEOF;
#ifdef  MDEBUG
                merror("matom");
#endif
                return(-1);
        }
        if (index < 0 || index > Curres.atomcnt) {
                err_midas = EM_OUTOFRANGE;
#ifdef  MDEBUG
                merror("matom");
#endif
                return(-1);
        }
        natom = 0;
        for (i = 0; i < Curres.atomcnt; i++) {
                if (i == index) {
                        for (j = 0; j < Curcon[i].tocnt; j++)
                                ibuf[natom++] = Curcon[i].to[j];
                } else {
                        for (j = 0; j < Curcon[i].tocnt; j++)
                                if (Curcon[i].to[j] == index)
                                        ibuf[natom++] = i;
                }
        }
        return(natom);
}
```

mseek.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mseek.c,v 3.3 84/12/06 13:53:27 arnol
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *

 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif
#include <stdio.h>

#define  STYPE   01              /* Residue sequence or type */
#define  SDIR    02              /* Direction of reads */
#define  SSTART  04              /* Whether to start at one end */
#define  SEOF    010             /* Which end if beginning of file */

/*
 * Mseekr seeks in the database to find the residue which matches the
 * user specification.  Binary search is used if the spec is for a
 * sequence number and linear search is used if it's for a residue type
 */
#ifndef  f77
mseekr(stno, residue, mode)
int     stno;
char    *residue;
int     mode;
#else
mseekr_(fstno, residue, fmode, len)
int     *fstno;
char    *residue;
int     *fmode;
long    len;
#endif
{
        int     save;
        int     i, j;
        register        MDBS    stream;
#ifdef  f77
        int     stno, mode;

        stno = *fstno;
        mode = *fmode;
#endif
        /* Set the direction of search first */
        stream = &_mstrm[stno];
        if ((mode & SDIR) && !(Mode & DIRECTION))
                Mode |= DIRECTION;
        if (!(mode & SDIR) && (Mode & DIRECTION))
                Mode &= ~DIRECTION;
#ifdef  f77
        mflush_(fstno);
#else
        mflush(stno);
#endif

        /* Find out where to start searching */
        save = Curseq;
#ifdef  f77
        termin_(residue, (int) len);
#endif
        if (mode & SSTART)
                Curseq = (mode & SEOF) ? Reshdr.lastres : Reshdr.firstres;
        else
                Curseq = NEXTR(Curseq);

        /* Search by type proceeds linearly, but by sequence
         * goes in binary search.  Note that several templates
         * may have the same name and all match the user's residue */
        if (mode & STYPE) {
                while (Curseq >= 0) {
                        if (!tmatch(Curtpl.restype, residue))
                                break;
                        Curseq = NEXTR(Curseq);
                }
        } else
                Curseq = mbsrch(stno, residue, &i, &j);

        /* If we failed then we restore the pointer */
        if (Curseq == -1) {
                Curseq = save;
                return(-1);
        }

        /* Check if this is the same as where we were before */
        if (Curseq != save)
                Mode &= ~DAT_IN_CORE;
        Mode &= ~ADVANCE;
        return(0);                      /* Done */
}
```

```
/*
 * Mseeka returns the index of the atom 'atomname'.  Binary search is
 * used since the atoms must be in alphabetical order.
 */
#ifndef	f77
mseeka(stno, atomname)
int	stno;
char	*atomname;
#else
mseeka_(fstno, atomname, len)
int	*fstno;
char	*atomname;
long	len;
#endif
{
	int	hi, lo, mid, comp;
	register		MDBS	stream;
#ifdef	f77
	int	stno;

	stno = *fstno;
#endif
	stream = &_mstrm[stno];
	if (Curseq < 0) {
		err_midas = EM_ATEOF;
#ifdef	MDEBUG
		merror("mseeka");
#endif
		return(-1);
	}

	/* Binary search */
#ifdef	f77
	termin_(atomname, (int) len);
#endif
	hi = Curtpl.natom;
	lo = 0;
	mid = (hi + lo) / 2;
	do {
		comp = amatch(Curcon[mid].atname, atomname);
		if (comp == 0)
			return(mid);
		else if (comp > 0)
			hi = mid;
		else
			lo = mid;
		mid = (hi + lo) / 2;
	} while (mid != lo);
	/* Note that mid can only be equal to lo and NOT hi */

	if (amatch(Curcon[mid].atname, atomname) == 0)
		return(mid);
	else
		return(-1);
}

/*
 * Mbsrch does a binary search on 'stream' and attempts to find residue
 * 'resseq'.  The last value looked at and the test result are stored
 * in *'mid' and *'comp' respectively.
 */
mbsrch(stno, resseq, mid, comp)
int	stno;
char	*resseq;
int	*mid, *comp;
{
	register		hi, lo;
	register		MDBS	stream;

	stream = &_mstrm[stno];
	hi = Header.rescnt;
	lo = 0;
	*mid = (hi + lo) / 2;
	*comp = 1;
	if (hi == 0)
		return(-1);
	do {
		*comp = smatch(Residue[*mid].seq, resseq);
		if (*comp == 0)
			return(*mid);
		else if (*comp > 0)
			hi = *mid;
		else
			lo = *mid;
		*mid = (hi + lo) / 2;
	} while (*mid != lo);
	/* Note that *mid can only be equal to lo and NOT hi */

	*comp = smatch(Residue[*mid].seq, resseq);
	if (*comp == 0)
		return(*mid);
	else
		return(-1);
```

mflush.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/mflush.c,v 3.2 84/12/06 13:52:57 arnol
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Mflush forces the in core data of the stream be written out to disk
 */
#ifndef  f77
mflush(stno)
int      stno;
#else
mflush_(fstno)
int      *fstno;
#endif
{
        register            MDBS    stream;
        long        offset;
#ifdef   f77
        int         stno;

        stno = *fstno;
#endif
        stream = &_mstrm[stno];
        if (!(Mode & MFLUSH))                   /* Flush only when necessary */
                return(0);
        GETLONG(offset, Curres.offset);
        if (Floc != offset)                     /* Get to correct offset */
                bseek(Fd, offset, 0);
        bwrite(Data, (int) Reshdr.atomsize, (int) Curres.atomcnt, Fd);
        Floc = btell(Fd);                       /* Reset pointer to new location */
        Mode &= ~MFLUSH;                        /* Need not flush next time */
        return(0);                              /* Done */
}
```

bio.c

```
/* $Header: bio.c,v 3.9 86/07/21 14:54:40 gregc Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 * Release 2.1                          27 Jul 1983
 */

#ifndef  vms
#include <mdbs.h>
extern int          errno;
extern char         *sys_errlist[];
#else
#include "MIDAS$H:mdbs.h"
extern noshare int          errno;
extern noshare char         *sys_errlist[];
#endif
#include <errno.h>
#include <stdio.h>

bopen(file, mode)
char    *file, *mode;
{
        int         rw, f;

        rw = mode[1] == '+';

        switch (*mode) {

        case 'w':
                f = create(file, rw);
                break;

        case 'a':
                if ((f = open(file, rw? 2: 1)) < 0) {
                        if (errno == ENOENT)
                                f = create(file, rw);
                }
                lseek(f, 0L, 2);
                break;

        case 'r':
                f = open(file, rw? 2: 0);
                break;

        default:
                err_midas = EM_ILLMODE;
#ifdef  MDEBUG
                merror("bopen");
#endif
                return(-1);
        }

        if (f == -1) {
                err_midas = errno;
#ifdef  MDEBUG
                merror("bopen");
#endif
        }

        return(f);
}

create(file, rw)
char    *file;
int     rw;
{
        int         f;

#ifndef  vms
        f = creat(file, 0644);
#else
        /*
         * MUST be mode 644 to prevent RMS
         * from zero filling last block.
         * MUST be rmf=stream_lf for lseek(EOF)
         * to work and to prevent insertion
         * of extraneous <CR>'s.
         */
        f = creat(file, 0644, "rfm=stmlf");
#endif
        if (rw && f>=0) {
                close(f);
                f = open(file, 2);
        }
        if (f == -1) {
                err_midas = errno;
#ifdef  MDEBUG
                merror("create");
#endif
        }
        return(f);
}
```

```
bread(data, size, count, fd)
char    *data;
int     size, count;
int     fd;
{
        long    total;
        int     n;

        total = ((long) size) * ((long) count);
#ifdef  CHECKTOOBIG
        if (total > 32000) {
                err_midas = EM_TOOBIG;
#ifdef  MDEBUG
                merror("bread");
#endif
                return(-1);
        }
#endif
        n = read(fd, data, (int) total);
        if (n == -1) {
                err_midas = errno;
#ifdef  MDEBUG
                merror("bread");
#endif
                return(-1);
        }
        return(n / size);
}

bwrite(data, size, count, fd)
char    *data;
int     size, count;
int     fd;
{
        long    total;

        total = ((long) size) * ((long) count);
#ifdef  CHECKTOOBIG
        if (total > 32000) {
                err_midas = EM_TOOBIG;
#ifdef  MDEBUG
                merror("bwrite");
#endif
                return(-1);
        }
#endif
        size = total;
        total = write(fd, data, size);
        if (total == size)
                return(count);
        if (total < 0)
                err_midas = errno;
        else
                err_midas = EINTR;
#ifdef  MDEBUG
        merror("bwrite");
#endif
        return(-1);
}

bclose(fd)
int     fd;
{
        register        i;

        i = close(fd);
        if (i == -1) {
                err_midas = errno;
#ifdef  MDEBUG
                merror("bclose");
#endif
        }
        return(i);
}

long
bseek(fd, loc, type)
int     fd;
long    loc;
int     type;
{
        long    addr;

        addr = lseek(fd, loc, type);
        if (addr == -1) {
                err_midas = errno;
#ifdef  MDEBUG
                merror("bseek");
#endif
        }
        return(addr);
}
```

bio.c

```
long    btell(fd)
int     fd;
{
        return(beeek(fd, 0L, 1));
}

brewind(fd)
int     fd;
{
        return(beeek(fd, 0L, 0));
}

merror(s)
char    *s;
{
        fputs(s, stderr);
        if (err_midas) {
                fputs(": ", stderr);
                fputs(err_midas > 0 ?
                        sys_errlist[err_midas] : midas_errlist[-err_midas],
                        stderr);
        }
        putc('\n', stderr);
        if (nargs() > 1)
                abort();
}

#ifdef  vms
/*
 * Exit() "write around" for funny VMS exit codes.
 * This routine is included here because bio is
 * included by lots of programs.
 */
#undef  exit
#include <ssdef.h>

exit_(n)
{
        exit(n ? SS$_ABORT : SS$_NORMAL);
}
#endif
```

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/match.c,v 3.2 84/12/06 13:52:43 arnold
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

/*
 * Smatch matches the strings as sequence numbers
 */
smatch(s1, s2)
register        char    *s1, *s2;
{
        register        i;

        if (*s1 == '*' || *s2 == '*')
                return(0);
        for (i = 0; i < RES_SEQ_SIZE; i++,s1++,s2++) {
                if (*s1 == '?' || *s2 == '?')
                        continue;
                if (*s1 != *s2)
                        return(*s1 - *s2);
                if (*s1 == '\0')
                        return(0);
        }
        return(0);
}

/*
 * Tmatch matches the strings as residue types
 */
tmatch(s1, s2)
register        char    *s1, *s2;
{
        register        i;
        register        hasmagic;

        if (*s1 == '*' || *s2 == '*')
                return(0);
        hasmagic = FALSE;
        for (i = 0; i < RES_TYPE_SIZE; i++,s1++,s2++) {
                if (*s1 == '?' || *s2 == '?') {
                        hasmagic = TRUE;
                        continue;
                }
                if (*s1 != *s2) {
                        if (hasmagic && smatch(s1, s2) == 0)
                                return(0);
                        else
                                return(*s1 - *s2);
                }
                if (*s1 == '\0')
                        return(0);
        }
        return(0);
}

/*
 * Amatch matches the strings as atom names
 */
amatch(s1, s2)
register        char    *s1, *s2;
{
        register        i;

        for (i = 0; i < AT_NAME_SIZE; i++,s1++,s2++) {
                if (*s1 == '\0' || *s2 == '\0')
                        return(*s1 - *s2);
                if (*s1 == '?' || *s2 == '?')
                        continue;
                if (*s1 != *s2)
                        return(*s1 - *s2);
        }
        return(0);
}
```

gettemp.c

```
/* $Header: /usr/src/local/midas/src/mdbs/RCS/gettemp.c,v 3.3 84/12/06 13:53:57 am
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#ifndef  vms
#include <mdbs.h>
#else
#include "MIDAS$H:mdbs.h"
#endif

static   char    tfile[MBASELEN];
static   int     init;

char     *
gettemp(suffix)
char     *suffix;
{
         register          char     *s, *dot;
         unsigned          pid;

         if (!init) {
                  init = TRUE;
                  strcpy(tfile, MBASE);
                  pid = getpid();
                  s = &tfile[strlen(tfile) - 1];
                  *s = 'a';
                  while (*--s == 'X') {
                           *s = (pid%10) + '0';
                           pid /= 10;
                  }
         }
         s = &tfile[strlen(tfile) - 1];
         dot = s + 1;
         strcat(tfile, suffix);
         while (access(tfile, 0) != -1) {
                  if (*s == 'z') {
                           err_midas = EM_NOTEMP;
#ifdef   MDEBUG
                           merror("gettemp");
#endif
                           return("/");
                  }
                  *s = (*s) + 1;
         }
         *dot = '\0';
         return(tfile);
}
```

block.c

```
/* $Header: /usr/src/local/mldas/src/mdbs/RCS/block.c,v 3.2 85/07/12 13:39:27 arnold
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          27 Jul 1983
 */

#ifdef    vms
DELIBERATE SYNTAX ERROR
(you should be using the file "block.mar" instead of this one
#endif

blkcpy(from, to, size)
char *from, *to;
int size;
{
#ifdef    vax
#ifdef    lint
        from = from, to = to, size = size;
#endif
        asm("     movc3     12(ap),*4(ap),*8(ap)");
#else
#ifdef    SYSV
        memcpy(to, from, size);
#else
        register int *f = (int *)from, *t = (int *)to;
        register count = size/sizeof(int);

        do
                *t++ = *f++;
        while(--count);
#endif
#endif
}

blkclr(buf, size)
char *buf;
int size;
{
#ifdef    vax
#ifdef    lint
        buf = buf, size = size;
#endif
        asm("movc5        $0,(r0),$0,8(ap),*4(ap)");
#else
#ifdef    SYSV
        memset(buf, 0, size);
#else
        register int *p = (int *)buf;
        register count = size/sizeof(int);

        do
                *p++ = 0;
        while (--count);
#endif
#endif
}
```

mdata.c

```
/* $Header: mdata.c,v 1.6 86/07/31 13:05:00 conrad Exp $ */
#ifndef VMS
#include <mdbs.h>
#else
#include "midas$h:mdbs.h"
#endif

/* Error number and messages */

int     err_midas;                          /* Mdbs error number */

char    *midas_errlist[] = {                /* Error messages */
        "No error",
        "Illegal mode",
        "Argument too large",
        "No temporary filename",
        "Invalid stream number",
        "Permission denied",
        "Index out of range",
        "Too many open streams",
        "Out of memory",
        "File type mismatch",
        "Empty database",
        "Illegal operation",
        "At end of database",
};

struct  stream_def      _mstrm[MAXSTREAM];  /* Streams */
#ifndef m68000
union   long_def        _longconv;          /* Conversion union */
#endif
```

/usr/src/local/midas/h/midas.h

```
/* $Header: midas.h,v 3.15 86/08/04 18:25:43 arnold Exp $ */
/*
 * midas.h
 *
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                  9 Apr 1982
 * Release 2.0                  13 May 1983
 * Release 2.1                  27 Jul 1983
 * Release 2.2                  12 Oct 1983
 */

/*
 * General constant definition
 */
#define  MAXMOD          MAXSTREAM       /* Maximum number of models */
#if      GRAFENG != IRIS
#define  MAXREPLY        5               /* Maximum # of replies */
#else
#define  MAXREPLY        14              /* Maximum # of replies */
#endif
#define  MAXINROT        8               /* Maximum # of internal rotations */
#define  MAXDIST         10              /* Maximum # of distance calcs */
#define  MAXANGLE        2               /* Maximum # of angle calcs */
#define  MAXPSOBS        (MAXINROT+MAXMOD+6)  /* Maximum # PS memory
                                         /* If you change this constant, you'll
                                          * have to recompile libgu, intr,
                                          * editor, and menu */
#define  CHEATSZ         512             /* sizeof "cheat" buffer */
#define  MOLNMSZ         40              /* Size of name of molecule */

#if      GRAFENG == PS2 || GRAFENG == MPS
#define  ATOPS           100.0           /* Angstroms to Picture System unit
#define  MAPCRD(z)       (_mapcrd = (z) * ATOPS, \
                          (_mapcrd < 0) ? (_mapcrd - 0.5) : (_mapcrd + 0.5))
#define  UNMAPCRD(z)     ((z) / ATOPS)
#define  MAPANGLE(a)     ((ps_t) (a * K16K.0 / 90.0))
#define  UNMAPANGLE(a)   (a * 90.0 / K16K.0)
#endif
#if      GRAFENG == PS300
#define  ATOPS           ((ps_t) 1)
#define  MAPCRD(z)       (z)
#define  UNMAPCRD(z)     (z)
#define  MAPANGLE(a)     (a)
#define  UNMAPANGLE(a)   (a)
#endif
#if      GRAFENG == IRIS
#define  ATOPS           ((ps_t) 1)
#define  MAPCRD(z)       (z)
#define  UNMAPCRD(z)     (z)
#define  MAPANGLE(a)     ((Angle) (a * 10))
#define  UNMAPANGLE(a)   (a / 10.0)
#endif

/*
 * The atom status bit definitions
 */
#define  SHOWBIT         01      /* This atom to be shown during display */
#define  BREAKBIT        02      /* MOVETO this atom */
#define  LABELBIT        04      /* This atom is to be labelled */
#define  DISTBIT         010     /* This atom needed for dist. calculation */
#define  WATCHBIT        020     /* This atom is being 'watched' */
#define  ROTBIT          040     /* This atom needed for forward rotation */
#define  ANGLEBIT        0100    /* This atom needed for angle calc. */
#define  SURFBIT         0200    /* Display the surface of this atom */
#define  NORLBIT         0400    /* No residue label on this atom */
#define  VDWBIT          01000   /* van der Waals surface bit */
#define  BONDBIT         020000  /* Atom is part of SS or HH bond */
#define  STARTBIT        040000  /* This atom starts a new chain */
#define  EXISTBIT        0100000 /* This atom is real (not place holder) */

/*
 * Mseekr modes
 */
#define  FBSEEK          04      /* Forward binary seek */
#define  BBSEEK          06      /* Backward binary seek */
#define  FLSEEK          05      /* Forward linear seek */
#define  BLSEEK          017     /* Backward linear seek */
#define  CFLSEEK         01      /* Continuing forward linear seek */
#define  CBLSEEK         03      /* Continuing backward linear seek */

/*
 * Electrostatic potential constants
 */
#define  ESPRNG          256     /* Range of normalized potentials */
#define  ESPMID          128     /* 1/2 ESPRNG */
#define  ESPMIN          -128    /* Minimum value in range */
#define  ESPMAX          127     /* Maximum value in range */
#define  ESPMAXPTS       512     /* Maximum number of pts per atom */
#define  SIGNORE         0200    /* We steal a bit from the color byte for
                                  * surfaces to allow display of dots of
                                  * selective potentials */
```

/usr/src/local/midas/h/midas.h

```
#if     GRAFENG == IRIS
#ifdef  color
/*
 * On some IRIS systems, color is a macro, not a function.  This causes
 * complaints, since it is also a structure field name in struct atom_def.
 * The easiest patch is to undefine the macro, and provide a interface
 * in libguiris.
 */
#undef  color
#define color   i_color
#endif
#endif  IRIS

/*
 * Atom data structure definition
 */
struct  atom_def            {
        float   x, y, z;                        /* Atom coordinates */
        float   tempfac;                        /* Temperature factor */
        short   status;                         /* Atom status (16 bits) */
        char    color, scolor, lcolor, pad1;    /* Colors (bond, label) */
        short   pad2;                           /* Force size to 4n bytes *
};

/*
 * Surface data structure definition
 */
struct  surf_def            {
        short   s_nentry;                       /* Number of entries in the
                                                 * surface description tabl
        long_t  s_addr;                         /* Surface points address
};

/*
 * Surface file header definition
 */
struct  srfhdr_def          {
        float   sh_minesp, sh_maxesp;           /* Min/max potentials in d
};

/*
 * Surface description table definition
 */
struct  srftab_def          {
        short   st_npts;                        /* Number of points */
        char    st_pot;                         /* Potential of these points
        char    st_color;                       /* Color of these points */
};

/*
 * Bond data structure definition
 */
struct  bond_def            {
        char    b_resseq[2][RES_SEQ_SIZE];      /* Residue sequence names */
        char    b_atname[2][AT_NAME_SIZE];      /* Atom names */
        long    b_pad;                          /* Pad to same size as
                                                 * ... atom_def */
};

double  _mapcrd;

#ifdef  vms
/* common defines for pipe I/O (a VMESS kludge!) */
#define TO_CHILD        96
#define FROM_CHILD      97
#define TO_PARENT       98
#define FROM_PARENT     99
#endif
```

editor.h

```
/* $Header: editor.h,v 3.28 86/09/25 11:17:30 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 * Release 2.1                           5 Jul 1983
 * Release 2.2                          28 Jul 1983
 * Release 2.3                          18 Aug 1983
 */

/*
 * Compilation control flags.
 *
 * #define SYSTEM - indicates "released" version of executables
 * #define EDITONLY - editor does data base operations only - no pictures
 * #define DRAWPIC - editor generates static pictures - intr module not used
 * NOTE: EDITONLY & DRAWPIC are mutually exclusive - do not define both!
 * #define ONEPROC - editor and storage module are one process
 * #define DOSTATS - collect command statistics
 */

#ifndef  EDITONLY
#ifndef  vms
/* The machine type is selected in the Makefile */
#else
#include "MIDAS$H:machdef.h"
#endif
#endif   EDITONLY

#ifndef  vms

#include <mdbs.h>
#include <midas.h>

#if      GRAFENG == PS2
#include <ps.h>
#endif   PS2
#if      GRAFENG == PS300
#include <ps300defs.h>
#endif   PS300
#if      GRAFENG == MPS
#include <mps.h>
#endif   MPS
#if GRAFENG == IRIS
#include <irisdefs.h>
#ifdef   EMULATE
#include <fakegl.h>
#endif   EMULATE
#endif IRIS

#else    vms

#include "MIDAS$H:mdbs.h"
#include "MIDAS$H:midas.h"

#if      GRAFENG == PS2
DELIBERATE SYNTAX ERROR (Do not know what to include yet)
#endif   PS2
#if      GRAFENG == MPS
#include "MIDAS$H:ps.h"
#endif   MPS
#if      GRAFENG == PS300
#include "MIDAS$H:ps300defs.h"
#endif   PS300
#if GRAFENG == IRIS
#include "MIDAS$H:irisdefs.h"
#endif IRIS

#endif   vms

#include <stdio.h>

#ifndef  vms
#ifndef IRIS
#define  DOSTATS
#endif
#else    vms
#define  ONEPROC          /* editor and storage module are one */
#define  sscanf   vmsscanf        /* Yet another VMS C2.0 bug */
/* #define          DRAWPIC         /* have editor draw pretty pictures itself */
#endif   vms

#define  MAXSTACK         10                          /* Input stack depth */
#define  MAXMSTCK         5                           /* Max nested inrot */
#define  MAXALIAS         20                          /* Number of aliases */
#define  MAXVALLEN        256                         /* Maximum token length */
#define  BUFLEN           256                         /* Temporary buffer size */
#define  PIPESIZE         4096                        /* Size of pipe buffer */
#define  ITEMSIZE         4                           /* Offset per inst */
#define  ONBITS  (SHOWBIT | SURFBIT | LABELBIT | VDWBIT)

/* Lexical codes of input tokens */
#define  NOMORECOM        0
```

```
#define NAME            1
#define SEMICOLON       2
#define TILDA           3
#define COMMA           4
#define AND             5
#define MODEL           6
#define RESIDUE         7
#define ATOM            8
#define COUNT           9
#define HYPHEN          10
#define EXCLPT          11
#define EOC             12
#define GCOMWORD        13
#define SCOMWORD        14
#define NCOMWORD        15
#define TFRANGE         16
#define ESPRANGE        17

/* Internal command codes */
#define DISPLAY 0               /* Needs bits */
#define CHAIN   1               /* Needs bits */
#define LABEL   2               /* Needs bits */
#define SHOW    3               /* Needs bits */
#define SURFACE 4               /* Needs bits */
#define PCOLOR  5
#define ALIAS   7
#define PREAD   8
#define STOP    9
#define OPEN    10
#define CHDIR   12
#define SAVE    13
#define FINROT  14              /* Needs bits */
#define BINROT  15              /* Needs bits */
#define DIST    16              /* Needs bits */
#define MATCH   17
#define ANGLE   18
#define REVERSE 19              /* Needs bits */
#define GETCRD  20
#define SETCOM  21
#define FIXREV  22              /* Needs bits */
#define RLABEL  23              /* Needs bits */
#define VDW     24              /* Needs bits */
#define VDWOPT  25
#define LINK    26              /* Needs bits */
#define REDRAW  27
#define HALFBND 28
#define ALIGN   29

#if     GRAFENG == IRIS
#define CPK     30
#else
#define PICK    30
#endif
#define ADDGRP  31
#define SWAPAA  32
#define SWAPNUC 33
#define ADDAA   34
#define ADDNUC  35
#define DELGRP  36
#if     GRAFENG == IRIS
#define MAPCOLORS       37
#endif

/* Modes of operation for gettoken() */
#define NORMAL  0               /* Normal */
#define KEYWRD  1               /* Identify keyword */
#define TILSPC  2               /* Read till white space */

/* Debugging bits */
#ifdef  DEBUG
#define D_PG            01
#define LD_PG           02
#define D_LEX           04
#define LD_LEX          010
#define D_PARSE         020
#define LD_PARSE        040
#define D_EDIT          0100
#define LD_EDIT         0200
#endif  DEBUG

/* Command codes to MIDAS-STORE */
#define S_QUIT          0
#define S_APPEND        1
#define S_SREPLACE      2
#define S_BREPLACE      3
#define S_DRAW          4
#define S_DELETE        5
#define S_REPLACE       6
#define S_OBJECT        7
#define S_SCHDIR        8
#define S_LREPLACE      9
#define S_HLFBND        10
#define S_IDENTRES      11
#define S_IDENTATM      12
#define S_ADD           13
```

```
#define  S_INSERT        14
#define  S_RENAME        15
#define  S_REMOVE        16
#define  S_BACKWARD      17
#define  S_FORWARD       18
#define  S_GETLEVEL      19
#define  S_SETLEVEL      20
#define  S_FINDNEXT      21

/* Modes of rotation and distance */
#define  INACTIVE        00
#define  ACTIVE          01
#define  HOLDALL         036
#define  HOLD(x)         (02 << (x))
#define  BACKWRD         040
#define  MCHAIN          0100
#define  REMAKEMAT       0200
#if      GRAFENG == PS300
#define  SENDCONN        0400

#endif

/* Spacifier status flags */
#define  DUPRES          01
#define  DUPMOD          02
#define  HASLTF          04
#define  HASHTF          010
#define  HASLESP         020
#define  HASHESP         040

/* Modes of coloring */
#define  BCOLOR          01
#define  SCOLOR          02
#define  LCOLOR          04
#define  VCOLOR          010
#define  ALLCOLOR        017

/* Flags associated with models */
#define  PSOB            01
#define  SETCM           02
#define  SETCNT          04
#define  SETRNG          010

/* Modes of "first mode" when drawing a residue */
#define  FM_NONE         0
#define  FM_MOVE         1
#define  FM_LINE         2

/* Rotation active */

/* Required atom not */
/* Backward rotation */
/* Mainchain rotation */
/* Remake matrix */

/* Send connection to func
 * network */

/* Vaious macro definitions */
#define  CURLINE cstck[stptr].c_input.c_line
#define  CURSTRM cstck[stptr].c_input.c_strm
#define  CURNDX  cstck[stptr].c_index
#define  NEXTC   (CURNDX < 0) ? getc(CURSTRM) : CURLINE [ CURNDX++ ]
#define  talloc(n, s)       (struct token_def *) alloc(n, s)
#define  walloc(n, s)       (struct wordrange_def *) alloc(n, s)
#define  salloc(n, s)       (struct spec_def *) alloc(n, s)
#define  spalloc(n, s)      (struct speclist_def *) alloc(n, s)
#define  challoc(n, s)      (struct chain_def *) alloc(n, s)

struct   alias_def          {
         char     *a_name;                    /* Alias name */
         char     *a_line;                    /* Alias expansion */
}        atab[MAXALIAS];

struct   wordrange_def      {
         char     *w_first;                   /* First word of range */
         char     *w_last;                    /* Last word of range */
         struct   wordrange_def    *w_next;
};

struct   speclist_def       {
         int      sp_cnt;                     /* Count of this range */
         struct   wordrange_def    *sp_range;/* Pointer to range */
};

struct   spec_def           {
         int      s_stat;                     /* Various flags */
         float    s_tfmin, s_tfmax;           /* Temp factor range */
         float    s_espmin, s_espmax;         /* ESP range */
         struct   speclist_def     *s_model; /* Model range */
         struct   speclist_def     *s_residue;/* Residue range */
         struct   speclist_def     *s_atom;  /* Atom range */
         struct   spec_def         *s_next;
};

struct   token_def          {
         int      t_type;                     /* Token lexical code */
         char     *t_value;                   /* Token string */
         struct   token_def        *t_next;
}        token;

struct   cstack_def         {
         int      c_index;                    /* Buffer location */
         union    {
```

```
                char    *c_line;        /* Input buffer (alias) */
                FILE    *c_stm;         /* ... or input stream */
        } c_input;
} cstck[MAXSTACK];

struct  command_def     {
        char    com_type;               /* Command type */
        char    com_num;                /* Command number */
        char    *com_name;              /* Command name */
        int     com_count;              /* Command usecnt */
};

struct  chain_def       {               /* Information for one chai
        int     ch_rotb;                /* First backward rotation
        int     ch_rotf;                /* First forward rotation */
        char    ch_start[RES_SEQ_SIZE+1];       /* Name of first residue */
        struct  chain_def       *ch_next;       /* Pointer to next chain */
};

struct  mdbs_def        {
        char    *m_fname;               /* File name */
        char    *m_aname;               /* Alternate file name */
        int     m_flags;                /* Various flags */
        int     m_fd;                   /* Coordinate db descripto
        int     s_fd;                   /* Surface db descriptor */
        int     sd_fd;                  /* Surface data descriptor
        int     m_active;               /* Number of active atoms
        int     m_nchain;               /* Number of chains */
        float   m_com[3];               /* Center of mass total */
        float   m_min[3];               /* Minimum coordinates */
        float   m_max[3];               /* Maximum coordinates *
        float   m_size;                 /* Greatest range from cof
        float   m_espmin, m_espmax;     /* Min/max ESP values */
        struct  chain_def       *m_chain;       /* Pointer to chains */
} mdb[MAXMOD];

#ifndef EDITONLY
struct  input_def       {
        char    i_type;                 /* Command code */
        char    i_molnum;               /* Model number */
        char    i_name[RES_SEQ_SIZE+1];         /* Residue sequence nam
        char    i_altname[RES_SEQ_SIZE+1];      /* Alternate name */
        int     i_size;                 /* Size of buffer */
        int     i_ndata;                /* Number of data */
        char    i_newchain;             /* New chain starting */
        char    i_firstmode;            /* No atom displayed */
#if     GRAFENG == PS300
        char    i_halfbond;             /* In halfbond mode ? */
#endif
        ps_t    i_x, i_y, i_z;          /* Coordinate of first atom */
        ps_t    i_lx, i_ly, i_lz;       /* Coordinate of last atom */
        char    i_color, i_lcolor;      /* First and last BOND color */
};

struct  invrot_def      {
        int     r_type;                 /* Type of rotation */
        int     r_linkf;                /* Link to next rotation */
        int     r_linkb;                /* Link to last rotation */
        char    r_natom;                /* Number of atoms specified */
        char    r_model;                /* Model number of rotation */
        char    r_resseq[4][RES_SEQ_SIZE+1];    /* Residue of rotation */
        char    r_atom[4][AT_NAME_SIZE+1];      /* Atoms of rotation */
        angle_t r_initangle;            /* Initial torsion angle */
        angle_t r_rotangle;             /* Actual rotation angle */
        double  r_loc[4][3];            /* Coordinates */
        ps_t    r_mat[4][4], r_invmat[4][4];    /* Matrices to use */
#if     GRAFENG == PS300
        char    *r_locname[2];          /* Name of rotation matrix */
        int     r_nloc;
#endif
};

struct  dist_def        {
        char    d_type;                 /* Distance active ? */
        char    d_model[2];             /* Model number of distance */
        char    d_resseq[2][RES_SEQ_SIZE+1];    /* Residues involved */
        char    d_atom[2][AT_NAME_SIZE+1];      /* Atoms involved */
        int     d_obloc[2];             /* Location in residue */
        int     d_psloc[2];             /* Location of coordinates */
#if     GRAFENG == IRIS
        ps_t    d_x[2], d_y[2], d_z[2];         /* Coordinates of atoms */
#endif
#if     GRAFENG == PS300
        int     d_vecnum[2];
        char    *d_vecname[2];
#endif
};

struct  angle_def       {
        char    a_type;                 /* Distance active ? */
        char    a_natom;                /* Number of atoms specified */
        char    a_model[4];             /* Model number of angle */
        char    a_resseq[4][RES_SEQ_SIZE+1];    /* Residues involved */
        char    a_atom[4][AT_NAME_SIZE+1];      /* Atoms involved */
```

```
        int     a_obloc[4];
        int     a_psloc[4];                     /* Location in residue */
#if     GRAFENG == IRIS                         /* Location of coordinates
        ps_t    a_x[4], a_y[4], a_z[4];         /* Coordinates of atoms */
#endif
};
#endif  EDITONLY

struct  bc_def  {
        float   b_x[2];                         /* Coordinates of hydroge
        float   b_y[2];                         /* ... or disulfide-bonded */
        float   b_z[2];                         /* ... atoms. */
        char    b_res[2][RES_SEQ_SIZE];         /* Residue sequence num
        char    b_atom[2][AT_NAME_SIZE];        /* Atom names */
        char    b_disp[2];                      /* Displayed ? */
        char    b_color;                        /* Color of bond */
};

#if     GRAFENG == MPS
struct  psobs   {
        struct  udl_preamble    preamble;
        struct  udl_entry       uentry[MAXPSOBS];
}       *psobs;
#endif

struct  spec_def        *spec();
struct  spec_def        *specs();
struct  token_def       *symbol();
struct  token_def       *symbols();
struct  wordrange_def   *wrange();
struct  wordrange_def   *wranges();
char    *word();
char    *alloc();
char    *delims();

char    *calloc(), *malloc();
char    *sprintf(), *strcpy(), *strcat();

# if GRAFENG == IRIS
Angle   to_Angle();
Object  numobj();
# endif IRIS

int     stptr;                                  /* Input stack depth */
int     nalias;                                 /* Number of aliases */
int     nbond[MAXMOD];                          /* Number of SS/HH bond
int     notcom;                                 /* ~ prefixed command */

int     curcom;                                 /* Command index */
int     comnum;                                 /* Command number */
#ifndef ONEPROC
int     store_fd[2];                            /* Pipe to MIDAS-STORE */
int     store_pid;                              /* Store process id */
#endif  ONEPROC
#ifdef  DRAWPIC
ps_t    wr, wl, wu, wd, wh, wy;                 /* Window values */
#endif  DRAWPIC
#ifdef  DEBUG
char    debug;                                  /* Debugging mode */
#endif  DEBUG
char    anychg;                                 /* Has picture changed */
char    verbose;                                /* Verbose mode */
char    *all, *nothing;                         /* Default values */
char    buf[BUFLEN];                            /* Output temp buffer */
char    isterm;                                 /* Is input a terminal */
char    nmol;                                   /* Number of molecules */
char    nrot;                                   /* Number of rotations */
char    ndist;                                  /* Number of distances */
char    nangle;                                 /* Number of angles */
#if     GRAFENG == PS300
char    newdist;
#endif
#ifndef EDITONLY
char    nreplies;                               /* Number of replies */
char    repbuf[MAXREPLY][BUFLEN];               /* Reply buffer */
struct  inrot_def       rotation[MAXINROT];     /* Rotation data */
struct  dist_def        distance[MAXDIST];      /* Distance data */
struct  angle_def       angles[MAXANGLE];       /* Angle data */
#endif  EDITONLY
float   maxgtf, mingtf;                         /* Global temp factor range */
float   maxgesp, mingesp;                       /* Global ESP range */
struct  bc_def          *bonds[MAXMOD];         /* SS/HH bond data */
extern  struct  command_def     comtab[];       /* Command table */
extern  int     com_bits[][2][4];               /* Bits to manipulate */
```

editor.c

```
/* $Header: editor.c,v 3.23 86/06/09 19:04:23 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          22 Jul 1983
 * Release 2.2                          28 Jul 1983
 */

char    id[]    =       "MIDAS EDITOR";

#include <signal.h>
#ifndef vms
#include <grafeng.h>
#else
#include "MIDAS$H:machdef.h"
#endif

#if     GRAFENG == IRIS
#include "iriscfs.h"
#include <device.h>

# undef REDRAW/* device.h defines a different REDRAW than editor.h */
#endif
#include "editor.h"

#if     GRAFENG == MPS
int     extdmem = FALSE;        /* default is to use non-refresh partition */

mpserr(errbuf)
short   *errbuf;
{
        eprint(errbuf);
        fprintf(stderr, "\n MODULE: %s\n", id);
        fprintf(stderr, " Aborting...\n");
        abort();
}
#endif  MPS

#if     GRAFENG == PS300
char    *usedev;
#endif

main(argc, argv)
int     argc;
char    *argv[];
{
        register        i, j;
        int     stop();
#ifndef EDITONLY
#if     GRAFENG == PS2 || GRAFENG == MPS
        int     psmems;
#endif
        extern  int     pstmp[150];
#endif
#ifdef  DRAWPIC
        int     matnam, count = 0;
#if     GRAFENG == PS2
        struct  psobs   psb[MAXPSOBS];
#endif
#if     GRAFENG == MPS
        struct  psobs   psb;
#endif
#endif  DRAWPIC
#if     GRAFENG == PS300
        int dnum=0, dtype=0;
#endif

        /* Initialize variables and process flags */

#ifndef EDITONLY
#if     GRAFENG == PS2 || GRAFENG == MPS
        psmems = FALSE;
#endif
#endif

        stptr = 0;
        CURNDX = -1;
        CURSTRM = stdin;
        nalias = 0;
        all = "*";
        nothing = "$";
#ifndef vms
        isterm = isatty(0);
#else
        {
                char    buf[128];

                isterm = (getname(FROM_PARENT, buf) == -1);
        }
#endif
        for (i = 1; i < argc; i++)
                if (argv[i][0] == '-')
```

```
                for (j = 1; argv[i][j] != '\0'; j++)
                        switch (argv[i][j]) {
#ifdef  DEBUG
                        case 'E':
                                debug |= LD_EDIT;
                        case 'e':
                                debug |= D_EDIT;
                                break;

                        case 'G':
                                debug |= LD_PG;
                        case 'g':
                                debug |= D_PG;
                                break;

                        case 'L':
                                debug |= LD_LEX;
                        case 'l':
                                debug |= D_LEX;
                                break;

                        case 'P':
                                debug |= LD_PARSE;
                        case 'p':
                                debug |= D_PARSE;
                                break;

#if     GRAFENG == PS300
                        case 'T':
                                pstraceon();
                                break;

#endif
#endif  DEBUG
#ifndef EDITONLY
#if     GRAFENG == PS2 || GRAFENG == MPS
                        case '?':
                                psmems = TRUE;
                                break;

#endif
#if     GRAFENG == MPS
                        case 'x':
                                extdmem = TRUE;
                                break;

#endif
#if     GRAFENG == PS300
                        case 'X':
                                /* subprocess device info */
                                dnum = atoi(argv[i+1]);
                                dtype = atoi(argv[i+2]);
                                break;

                        case 't':
                                /* alternate device name */
                                usedev = argv[i+1];
                                break;

#endif
#endif  EDITONLY
                        case 'v':
                                verbose = TRUE;
                                break;

                        default:
                                fprintf(stderr, "'%c' ???\n",
                                        argv[i][j]);
                                break;
                        }
        l_init();               /* initialize lexical analyzer */

#ifdef  DRAWPIC
#if     GRAFENG == IRIS
        ginit();
        /*
         * This dummy command is here because the IRIS terminal traps
         * the signals itself when the first graphics command is
         * given.  This command will therefore force that trap to be
         * set up before we set up ours, making ours the effective
         * one.
         */
        color(BLACK);
#endif
#endif

        (void) signal(SIGTERM, stop);
        (void) signal(SIGHUP, stop);
#ifdef EDITONLY
        (void) signal(SIGINT, stop);
#else
#if     GRAFENG == PS2 || GRAFENG == MPS
        (void) signal(SIGINT, SIG_IGN);
        if (ipsmems) {
                fprintf(stderr, "editor: '-?' flag missing\n");
#ifdef  vms
                fprintf(stderr, "crtlib.olb/yfork.exe version mismatch?\n");
                fprintf(stderr, "(try relinking intr and editor modules)\n");
#endif
```

```
        getcomnum();                    /* Initialize rotation handler */
#ifdef  vms
        if (listerm) {
                char    buf[80];

                /* Set up standard pipe environment */

                if (close(FROM_CHILD) < 0) goto bad;
                if (close(TO_CHILD) < 0) goto bad;

                if (close(0) < 0) goto bad;
                if (dup2(FROM_PARENT, 0) < 0) goto bad;
                if (close(FROM_PARENT) < 0) goto bad;

                if (close(1) < 0) goto bad;
                if (dup2(TO_PARENT, 1) < 0) goto bad;
                if (close(TO_PARENT) < 0) {
                  bad:
                        fprintf(stderr, "EDITOR: can't set up pipes\n");
                        perror("pipe setup");
                        exit(1);
                }

                /* Apparently the first read on a pipe returns
                 * NULL.  So we consume that here instead of in
                 * the lexical analyzer */
                fgets(buf, sizeof buf, stdin);
        }
#endif  vms

#ifndef DRAWPIC
        setuppe();
#endif

#if     GRAFENG == PS300
        if (dnum != 0)
                psinit((char *)0, dnum, dtype);
        else
                psinit(usedev);
#ifdef  vms
        /* Need to call color_hook() to force the linker to
         * load in the PS300 color map module from the library */
#endif  PS300

#if     GRAFENG == MPS
        if (extdmem)
                mpinit(&0, &1);
        else
                mpinit(&NRMEMSZ);
        einit(mpserr);

#ifdef  DRAWPIC
        psobs = &psb;
#else
        /*
         * Have already mapped the UDL structure into our
         * address space, but need to synchronize the
         * internal mpuntran tables as well.
         */
        getps();
#endif


        /*
         * The following is the only call to ucontx().
         * This means that we are always in the MPSCTX
         * context and hence never need to recall ucontx().
         */
        if (extdmem)
                ucontx(&MAXPSOBS, psobs, &MPSCTX);
        else
                ucontx(&MAXPSOBS, psobs, &NORFCTX);
#ifndef DRAWPIC
        putps();                /* restore context for intr process */
#endif
#endif  MPS

#ifndef DRAWPIC
        if (!listerm) {
#if     GRAFENG == IRIS
                fileno(stdout) = IRIS_STDOUT;
#endif
                printf("SYNC\n");               /* sync up with intr */
                fflush(stdout);
        }
```

```
#else   DRAWPIC
#if     GRAFENG == PS2
        psinit();
        BUFRPS(150);
        setps((psaddr_t) MAXPSMEM, MAXPSOBS, psb);
        vwport(-2048, 2047, -2048, 2047, 255, 0);
        huesat(YELLOW);
        charsz(3, 0);
        setbase();

        /* Set up the initial matrices as identity matrices. */
        for (i = 0; i < MAXMOD; i++) {
                matnam = ('m' << 8) | ('0' + i);
                maksps(matnam, &count);
                tran(0, 0, 0);
                stopps();
        }
        for (i = 0; i < MAXINROT; i++) {
                matnam = ('r' << 8) | ('0' + i);
                maksps(matnam, &count);
                tran(0, 0, 0);
                stopps();
        }
        for (i = 0; i < 2; i++) {
                matnam = ('g' << 8) | ('0' + i);
                maksps(matnam, &count);
                tran(0, 0, 0);
                stopps();
        }
        makeps('pc', &count);
        entrype('bd');
        tran(0, 0, 0);                          /* Make sure there's something ther
        stopps();
#endif  PS2
#if     GRAFENG == MPS
        sdbuff();
        vbound(&-2048, &2047, &-2048, &2047);
        vinten(&63, &0);
        lspeed(&1);
        lcolor(&YELLOW);
        csize(&3);
        setbase();

        /* Set up the initial matrices as identity matrices */
        for (i = 0; i < MAXMOD; i++) {
                matnam = ('m' << 8) | ('0' + i);
                ubegin(&matnam, &count);
                        ttran(&0, &0, &0);
                        uend();
        }
        for (i = 0; i < MAXINROT; i++) {
                matnam = ('r' << 8) | ('0' + i);
                ubegin(&matnam, &count);
                ttran(&0, &0, &0);
                uend();
        }
        for (i = 0; i < 2; i++) {
                matnam = ('g' << 8) | ('0' + i);
                ubegin(&matnam, &count);
                ttran(&0, &0, &0);
                uend();
        }
        /* Define initial UDL */
        ubegin(&'pc', &count);
        ttran(&0, &0, &0);
        uend();
        ubegin(&'bd', &count);
        ttran(&0, &0, &0);
        uend();
#endif  MPS
#if     GRAFENG == IRIS
        choose_colors();
        iris_colors();
        pagecolor(ir_colornum(BLUE));
        textcolor(ir_colornum(WHITE));
        unqdevice(KEYBD);
        qdevice(MOUSE1);
        qdevice(MOUSE2);
        qdevice(MOUSE3);
        viewport(0, XMAXSCREEN, 0, YMAXSCREEN);

        /* Set up the initial matrices as identity matrices. */
        for (i = 0; i < MAXMOD; i++) {
                matnam = ('m' << 8) | ('0' + i);
                makeobj((Object) matnam);
                        translate(0.0, 0.0, 0.0);
                closeobj();
        }
        for (i = 0; i < MAXINROT; i++) {
                matnam = ('r' << 8) | ('0' + i);
                makeobj((Object) matnam);
                        translate(0.0, 0.0, 0.0);
                closeobj();
        }
```

```
                for (i = 0; i < 2; i++) {
                        matnam = ('g' << 8) | ('0' + i);
                        makeobj((Object) matnam);
                                translate(0.0, 0.0, 0.0);
                        closeobj();
                }
                makeobj(O_LABEL);
                closeobj();
                makeobj(O_SURFACE);
                closeobj();
                makeobj(O_BOND);
                        translate((Coord) 0, (Coord) 0, (Coord) 0);
                closeobj();
#endif  IRIS
#endif  DRAWPIC
#endif  EDITONLY

                /* Run the program */

                input();                /* see recdes.c */
                if (isterm)
                        printf("\n");
                notcom = FALSE;
                execstop((struct token_def *) NULL);
}

#ifdef  DEBUG
/*
 * printspec - print specifier
 */
printspec(s)
struct  spec_def        *s;
{
        struct  spec_def        *s1;

        for (s1 = s; s1 != NULL; s1 = s1->s_next) {
                fprintf(stderr, "\tModel ");
                printwords(s1->s_model->sp_range);
                fprintf(stderr, "(every %dth)\n", s1->s_model->sp_cnt);
                fprintf(stderr, "\tResidue ");
                printwords(s1->s_residue->sp_range);
                fprintf(stderr, "(every %dth)\n", s1->s_residue->sp_cnt);
                fprintf(stderr, "\tAtom ");
                printwords(s1->s_atom->sp_range);
                fprintf(stderr, "(every %dth)\n", s1->s_atom->sp_cnt);
        }
        fprintf(stderr, "%.3f < B < %.3f\n", s->s_tfmin, s->s_tfmax);
        fprintf(stderr, "%.3f < ESP < %.3f\n", s->s_espmin, s->s_espmax);
}

/*
 * printwords - Print a range
 */
printwords(w)
struct  wordrange_def   *w;
{
        struct  wordrange_def   *w1;

        for (w1 = w; w1 != NULL; w1 = w1->w_next) {
                fprintf(stderr, "'%s'", w1->w_first);
                if (w1->w_first != w1->w_last)
                        fprintf(stderr, "-'%s'", w1->w_last);
                fprintf(stderr, " ");
        }
}

/*
 * printtoken - Print a token
 */
printtoken(t)
struct  token_def       *t;
{
        struct  token_def       *t1;

        for (t1 = t; t1 != NULL; t1 = t1->t_next)
                fprintf(stderr, "'%s'", t1->t_value);
}
#endif  DEBUG

#ifdef  vms
#define index   strchr
#endif

/*
 * reply - Send a message to the user
 */
reply(s)
char    *s;
{
#ifndef EDITONLY
        char    *index();

        if (nreplies >= MAXREPLY)
                return;
```

```
            strcpy(repbuf[nreplies], s);
            if (index(s, '\n') == NULL) {
                    strcat(repbuf[nreplies], "\n");
                    fprintf(stderr, "Bad error message!\n%s\nSee MIDAS guru.\n", s);
            }
            nreplies++;
#else
            fputs(s, stdout);
#endif
}

/*
 * prompt - Prompt he user
 */
prompt()
{
            if (isterm)
                    printf("> ");
}
```

lex.c

```
/* $Header: lex.c,v 3.2 86/05/29 16:32:38 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include <ctype.h>
#include "editor.h"

static  char    tokenval[MAXVALLEN];

/*
 * l_init - the lexical analyzer initializing routine
 */
l_init()
{
        token.t_value = tokenval;
}

/*
 * gettoken - reads in the next token and determines its lexical code.
 * Both the lexical code and the actual string are stored in the
 * global variable 'token'
 */
gettoken(mode)
int     mode;
{
        char    lastc, nc, getch();
        int     i, checkcom, tilspc;

        /* Aliases are expanded by default */
        checkcom = mode == KEYWRD;
        tilspc = mode == TILSPC;

        /* Skip white space */
        do {
                lastc = getch();
        } while (lastc == ' ' || lastc == '\t');

        /* Store the character and determine the token lexical code */
        token.t_value[0] = lastc;
        token.t_value[1] = '\0';
        if (tilspc && lastc != '\n' && lastc != '\0' && lastc != ';')
                lastc = 'a';                    /* Fake like we got a name */

        switch (lastc) {

        case ';' :
                token.t_type = SEMICOLON;
                break;

        case '~' :
                token.t_type = TILDA;
                break;

        case ',' :
                token.t_type = COMMA;
                break;

        case '&' :
                token.t_type = AND;
                break;

        case '#' :
                token.t_type = MODEL;
                break;

        case ':' :
                token.t_type = RESIDUE;
                break;

        case '@' :
                token.t_type = ATOM;
                break;

        case '%' :
                token.t_type = COUNT;
                break;

        case '-' :
                token.t_type = HYPHEN;
                break;

        case '!' :
                token.t_type = EXCLPT;
                break;

        case '\0' :
        case '\n' :
                token.t_type = EOC;
                break;
```

```
		do
			nc = getch();
		while (isspace(nc) && nc != '\n');
		if (nc == '<' || nc == '>') {
			if (lastc == 'B' || lastc == 'b')
				token.t_type = TFRANGE;
			else
				token.t_type = ESPRANGE;
			token.t_value[1] = nc;
			token.t_value[2] = '\0';
			break;
		}
		backspace(nc);
		/* Fall through sine 'b' and 'e' are legal letters */

	default :
		/* The token is a name.  We must do some more work */
		if (lisletter(lastc)) {
			fprintf(stderr, "%c (%o) ignored\n",
				lastc, lastc);
			gettoken(checkcom);
			break;
		}

		/* Read in the entire token and put back the
		 * unused character */
		for (i = 1; i < MAXVALLEN - 1; i++) {
			lastc = getch();
			if (!tilspc && lisletter(lastc)) {
				backspace(lastc);
				break;
			}
			if (tilspc && (isspace(lastc) || lastc == ';')) {
				backspace(lastc);
				break;
			}
			token.t_value[i] = lastc;
		}
		token.t_value[i] = '\0';

		/* Try to expand the name */
		if (tilspc || !expand(checkcom))
			token.t_type = NAME;
		if (checkcom && token.t_type == NAME)
			for (i = 0; comtab[i].com_type != NOMORECOM; i++)
				if (strncmp(token.t_value, comtab[i].com_name,
					strlen(token.t_value)) == 0) {
					token.t_type = comtab[i].com_type;
					curcom = i;
					comnum = comtab[i].com_num;
					comtab[i].com_count++;
					break;
				}
#ifdef	DEBUG
		if (debug & D_LEX) fprintf(stderr, "Lex: (%d) >%s<\n",
			token.t_type, token.t_value);
#endif
		return;
}

/*
 * getch - gets the next character from the input
 */
char
getch()
{
	char	c;

	if (stptr < 0)				/* Hit bottom */
		return('\0');
	while ((c = NEXTC) == '\0')		/* Hit end of level */
		stptr--;
	if ((int) c == -1) {			/* Read failed */
		(void) fclose(CURSTRM);
		stptr--;
		return('\0');
	}
	return(c);
}

/*
 * backspace - puts 'c' back onto the input stream
 */
backspace(c)
char	c;
{
```

```
        if (CURNDX > 0)
                CURNDX--;
        else if (CURNDX < 0)
                (void) ungetc(c, CURSTRM);
}

/*
 * expand - expands the current token if it is an alias
 */
expand(checkcom)
int     checkcom;
{
        int     i;

        for (i = 0; i < nalias; i++)
                if (!strcmp(token.t_value, atab[i].a_name))
                        break;
        if (i >= nalias)
                return(FALSE);
        if (stptr+1 >= MAXSTACK) {
                reply("Alias nesting too deep.\n");
                return(FALSE);
        }
        stptr++;
        CURNDX = 0;
        CURLINE = atab[i].a_line;
        gettoken(checkcom);
        return(TRUE);
}

/*
 * isletter - returns TRUE if it is not a special character
 */
isletter(c)
char    c;
{
        if (c == ';' || c == '~' || c == '&' || c == '#' || c == '\t' ||
                c == ' ' || c == ':' || c == '@' || c == '%' || c == '<' ||
                c == '>' || c == ',' || c == '-' || c == '|' || c == '\n' ||
                c == '\0')
                        return(FALSE);
        return(TRUE);
}
```

recdes.c

```
/* $Header: recdes.c,v 3.14 86/06/09 19:04:53 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                  9 Apr 1982
 * Release 2.0                  13 May 1983
 * Release 2.1                  16 May 1983
 * Release 2.2                  28 Jul 1983
 */

/*
 * Recursive descent parser for MIDAS.
 *
 * See  Aho, Ullman, Principles of Compiler Design,
 *      Addison-Wesley Publishing Company, 1979, pp 180-183
 */
#include <ctype.h>
#ifndef  vms
#include <grafeng.h>
#else
#include "MIDAS$H:machdef.h"
#endif

#if      GRAFENG == IRIS
#include <device.h>

#undef   REDRAW/* device.h defines a different REDRAW than editor.h */
#endif
#include "editor.h"

input()
{
        int     save;
        static short    junk;

#ifdef   DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Input()\n");
#endif
        save = stptr;
        anychg = FALSE;
        do {
                if (stptr == 0)
                        prompt();
                gettoken(KEYWRD);
                if (token.t_type == EOC)
                        continue;
#ifndef  EDITONLY
#ifndef  DRAWPIC
                if (stptr == 0)
                        getps();
#endif
#endif
                if (command() != 0)
                        clearin();
                while (token.t_type == SEMICOLON) {
                        gettoken(KEYWRD);
                        if (command() != 0)
                                clearin();
                }
#ifndef EDITONLY
#ifdef   DEBUG
                if (debug & D_PARSE)
                        fprintf(stderr, "Input level = %d\n", stptr);
#endif
                if (save == 0 && stptr == 0) {
                        if (anychg)
                                drawpic();
#ifndef  DRAWPIC
                        putps();
#endif
                        writeeo();
                        nreplies = 0;
#ifdef   DRAWPIC
#if      GRAFENG == PS2
                        push();
                        window(wl, wr, wd, wu, wh, wy);
                        drawps('pc');
                        pop();
                        nufram();
#endif
#if      GRAFENG == MPS
                        tpush();
                        csize(&3);
                        twind(&wl, &wr, &wd, &wu, &wh, &wy);
                        udata(&'pc', psobs);
                        udata(&'bd', psobs);
                        tpop();
                        sdrep();
#endif
#if      GRAFENG == IRIS
                        if (wl != wr && wd != wu && wy != wh) {
                                pushmatrix();
```

```
                                        /*
                                         * We must invert "hither" and "yon" since the
                                         * IRIS uses right hand co-ordinates
                                         */
                                        ortho(wl, wr, wd, wu, wy, wh);
                                        depthcue(TRUE);
                                        callobj(O_ALL);
                                        depthcue(FALSE);
                                        popmatrix();
                                }
                                tpoff();
                                gflush();
                                qreset();
                                (void) qread(&junk);
                                tpon();
#endif
#if     GRAFENG == PS300
                                printf("{ display all models }\n");
#endif
#endif  DRAWPIC
                        }
#endif  EDITONLY
        } while (stptr >= save);
        return;
}

command()
{
        int     stat;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Command()\n");
#endif
        if (token.t_type == TILDA) {
                notcom = TRUE;
                gettoken(KEYWRD);
        } else if (token.t_type == MODEL) {             /* # starts a comment */
                return(-1);
        } else
                notcom = FALSE;
        if ((stat = gcom()) >= 0)
                return(stat);
        if ((stat = scom()) >= 0)
                return(stat);
        if ((stat = ncom()) >= 0)
                return(stat);
        if (token.t_type != EXCLPT) {
                (void) sprintf(buf, "Unknown command '%s'\n", token.t_value);
                reply(buf);
                return(-1);
        }
        return(shell());
}

gcom()
{
        struct  spec_def        *s;
#ifdef  DRAWPIC
        int     i;
#endif

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Gcom()\n");
#endif
        if (token.t_type != GCOMWORD)
                return(-1);
        gettoken(NORMAL);
        if ((s = specs()) == NULL) {
                freespec(s);
                return(1);
        }
        if (token.t_type != SEMICOLON && token.t_type != EOC) {
                freespec(s);
                (void) sprintf(buf,
                        "Expecting atom specifier (#:@,) and got '%s'.\n",
                        token.t_value);
                reply(buf);
                return(1);
        }
        execgcom(s);
        freespec(s);
#ifdef  DRAWPIC
        for (i = 0; i < MAXMOD; i++)
                if (mdb[i].m_fname != NULL)
                        mflush(mdb[i].m_fd);
#endif
        return(0);
}

struct  spec_def        *
specs()
{
```

```
        struct  spec_def                *s, *s1;
#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Specs()\n");
#endif
        s1 = s = spec((struct spec_def *) NULL);
        if (s == NULL)
                return(NULL);
        while (token.t_type == MODEL || token.t_type == RESIDUE ||
                token.t_type == ATOM || token.t_type == AND) {
                        if (token.t_type == AND)
                                gettoken(NORMAL);
                        s1->s_next = spec(s1);
                        s1 = s1->s_next;
                        if (s1 == NULL) {
                                freespec(s);
                                return(NULL);
                        }
        }
        s1->s_next = NULL;
        return(s);
}

struct  spec_def        *
spec(s1)
struct  spec_def        *s1;
{
        struct  spec_def        *s;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Spec()\n");
#endif
        s = salloc(1, sizeof (struct spec_def));
        s->s_stat = 0;
        s->s_next = NULL;
        s->s_model = s->s_residue = s->s_atom = NULL;
        if (getspec(MODEL, &s->s_model) == -1) {
                freespec(s);
                reply("Model(#) specifier mangled.\n");
                return(NULL);
        } else if (s->s_model == NULL) {
                if (s1 == NULL) {
                        /* default to all models */
                        s->s_model = spalloc(1, sizeof (struct speclist_def));
                        s->s_model->sp_range =
                                walloc(1, sizeof (struct wordrange_def));
                        s->s_model->sp_range->w_first = all;
                        s->s_model->sp_range->w_last = all;
                        s->s_model->sp_range->w_next = NULL;
                        s->s_model->sp_cnt = 1;
                } else {
                        s->s_model = s1->s_model;
                        s->s_stat |= DUPMOD;
                }
        }
        if (getspec(RESIDUE, &s->s_residue) == -1) {
                freespec(s);
                reply("Residue(:) specifier mangled.\n");
                return(NULL);
        } else if (s->s_residue == NULL) {
                if (!(s->s_stat & DUPMOD)) {
                        /* default to all residues */
                        s->s_residue = spalloc(1, sizeof (struct speclist_def));
                        s->s_residue->sp_range =
                                walloc(1, sizeof (struct wordrange_def));
                        s->s_residue->sp_range->w_first = all;
                        s->s_residue->sp_range->w_last = all;
                        s->s_residue->sp_range->w_next = NULL;
                        s->s_residue->sp_cnt = 1;
                } else {
                        s->s_residue = s1->s_residue;
                        s->s_stat |= DUPRES;
                }
        }
        if (getspec(ATOM, &s->s_atom) == -1) {
                freespec(s);
                reply("Atom(@) specifier mangled.\n");
                return(NULL);
        } else if (s->s_atom == NULL) {
                /* default to all atoms */
                s->s_atom = spalloc(1, sizeof (struct speclist_def));
                s->s_atom->sp_range =
                        walloc(1, sizeof (struct wordrange_def));
                s->s_atom->sp_range->w_first = all;
                s->s_atom->sp_range->w_last = all;
                s->s_atom->sp_range->w_next = NULL;
                s->s_atom->sp_cnt = 1;
        }
        s->s_tfmin = mingtf;
        s->s_tfmax = maxgtf;
        s->s_espmin = mingesp;
        s->s_espmax = maxgesp;
```

```
        while (tferange(s))
                ;
        return(s);
}


getspec(type, ptr)
int     type;
struct  speclist_def        **ptr;
{
        struct  speclist_def        *sp;
        struct  wordrange_def       *w;
        int     i;

        *ptr = NULL;
        if (token.t_type != type)
                return(0);
#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Getspec(%d)\n", type);
#endif
        gettoken(NORMAL);
        sp = spalloc(1, sizeof (struct speclist_def));
        if ((sp->sp_range = wranges()) == NULL) {
                free((char *) sp);
                return(-1);
        }
        sp->sp_cnt = count();
        if (sp->sp_cnt == -1) {
                freelist(sp);
                return(-1);
        }
        if (type != MODEL)
                for (w = sp->sp_range; w != NULL; w = w->w_next) {
                        for (i = 0; w->w_first[i] != '\0'; i++)
                                if (islower(w->w_first[i]))
                                        w->w_first[i] = toupper(w->w_first[i]);
                        if (w->w_last != w->w_first)
                                for (i = 0; w->w_last[i] != '\0'; i++)
                                        if (islower(w->w_last[i]))
                                                w->w_last[i] =
                                                        toupper(w->w_l
                }
        *ptr = sp;
        return(0);
}

struct  wordrange_def       *
```

```
wranges()
{
        struct  wordrange_def       *w, *w1;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Wranges()\n");
#endif
        if ((w = wrange()) == NULL)
                return(NULL);
        w1 = w;
        while (token.t_type == COMMA) {
                gettoken(NORMAL);
                if ((w1->w_next = wrange()) == NULL) {
                        (void) sprintf(buf, "Comma before '%s' ignored.\n",
                                token.t_value);
                        reply(buf);
                } else
                        w1 = w1->w_next;
        }
        w1->w_next = NULL;
        return(w);
}

struct  wordrange_def       *
wrange()
{
        struct  wordrange_def       *w;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Wrange()\n");
#endif
        w = walloc(1, sizeof (struct wordrange_def));
        if ((w->w_first = word(NORMAL)) == NULL) {
                free((char *) w);
                return(NULL);
        }
        if (token.t_type == HYPHEN) {
                gettoken(NORMAL);
                if ((w->w_last = word(NORMAL)) == NULL) {
                        free((char *) w->w_first);
                        free((char *) w);
                        return(NULL);
                }
        } else
                w->w_last = w->w_first;
```

```
        return(w);
}

char    *

word(type)
int     type;
{
        char    *c;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Word()\n");
#endif
        if (token.t_type == NAME) {
                c = alloc(strlen(token.t_value)+1, sizeof (char));
                strcpy(c, token.t_value);
                gettoken(type);
                return(c);
        }
        return(NULL);
}

count()
{
        int     i;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Count()\n");
#endif
        if (token.t_type != COUNT)
                return(1);
        gettoken(NORMAL);
        if (token.t_type != NAME)
                return(-1);
        for (i = 0; token.t_value[i] != '\0'; i++)
                if (!isdigit(token.t_value[i]))
                        return(-1);
        i = atoi(token.t_value);
        gettoken(NORMAL);
        return(i);
}

tferange(s)
struct  spec_def        *s;
{
        register        i;
        int     mgtype;
        char    type, negative;
        double  value, atof();

        if (token.t_type != TFRANGE && token.t_type != ESPRANGE)
                return(FALSE);
#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Tfrange()\n");
#endif
        mgtype = token.t_type;
        type = token.t_value[1];
        gettoken(NORMAL);
        if (token.t_type == HYPHEN) {
                negative = TRUE;
                gettoken(NORMAL);
        } else
                negative = FALSE;
        if (token.t_type != NAME)
                return(FALSE);
        for (i = 0; token.t_value[i] != '\0'; i++)
                if (!isdigit(token.t_value[i]) && token.t_value[i] != '.')
                        return(-1);
        value = negative ? -atof(token.t_value) : atof(token.t_value);
        if (mgtype == TFRANGE) {
                if (type == '<')
                        s->s_tfmax = value;
                else
                        s->s_tfmin = value;
        } else {
                if (type == '<')
                        s->s_espmax = value;
                else
                        s->s_espmin = value;
        }
        gettoken(NORMAL);
        return(TRUE);
}

scom()
{
        struct  wordrange_def   *w;
        struct  spec_def        *s;
#ifdef  DRAWPIC
        int     i;
#endif
```

```
#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Scom()\n");
#endif
        if (token.t_type != SCOMWORD)
                return(-1);
        gettoken(NORMAL);
        w = wranges();
        if ((s = specs()) == NULL) {
                freespec(s);
                return(1);
        }
        if (token.t_type != SEMICOLON && token.t_type != EOC) {
                freerange(w);
                freespec(s);
                (void) sprintf(buf,
                        "Expecting atom specifier (#:@,) and got '%s'.\n",
                        token.t_value);
                reply(buf);
                return(1);
        }
        execscom(w, s);
        freerange(w);
        freespec(s);
#ifdef  DRAWPIC
        for (i = 0; i < MAXMOD; i++)
                if (mdb[i].m_fname != NULL)
                        mflush(mdb[i].m_fd);
#endif
        return(0);
}

ncom()
{
        struct  token_def       *t;
#ifndef EDITONLY
        extern  int     halfbond;
#endif

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Ncom()\n");
#endif
        if (token.t_type != NCOMWORD)
                return(-1);
        gettoken(TILSPC);
        t = symbols();
        switch (comnum) {

        case ALIAS :
                execalias(t);
                break;

        case PREAD :
                execread(t);
                break;

        case STOP :
                execstop(t);
                break;

        case OPEN :
                execopen(t);
                break;

        case CHDIR :
                exechdir(t);
                break;

        case SAVE :
                execsave(t);
                break;
#ifndef EDITONLY
        case SETCOM :
                execsetc(t);
                break;

        case VDWOPT :
                execvopt(t);
                break;

        case HALFBND :
                if (halfbond == notcom)
                        halfbond = !notcom;
                if (halfbond)
                        reply("Halfbond mode is on.\n");
                else
                        reply("Halfbond mode is off.\n");
                break;

#if     GRAFENG == PS2 || GRAFENG == MPS
        case PICK :
                execpick(t);
                break;
```

```
#endif
#ifdef GRAFENG == IRIS
        case MAPCOLORS:
                execmapcolors(t);
                break;
#endif
#endif  EDITONLY
        }
        freetoken(t);
        return(0);
}

struct  token_def       *
symbols()
{
        struct  token_def       *t, *t1;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Symbols()\n");
#endif
        if ((t = symbol()) == NULL)
                return(NULL);
        t1 = t;
        while ((t1->t_next = symbol()) != NULL)
                t1 = t1->t_next;
        return(t);
}

struct  token_def       *
symbol()
{
        char    *c;
        int     save;
        struct  token_def       *t;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Symbol()\n");
#endif
        save = token.t_type;
        if ((c = word(TILSPC)) == NULL && (c = delims()) == NULL)
                return(NULL);
        t = talloc(1, sizeof (struct token_def));
        t->t_type = save;
        t->t_value = c;
        return(t);
}

char    *
delims()
{
        char    *c;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Delims()\n");
#endif
        if (token.t_type == COMMA || token.t_type == AND ||
                token.t_type == MODEL   || token.t_type == RESIDUE ||
                token.t_type == ATOM    || token.t_type == COUNT ||
                token.t_type == TFRANGE || token.t_type == HYPHEN ||
                token.t_type == TILDA   || token.t_type == EXCLPT) {
                        c = alloc(strlen(token.t_value)+1, sizeof (char));
                        strcpy(c, token.t_value);
                        gettoken(TILSPC);
                        return(c);
        }
        return(NULL);
}

shell()
{
        struct  token_def       *t;

#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Shell()\n");
#endif
        if (token.t_type != EXCLPT)
                return(-1);
        gettoken(TILSPC);
        t = symbols();
        execshell(t);
        freetoken(t);
        return(0);
}

clearin()
{
#ifdef  DEBUG
        if (debug & LD_PARSE)
                fprintf(stderr, "Stopped parsing at %d ('%s')\n",
                        token.t_type, token.t_value);
```

```
#endif

        while (token.t_type != SEMICOLON && token.t_type != EOC)
                gettoken(NORMAL);
}
```

edit.c

```
/* $Header: edit.c,v 3.8 86/04/25 09:53:54 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          6 Jul 1983
 * Release 2.2                          15 Jul 1983
 * Release 2.3                          28 Jul 1983
 * Release 2.4                          23 Aug 1983
 */

#include <ctype.h>
#ifndef  vms
#include <assert.h>
#else
#include "MIDAS$H:assert.h"
#endif
#include "editor.h"

#define  ALLSPEC sp=s,p=(&first);sp!=NULL;sp=sp->s_next,p=p->p_next
#define  BEGIN   0
#define  END     1

/*
 ** 'p_*sel' are flags indicating whether the current object should
 * be selected be default. 'p_*cnt' are the counters keeping track of
 * the number of matches for this object
 */
struct   pick_def            {
         int         p_modsel, p_modcnt;
         int         p_ressel, p_rescnt;
         int         p_atmsel, p_atmcnt;
         struct      pick_def          *p_next;
};

static   struct      spec_def          *s;
static   struct      pick_def          first;
static   float       tfmin, tfmax;
static   float       espmin, espmax;
static   int         mol;
static   char        reseseq[RES_SEQ_SIZE+1], restype[RES_TYPE_SIZE+1];
static   int         natom;
static   struct      atom_def          *atomdata;
static   struct      surf_def          *surfdata;
static   int         p_on, p_off, n_on, n_off;
static   int         chkendatm;
static   int         (*editres)(), (*realedit)(), (*prepres)();
static   int         firstcompr;

int      allres(), selres(), compr();
int      endselatm(), selatm(), endpickatm(), pickatm();
int      flipbits(), colores(), chcolor(), null(), setsurf();
int      edltrot(), stcoord(), editdist(), editang();
int      editadd(), editgetres();

/*
 * edit - correspond to the main program of the editing portion.
 */
edit(sp)
struct   spec_def            *sp;
{
         struct      pick_def          *p, *q;
         struct      wordrange_def     *w;
         int         selected;
         int         lastsel, lastcnt;

         /* Initialize variables.  Nothing selected by default */

         chkendatm = FALSE;
         s = sp;
         first.p_modsel = FALSE;
         first.p_modcnt = 0;
         first.p_ressel = FALSE;
         first.p_rescnt = 0;
         first.p_atmsel = FALSE;
         first.p_atmcnt = 0;
         if (!(sp->s_stat & DUPMOD))
                  for (w = sp->s_model->sp_range; w != NULL;
                           w = w->w_next) {
                                    insmol(w->w_first);
                                    insmol(w->w_last);
                  }
         if (!(sp->s_stat & DUPRES))
                  for (w = sp->s_residue->sp_range; w != NULL;
                           w = w->w_next) {
                                    insres(w->w_first);
                                    insres(w->w_last);
                  }
         for (w = sp->s_atom->sp_range; w != NULL; w = w->w_next) {
                  insatm(w->w_first);
                  insatm(w->w_last);
                  if (strcmp(w->w_last, all) == 0 && strcmp(w->w_first, all) != 0)
```

```
                        chkendatm = TRUE;
        }

        /* Allocate markers for all specifiers */
        p = &first;
        for (sp = s->s_next; sp != NULL; sp = sp->s_next) {
                p->p_next = (struct pick_def *)
                        alloc(1, sizeof (struct pick_def));
                p = p->p_next;
                p->p_modsel = FALSE;
                p->p_modcnt = 0;
                p->p_ressel = FALSE;
                p->p_rescnt = 0;
                p->p_atmsel = FALSE;
                p->p_atmcnt = 0;
                if (!(sp->s_stat & DUPMOD))
                        for (w = sp->s_model->sp_range; w != NULL;
                                w = w->w_next) {
                                        insmol(w->w_first);
                                        insmol(w->w_last);
                        }
                if (!(sp->s_stat & DUPRES))
                        for (w = sp->s_residue->sp_range; w != NULL;
                                w = w->w_next) {
                                        insres(w->w_first);
                                        insres(w->w_last);
                        }
                for (w = sp->s_atom->sp_range; w != NULL; w = w->w_next) {
                        insatm(w->w_first);
                        insatm(w->w_last);
                        if (strcmp(w->w_last, all) == 0
                           && strcmp(w->w_first, all) != 0)
                                chkendatm = TRUE;
                }
        }
        p->p_next = NULL;

        /* Go through all models */
        for (mol = 0; mol < MAXMOD; mol++) {
                if (mdb[mol].m_fname == NULL || mdb[mol].m_flags & PSOB)
                        continue;
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Editing model %d\n", mol);
#endif

                /* Try to match the front end of a range */
                selected = FALSE;
#ifdef  lint
                lastsel = FALSE;
                lastcnt = 0;
#endif
                for (ALLSPEC) {
                        if (sp->s_stat & DUPMOD) {
                                p->p_modsel = lastsel;
                                p->p_modcnt = lastcnt;
                        } else {
                                if (!p->p_modsel)
                                        if (matchm(sp->s_model, BEGIN))
                                                p->p_modsel = TRUE;
                                if (p->p_modsel) {
                                        if (p->p_modcnt == 0)
                                                selected = TRUE;
                                        p->p_modcnt = (p->p_modcnt + 1) %
                                                sp->s_model->sp_cnt;
                                }
                                lastsel = p->p_modsel;
                                lastcnt = p->p_modcnt;
                        }
                }

                if (selected) {
                        firstcompr = TRUE;
                        (*editree)();           /* Do the editing */
                }

                /* Try to match the back end of a range */
                for (ALLSPEC) {
                        if (sp->s_stat & DUPMOD)
                                p->p_modsel = lastsel;
                        else {
                                if (p->p_modsel && matchm(sp->s_model, END))
                                        p->p_modsel = FALSE;
                                lastsel = p->p_modsel;
                        }
                }
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Done editing mol %d\n", mol);
#endif
        }

        /* Get rid of the markers */
        for (p = first.p_next; p != NULL; p = q) {
```

```
                q = p->p_next;
                free((char *) p);
        }
        prunused();
}

/*
 * allres - go through the database sequentially and edits ALL the atoms
 */
allres()
{
        struct  spec_def        *sp;
        struct  pick_def        *p;
        int     selected;
        int     lastsel, lastcnt;

        /* Start off non-selected and at the beginning of the database */
        for (ALLSPEC) {
                p->p_ressel = FALSE;
                p->p_rescnt = 0;
        }
        (void) mseekr(mdb[mol].m_fd, all, FLSEEK);

        /* Step through the database */
        while ((natom = mreadr(mdb[mol].m_fd, resseq, restype)) > 0) {
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Residue %s (%s)", resseq, restype);
#endif

                /* If it is a BOND residue, skip it */
                if (tmatch(restype, "BOND") == 0)
                        continue;

                /* Go for a match at the beginning of a range
                 * (Doesn't this look familiar ?) */
                selected = FALSE;
#ifdef  lint
                lastsel = FALSE;
                lastcnt = 0;
#endif
                for (ALLSPEC) {
                        if (!p->p_modsel)
                                continue;
                        if (sp->s_stat & DUPRES) {
                                p->p_ressel = lastsel;
                                p->p_rescnt = lastcnt;
                        } else {
                                if (!p->p_ressel)
                                        if (matchr(sp->s_residue, BEGIN))
                                                p->p_ressel = TRUE;
                                if (p->p_ressel) {
                                        if (p->p_rescnt == 0)
                                                selected = TRUE;
                                        p->p_rescnt = (p->p_rescnt + 1) %
                                                sp->s_residue->sp_cnt;
                                }
                                lastsel = p->p_ressel;
                                lastcnt = p->p_rescnt;
                        }
                }
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, " %s\n", selected ? "selected"
                                : "not selected");
#endif

                /* Read in the atomic data, edit the residue, and
                 * write the data back out */
                assert (mreada(mdb[mol].m_fd) != -1);
                atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#ifdef  DEBUG
                if (debug & LD_EDIT)
                        fprintf(stderr, "Atomdata = %o\n", atomdata);
#endif
                if (selected) {
                        if (chkendatm)
                                (void) mtrav(mdb[mol].m_fd, selatm, endselatm);
                        else
                                (void) mtrav(mdb[mol].m_fd, selatm, null);
                } else
                        allatm(mdb[mol].m_fd);
                (void) mwrta(mdb[mol].m_fd);
#ifndef EDITONLY
                remakeplc();
#endif

                /* Try to match the last residue of any range */
                for (ALLSPEC) {
                        if (!p->p_modsel)
                                continue;
                        if (sp->s_stat & DUPRES)
                                p->p_ressel = lastsel;
                        else {
```

```
                                        if (p->p_ressel && matchr(sp->s_residue, EN
                                                p->p_ressel = FALSE;
                                        lastsel = p->p_ressel;
                                }
                        }
                }
}

/*
 * selatm - used to edit residues which were selected.
 */
selatm(mfd, index, ischief, islinkage, nson, firstime)
int     mfd;
int     index;
int     ischief, islinkage, nson, firstime;
{
        struct  spec_def        *sp;
        struct  pick_def        *p;
        int     selected;
        char    atname[AT_NAME_SIZE+1];

        /* And one more time -- try for a match at the start of a range */
        (void) matom(mfd, index, atname);
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "At atom %s %o (%f %f %f %f)\n", atname,
                        atomdata[index].status,
                        atomdata[index].x, atomdata[index].y,
                        atomdata[index].z, atomdata[index].tempfac);
#endif

        selected = FALSE;
        for (ALLSPEC) {
                if (!p->p_modsel || !p->p_ressel)
                        continue;
                if (!p->p_atmsel)
                        if (matcha(sp->s_atom, atname, BEGIN, ischief))
                                p->p_atmsel = TRUE;
                if (p->p_atmsel) {
                        tfmin = sp->s_tfmin;
                        tfmax = sp->s_tfmax;
                        espmin = sp->s_espmin;
                        espmax = sp->s_espmax;
                        if (atomdata[index].tempfac >= tfmin &&
                            atomdata[index].tempfac <= tfmax)
                                selected = TRUE;
                        else
                                selected = FALSE;
                        if (selected) {
                                if (p->p_atmcnt != 0)
                                        selected = FALSE;
                                p->p_atmcnt = (p->p_atmcnt + 1) %
                                        sp->s_atom->sp_cnt;
                        }
                        if (matcha(sp->s_atom, atname, END, ischief))
                                p->p_atmsel = FALSE;
                }
        }

        if (firstime && atomdata[index].status & EXISTBIT)
                (*realedit)(selected, index, atname);
        /* ARGSUSED */
}

/*
 * endselatm - used to match '*' in specifiers
 */
endselatm(mfd, index, ischief, islinkage, nson)
int     mfd;
int     index;
int     ischief, islinkage, nson;
{
        register        struct  wordrange_def   *w;
        struct  spec_def        *sp;
        struct  pick_def        *p;
        char    atname[AT_NAME_SIZE+1];

        if (nson != 0)
                return;

        (void) matom(mfd, index, atname);
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "Return atom %s (%f %f %f)\n", atname,
                        atomdata[index].x, atomdata[index].y,
                        atomdata[index].z);
#endif

        for (ALLSPEC) {
                if (!p->p_modsel || !p->p_ressel || !p->p_atmsel)
                        continue;
                for (w = sp->s_atom->sp_range; w != NULL; w = w->w_next) {
                        if (strcmp(w->w_last, all) != 0)
                                continue;
```

```
                        if (amatch(w->w_first, atname) == 0)
                                p->p_atmsel = FALSE;
                }
        }
        /* ARGSUSED */
}

/*
 * allatm - used to edit the non-selected residues where the
 * traversal makes no difference
 */
allatm(mfd)
int     mfd;
{
        int     i;
        char    atname[AT_NAME_SIZE+1];

        for (i = 0; i < natom; i++) {
                (void) matom(mfd, i, atname);
                (*realedit)(FALSE, i, atname);
        }
}

/*
 * selres - only looks at the selected residues
 */
selres()
{
        struct  spec_def        *sp;
        struct  pick_def        *p;
        struct  wordrange_def   *w;

        /* Nothing selected by default */
        for (ALLSPEC) {
                p->p_ressel = FALSE;
                p->p_rescnt = 0;
        }

        /* Go through all ranges and edit them */
        for (ALLSPEC) {
                if (!p->p_modsel)
                        continue;
                tfmin = sp->s_tfmin;
                tfmax = sp->s_tfmax;
                espmin = sp->s_espmin;
                espmax = sp->s_espmax;
                for (w = sp->s_residue->sp_range; w != NULL; w = w->w_next) {
                        if (isalpha(w->w_first[0]) || w->w_first[0] == all[0]
                           || w->w_first[0] == '?')
                                editype(sp->s_atom, p, w,
                                        sp->s_residue->sp_cnt);
                        else
                                editseq(sp->s_atom, p, w,
                                        sp->s_residue->sp_cnt);
                }
        }
}

static  struct  speclist_def    *atomlist;
static  struct  pick_def        *pick_p;

/*
 * editseq - used if the residue specifier was a residue sequence number
 */
editseq(atoms, p, words, skipcnt)
struct  speclist_def    *atoms;
struct  pick_def        *p;
struct  wordrange_def   *words;
int     skipcnt;
{
        int     count;
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "Looking for seq %s\n", words->w_first);
#endif
        if (mseekr(mdb[mol].m_fd, words->w_first, FBSEEK) == -1)
                return;
        delres(words->w_first);
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "Found %s\n", words->w_first);
#endif
        atomlist = atoms;
        pick_p = p;
        pick_p->p_ressel = TRUE;
        do {
                natom = mreadr(mdb[mol].m_fd, reseseq, restype);
                if (natom < 0)
                        break;

                /* If it is a BOND residue, skip it */
                if (tmatch(restype, "BOND") == 0)
                        continue;
```

```
                count = pick_p->p_rescnt;
                pick_p->p_rescnt = (pick_p->p_rescnt + 1) % skipcnt;
                if (count != 0)
                        continue;
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Editing %s (%s)\n", resseq, restype);
#endif
                assert (mreada(mdb[mol].m_fd) != -1);
                atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                if (prepres != NULL)
                        (*prepres)();
#ifdef  DEBUG
                if (debug & LD_EDIT)
                        fprintf(stderr, "Atomdata = %o\n", atomdata);
#endif
                pick_p->p_atmsel = FALSE;
                if (chkendatm)
                        (void) mtrav(mdb[mol].m_fd, pickatm, endpickatm);
                else
                        (void) mtrav(mdb[mol].m_fd, pickatm, null);
                (void) mwrta(mdb[mol].m_fd);
#ifndef EDITONLY
                remakepic();
#endif
        } while (!compr(words, END));
}

/*
 * editype - used when the residue specifier was a residue type.
 * Note how editseq and editype differ.  Editseq only seeks once
 * since sequence numbers are unique, whereas residue types are not.
 * Thus, editype must repeatedly seek until there are no more.
 */
editype(atoms, p, words, skipcnt)
struct  speclist_def    *atoms;
struct  pick_def        *p;
struct  wordrange_def   *words;
int     skipcnt;
{
        int     count;

        if (mseekr(mdb[mol].m_fd, words->w_first, FLSEEK) == -1)
                return;
        delres(words->w_first);
        atomlist = atoms;
        pick_p = p;
        pick_p->p_ressel = TRUE;
        do {
                do {
                        natom = mreadr(mdb[mol].m_fd, resseq, restype);
                        if (natom < 0)
                                break;

                        /* If it is a BOND residue, skip it */
                        if (tmatch(restype, "BOND") == 0)
                                continue;

                        count = pick_p->p_rescnt;
                        pick_p->p_rescnt = (pick_p->p_rescnt + 1) % skipcnt;
                        if (count != 0)
                                continue;
#ifdef  DEBUG
                        if (debug & D_EDIT)
                                fprintf(stderr, "Editing %s (%s)\n",
                                        resseq, restype);
#endif
                        assert (mreada(mdb[mol].m_fd) != -1);
                        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                        if (prepres != NULL)
                                (*prepres)();
                        else
                                surfdata = NULL;
#ifdef  DEBUG
                        if (debug & LD_EDIT)
                                fprintf(stderr, "Atomdata = %o\n", atomdata);
#endif
                        pick_p->p_atmsel = FALSE;
                        if (chkendatm)
                                (void) mtrav(mdb[mol].m_fd, pickatm,endpickatm);
                        else
                                (void) mtrav(mdb[mol].m_fd, pickatm, null);
                        (void) mwrta(mdb[mol].m_fd);
#ifndef EDITONLY
                        remakepic();
#endif
                } while (!compr(words, END));
                if (isdigit(words->w_last[0]))
                        break;
        } while (mseekr(mdb[mol].m_fd, words->w_first, CFLSEEK) != -1);
}

/*
 * pickatm - analogous to selatm
```

```
 */
pickatm(mfd, index, ischief, islinkage, nson, firstime)
int     mfd;
int     index;
int     ischief, islinkage, nson, firstime;
{
        int     selected;
        char    atname[AT_NAME_SIZE+1];

        (void) matom(mfd, index, atname);
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "At atom %s (%f %f %f)\n", atname,
                        atomdata[index].x, atomdata[index].y,
                        atomdata[index].z);
#endif

        selected = FALSE;
        if (!pick_p->p_atmsel)
                if (matcha(atomlist, atname, BEGIN, ischief))
                        pick_p->p_atmsel = TRUE;
        if (pick_p->p_atmsel) {
                selected = TRUE;
                if (atomdata[index].tempfac < tfmin)
                        selected = FALSE;
                if (atomdata[index].tempfac > tfmax)
                        selected = FALSE;
                if (selected) {
                        if (pick_p->p_atmcnt != 0)
                                selected = FALSE;
                        pick_p->p_atmcnt = (pick_p->p_atmcnt + 1) %
                                atomlist->sp_cnt;
                }
                if (matcha(atomlist, atname, END, ischief))
                        pick_p->p_atmsel = FALSE;
        }

        if (firstime && atomdata[index].status & EXISTBIT)
                (*realedit)(selected, index, atname);
        /* ARGSUSED */
}

/*
 * endpickatm - analogous to endselatm
 */
endpickatm(mfd, index, ischief, islinkage, nson)
int     mfd;
int     index;
int     ischief, islinkage, nson;
{
        register        struct  wordrange_def   *w;
        char    atname[AT_NAME_SIZE+1];

        if (nson != 0)
                return;

        (void) matom(mfd, index, atname);
#ifdef  DEBUG
        if (debug & LD_EDIT)
                fprintf(stderr, "Return atom %s (%f %f %f)\n", atname,
                        atomdata[index].x, atomdata[index].y,
                        atomdata[index].z);
#endif

        if (!pick_p->p_modsel || !pick_p->p_ressel || !pick_p->p_atmsel)
                return;
        for (w = atomlist->sp_range; w != NULL; w = w->w_next) {
                if (strcmp(w->w_last, all) != 0)
                        continue;
                if (amatch(w->w_first, atname) == 0)
                        pick_p->p_atmsel = FALSE;
        }
        /* ARGSUSED */
}

#ifndef  EDITONLY
/*
 * remakepic - remake the picture for the current residue
 * as efficiently as possible
 */
remakepic()
{
        int     needbond, needsurf, needlabel;

        switch (comnum) {

        case DISPLAY :
                needsurf = FALSE;
                needlabel = notcom;
                needbond = TRUE;
                break;

        case CHAIN :
                needsurf = FALSE;
```

```
            needlabel = !notcom;
            needbond = TRUE;
            break;

        case RLABEL :
        case LABEL :
            needsurf = FALSE;
            needlabel = TRUE;
            needbond = FALSE;
            break;

        case SURFACE :
        case VDW :
            needsurf = TRUE;
            needlabel = FALSE;
            needbond = FALSE;
            break;

        case FINROT :           /* These cases are taken care of */
        case BINROT :           /* by their associated routines */
        case REVERSE :          /* because the structure of the */
        case FIXREV :           /* object associated with the */
        case ADDGRP :           /* residue changes */
        case SWAPAA :
        case ADDAA :
        case SWAPNUC :
        case ADDNUC :
        case LINK :
            needsurf = FALSE;
            needlabel = FALSE;
            needbond = FALSE;
            break;

        case SHOW :
        default :
            needsurf = TRUE;
            needlabel = TRUE;
            needbond = TRUE;
            break;

        case DIST :
        case ANGLE :
            needsurf = FALSE;
            needlabel = FALSE;
            needbond = !notcom;
            break;

        case PCOLOR :
        case REDRAW :
            needsurf = (p_off & (SCOLOR | VCOLOR));
            needlabel = (p_off & LCOLOR);
            needbond = (p_off & BCOLOR);
            break;

        case MATCH :
        case GETCRD :
            needsurf = FALSE;
            needlabel = FALSE;
            needbond = TRUE;
            break;
        }
        if (needbond)
            drawbond(mol, resseq, restype, atomdata);
        if (needlabel)
            drawlabel(mol, resseq, restype, atomdata);
        if (needsurf) {
            if (mdb[mol].s_fd >= 0 &&
                mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                    assert (mreada(mdb[mol].s_fd) != -1);
                    surfdata = (struct surf_def *)
                            mdatptr(mdb[mol].s_fd);
            } else
                surfdata = NULL;
            drawsurf(mol, resseq, restype, atomdata, surfdata);
        }
}
#endif

/*
 * matchm - returns TRUE if the current models matches the 'type' end
 * of the given range (where type is front [ BEGIN ] or back [ END ] )
 */
matchm(model, type)
struct  speclist_def        *model;
int     type;
{
        struct  wordrange_def       *w;
        char    *name;

        for (w = model->sp_range; w != NULL; w = w->w_next) {
            if (type == END) {
                if (strcmp(w->w_last, all) == 0)
                    continue;
                name = w->w_last;
```

```
                } else {
                        if (strcmp(w->w_first, all) == 0) {
                                if (w->w_first != all)
                                        free(w->w_first);
                                w->w_first = nothing;           /* Match no mor
                                return(TRUE);
                        }
                        name = w->w_first;
                }
                if (atoi(name) == mol) {
                        delmod(name);
                        return(TRUE);
                }
                if (strcmp(name, mdb[mol].m_fname) == 0) {
                        delmod(name);
                        return(TRUE);
                }
        }
        return(FALSE);
}

/*
 * matchr - returns TRUE if the current residue matches the appropriate end
 * of the given range
 */
matchr(res, type)
struct  speclist_def        *res;
int     type;
{
        struct  wordrange_def       *w;

        for (w = res->sp_range; w != NULL; w = w->w_next)
                if (compr(w, type))
                        return(TRUE);
        return(FALSE);
}

/*
 * compr - uses the appropriate match routine to compare the residue specifiers
 */
compr(w, type)
struct  wordrange_def       *w;
int     type;
{
        char    *name;
        int     retval;

        if (type == END) {
                if (strcmp(w->w_last, all) == 0)
                        return(FALSE);
                name = w->w_last;
        } else {
                if (strcmp(w->w_first, all) == 0) {
                        if (w->w_first != all) {
                                free(w->w_first);
                                w->w_first = all;
                        }
                        if (firstcompr) {
                                firstcompr = FALSE;
                                return(TRUE);
                        } else
                                return(FALSE);
                }
                name = w->w_first;
        }
#ifdef  DEBUG
        if (debug & D_EDIT)
                fprintf(stderr, "Looking at %s (vs %s)\n", name, resseq);
#endif
        if (isalpha(name[0]) || name[0] == all[0] || name[0] == '?')
                retval = tmatch(name, restype) == 0;
        else
                retval = smatch(name, resseq) == 0;
        if (retval)
                delres(name);
        return(retval);
}

/*
 * matcha - yet another matching routine -- this one for atom names
 */
matcha(atom, atname, type, ischief)
struct  speclist_def        *atom;
char    *atname;
int     type, ischief;
{
        register        struct  wordrange_def       *w;

        if (type == END) {
                for (w = atom->sp_range; w != NULL; w = w->w_next) {
                        if (strcmp(w->w_last, all) == 0)
                                continue;
                        if (amatch(w->w_last, atname) == 0) {
                                delatm(w->w_last);
```

```
                                return(TRUE);
                        }
                }
        } else {
                for (w = atom->sp_range; w != NULL; w = w->w_next) {
                        if (strcmp(w->w_first, all) == 0) {
                                if (ischief)
                                        return(TRUE);
                                else
                                        return(FALSE);
                        }
                        if (amatch(w->w_first, atname) == 0) {
                                delatm(w->w_first);
                                return(TRUE);
                        }
                }
        }
        return(FALSE);
}

/*
 * setupbits - set the global variables for the editor
 */
setupbits()
{
        register int            ndx;
        /* Record bits that are to be manipulated */

        ndx = notcom ? 1 : 0;
        p_on  = com_bits[curcom][ndx][0];
        p_off = com_bits[curcom][ndx][1];
        n_on  = com_bits[curcom][ndx][2];
        n_off = com_bits[curcom][ndx][3];
#ifdef  DEBUG
        if (debug & D_EDIT) {
                fprintf(stderr, "Select: on = %o off = %o\n", p_on, p_off);
                fprintf(stderr, "Others: on = %o off = %o\n", n_on, n_off);
        }
#endif

        /* If the non-selected atoms are to be ignored, then we
         * use 'selres' which only looks at the selected residues.
         * 'allres' goes through the database sequentially and is
         * MUCH slower. */
        if (n_on == 0 && n_off == 0)
                editres = selres;
        else
                editres = allres;
        prepres = NULL;                 /* No special thing for residues */
        realedit = flipbits;
}

/*
 * flipbits - does the actual editing (called by edit??? as (*realedit)())
 */
flipbits(selected, index, atname)
int     selected, index;
char    *atname;
{
#ifndef EDITONLY
        int     wason, ison;
#endif
        register                i, j;
        register struct atom_def                *ap;

        /* Ignore non-existent atoms */
        ap = &atomdata[index];
        if (!(ap->status & EXISTBIT))
                return;

#ifndef EDITONLY
        wason = (ap->status & ONBITS);
#endif
        /* Flip the bits */
        if (selected) {
                ap->status &= ~p_off;
                ap->status |= p_on;
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr,"Atom %s selected (%o %f)\n",
                                atname, ap->status, ap->tempfac);
#endif
        } else {
                ap->status &= ~n_off;
                ap->status |= n_on;
#ifdef  DEBUG
                if (debug & LD_EDIT)
                        fprintf(stderr, "Atom %s not selected (%o).\n",
                                atname, ap->status);
#endif
        }
#ifndef EDITONLY
        ison = (ap->status & ONBITS);
        if (!wason && ison) {
```

```
                if (!(mdb[mol].m_flags & SETCNT))
                        mdb[mol].m_active++;
                if (!(mdb[mol].m_flags & SETCM)) {
                        mdb[mol].m_com[0] += ap->x;
                        mdb[mol].m_com[1] += ap->y;
                        mdb[mol].m_com[2] += ap->z;
                }
        } else if (wason && lison) {
                if (!(mdb[mol].m_flags & SETCNT))
                        mdb[mol].m_active--;
                if (!(mdb[mol].m_flags & SETCM)) {
                        mdb[mol].m_com[0] -= ap->x;
                        mdb[mol].m_com[1] -= ap->y;
                        mdb[mol].m_com[2] -= ap->z;
                }
        }
        if (ap->status & BONDBIT) {
                for (i = 0; i < nbond[mol]; i++)
                        for (j = 0; j < 2; j++)
                                if (ap->x == bonds[mol][i].b_x[j]
                                && ap->y == bonds[mol][i].b_y[j]
                                && ap->z == bonds[mol][i].b_z[j])
                                        bonds[mol][i].b_disp[j] =
                                                (ap->status & SHOWBIT);
        }
#endif
        return;
        /* ARGSUSED */
}

/*
 * setupredraw - sets the editing routine to null
 */
setupredraw(which)
int     which;
{
        p_off = which;
        prepres = NULL;
        editres = selres;
        realedit = null;
}

/*
 * setupcolor - records color for the color editing routine
 */
setupcolor(colnum, optnum, which)
int     colnum, optnum, which;
{
        p_on = colnum;
        n_on = optnum;
        p_off = which;
        editres = selres;
        if (p_off & SCOLOR)
                prepres = colores;
        else
                prepres = NULL;
        realedit = chcolor;
}

/*
 * colores - preparing a residue for being colored
 */
colores()
{
        if (mdb[mol].s_fd < 0) {
                surfdata = NULL;
                return;
        }
        if (mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                assert (mreada(mdb[mol].s_fd) != -1);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        } else
                surfdata = NULL;
}

/*
 * chcolor - color editing routine
 */
chcolor(selected, index, atname)
int     selected, index;
char    *atname;
{
        register        i, ndx;
        int     col;
        struct  srftab_def      stab[ESPRNG];   /* Maximum table size */
        int     max, min;
        float   fac;
        struct  mdbs_def        *db;
        long    addr;
        register struct atom_def                *ap;

        if (selected) {
                ap = &atomdata[index];
                if (n_on != -1)
```

```
                        col = p_on + (ap->tempfac - tfmin) / (tfmax - tfmin) *
                                (n_on - p_on);
                else
                        col = p_on;
                if (p_off & BCOLOR) {
                        ap->color = col;
#ifndef EDITONLY
                        if (ap->status & BONDBIT) {
                                for (i = 0; i < nbond[mol]; i++)
                                        if (ap->x == bonds[mol][i].b_x[1] &&
                                            ap->y == bonds[mol][i].b_y[1] &&
                                            ap->z == bonds[mol][i].b_z[1])
                                                bonds[mol][i].b_color = col
                        }
#endif
                }
                if (p_off & LCOLOR)
                        ap->lcolor = col;
                if (p_off & VCOLOR)
                        ap->scolor = col;

                if (p_off & SCOLOR && surfdata != NULL) {
                        db = &mdb[mol];
                        ndx = mseeka(db->s_fd, atname);
                        if (ndx != -1 && surfdata[ndx].s_nentry > 0) {
                                min = (espmin - db->m_espmin) /
                                        (db->m_espmax - db->m_espmin) *
                                        ESPRNG - ESPMID;
                                max = (espmax - db->m_espmin) /
                                        (db->m_espmax - db->m_espmin) *
                                        ESPRNG - ESPMID;
                                if (n_on != -1 && max != min)
                                        fac = (n_on - p_on) /
                                                (double) (max - min);
                                else
                                        fac = 0;

                                GETLONG(addr, surfdata[ndx].s_addr);
                                (void) lseek(db->sd_fd, addr, 0);
                                (void) read(db->sd_fd, (char *) stab,
                                        surfdata[ndx].s_nentry *
                                        sizeof (struct srftab_def));
                                for (i = 0; i < surfdata[ndx].s_nentry; i++) {
                                        if (stab[i].st_pot <= min ||
                                            stab[i].st_pot >= max)
                                                continue;
                                        if (n_on != -1 && max != min)
                                                stab[i].st_color = p_on + fac *
                                                  (stab[i].st_pot - min);
                                        else
                                                stab[i].st_color = p_on;
                                }
                                (void) lseek(db->sd_fd, addr, 0);
                                (void) write(db->sd_fd, (char *) stab,
                                        surfdata[ndx].s_nentry *
                                        sizeof (struct srftab_def));
                        }
                }
        }
        /* ARGSUSED */
}

/*
 * setupsurface - prepares editor to change surface status of atoms
 */
setupsurface()
{
        int     ndx;

        ndx = notcom ? 1 : 0;
        p_on  = com_bits[curcom][ndx][0];
        p_off = com_bits[curcom][ndx][1];

        editres = selres;
        prepres = colores;          /* Always grab the surface stuff */
        realedit = setsurf;
}

/*
 * setsurf - set the surface attributes
 */
setsurf(selected, index, atname)
int     selected, index;
char    *atname;
{
        register        i, ndx;
        int     ndot;
        struct  srftab_def      stab[ESPRNG];   /* Maximum table size */
        int     max, min;
        struct  mdbs_def        *db;
        long    addr;

        /* Ignore non-existent atoms */
        if (!(atomdata[index].status & EXISTBIT) || !selected)
```

```
                return;

        /* Ignore atoms with no surface data */
        if (surfdata == NULL)
                return;
        db = &mdb[mol];
        ndx = mseeka(db->s_fd, atname);
        if (ndx == -1 || surfdata[ndx].s_nentry <= 0)
                return;

        /* Calculate the range of potentials selected */
        min = (espmin - db->m_espmin) / (db->m_espmax - db->m_espmin) * ESPR
                - ESPMID;
        max = (espmax - db->m_espmin) / (db->m_espmax - db->m_espmin) * ESP
                - ESPMID;

        /* Read the atom surface data */
        GETLONG(addr, surfdata[ndx].s_addr);
        (void) lseek(db->sd_fd, addr, 0);
        (void) read(db->sd_fd, (char *) stab,
                surfdata[ndx].s_nentry * sizeof (struct srftab_def));

        /* Set the bits and determine whether any dots would be
         * displayed if SURFBIT were on */
        ndot = 0;
        for (i = 0; i < surfdata[ndx].s_nentry; i++) {
                if (stab[i].st_pot < min || stab[i].st_pot > max) {
                        if ((stab[i].st_color & SIGNORE) == 0)
                                ndot++;
                        continue;
                }
                if (notcom)
                        stab[i].st_color |= SIGNORE;
                else {
                        stab[i].st_color &= ~SIGNORE;
                        ndot++;
                }
        }
        (void) lseek(db->sd_fd, addr, 0);
        (void) write(db->sd_fd, (char *) stab,
                surfdata[ndx].s_nentry * sizeof (struct srftab_def));

        /*
         * Flip the actual bits for two cases:
         * 1.   If we are turning surface on.
         * 2.   If there are not dots remaining after turning selected
         *      potentials off
         */
        if (!notcom || ndot == 0)
                flipbits(TRUE, index, atname);
}

#include "struct.i"

#ifndef  EDITONLY

#include "fixrot.i"
#include "coord.i"
#include "rotation.i"
#include "distance.i"
#include "angle.i"

#endif   EDITONLY
```

comtab.c

```
/* $Header: comtab.c,v 3.9 86/06/09 19:04:56 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                  9 Apr 1982
 * Release 2.0                  13 May 1983
 * Release 2.1                  5 Jul 1983
 * Release 2.2                  23 Aug 1983
 */

#include "editor.h"

#define  SH     SHOWBIT
#define  BR     BREAKBIT
#define  LA     LABELBIT
#define  SL     (LABELBIT|SHOWBIT)
#define  LB     (LABELBIT|BREAKBIT|SHOWBIT)
#define  SB     SURFBIT
#define  RB     ROTBIT
#define  DB     DISTBIT
#define  AB     ANGLEBIT
#define  NR     NORLBIT
#define  VB     VDWBIT
#define  CB     STARTBIT        /* CHAINBIT (should be SB but it was taken) *

struct   command_def     comtab[]        =       {
         GCOMWORD,       DISPLAY,        "display",      0,
         GCOMWORD,       CHAIN,          "chain",        0,
         GCOMWORD,       LABEL,          "label",        0,
         GCOMWORD,       SHOW,           "show",         0,
         SCOMWORD,       SURFACE,        "surface",      0,
         GCOMWORD,       RLABEL,         "rlabel",       0,
         GCOMWORD,       VDW,            "vdw",          0,
         SCOMWORD,       LINK,           "link",         0,
#ifndef  EDITONLY
         SCOMWORD,       FINROT,         "rotation",     0,
         SCOMWORD,       FINROT,         "frotation",    0,
         SCOMWORD,       BINROT,         "brotation",    0,
         SCOMWORD,       DIST,           "distance",     0,
         SCOMWORD,       REVERSE,        "reverse",      0,
         SCOMWORD,       FIXREV,         "fixreverse",   0,
         SCOMWORD,       ANGLE,          "angle",        0,
#endif   EDITONLY
         /* End of commands which have associated status bits */

         SCOMWORD,       PCOLOR,         "color",        0,
         NCOMWORD,       ALIAS,          "alias",        0,
         NCOMWORD,       PREAD,          "read",         0,
         NCOMWORD,       STOP,           "stop",         0,
         NCOMWORD,       OPEN,           "open",         0,
         NCOMWORD,       CHDIR,          "cd",           0,
         NCOMWORD,       SAVE,           "save",         0,
#ifndef  EDITONLY
         SCOMWORD,       GETCRD,         "getcrd",       0,
         SCOMWORD,       ALIGN,          "align",        0,
         SCOMWORD,       MATCH,          "match".        0,
         NCOMWORD,       SETCOM,         "setcom",       0,
         NCOMWORD,       VDWOPT,         "vdwopt",       0,
         SCOMWORD,       REDRAW,         "redraw",       0,
         NCOMWORD,       HALFBND,        "halfbond",     0,
#if GRAFENG == PS2 || GRAFENG == MPS
         NCOMWORD,       PICK,           "pick",         0,
#endif
#if GRAFENG == PS300
         SCOMWORD,       PICK,           "pick",         0,
#endif
#if GRAFENG == IRIS
         SCOMWORD,       CPK,            "cpk",          0,
         NCOMWORD,       MAPCOLORS,      "mapcolors",    0,
#endif
#endif   EDITONLY
         SCOMWORD,       ADDGRP,         "addgrp",       0,
         SCOMWORD,       SWAPAA,         "swapaa",       0,
         SCOMWORD,       ADDAA,          "addaa",        0,
         SCOMWORD,       SWAPNUC,        "swapna",       0,
         SCOMWORD,       ADDNUC,         "addna",        0,
         SCOMWORD,       DELGRP,         "delete",       0,
         NOMORECOM,      0,              0,              0
};

int      com_bits[][2][4]                =       {
         SH,BR,00,00,    BR,SL,00,00,            /* display */
         SH,BR,00,LB,    BR,00,00,00,            /* chain */
         LA,00,00,00,    00,LA,00,00,            /* label */
         SH,BR,BR,SL,    BR,SL,00,00,            /* show */
         SB,00,00,00,    00,SB,00,00,            /* surface */
         00,NR,00,00,    NR,00,00,00,            /* rlabel */
         VB,00,00,00,    00,VB,00,00,            /* vdw */
         00,CB,00,00,    CB,00,00,00,            /* link */
#ifndef  EDITONLY
         RB,00,00,00,    00,RB,00,00,            /* rotation */
         RB,00,00,00,    00,RB,00,00,            /* frotation */
         RB,00,00,00,    00,RB,00,00,            /* brotation */
```

comtab.c

```
        DB,00,00,00,    00,DB,00,00,    /* distance */
        RB,00,00,00,    00,RB,00,00,    /* reverse */
        RB,00,00,00,    00,RB,00,00,    /* fixreverse */
        AB,00,00,00,    00,AB,00,00     /* angle */
#endif  EDITONLY
};
```

drawpic.c

```
/* $Header: drawpic.c,v 3.44 86/09/25 13:31:48 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          16 May 1983
 * Release 2.2                          23 May 1983
 * Release 2.3                          1 Aug 1983
 * Release 2.4                          18 Aug 1983
 * Release 2.5                          26 Sep 1983
 * Converted to RCS                     October 1984
 */

#include "editor.h"

#if     GRAFENG == PS2
#include <ps2.h>
#define pushmatrix()        _doint(PUSHR)
#define popmatrix()         _doint(POPR)
#define setcolor(x)         ((x)==0? huesat(WHITE):huesat(x))
#endif

#if     GRAFENG == MPS
int     lastcall;           /* used by overflow() */
#define pushmatrix()        tpush()
#define popmatrix()         tpop()
#define setcolor(x)         ((x)==0? lcolor(&WHITE):lcolor(&(x), &FULSAT))
#endif

#if     GRAFENG == IRIS
#include <irisdefs.h>
#define hittag(x)
#define setcolor(x)         ir_srange((Colorindex)(x))
#define incoffset(x)        /* null */
#endif

#if     GRAFENG == PS300
#ifndef vms
#include <assert.h>
#include <ps300/gsr.h>
#else   vms
#include "MIDAS$H:assert.h"
#include "MIDAS$H:fakegsr.h"
#endif  vms
/*
 * someday setcolor() will be used with the
 * vector manager function to indicate to the
 * manager to end off a data block and allocate
 * another on the linked list with an imbeded
 * "set color i" color command in it.
 */
#define setcolor(x)         flushcrd()
#define incoffset(x)        /* null */
#endif  PS300

#ifdef  vms
#define rindex  strrchr
#endif

#define OBSIZE              12000           /* max size of incore OB array */
#define ERBITS              (ROTBIT | EXISTBIT)
#define D_BOND              1
#define D_LABEL             2
#define D_SURF              3

int     halfbond;

/* global variables for the various residue traversal routines */

char                *rseq, *rtype;
int                 molnum;
struct atom_def     *atoms;

static  char    newlink, newchain;
static  char    startch, rlabeled;
static  int     firstmode, firstcolor;
static  int     curcolor;
static  ps_t    nx, ny, nz, lx, ly, lz;
static  ps_t    surf[ESPMAXPTS][3];
static  struct  surf_def        *surfs;
static  struct  srftab_def      stab[ESPRNG];

#if     GRAFENG == IRIS
int     mat_stack = 0;      /* keep track of push/pop matrix */
#endif

#if     GRAFENG == PS300
static  int     subnum, drawtype;
static  char    curobjname[128], baseobjname[128];
static  char    namebuf[128];
static  int     segnum[MAXINROT + 1];
static  int     depth;
```

drawpic.c

```
static  int     firstcoord;
static  int     vecnum, curvecnum;
char            *map_name();

#define sizeofob()          0
#define copyob(x)           ((char *) NULL)
#ifdef  DEBUG
#define tabto(z)            /* null */
#else
#define tabto(z)            { register i; for(i=0;i<(z)+1;i++)pprintf("\t"); }
#endif
#endif  PS300

#if     GRAFENG == IRIS
#define sizeofob()          0
#define copyob(x)           ((char *) NULL)
#endif

#if     GRAFENG == PS2 || GRAFENG == MPS
static int          obsize;
static ps_t         ob[OBSIZE + 1];
int                 overflow();
char                *copyob();
#endif

/*
 * drawpic - output the selected parts of the selected models with the
 * adjusted window to PicSys memory (via the storage module)
 */
drawpic()
{
        register            i;
        int         j;
        long        natom;
#if     GRAFENG != PS300
        struct input_def            in;
#if     GRAFENG != IRIS
        int         cnt;
#endif
#endif

        /* Find number of active atoms */
        natom = 0;
        j = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (mdb[i].m_fname == NULL)
                        continue;
                if (mdb[i].m_flags & PSOB && mdb[i].m_active == 0)
                        j++;
                else
                        natom += mdb[i].m_active;
        }

        /* Create the picture */
        if (natom > 0 || j > 0) {
                drawssbond();

#if     GRAFENG != PS300
                in.i_type = S_DRAW;
#ifdef  ONEPROC
                /* "storage" routine does all the work */
                process(&in, (char *) NULL);
#else   ONEPROC
                putps();
                (void) write(store_fd[1], (char *) &in, sizeof in);
                (void) write(store_fd[1], (char *) mdb,
                        sizeof (struct mdbs_def)*MAXMOD);
                (void) write(store_fd[1], (char *) distance,
                        sizeof(struct dist_def)*MAXDIST);
                (void) write(store_fd[1], (char *) angles,
                        sizeof(struct angle_def)*MAXANGLE);

                (void) read(store_fd[0], (char *) mdb,
                        sizeof (struct mdbs_def)*MAXMOD);
                (void) read(store_fd[0], (char *) distance,
                        sizeof (struct dist_def)*MAXDIST);
                (void) read(store_fd[0], (char *) angles,
                        sizeof(struct angle_def)*MAXANGLE);
                (void) read(store_fd[0], (char *) &in, sizeof in);
                getps();
#endif  ONEPROC
                if (in.i_type)
                        reply("Display overflow.\n");

                if (natom == 0)
                        reply("Warning: no reference point.\n");
#else   PS300
                if (ndist > 0) {
                        reset_disttab();
                        for (i = 0; i < MAXDIST; i++) {
                                if (distance[i].d_type == INACTIVE)
                                        continue;
                                if (distance[i].d_type & HOLDALL)
                                        continue;
```

```
                                        disttab(distance[i].d_vecname[0],
                                                distance[i].d_vecname[1],
                                                distance[i].d_vecnum[0],
                                                distance[i].d_vecnum[1]);
                        }
                }
#endif
        } else {
                /* "null" picture */
#if     GRAFENG == PS2
                makepe('pc', &cnt);
                tran(0, 0, 0);
                stoppe();
#endif
#if     GRAFENG == MPS
                udelet(&'bd');
                udelet(&'pc');
                ubegin(&'pc', &cnt);
                ttran(&0, &0, &0);
                uend();
                ubegin(&'bd', &cnt);
                ttran(&0, &0, &0);
                uend();
#endif
#if     GRAFENG == IRIS
                makeobj(O_ALL);
                closeobj();
#endif
#if     GRAFENG == PS300
                /* Nothing to do */
#endif
        }

        (void) sprintf(buf, "%ld atoms/surface %d objects displayed.\n",
                natom, );
        reply(buf);
        anychg = FALSE;
}

#if     GRAFENG == PS300
/*
 * reset_disttab:
 *      Reset the distance table
 */
reset_disttab()
{
        PPurge();
        sleep(1);
        PSndReal(1000.0, 6, "disttab_sync");
        PSndReal(0.0, 5, "disttab_sync");
        PSndStr("none", 4, "disttab_sync");
        PSndStr("none", 3, "disttab_sync");
        PSndFix(0, 2, "disttab_sync");
        PSndStr("distance", 1, "disttab_sync");
}

/*
 * disttab:
 *      Send data to distance table function
 */
disttab(name1, name2, vec1, vec2)
char    *name1, *name2;
int     vec1, vec2;
{
        PSndReal(1000.0, 6, "disttab_sync");
        PSndReal(0.0, 5, "disttab_sync");
        PSndStr(name1, 4, "disttab_sync");
        PSndStr(name2, 3, "disttab_sync");
        PSndFix(vec1, 2, "disttab_sync");
        PSndFix(vec2, 1, "disttab_sync");
}
#endif

/*
 * drawbond - construct the PS2 object of the given residue's bonds
 * and sends it to MIDAS-STORE; called from edit()
 */
drawbond(mol, seq, type, atomdat)
int     mol;
char    *seq, *type;
struct  atom_def        *atomdat;
{
        struct  input_def       in;
        int     bvisit(), bagain();
#if     GRAFENG == PS300
        register        l;
#endif
#if     GRAFENG == IRIS
        int     pvisit(), pagain();
        register unsigned long  name;
        register Object         o;
#endif

        /* Set up global variables */
```

```
        molnum = mol;
        rseq = seq;
        rtype = type;
        atoms = atomdat;
        firstcolor = -1;
        curcolor = -1;
        firstmode = FM_NONE;
        newlink = FALSE;
        newchain = FALSE;                       /* Start with what is appropriate */
        startch = FALSE;
#if     GRAFENG == IRIS
        mat_stack = 0;
#endif

        initcrdbuf();
#ifdef  DEBUG
        if (debug & D_PG)
                fprintf(stderr, "Bond residue %s %s\n", rseq, rtype);
#endif

        /* Make the object for this residue */
#if     GRAFENG == PS2
        prepmkob();
        makeob(ob, OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, bvisit, bagain);
        flushcrd();
        stopob();
#endif
#if     GRAFENG == MPS
        prepmkob();
        makeob(ob, &OBSIZE, &obsize, overflow);
        setbase();
        (void) mtrav(mdb[molnum].m_fd, bvisit, bagain);
        flushcrd();
        lastcall = TRUE;
        stopob();
        voidpp();
#endif
#if     GRAFENG == IRIS
        o = numobj(molnum, rseq, O_BOND);
        makeobj(o);
        (void) mtrav(mdb[molnum].m_fd, bvisit, bagain);
        flushcrd();
        closeobj();
        if (curcolor == -1 && mat_stack == 0)           /* nothing is in this object
                delobj(o);
        else {                                  /* make the picking object */
                makeobj(numobj(molnum, rseq, O_PNTS));
                name = ra_name_to_num(rseq);
                loadname((short) name);
                pushname((short) (name >> 16));
                pushname(0);        /* dummy name so pvisit() can just loadname() */
                (void) mtrav(mdb[molnum].m_fd, pvisit, pagain);
                popname();
                popname();
                closeobj();
        }
#endif
#if     GRAFENG == PS300
        firstcoord = TRUE;
        curvecnum = vecnum = 0;
        subnum = 1;
        depth = 0;
        segnum[0] = 1;
        (void) sprintf(baseobjname, "m%dr%s", molnum, map_name(rseq));
        (void) strcpy(curobjname, baseobjname);
        in.i_type = S_GETLEVEL;
        in.i_molnum = molnum;
        strcpy(in.i_name, rseq);
#ifdef  ONEPROC
        process(&in, (char *) NULL);
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        for (i = 0; i < in.i_ndata; i++) {
                char    buf[10];

                (void) sprintf(buf, ".sub%d", subnum++);
                strcat(curobjname, buf);
                segnum[++depth] = 1;
        }
        drawtype = D_BOND;
        sprintf(namebuf, "%s.bond%d", curobjname, segnum[depth]++);
        (void) mtrav(mdb[molnum].m_fd, bvisit, bagain);
        flushcrd();
        if (firstcoord)
                PNameNil(namebuf);
        else
                MVecEnd();
#endif  PS300

        /* Send the stuff to 'store' for safe keeping */
        in.i_type = S_BREPLACE;
        in.i_molnum = molnum;
```

```
        strncpy(in.i_name, rseq, RES_SEQ_SIZE);
        in.i_size = sizeofob();
        in.i_ndata = getoffset();
        in.i_newchain = startch;
        in.i_firstmode = firstmode;
        in.i_x = nx;
        in.i_y = ny;
        in.i_z = nz;
        in.i_lx = lx;
        in.i_ly = ly;
        in.i_lz = lz;
        in.i_color = firstcolor;
        in.i_lcolor = curcolor;
#if     GRAFENG == PS300
        in.i_halfbond = halfbond;
#endif
#ifdef  ONEPROC
        process(&in, copyob(ob));
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        sendob(store_fd[1], ob, sizeof ob);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        anychg = TRUE;
}

#if     GRAFENG == IRIS
/* ARGSUSED */
static
pvisit(db, index, ischelf, islink, nsons, firstime)
int     db;
int     index, ischelf, islink, nsons, firstime;
{
        register struct atom_def        *data;
        char                            atname[AT_NAME_SIZE+1];

        if (!firstime)
                return;

        data = &atoms[index];
        if (!(data->status & EXISTBIT))
                return;

        (void) matom(db, index, atname);
        loadname((short) ra_name_to_num(atname));
        pnt(MAPCRD(data->x), MAPCRD(data->y), MAPCRD(data->z));

        /* Check for rotations */
        if (data->status & ROTBIT) {
                pushrot(db, index);
                if (nsons == 0)
                        poprot(db, index);
        }
}

/* ARGSUSED */
static
pagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        register struct atom_def        *data;

        data = &atoms[index];
        if (!(data->status & EXISTBIT))
                return;

        /* Check rotation (if there are no more branches) */
        if (data->status & ROTBIT && nsons == 0)
                poprot(db, index);
}

#endif  GRAFENG == IRIS

/*
 * bvisit - front half of the object-creation traversal routine
 */
/* ARGSUSED */
static
bvisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
        register        struct  atom_def        *data;
        char    atname[AT_NAME_SIZE+1];
        ps_t    x, y, z;
        ps_t    midx, midy, midz;
        int     mode;

        /* Initialize and check for chaining status */
#ifdef  DEBUG
        if (debug & LD_PG)
                fprintf(stderr, "Atom %d (%o,%d)\n", index,
                        atoms[index].status, mchain(db, index));
```

```
#endif
        data = &atoms[index];
        if (data->status & STARTBIT && firstime)
                startch = newchain = TRUE;
        if (!(data->status & EXISTBIT))
                return;

#ifdef  DEBUG
        if (data->status & (DISTBIT | ANGLEBIT) || debug & LD_PG)
#else
        if (data->status & (DISTBIT | ANGLEBIT))
#endif
                (void) matom(db, index, atname);

        /* Get the Picture System coordinates */
        x = MAPCRD(data->x);
        y = MAPCRD(data->y);
        z = MAPCRD(data->z);

        /* Check if we have to break the chain here */
        if (curcolor == -1 && data->status & BREAKBIT)
                startch = TRUE;

        /* If not selected, don't draw the atom */
        if (data->status & SHOWBIT) {

#if     GRAFENG == PS300
                if (firstcoord) {
                        MVecBegn(namebuf, P_Item, vecnum);
                        curvecnum = 0;
                        firstcoord = FALSE;
                }
#endif  PS300
                /* Decide whether to move or to draw */
                if ((data->status & BREAKBIT || newchain) && firstime) {
                        mode = MOVETO;
                        /* Change the color if necessary */
                        if (curcolor != data->color) {
                                /* Record the first atom's location */
                                if (curcolor == -1) {
                                        nx = x;
                                        ny = y;
                                        nz = z;
                                        firstmode = FM_MOVE;
                                        firstcolor = data->color;
                                }
                                curcolor = data->color;
                                flushcrd(); /* not always needed for ps300 */
                                setcolor(curcolor);
                                incoffset(ITEMSIZE);
                        }
                        newchain = FALSE;
#ifdef  DEBUG
                        if (debug & LD_PG)
                                fprintf(stderr, "Moveto %s\n", atname);
#endif
                } else {
                        mode = LINETO;
                        /* Change the color if necessary */
                        if (curcolor != data->color) {
                                /* Record the first atom's location */
                                if (curcolor == -1) {
                                        nx = x;
                                        ny = y;
                                        nz = z;
                                        firstmode = FM_LINE;
                                        firstcolor = data->color;
#if     GRAFENG == PS300
                                        /* Have to do this because the ps300
                                         * does not permit a lineto at the
                                         * beginning of a vector list.
                                         * It will be taken care of by
                                         * store.c */
                                        mode = MOVETO;
#endif
                                } else if (halfbond) {
                                        midx = (lx + x) / 2;
                                        midy = (ly + y) / 2;
                                        midz = (lz + z) / 2;
                                        savecrd(LINETO, midx, midy, midz,
                                                curcolor);
#if     GRAFENG == PS300
                                        vecnum++;
                                        curvecnum++;
#endif
                                }
                                curcolor = data->color;
                                flushcrd(); /* not always needed for ps300 */
                                setcolor(curcolor);
                                incoffset(ITEMSIZE);
                        }
#ifdef  DEBUG
                        if (debug & LD_PG)
                                fprintf(stderr, "Lineto %s\n", atname);
```

```
#endif
                }
                savecrd(mode, x, y, z, curcolor);
#if     GRAFENG == PS300
                vecnum++;
                curvecnum++;
#endif
                lx = x;
                ly = y;
                lz = z;
        }

        /* The rest of the stuff need only be done once per atom */
        if (!firstime)
                return;

        /* Check for distances and angles */
        if (data->status & DISTBIT)
                markdist(index, molnum, rseq, atname, getoffset(), x, y, z);
        if (data->status & ANGLEBIT)
                markangle(index, molnum, rseq, atname, getoffset(), x, y, z);

        /* Check for rotations */
        if (data->status & ROTBIT) {
                pushrot(db, index);
                if (nsons == 0)
                        poprot(db, index);
        }
}

/*
 * markdist - mark an atom for distance calculation
 */
/* ARGSUSED */
markdist(index, mol, seq, atname, cnt, x, y, z)
int     index, mol;
char    *seq, *atname;
int     cnt;
ps_t    x, y, z;
{
        register        struct  dist_def        *dp;
        register        int     j;
        register        struct  atom_def        *data;

        data = &atoms[index];
        for (dp = distance; dp < &distance[MAXDIST]; dp++) {
                if (dp->d_type == INACTIVE)
                        continue;
                for (j = 0; j < 2; j++) {
                        if (dp->d_model[j] != mol
                        || smatch(seq, dp->d_reseq[j]) != 0
                        || amatch(atname, dp->d_atom[j]) != 0)
                                continue;

                        if ((data->status & SHOWBIT) == 0) {
                                dp->d_type |= HOLD(j);
                                continue;
                        }

                        dp->d_type &= ~HOLD(j);
#if     GRAFENG == PS2 || GRAFENG == MPS
                        dp->d_obloc[j] = cnt - ITEMSIZE;
#endif
#if     GRAFENG == PS300
                        if (dp->d_vecnum[j] == curvecnum
                        && dp->d_vecname[j] != NULL
                        && strcmp(dp->d_vecname[j], namebuf) == 0)
                                continue;
                        if (dp->d_vecname[j] != NULL)
                                free(dp->d_vecname[j]);
                        dp->d_vecname[j] = malloc(strlen(namebuf) + 1);
                        strcpy(dp->d_vecname[j], namebuf);
                        dp->d_vecnum[j] = curvecnum;
                        newdist = TRUE;
#endif
#if     GRAFENG == IRIS
                        callfunc(DISTFUNC, 2L, (long) (dp - distance),
                                (long) j);
                        dp->d_x[j] = x;
                        dp->d_y[j] = y;
                        dp->d_z[j] = z;
#endif
                }
        }
}

/*
 * markangle - mark an atom for angle calculation
 */
/* ARGSUSED */
markangle(index, mol, seq, atname, cnt, x, y, z)
int     index, mol;
char    *seq, *atname;
int     cnt;
```

```
pe_t    x, y, z;
{
        register        struct  angle_def       *ap;
        register        ];
        register        struct  atom_def        *data;

        data = &atoms[index];
        for (ap = angles; ap < &angles[MAXANGLE]; ap++) {
                if (ap->a_type == INACTIVE)
                        continue;
                for (j = 0; j < ap->a_natom; j++) {
                        if (ap->a_model[j] != mol
                        || smatch(seq, ap->a_resseq[j]) != 0
                        || amatch(atname, ap->a_atom[j]) != 0)
                                continue;

                        if ((data->status & SHOWBIT) == 0) {
                                ap->a_type |= HOLD(j);
                                continue;
                        }

                        ap->a_type &= ~HOLD(j);
#if     GRAFENG == PS2 || GRAFENG == MPS
                        ap->a_obloc[j] = cnt - ITEMSIZE;
#endif
#if     GRAFENG == PS300
                        /* Punt */
#endif
#if     GRAFENG == IRIS
                        callfunc(ANGLEFUNC, 2L, (long) (ap - angles), (long) j)
                        ap->a_x[j] = x;
                        ap->a_y[j] = y;
                        ap->a_z[j] = z;
#endif
                }
        }
}

/*
 * bagain - second half of the object-creation routine
 */
/* ARGSUSED */
static
bagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        register        struct  atom_def        *data;

        /* Initialize link information */
        if (islink)
                newlink = TRUE;
        data = &atoms[index];
        if (!(data->status & EXISTBIT))
                return;

        /* Move back only if there are more things to draw */
        if ((data->status & SHOWBIT) && (nsons > 0 || newlink)) {
                lx = MAPCRD(data->x);
                ly = MAPCRD(data->y);
                lz = MAPCRD(data->z);
#ifdef  DEBUG
                if (debug & LD_PG) {
                        char    atname[AT_NAME_SIZE + 1];

                        (void) matom(db, index, atname);
                        fprintf(stderr, "MoveTo %s\n", atname);
                }
#endif
                /* Change the color if necessary */
                if (halfbond && curcolor != data->color) {
                        curcolor = data->color;
                        flushcrd(); /* not always needed for ps300 */
                        setcolor(curcolor);
                        incoffset(ITEMSIZE);
                }
                savecrd(MOVETO, lx, ly, lz, curcolor);
#if     GRAFENG == PS300
                vecnum++;
                curvecnum++;
#endif
                newlink = FALSE;
        }

        /* Check rotation (if there are no more branches) */
        if (data->status & ROTBIT && nsons == 0)
                poprot(db, index);
}

/*
 * drawsurf - construct the PS2 object of the given residue's surface
 * and sends it to MIDAS-STORE
 */
drawsurf(mol, seq, type, atomdat, surfdat)
```

```
int     mol;
char    *seq, *type;
struct  atom_def        *atomdat;
struct  surf_def        *surfdat;
{
        struct  input_def       in;
        int     svisit(), sagain();
#if     GRAFENG == PS300
        register        i;
#endif
#if     GRAFENG == IRIS
        register Object     o;
#endif

#ifdef  DEBUG
        if (debug & D_PG)
                fprintf(stderr, "Surf residue %s %s\n", seq, type);
#endif

        /* Set up global variables */
        molnum = mol;
        rseq = seq;
        rtype = type;
        atoms = atomdat;
        surfs = surfdat;
        curcolor = -1;
#if     GRAFENG == IRIS
        mat_stack = 0;
#endif

        /*
         * Make the object for this residue.
         *
         * We first set the software copy of the LG
         * status words to dot mode.  This prevents
         * the draw3d call in svisit from outputing
         * any LG status, which would otherwise
         * be done for every residue.  We later then
         * do a single texture(DOTS) call when we
         * create the PicSys resident UDL.
         *
         * On the MPS things are a little different.
         * Here the d3data() routine UNCONDITIONALLY
         * outputs a new LG status whenever fsm1=DOTTED
         * then restores the previous status after the
         * draw is completed.  Thus there are 4 extra
         * refresh words for every residue in the MPS!
         * There still is a purpose for fixlg call
         * however, since now we must insure that the
         * LG is running at full speed.
         * Too bad about this, but nothing can be done
         * short of modifying the E&S software.
         */
#if     GRAFENG == PS2
        prepmkob();
        fixlg(DOTTED);
        makeob(ob, OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, svisit, sagain);
        stopob();
        fixlg(!DOTTED);
#endif
#if     GRAFENG == MPS
        prepmkob();
        fixlg(DOTTED);
        makeob(ob, &OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, svisit, sagain);
        lastcall = TRUE;
        stopob();
        voidpp();
        fixlg(!DOTTED);
#endif
#if     GRAFENG == IRIS
        o = numobj(molnum, rseq, O_SURFACE);
        makeobj(o);
        (void) mtrav(mdb[molnum].m_fd, svisit, sagain);
        closeobj();
        if (curcolor == -1 && mat_stack == 0)           /* nothing in this object */
                delobj(o);
#endif
#if     GRAFENG == PS300
        firstcoord = TRUE;
        subnum = 1;
        depth = 0;
        segnum[0] = 1;
        (void) sprintf(baseobjname, "m%dr%s", molnum, map_name(rseq));
        (void) strcpy(curobjname, baseobjname);
        in.i_type = S_GETLEVEL;
        in.i_molnum = molnum;
        strcpy(in.i_name, rseq);
#ifdef  ONEPROC
        process(&in, (char *) NULL);
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        (void) read(store_fd[0], (char *) &in, sizeof in);
```

```
#endif
        for (i = 0; i < in.i_ndata; i++) {
                char    buf[10];

                (void) sprintf(buf, ".sub%d", subnum++);
                strcat(curobjname, buf);
                segnum[++depth] = 1;
        }
        drawtype = D_SURF;
        sprintf(namebuf, "%s.surf%d", curobjname, segnum[depth]++);
        (void) mtrav(mdb[molnum].m_fd, svisit, sagain);
        if (firstcoord)
                PNameNil(namebuf);
        else
                MVecEnd();
#endif  PS300

        /* Send the stuff to 'store' for safe keeping */
        in.i_type = S_SREPLACE;
        in.i_molnum = molnum;
        strncpy(in.i_name, rseq, RES_SEQ_SIZE);
        in.i_size = sizeofob();
        if (in.i_size > 65536)
                fprintf(stderr, "Panic: object size = %d > 64K\n", in.i_size);
#ifdef  ONEPROC
        process(&in, copyob(ob));
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        sendob(store_fd[1], ob, sizeof ob);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        anychg = TRUE;
}

/*
 * svisit - Surface visiting routine
 */
/* ARGSUSED */
static
svisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
        register        i;
        register        struct  atom_def        *data;
        int     ndx, npoints, offset;
        char    atname[AT_NAME_SIZE+1];
        long    addr;

        /* Initialize and check for chaining status */
#ifdef  DEBUG
        if (debug & LD_PG)
                fprintf(stderr, "Atom %d (%o,%d)\n", index,
                        atoms[index].status, mchain(db, index));
#endif
        data = &atoms[index];
        if (!(data->status & EXISTBIT) || !firstime)
                return;

        /* Check for rotations */
        if (data->status & ROTBIT)
                pushrot(db, index);

        /* Is surface to be displayed ? */
        if (data->status & SURFBIT && surfs != NULL) {
                (void) matom(db, index, atname);
                ndx = mseeka(mdb[molnum].s_fd, atname);
                if (ndx != -1 && surfs[ndx].s_nentry > 0) {
#if     GRAFENG == PS300
                        if (firstcoord) {
                                MVecBegn(namebuf, P_Dots, 0);
                                firstcoord = FALSE;
                        }
#endif  PS300
                        GETLONG(addr, surfs[ndx].s_addr);
                        (void) lseek(mdb[molnum].sd_fd, addr, 0);
                        (void) read(mdb[molnum].sd_fd, (char *) stab,
                          surfs[ndx].s_nentry *
                          sizeof (struct srftab_def));
                        npoints = 0;
                        for (i = 0; i < surfs[ndx].s_nentry; i++)
                                npoints += stab[i].st_npts;
                        (void) read(mdb[molnum].sd_fd, (char *) surf,
                          npoints * 3 * sizeof (ps_t));
                        npoints = 0;
                        offset = 0;
                        for (i = 0; i < surfs[ndx].s_nentry; i++) {
                                if (stab[i].st_npts <= 0
                                  || (stab[i].st_color & SIGNORE))
                                        continue;
                                if (curcolor != stab[i].st_color) {
                                        if (npoints > 0) {
#if     GRAFENG == PS2
                                                draw3d(surf[offset], npoints,
```

```
#endif                                                  DOTTED, OFFSET);
#if     GRAFENG == MPS
                                                        d3data(surf[offset], &npoi
                                                          &DOTTED, &ABSOLU
#endif
#if     GRAFENG == IRIS
                                                        drawpnts(surf[offset],
                                                                npoints);
#endif
#if     GRAFENG == PS300
                                                        drawdots(surf[offset],
                                                          npoints, curcolor);
#endif
                                                        offset += npoints;
                                                        npoints = 0;
                                                }
                                                curcolor = stab[i].st_color;
                                                setcolor(curcolor);
                                        }
                                        npoints += stab[i].st_npts;
                                }
                                if (npoints > 0)
#if     GRAFENG == PS2
                                        draw3d(surf[offset], npoints,
                                                DOTTED, OFFSET);
#endif
#if     GRAFENG == MPS
                                        d3data(surf[offset], &npoints,
                                                &DOTTED, &ABSOLUTE);
#endif
#if     GRAFENG == IRIS
                                        drawpnts(surf[offset], npoints);
#endif
#if     GRAFENG == PS300
                                        drawdots(surf[offset], npoints, curcolor);
#endif
                        }
                }
                if (data->status & VDWBIT) {
                        npoints = mksurf(db, index, islink, ischief, rseq, rtype, surf);
                        if (npoints > 0) {
#if     GRAFENG == PS300
                                if (firstcoord) {
                                        MVecBegn(namebuf, P_Dots, 0);
                                        firstcoord = FALSE;
                                }
#endif  PS300
                                if (curcolor != data->scolor) {
                                        curcolor = data->scolor;
                                        setcolor(curcolor);
                                }
#if     GRAFENG == PS2
                                draw3d(surf[0], npoints, DOTTED, OFFSET);
#endif
#if     GRAFENG == MPS
                                d3data(surf[0], &npoints, &DOTTED, &ABSOLUTE);
#endif
#if     GRAFENG == IRIS
                                drawpnts(surf[0], npoints);
#endif
#if     GRAFENG == PS300
                                drawdots(surf[0], npoints, curcolor);
#endif
                        }
                }
                if (data->status & ROTBIT && nsons == 0)
                        poprot(db, index);
}

/*
 * sagain - second half of the object-creation routine
 */
/* ARGSUSED */
static
sagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        /* Check rotation (if there are no more branches) */
        if ((atoms[index].status & ERBITS) == ERBITS && nsons == 0)
                poprot(db, index);
}

/*
 * drawlabel - construct the PS2 object of the given residue's label
 * and sends it to MIDAS-STORE
 */
drawlabel(mol, seq, type, atomdat)
int     mol;
char    *seq, *type;
struct  atom_def        *atomdat;
{
```

```
        struct  input_def       in;
        int     lvisit(), lagain();
#if     GRAFENG == PS300
        register        i;
#endif
#if     GRAFENG == IRIS
        register Object     o;
#endif

#ifdef  DEBUG
        if (debug & D_PG)
                fprintf(stderr, "Label residue %s %s\n", seq, type);
#endif

        /* Set up global variables */
        molnum = mol;
        rseq = seq;
        rtype = type;
        atoms = atomdat;
        rlabeled = FALSE;
        curcolor = -1;
#if     GRAFENG == IRIS
        mat_stack = 0;
#endif

        /* Make the object for this residue */
#if     GRAFENG == PS2
        prepmkob();
        makeob(ob, OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, lvisit, lagain);
        stopob();
#endif
#if     GRAFENG == MPS
        prepmkob();
        makeob(ob, &OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, lvisit, lagain);
        lastcall = TRUE;
        stopob();
        voidpp();
#endif
#if     GRAFENG == IRIS
        o = numobj(molnum, rseq, O_LABEL);
        makeobj(o);
        (void) mtrav(mdb[molnum].m_fd, lvisit, lagain);
        closeobj();
        if (curcolor == -1 && mat_stack == 0)           /* nothing in this object */
                delobj(o);
#endif
#if     GRAFENG == PS300
        firstcoord = TRUE;
        subnum = 1;
        depth = 0;
        segnum[0] = 1;
        (void) sprintf(baseobjname, "m%dr%s", molnum, map_name(rseq));
        (void) strcpy(curobjname, baseobjname);
        in.i_type = S_GETLEVEL;
        in.i_molnum = molnum;
        strcpy(in.i_name, rseq);
#ifdef  ONEPROC
        process(&in, (char *) NULL);
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        for (i = 0; i < in.i_ndata; i++) {
                char    buf[10];

                (void) sprintf(buf, ".sub%d", subnum++);
                strcat(curobjname, buf);
                segnum[++depth] = 1;
        }
        drawtype = D_LABEL;
        sprintf(namebuf, "%s.label%d", curobjname, segnum[depth]++);
        (void) mtrav(mdb[molnum].m_fd, lvisit, lagain);
        if (firstcoord)
                PNameNil(namebuf);
        else
                MVecEnd();
#endif  PS300

        /* Send the stuff to 'store' for safe keeping */
        in.i_type = S_LREPLACE;
        in.i_molnum = molnum;
        strncpy(in.i_name, rseq, RES_SEQ_SIZE);
        in.i_size = sizeofob();
#ifdef  ONEPROC
        process(&in, copyob(ob));
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        sendob(store_fd[1], ob, sizeof ob);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        anychg = TRUE;
}
```

```
/*
 * lvisit - Label visiting routine
 */
/* ARGSUSED */
static
lvisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
        register        struct  atom_def        *data;
        ps_t    x, y, z;
        char    atname[AT_NAME_SIZE+1];
        extern int      ps300cm[];

        data = &atoms[index];
        if (!(data->status & EXISTBIT) || !firstime)
                return;

        /* Check for rotations */
        if (data->status & ROTBIT)
                pushrot(db, index);

        /* Is label required ? */
        if (data->status & LABELBIT) {
#if     GRAFENG == PS300
                if (firstcoord) {
                        /* PBeginS(namebuf); */
                        MVecBegn(namebuf, P_Item, 0);
                        firstcoord = FALSE;
                }
#endif
                x = MAPCRD(data->x);
                y = MAPCRD(data->y);
                z = MAPCRD(data->z);
                (void) matom(db, index, atname);
#ifdef  DEBUG
                if (debug & LD_PG)
                        fprintf(stderr, "Label %s\n", atname);
#endif
                if (rlabeled || (data->status & NORLBIT))
                        (void) sprintf(buf, "%s", atname);
                else {
                        (void) sprintf(buf, "%s%c(%s %s)%c", atname,
                                SUB, rseq, rtype, RSB);
                        rlabeled = TRUE;
                }
                        /* Change to label color */
                        if (curcolor != data->lcolor) {
                                curcolor = data->lcolor;
#ifndef GL2_3BUGS       /* this can only be defined on the IRIS */
                                setcolor(curcolor);
#else
                                ir_color(curcolor);
#endif
                        }
#if     GRAFENG == PS2
                        moveto(x, y, z);
                        text(buf);
#endif
#if     GRAFENG == MPS
                        damove(&x, &y, &z);
                        ctext(buf);
#endif
#if     GRAFENG == IRIS
                        cmov(x, y, z);
                        irlsout(buf);
#endif
#if     GRAFENG == PS300
                        /* PChars(NULL, x, y, z, 1.0, 0.0, buf); */
                        MLabel(x, y, z, ps300cm[curcolor], buf);
#endif
        }

        if (data->status & ROTBIT && nsons == 0)
                poprot(db, index);
}

/*
 * lagain - second half of the object-creation routine
 */
/* ARGSUSED */
static
lagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        /* Check rotation (if there are no more branches) */
        if ((atoms[index].status & ERBITS) == ERBITS && nsons == 0)
                poprot(db, index);
}

#if     GRAFENG == PS2 || GRAFENG == MPS || GRAFENG == IRIS
```

```
/*
 * drawtag - construct the PS2 object of the given residue's bonds
 * and sends it to MIDAS-STORE for identification
 */
drawtag(mol, seq, type, atomdat, hwx, hwy)
int     mol;
char    *seq, *type;
struct  atom_def        *atomdat;
int     hwx, hwy;
{
        struct  input_def       in;
        int     tagvisit(), tagagain();
#if     GRAFENG == IRIS
        register Object     o;
#endif

        /* Set up global variables */
        molnum = mol;
        rseq = seq;
        rtype = type;
        atoms = atomdat;
        newlink = FALSE;
        curcolor = -1;
#if     GRAFENG == IRIS
        mat_stack = 0;
#endif
#ifdef  DEBUG
        if (debug & D_PG)
                fprintf(stderr, "Bond residue %s %s\n", rseq, rtype);
#endif

        /* Make the object for this residue */
#if     GRAFENG == PS2
        prepmkob();
        makeob(ob, OBSIZE, &obsize, overflow);
        (void) mtrav(mdb[molnum].m_fd, tagvisit, tagagain);
        flushcrd();
        stopob();
#endif
#if     GRAFENG == MPS
        prepmkob();
        makeob(ob, &OBSIZE, &obsize, overflow);
        setbase();
        (void) mtrav(mdb[molnum].m_fd, tagvisit, tagagain);
        flushcrd();
        lastcall = TRUE;
        stopob();
        voidpp();
#endif
#if     GRAFENG == IRIS
        o = numobj(molnum, seq, O_BOND);
        makeobj(o);
        (void) mtrav(mdb[molnum].m_fd, tagvisit, tagagain);
        flushcrd();
        closeobj();
        if (curcolor == -1 && mat_stack == 0)
                delobj(o);
#endif

        /* Send the stuff to 'store' for safe keeping */
        in.i_type = S_IDENTATM;
        in.i_molnum = molnum;
        strncpy(in.i_name, rseq, RES_SEQ_SIZE);
        in.i_size = sizeofob();
        in.i_x = hwx;
        in.i_y = hwy;
#ifdef  ONEPROC
        process(&in, copyob(ob));
#else
        (void) write(store_fd[1], (char *) &in, sizeof in);
        sendob(store_fd[1], ob, sizeof ob);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        return(in.i_type - 1);                  /* Tag = index + 1 */
}

/*
 * tagvisit - front half of the object-creation traversal routine
 */
/* ARGSUSED */
static
tagvisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
        register        struct  atom_def        *data;
        ps_t    x, y, z;

        /* Initialize and check for chaining status */
        data = &atoms[index];
#ifdef  DEBUG
        if (debug & LD_PG)
                fprintf(stderr, "Atom %d (%o,%d)\n", index,
                        data->status, mchain(db, index));
```

```
#endif
        if (data->status & STARTBIT && firstime)
                startch = newchain = TRUE;
        if (!(data->status & EXISTBIT))
                return;

        /* Get the Picture System coordinates */
        x = MAPCRD(data->x);
        y = MAPCRD(data->y);
        z = MAPCRD(data->z);

        /* If not selected, don't draw the atom */
        if (data->status & SHOWBIT) {
                /* Tag this atom with the proper index */
                hittag(index + 1);
#if     GRAFENG == PS2
                moveto(x, y, z);
#endif
#if     GRAFENG == MPS
                damove(&x, &y, &z);
#endif
#if     GRAFENG == IRIS
                move(x, y, z);
#endif
        }

        /* The rest of the stuff need only be done once per atom */
        if (!firstime)
                return;

        /* Check for rotations */
        if (data->status & ROTBIT) {
                pushrot(db, index);
                if (nsons == 0)
                        poprot(db, index);
        }
}

/*
 * tagagain - second half of the object-creation routine
 */
/* ARGSUSED */
static
tagagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        register        struct  atom_def        *data;

        /* Initialize link information */
        data = &atoms[index];
        if (!(data->status & EXISTBIT))
                return;

        /* Check rotation (if there are no more branches) */
        if (data->status & ROTBIT && nsons == 0)
                poprot(db, index);
}
#endif  GRAFENG == PS2 || GRAFENG == MPS || GRAFENG == IRIS

/*
 * pushrot - push the rotation matrix on
 */
pushrot(db, index)
int     db, index;
{
        register        i;
#if     GRAFENG == PS2 || GRAFENG == MPS || GRAFENG == IRIS
        int     matnam;
#endif
        char    atname[AT_NAME_SIZE + 1];

        /* Get the atom name */
        (void) matom(db, index, atname);

        /* Check for backward rotation */
        i = findrot(molnum, rseq, atname, 2, 0);
        if (i >= 0) {
                if (atoms[index].status & SHOWBIT)
                        rotation[i].r_type &= ~HOLD(0);
                else
                        rotation[i].r_type |= HOLD(0);
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr, "Backward rotation %d.\n", i);
#endif
                flushcrd();

#if     GRAFENG != PS300
                popmatrix();
#if     GRAFENG == IRIS
                mat_stack--;
#endif
#else
```

```
                switch (drawtype) {

                case D_BOND :
                        if (firstcoord) {
                                PNameNil(namebuf);
                                sprintf(namebuf, "%s.bond%d",
                                        curobjname, segnum[depth]++);
                                curvecnum = 0;
                        } else {
                                MVecEnd();
                                sprintf(namebuf, "%s.bond%d",
                                        curobjname, segnum[depth]++);
                                MVecBegn(namebuf, P_Item, vecnum);
                                savecrd(MOVETO, lx, ly, lz, curcolor);
                                curvecnum = 1;
                        }
                        break;

                case D_LABEL :
                        if (firstcoord)
                                PNameNil(namebuf);
                        else
                                /* PEndS(); */
                                MVecEnd();
                        sprintf(namebuf, "%s.label%d", curobjname,
                                segnum[depth]++);
                        firstcoord = TRUE;
                        break;

                case D_SURF :
                        if (firstcoord)
                                PNameNil(namebuf);
                        else
                                MVecEnd();
                        sprintf(namebuf, "%s.surf%d", curobjname,
                                segnum[depth]++);
                        firstcoord = TRUE;
                        break;

                }
#endif  PS300
        } /* end backwards rot */

        /* check for forward rotation */
        i = findrot(molnum, rseq, atname, 1, 0);
        if (i >= 0) {
                if (atoms[index].status & SHOWBIT)
                        rotation[i].r_type &= ~HOLD(0);
                else
                        rotation[i].r_type |= HOLD(0);
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr, "Forward rotation %d.\n", i);
#endif
                flushcrd();

#if     GRAFENG == PS2
                pushmatrix();
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                bldcon(CONCATMAT, rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                subps(matnam);
                bldcon(CONCATMAT, rotation[i].r_mat);
                setbase();                      /* Reset base values */
#endif
#if     GRAFENG == MPS
                pushmatrix();
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                tcon(rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                subps(matnam);
                tcon(rotation[i].r_mat);
                setbase();                      /* Reset base values */
#endif
#if     GRAFENG == IRIS
                pushmatrix();
                mat_stack++;
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                multmatrix(rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                callobj((Object) matnam);
                multmatrix(rotation[i].r_mat);
#endif
#if     GRAFENG == PS300
                pushname();
                segnum[++depth] = 1;
                switch (drawtype) {

                case D_BOND :
```

```
                        sprintf(namebuf, "%s.bond%d", curobjname,
                                segnum[depth]++);
                        MVecBegn(namebuf, P_Item, vecnum);
                        curvecnum = 0;
                } else {
                        MVecEnd();
                        sprintf(namebuf, "%s.bond%d", curobjname,
                                segnum[depth]++);
                        MVecBegn(namebuf, P_Item, vecnum);
                        savecrd(MOVETO, lx, ly, lz, curcolor);
                        curvecnum = 1;
                }
                break;

        case D_LABEL :
                if (firstcoord)
                        PNameNil(namebuf);
                else
                        /* PEndS(); */
                        MVecEnd();
                sprintf(namebuf, "%s.label%d", curobjname,
                        segnum[depth]++);
                firstcoord = TRUE;
                break;

        case D_SURF :
                if (firstcoord)
                        PNameNil(namebuf);
                else
                        MVecEnd();
                sprintf(namebuf, "%s.surf%d", curobjname,
                        segnum[depth]++);
                firstcoord = TRUE;
                break;

        }
#endif  PS300
        } /* end forward rot */
}

/*
 * poprot - pop the rotation matrix off
 */
poprot(db, index)
int     db, index;
{
#if     GRAFENG == PS2 || GRAFENG == MPS || GRAFENG == IRIS
        int     matnam;
#endif
        char    atname[AT_NAME_SIZE+1];

        if (!mchain(db, index)) {
                (void) matom(db, index, atname);
                i = findrot(molnum, rseq, atname, 1, 0);
                if (i >= 0) {
#ifdef  DEBUG
                        if (debug & D_PG)
                                fprintf(stderr,
                                        "End forward rotation.\n");
#endif
                        flushcrd();
#if     GRAFENG != PS300
                        popmatrix();
#if     GRAFENG == IRIS
                        mat_stack--;
#endif
#else

                        popname();
                        depth--;
                        switch (drawtype) {

                        case D_BOND :
                                if (firstcoord) {
                                        PNameNil(namebuf);
                                        sprintf(namebuf, "%s.bond%d",
                                                curobjname, segnum[depth]++);
                                        curvecnum = 0;
                                } else {
                                        MVecEnd();
                                        sprintf(namebuf, "%s.bond%d",
                                                curobjname, segnum[depth]++);
                                        MVecBegn(namebuf, P_Item, vecnum);
                                        savecrd(MOVETO, lx, ly, lz, curcolor);
                                        curvecnum = 1;
                                }
                                break;

                        case D_LABEL :
                                if (firstcoord)
                                        PNameNil(namebuf);
                                else
```

```
                        sprintf(namebuf, "%s.label%d",
                                curobjname, segnum[depth]++);
                        firstcoord = TRUE;
                        break;

                case D_SURF :
                        if (firstcoord)
                                PNameNil(namebuf);
                        else
                                MVecEnd();
                        sprintf(namebuf, "%s.surf%d", curobjname,
                                segnum[depth]++);
                        firstcoord = TRUE;
                        break;

                }
#endif  PS300
        } /* end of "end forward rot" */

        i = findrot(molnum, rseq, atname, 2, 0);
        if (i >= 0) {
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr,
                                "Start forward rotation.\n");
#endif
                flushcrd();

#if     GRAFENG == PS2
                pushmatrix();
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                bldcon(CONCATMAT, rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                subps(matnam);
                bldcon(CONCATMAT, rotation[i].r_mat);
                setbase();                      /* Reset base values */
#endif
#if     GRAFENG == MPS
                pushmatrix();
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                tcon(rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                subps(matnam);
```

```
                setbase();                      /* Reset base values */
#endif
#if     GRAFENG == IRIS
                pushmatrix();
                mat_stack++;
                if (rotation[i].r_type & REMAKEMAT)
                        makemats(i);
                multmatrix(rotation[i].r_invmat);
                matnam = ('r' << 8) | ('0' + i);
                callobj((Object) matnam);
                multmatrix(rotation[i].r_mat);
#endif
#if     GRAFENG == PS300
                pushname();
                segnum[++depth] = 1;
                switch (drawtype) {

                case D_BOND :
                        if (firstcoord) {
                                PNameNil(namebuf);
                                sprintf(namebuf, "%s.bond%d",
                                        curobjname, segnum[depth]++);
                                curvecnum = 0;
                        } else {
                                MVecEnd();
                                sprintf(namebuf, "%s.bond%d",
                                        curobjname, segnum[depth]++);
                                MVecBegn(namebuf, P_Item, vecnum);
                                savecrd(MOVETO, lx, ly, lz, curcolor);
                                curvecnum = 1;
                        }
                        break;

                case D_LABEL :
                        if (firstcoord)
                                PNameNil(namebuf);
                        else
                                /* PEndS(); */
                                MVecEnd();
                        sprintf(namebuf, "%s.label%d",
                                curobjname, segnum[depth]++);
                        firstcoord = TRUE;
                        break;

                case D_SURF :
                        if (firstcoord)
```

```
                                        MVECENO();
                                        sprintf(namebuf, "%s.surf%d",
                                                curobjname, segnum[depth]++);
                                        firstcoord = TRUE;
                                        break;
                                }
#endif  PS300
                        } /* end of "start forward rot" */
                } /* end of lmainchain */
}

#if     GRAFENG != PS300
/*
 * dostack - generates the preceding pushes, succeeding pops, and any
 * rotation matrices required to make the structure work
 */
dostack()
{
        struct  input_def       in;
        struct  chain_def       *ch;
        int     npop;
        int     matnam, submatnam;
        int     count;
        register                i;
        static char             buf[10];

        /* generate this stuff for each individual molecule */
        for (molnum = 0; molnum < MAXMOD; molnum++) {
                if (mdb[molnum].m_fname == NULL)
                        continue;

                matnam = ('m' << 8) | ('0' + molnum);
                count = 0;

                for (ch = mdb[molnum].m_chain; ch != NULL; ch = ch->ch_next) {
#ifdef  DEBUG
                        if (debug & D_PG)
                                fprintf(stderr, "Chain '%s'.\n", ch->ch_start);
#endif
                        /* At least one pop to match the initial push */
                        count++;
                        npop = 1;

#if     GRAFENG == PS2
                        prepmkob();
                        pushmatrix();
                        subps(matnam);
#endif
#endif  PS2
#if     GRAFENG == MPS
                        prepmkob();
                        makeob(ob, &OBSIZE, &obsize, overflow);
#ifndef DRAWPIC
                        pushmatrix();
                        subps(matnam);
#endif
#endif  MPS
#if     GRAFENG == IRIS
                        (void) sprintf(buf, "psh%d", count);
                        makeobj(numobj(molnum, buf, 0));
#ifndef DRAWPIC
                        pushmatrix();
                        callobj((Object) matnam);
#endif
#endif  IRIS


                        /* Generate a rotation matrix if the rotation is a
                         * forward mainchain or backward sidechain rotation
                         * going from the first atom to
                         * somewhere in the middle of the molecule */
                        for (i = ch->ch_rotb; i != -1; i = rotation[i].r_linkb) {
#ifdef  DEBUG
                                if (debug & D_PG)
                                        fprintf(stderr,
                                                "Backward matrix %d.\n", i);
#endif

                                pushmatrix();
                                if (rotation[i].r_type & REMAKEMAT)
                                        makemats(i);
#if     GRAFENG == PS2
                                bldcon(CONCATMAT, rotation[i].r_invmat);
                                submatnam = ('r' << 8) | ('0' + i);
                                subps(submatnam);
                                bldcon(CONCATMAT, rotation[i].r_mat);

#endif
#if     GRAFENG == MPS
                                tcon(rotation[i].r_invmat);
                                submatnam = ('r' << 8) | ('0' + i);
                                subps(submatnam);
                                tcon(rotation[i].r_mat);
```

```
                        submatnam = ('r' << 8) | ('0' + i);
                        callobj((Object) submatnam);
                        multmatrix(rotation[i].r_mat);
#endif
                }
                for (i=ch->ch_rotf; i != -1; i = rotation[i].r_linkf) {
#ifdef  DEBUG
                        if (debug & D_PG)
                                fprintf(stderr, "Mainchain pop %d.\n"
#endif
                        npop++;
                }
#if     GRAFENG == PS2
                setbase();
                stopob();
#endif
#if     GRAFENG == MPS
                setbase();              /* probably does no good on MPS */
                lastcall = TRUE;
                stopob();
                voidpp();
#endif
#if     GRAFENG == IRIS
                closeobj();
#endif

                /* Send this down to 'store' */
                in.i_type = S_REPLACE;
                in.i_molnum = molnum;
                (void) sprintf(in.i_name, "psh%d", count);
                in.i_size = sizeofob();
                in.i_ndata = 0;
                in.i_firstmode = FM_NONE;
                in.i_newchain = FALSE;
                in.i_color = in.i_lcolor = 0;
#ifdef  ONEPROC
                process(&in, copyob(ob));
#else
                (void) write(store_fd[1], (char *) &in, sizeof in);
                sendob(store_fd[1], ob, sizeof ob);
                (void) read(store_fd[0], (char *) &in, sizeof in);
#endif

                /* Tack on the trailing pops */
                makeob(ob, OBSIZE, &obsize, overflow);
#endif
#if     GRAFENG == MPS
                prepmkob();
                makeob(ob, &OBSIZE, &obsize, overflow);
#endif
#if     GRAFENG == IRIS
                (void) sprintf(buf, "pop%d", count);
                makeobj(numobj(molnum, buf, 0));
#endif

#ifndef DRAWPIC
                while (npop--)
                        popmatrix();
#endif

#if     GRAFENG == PS2
                stopob();
#endif
#if     GRAFENG == MPS
                lastcall = TRUE;
                stopob();
#endif
#if     GRAFENG == IRIS
                closeobj();
#endif

                /* Send this down to 'store' also */
                in.i_type = S_REPLACE;
                in.i_molnum = molnum;
                (void) sprintf(in.i_name, "pop%d", count);
                in.i_size = sizeofob();
                in.i_ndata = 0;
                in.i_firstmode = FM_NONE;
                in.i_newchain = FALSE;
                in.i_color = in.i_lcolor = 0;
#ifdef  ONEPROC
                process(&in, copyob(ob));
#else
                (void) write(store_fd[1], (char *) &in, sizeof in);
                sendob(store_fd[1], ob, sizeof ob);
                (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        } /* end foreach(chain) */
} /* end foreach(model) */
```

```
#else	GRAFENG == PS300

/*
 * pushname - increase subobject count and create new object name
 */
static
pushname()
{
	char	buf[10];

	(void) sprintf(buf, ".sub%d", subnum++);
	strcat(curobjname, buf);
}

/*
 * popname - create name of object containing current object
 */
static
popname()
{
	register		char	*p;
	char	*rindex();

	p = rindex(curobjname, '.');
	if (p == NULL) {
		fprintf(stderr, "Popping top level object!!!\n");
		return;
	}
	*p = '\0';
}

/*
 * drawresidue - construct the PS300 object of the given residue
 * Called from touch() during open commands and from things like
 * dorot() when adding/fixing/deleting rotations.
 */
drawresidue(mol, seq, type, atomdat)
int	mol;
char	*seq, *type;
struct	atom_def		*atomdat;
{
	struct	input_def		in;
	struct	chain_def		*ch, *prevch;
	register			i, j, comp;
	int	drvisit(), dragain();

	/* Set up global variables */
	molnum = mol;
	rseq = seq;
	rtype = type;
	atoms = atomdat;
	prevch = mdb[molnum].m_chain;
	for (ch = prevch->ch_next; ch != NULL; ch = ch->ch_next)
		if (infront(mdb[molnum].m_fd, rseq, ch->ch_start))
			break;
		else
			prevch = ch;
	ch = prevch;			/* Previous chain contains this residue */
#ifdef	DEBUG
	if (debug & D_PG)
		fprintf(stderr, "Draw residue %s %s\n", rseq, rtype);
#endif
	subnum = 1;
	depth = 0;
	segnum[0] = 1;
	(void) sprintf(baseobjname, "m%d:r%s", molnum, map_name(rseq));
	(void) strcpy(curobjname, baseobjname);
	/*
	 * See comment in touch() for reason for putting in this
	 * "empty" declaration
	 */
	sprintf(namebuf, "%slink", baseobjname);
	nullnode(namebuf);
	PBeginS(baseobjname);

	for (i = ch->ch_rotb; i != -1; i = rotation[i].r_linkb) {
		if (rotation[i].r_model != molnum)
			break;
		switch (rotation[i].r_natom) {

		case 2 :
			comp = smatch(rotation[i].r_resseq[1], rseq);
			break;

		case 4 :
			comp = smatch(rotation[i].r_resseq[2], rseq);
			break;
		}
		if (comp != 0)
			continue;
		sprintf(namebuf, "sub%d", subnum);
		PBeginS(namebuf);
```

```
                dump_rot(i, ++depth);
                segnum[depth] = 1;
        }

        in.i_ndata = depth;
        sprintf(namebuf, "%sprev", baseobjname);
        PInst(NULL, namebuf);
        sprintf(namebuf, "%slink", baseobjname);
        PInst(NULL, namebuf);
        sprintf(namebuf, "bond%d", segnum[depth]);
        nullnode(namebuf);
        sprintf(namebuf, "label%d", segnum[depth]);
        nullnode(namebuf);
        sprintf(namebuf, "surf%d", segnum[depth]++);
        nullnode(namebuf);

        (void) mtrav(mdb[molnum].m_fd, drvisit, dragain);

        sprintf(namebuf, "%snext", baseobjname);
        PInst(NULL, namebuf);
        while (depth--) {
                PEndS();
                popname();
        }
        PEndS();

        /* Now send out all the connections for rotations */
        for (i = 0; i < MAXINROT; i++) {
                if (rotation[i].r_type == INACTIVE)
                        continue;
                if ((rotation[i].r_type & SENDCONN) == 0)
                        continue;
                rotation[i].r_type &= ~SENDCONN;
                sprintf(namebuf, "brot%d_mat$", i + 1);
                PDiscOut(namebuf, 1);
                for (j = 0; j < rotation[i].r_nloc; j++)
                        PConnect(namebuf, 1, 1, rotation[i].r_locname[j]);
                rotation[i].r_nloc = 0;
        }

        /* Set the level for the storage module */
        in.i_type = S_SETLEVEL;
        in.i_molnum = molnum;
        strcpy(in.i_name, rseq);
#ifdef  ONEPROC
        process(&in, (char *) NULL);
        (void) write(store_fd[1], (char *) &in, sizeof in);
        (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
#if     GRAFENG == PS300
        newdist = TRUE;
#endif
}

/*
 * dump_rot - dumps the rotation out to the PS300
 */
/* ARGSUSED */
dump_rot(i, depth)
register        i, depth;
{
        register int        j, k;
        P_VectorType        vec;
        P_MatrixType        mat;
        char                **cp;

        /* Send out the matrices and translations */
        if (rotation[i].r_type & REMAKEMAT)
                makemats(i);

        /* Dump the inverse matrix */
        vec.V4[0] = rotation[i].r_invmat[3][0];
        vec.V4[1] = rotation[i].r_invmat[3][1];
        vec.V4[2] = rotation[i].r_invmat[3][2];
        PTransBy(NULL, &vec, NULL);
#ifdef  COLUMN_MAJOR
        /* used by pascal GSR's on vms??? */
        for (j = 0; j < 3; j++)
          for (k = 0; k < 3; k++)
                mat[k][j] = (double) rotation[i].r_invmat[j][k];
#else
        for (j = 0; j < 3; j++)
          for (k = 0; k < 3; k++)
                mat[j][k] = (double) rotation[i].r_invmat[j][k];
#endif
        PMat3x3(NULL, mat, NULL);

        /* Set the variable matrix */
        PRotZ("inrot", 0.0, NULL);

        /* Dump the matrix that takes us to the z axis */
        vec.V4[0] = rotation[i].r_mat[3][0];
```

```
        PTransBy(NULL, &vec, NULL);
#ifdef  COLUMN_MAJOR
        for (j = 0; j < 3; j++)
          for (k = 0; k < 3; k++)
                mat[k][j] = (double) rotation[i].r_mat[j][k];
#else
        for (j = 0; j < 3; j++)
          for (k = 0; k < 3; k++)
                mat[j][k] = (double) rotation[i].r_mat[j][k];
#endif
        PMat3x3(NULL, mat, NULL);


        /* Store away the name to which the function network connects */
        cp = &rotation[i].r_locname[rotation[i].r_nloc++];
        assert(rotation[i].r_nloc <= 2);
        if (*cp == NULL)
                *cp = malloc(strlen(curobjname) + 7);
        else
                *cp = realloc(*cp, strlen(curobjname) + 7);
        assert(*cp != NULL);
        (void) sprintf(*cp, "%s.inrot", curobjname);
        rotation[i].r_type |= SENDCONN;
}

/*
 * drvisit - Draw residue visiting routine
 */
/* ARGSUSED */
static
drvisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
        register        struct  atom_def        *data;

        data = &atoms[index];
        if (!(data->status & EXISTBIT) || !firstime)
                return;

        /* Check for rotations */
        if (data->status & ROTBIT) {
                startrot(db, index);
                if (nsons == 0)
                        endrot(db, index);
}

/*
 * dragain - second half of the object-creation routine
 */
/* ARGSUSED */
static
dragain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        /* Check rotation (if there are no more branches) */
        if ((atoms[index].status & ERBITS) == ERBITS && nsons == 0)
                endrot(db, index);
}

/*
 * startrot - push the rotation matrix on
 */
static
startrot(db, index)
int     db, index;
{
        register        i;
        char    atname[AT_NAME_SIZE + 1];

        /* Get the atom name */
        (void) matom(db, index, atname);

        /* Identify main chain rotation */
        if (mchain(db, index)) {
                i = -1;
                while ((i = findrot(molnum, rseq, atname, 0, i + 1)) >= 0) {
#ifdef  DEBUG
                        if (debug & D_PG)
                                fprintf(stderr, "Mainchain rotation %d.\n", i);
#endif
                        rotation[i].r_type |= MCHAIN;
                }
        }

        /* Check for backward rotation */
        i = findrot(molnum, rseq, atname, 2, 0);
        if (i >= 0) {
#ifdef  DEBUG
                if (debug & D_PG)
```

```
#endif
                depth--;
                PEndS();
                popname();
                sprintf(namebuf, "bond%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "label%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "surf%d", segnum[depth]++);
                nullnode(namebuf);
        }

        /* check for forward rotation */
        i = findrot(molnum, rseq, atname, 1, 0);
        if (i >= 0) {
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr, "Forward rotation %d.\n", i);
#endif
                sprintf(namebuf, "sub%d", subnum);
                PBeginS(namebuf);
                pushname();
                dump_rot(i, ++depth);
                segnum[depth] = 1;
                sprintf(namebuf, "bond%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "label%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "surf%d", segnum[depth]++);
                nullnode(namebuf);
        }
}

/*
 * endrot - pop the rotation matrix off
 */
static
endrot(db, index)
int     db, index;
{
        register        i;
        char    atname[AT_NAME_SIZE+1];

        if (!mchain(db, index)) {
                (void) matom(db, index, atname);
                i = findrot(molnum, rseq, atname, 1, 0);
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr,
                                "End forward rotation.\n");
#endif
                depth--;
                PEndS();
                popname();
                sprintf(namebuf, "bond%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "label%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "surf%d", segnum[depth]++);
                nullnode(namebuf);
        }
        i = findrot(molnum, rseq, atname, 2, 0);
        if (i >= 0) {
#ifdef  DEBUG
                if (debug & D_PG)
                        fprintf(stderr,
                                "Start forward rotation.\n");
#endif
                sprintf(namebuf, "sub%d", subnum);
                PBeginS(namebuf);
                pushname();
                dump_rot(i, ++depth);
                segnum[depth] = 1;
                sprintf(namebuf, "bond%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "label%d", segnum[depth]);
                nullnode(namebuf);
                sprintf(namebuf, "surf%d", segnum[depth]++);
                nullnode(namebuf);
        }
}

makechains(molnum)
int     molnum;
{
        register        i;
        register        struct  chain_def       *ch;

        sprintf(namebuf, "model%d", molnum);
        PBeginS(namebuf);
        for (ch = mdb[molnum].m_chain; ch != NULL; ch = ch->ch_next) {
```

```
                        sprintf(namebuf, "m%dr%s", molnum,
                                map_name(ch->ch_start));
                        PInst(NULL, namebuf);
                } else {
                        i = ch->ch_rotb;
                        switch (rotation[i].r_natom) {

                        case 2 :
                                sprintf(namebuf, "m%dr%s", molnum,
                                        map_name(rotation[i].r_resseq[1]));
                                PInst(NULL, namebuf);
                                break;

                        case 4 :
                                sprintf(namebuf, "m%dr%s", molnum,
                                        map_name(rotation[i].r_resseq[2]));
                                PInst(NULL, namebuf);
                                break;
                        }
                }
        }
        PEndS();
}

/*
 * nullnode - create a data node with no data in it
 * (this is the same as "name := vec 0,0;")
 * Note that we CANNOT use the color + depth-cueing vector
 * list functions because they do not pass through the CI
 * which actually creates the structure.
 */
nullnode(name)
char *name;
{
        static P_VectorType nulldat;

        PVecBegn(name, 1, TRUE, TRUE, 2, P_Item);
        PVecList(1, &nulldat);
        PVecEnd();
}
#endif  PS300

drawssbond()
{
        struct  input_def       in;
        ps_t    x, y, z;
#if GRAFENG != PS300
        int     matnam;
#endif
        register        i;

        for (molnum = 0; molnum < MAXMOD; molnum++) {
                if (mdb[molnum].m_fname == NULL || nbond[molnum] <= 0)
                        continue;

                /* Draw the special SS or HH bonds */
                curcolor = -1;
                initcrdbuf();

#if     GRAFENG == PS2
                matnam = ('m' << 8) | ('0' + molnum);
                prepmkob();
                makeob(ob, OBSIZE, &obsize, overflow);
                pushmatrix();
                subps(matnam);
                setbase();
#endif
#if     GRAFENG == MPS
                matnam = ('m' << 8) | ('0' + molnum);
                prepmkob();
                makeob(ob, &OBSIZE, &obsize, overflow);
                pushmatrix();
                subps(matnam);
                setbase();
#endif
#if     GRAFENG == IRIS
                matnam = ('m' << 8) | ('0' + molnum);
                makeobj(numobj(molnum, "bond", O_BOND));
                pushmatrix();
                callobj((Object) matnam);
#endif
#if     GRAFENG == PS300
                if (nbond[molnum] > 0) {
                        sprintf(namebuf, "m%drbond", molnum);
                        MVecBegn(namebuf, P_Item, 0);
                }
#endif
                for (i = 0; i < nbond[molnum]; i++) {
                        if (bonds[molnum][i].b_disp[0]
                          && bonds[molnum][i].b_disp[1]) {
                                if (curcolor != bonds[molnum][i].b_color) {
                                        flushcrd();
                                        curcolor = bonds[molnum][i].b_color;
```

```
                x = MAPCRD(bonds[molnum][i].b_x[0]);
                y = MAPCRD(bonds[molnum][i].b_y[0]);
                z = MAPCRD(bonds[molnum][i].b_z[0]);
                /*
                 * Always move/draw, move/draw here;
                 * we still do cordinate buffering
                 * because bonds are part of the data
                 * transformed for distance and angle
                 * calculations (so we need to calculate
                 * the amount of data in rbtmem that is
                 * due to this stuff).
                 */
                savecrd(MOVETO, x, y, z, curcolor);
                x = MAPCRD(bonds[molnum][i].b_x[1]);
                y = MAPCRD(bonds[molnum][i].b_y[1]);
                z = MAPCRD(bonds[molnum][i].b_z[1]);
                savecrd(LINETO, x, y, z, curcolor);
                        }
                }
#if     GRAFENG == PS2
                flushcrd();
                popmatrix();
                stopob();
#endif
#if     GRAFENG == MPS
                flushcrd();
                popmatrix();
                lastcall = TRUE;
                stopob();
                voidpp();
#endif
#if     GRAFENG == IRIS
                flushcrd();
                popmatrix();
                closeobj();
#endif
#if     GRAFENG == PS300
                if (nbond[molnum] > 0) {
                        flushcrd();
                        MVecEnd();
                }
#endif

                /* Send this down to 'store' also */
                in.i_molnum = molnum;
                strcpy(in.i_name, "bond");
                in.i_size = sizeofob();
                in.i_ndata = getoffset();
                in.i_firstmode = FM_NONE;
                in.i_newchain = FALSE;
                in.i_color = in.i_lcolor = 0;
#ifdef  ONEPROC
                process(&in, copyob(ob));
#else
                (void) write(store_fd[1], (char *) &in, sizeof in);
                sendob(store_fd[1], ob, sizeof ob);
                (void) read(store_fd[0], (char *) &in, sizeof in);
#endif
        }
}

#if     GRAFENG == MPS
#define PUSHJ   04360
#define MATCTRL 6       /* 6 control words preface UDL data */
/*
 * Call a previously defined PicSys resident UDL as a subobject.
 * Since this is done while we are making a host resident
 * object using the makeob() facility, and since the MPS
 * software does not know that we will end up transfering
 * this object to PS memory before sending it to the MAP,
 * this type of subobject call is normally not allowed.
 * However, in this case we really know what we are doing
 * and so provide a special facility to do it.
 *
 * Unfortunately, however, this additionally functionality
 * requires a minor modification in the E&S graphics library.
 * In the file MPUNTRAN.MAR, at label DO010, the next 3
 * statements which only allow relative jump RSR commands
 * must be commented out (using ";" chars at the beginning
 * of each line) so that both relative and absolute jumps
 * are allowed.  The relevent source statements are shown
 * below:
 *      DO010:                                  ;JUMP/PUSHJ
 *      ;       BITW    #^O400,R0               ;RELATIVE?
 *      ;       BNEQ    1$                      ;IF NEQ, OK.
 *      ;       BRW     DO420                   ;ELSE, ERROR
 *
 * The MPUNTRAN module is then reassembled using the following
 * command:
 *      macro mpsysmac+mpuntran/obj
```

```
 * and the resulting object module included in the list of files
 * that are linked with the MIDAS editor.
 *
 * The above change has little or no effect on any other programs
 * using the MPS.
 *
 * Tom Ferrin        27may83
 */
subps(name)
{
        struct udl_entry *p;
        static int initialized, ubase;
        extern struct u0com u0com;
        extern int extdmem;

        if (!initialized) {
                initialized++;
                if (!extdmem) {
                        /*
                         * Since subps() stores absolute mode PUSHJ's,
                         * we must know the physical psmem address of
                         * our UDL partition.  On an extended memory
                         * MPS this is easy as the hardware does all the
                         * work, but on a non-extended memory system it's
                         * the device driver that translates logical
                         * user addresses into physical MPS addresses.
                         * All we need to know here is the offset from
                         * phys 0 to user logical 0; the MPS driver tells
                         * us this.
                         */
                        ubase = u0com.gtrbuf;
                        if (ubase == 0) {
                                fprintf(stderr,
                                 "PANIC: mplib rev level must be >= A4.V02\n
                                exit(1);
                        }
                }
        }
        for (p = &psobs->uentry[0]; p < &psobs->uentry[MAXPSOBS]; p++) {
                if (p->u_name < 0) {
        err:
                        fprintf(stderr, "PANIC: can't resolve UDL name\n");
                        abort();
                }
                if (p->u_name == name)
                        break;
        }
        if (p == &psobs->uentry[MAXPSOBS] || (p->u_obs&U_DEFINED) == 0)
                goto err;
        if (p->u_netpp != 0)
                goto err;               /* avoid keeping a tally */
        if (obsize >= OBSIZE-1) {
                ob[obsize++] = 0;               /* intermediate sentinel */
                ob[0] = (ps_t)obsize;           /* include count in array */
                overflow(ob, &obsize);          /* call fulsub */
                obsize = 1;             /* reset length to 1 */
        }
        ob[obsize++] = PUSHJ;           /* absolute mode push jump */
        if (obsize >= OBSIZE-1) {
                ob[obsize++] = 0;
                ob[0] = (ps_t)obsize;
                overflow(ob, &obsize);
                obsize = 1;
        }
        ob[obsize++] = (ps_t) (p->u_pbase + MATCTRL + ubase);           /* subobject starting a
        dbase(&0, &0, &0, &K32K);               /* pushj toggled IB bit; lost base reg */
}

/*
 * Void the push/pop info stored at the end of a OB.
 * This is necessary because the drawob() routine does
 * several things wrong: i) it thinks the MPS hardware
 * stack is only 9 matrices deep (it's really 14),
 * ii) it thinks the stack pointer is already at a
 * depth of 6 (it's really 1 in the beginning), and
 * iii) it doesn't understand when a ubegin() is in
 * effect and thus checks for stack overflow/underflow
 * too early (it should really be done by the udata()
 * routine).
 *
 * Of course it took hours to figure all this out.
 * Thomas Ferrin  12apr84
 */
voidpp()
{
        int     i = ob[0];

        if (i<2 || ob[i-1]!=SENTINEL) {
                fprintf(stderr, "PANIC: ob ptr invalid\n");
                abort();
        }
        ob[i-2] = ob[i-3] = ob[i-4] = 0;
}
```

drawpic.c

```
#ifdef	notdef
setcolor(hue)
{
	if (hue == 0)
		lcolor(&WHITE);	/* special case for white */
	else
		lcolor(&hue, &FULSAT);
}
#endif

hittag(name)
{
	hname(&name);
}
#endif	MPS

#if	GRAFENG == IRIS
/*
 * irisout:
 *	Output a string to the IRIS 1400 terminal, taking into account
 *	imbeded commands.
 */
irisout(str)
register char	*str;
{
	charstr(str);
#ifdef	IRISTERM
	register int		index;

	index = 0;
	while (str[index])
		switch (str[index]) {
		 default:
			index++;
			break;
		 case STX:
			str++;
			break;
		 case SUP:
		 case RSP:
		 case SUB:
		 case RSB:
		 case SIT:
		 case RIT:
		 case CUR:
			do_istr(str, index);
			str += index + 1;
			index = 0;
			break;
		}
#endif
}

do_istr(str, index)
char	*str;
int	index;
{
	register int		origch;

	origch = str[index];
	str[index] = '\0';
	charstr(str);
	str[index] = origch;
}

/*
 * hash table for numbering objects.  Numbers start in the high
 * word of the Object word (which is at least 31 bits long) so as
 * to avoid collision with two character constant names.
 */
# define		HTABSIZE		32749

typedef struct {
	char	h_used;
	int	h_molnum;
	char	h_rseq[RES_SEQ_SIZE+1];
} HASH;

HASH	htab[HTABSIZE];

Object
numobj(molnum, rseq, type)
int	molnum;
char	*rseq;
Object	type;
{
	if (rseq[0] == 'p')				/* is it a "psh" or "pop"? */
		return checkobj(molnum, rseq, TRUE);
	else
		return checkobj(molnum, rseq, TRUE) + type;
}

is_validobj(molnum, rseq)
int	molnum;
```

```
}

Object
checkobj(molnum, rseq, ins)
int     molnum;
char    *rseq;
int     ins;
{
        register unsigned int       hval;
        register HASH               *hp;
        register char               *sp;
        register HASH               *orig_hp;

        hval = molnum;
        for (sp = rseq; *sp; sp++)
                hval = (hval << 2) ^ *sp;
        hval %= HTABSIZE;

        orig_hp = &htab[hval];
        hp = &htab[hval];
        while (hp->h_used) {
                if (hp->h_molnum == molnum && strcmp(rseq, hp->h_rseq) == 0)
                        break;
                if (++hp >= &htab[HTABSIZE])
                        hp = htab;
                if (orig_hp == hp) {
                        fprintf(stderr, "out of hash table space\n");
                        exit(1);
                }
        }

        if (!ins)
                return hp->h_used;

        if (!hp->h_used) {
                hp->h_used = TRUE;
                hp->h_molnum = molnum;
                strcpy(hp->h_rseq, rseq);
        }
        /*
         * the object number is the index in the array * 5.  The
         * first index (0) is reserved for the global objects
         */
        return (((Object) ((hp - htab) + 1) * 5) << O_SAFEBITS);
```

misc.c

```
/* $Header: misc.c,v 3.10 86/07/24 12:45:04 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          6 Jul 1983
 */

#include <ctype.h>
#include "editor.h"

/*
 * getnum - gets a number from the wordrange (minor check of validity included)
 */
getnum(w)
struct  wordrange_def   *w;
{
        int     i;

        if (w == NULL || w->w_first != w->w_last)
                return(-1);
        for (i = 0; w->w_first[i] != '\0'; i++)
                if (!isdigit(w->w_first[i]))
                        return(-1);
        return(atoi(w->w_first));
}

static  struct  map_def {
        char    *m_color;
        int     m_code;
}       colormap[]      =       {
        "white",        0,
        "green",        1,                      /* Zero is saved for WHIT
#if     GRAFENG != MPS
        "cyan",         CYAN,
        "blue",         BLUE,
        "magenta",      MAGENTA,
        "red",          RED,
        "yellow",       YELLOW,
#else
        /*
         * Can't use the predefined values here, since they
         * include a saturation value as a 2nd argument.
         * This is because E&S chose a default saturation
         * value of zero (= WHITE), so you never get any
         * color unless you supply a saturation argument.
         */
        "cyan",         8,
        "blue",         16,
        "magenta",      24,
        "red",          32,
        "yellow",       48,
#endif
        0,              -1
};

/*
 * colornum - gets a color number from the argument
 */
colornum(w, n1, n2)
struct  wordrange_def   *w;
int     *n1, *n2;
{
        if (w == NULL) {
                *n1 = -1;
                *n2 = -1;
                return;
        }
        *n1 = findnum(w->w_first);
        if (w->w_first != w->w_last && *n1 != -1)
                *n2 = findnum(w->w_last);
        else
                *n2 = -1;
}

/*
 * findnum - find the number corresponding to the given string
 */
findnum(s)
char    *s;
{
        register        char    *p;
        register        i;
        int     isnum;

        isnum = TRUE;
        for (p = s; *p != '\0'; p++)
                if (!isdigit(*p)) {
                        isnum = FALSE;
                        break;
                }
        if (isnum)
```

```
                return(atoi(s));
        else
                for (i = 0; colormap[i].m_code >= 0; i++)
                        if (!strcmp(s, colormap[i].m_color))
                                return(colormap[i].m_code);
        return(-1);
}

/*
 * colormode - finds which part of atom (surface, bond or label)
 * is to be colored
 */
colormode(w)
struct  wordrange_def   *w;
{
        int     which;

        if (w == NULL)
                return(ALLCOLOR);
        which = 0;
        while (w != NULL) {
                switch (w->w_first[0]) {

                case 's' :
                case 'S' :
                        which |= SCOLOR;
                        break;

                case 'b' :
                case 'B' :
                        which |= BCOLOR;
                        break;

                case 'l' :
                case 'L' :
                        which |= LCOLOR;
                        break;

                case 'v' :
                case 'V' :
                        which |= VCOLOR;
                        break;
                }
                w = w->w_next;
        }
        return(which);
}

/*
 * matchmol - matches the token to a molecule (if possible)
 */
matchmol(t)
struct  token_def       *t;
{
        int     i;

        for (i = 0; t->t_value[i] != '\0'; i++)
                if (!isdigit(t->t_value[i]))
                        break;
        if (t->t_value[i] == '\0')
                i = atoi(t->t_value);
        else
                for (i = 0; i < MAXMOD; i++)
                        if (mdb[i].m_fname != NULL &&
                                !strcmp(mdb[i].m_fname, t->t_value))
                                        break;
        return(i);
}

#ifndef  EDITONLY
/*
 * findrot - finds rotation which matches the given data
 */
findrot(molnum, rseq, atname, dir, first)
int     molnum;
char    *rseq, *atname;
int     dir, first;
{
        register        i, j;

        for (i = first; i < MAXINROT; i++) {
                if (rotation[i].r_type == INACTIVE)
                        continue;
                switch (dir) {

                case 2 :
                        if ((rotation[i].r_type & BACKWRD) == 0)
                                continue;
                        break;

                case 1 :
                        if (rotation[i].r_type & BACKWRD)
                                continue;
                        break;
```

```
                case 0 :
                        break;
                }
                if (rotation[i].r_model != molnum)
                        continue;
                switch (rotation[i].r_natom) {

                case 1 :
                        j = 0;
                        break;

                case 2 :
                        j = 1;
                        break;

                case 4 :
                        j = 2;
                        break;
                }
                if (smatch(rseq, rotation[i].r_reseq[j]))
                        continue;
                if (amatch(atname, rotation[i].r_atom[j]))
                        continue;
                return(i);
        }
        return(-1);
}
#endif  EDITONLY

usage(com, not)
int     com, not;
{
        switch (com) {

        case OPEN :
                if (!not)
                        reply("Usage: open [ model surface object ] model file_n
                else
                        reply("Usage: ~open [ model surface object ] model [ , ...
                break;

        case FINROT :
                if (!not)
                        reply("Usage: [ rotat brotat ] rotation_number atom1(#:
                else
                        reply("Usage: ~ [ rotat brotat ] rotation_number [ , ... ]\n")
                break;

case REVERSE :
        if (!not)
                reply("Usage: reverse rotation_number [ , ... ]\n");
        else
                reply("No such command as ~ reverse.\n");
        break;

case DIST :
        if (!not)
                reply("Usage: distance distance_number atom1(#:@) atom2(#:@)\n");
        else
                reply("Usage: ~ distance distance_number [ , ... ]\n");
        break;

case GETCRD :
        if (!not)
                reply("Usage: getcrd atom(#:@)\n");
        else
                reply("Getcrd: ~ ignored.\n");
        break;

case ALIGN :
        if (!not)
                reply("Usage: align atom1(#:@) atom2(#:@)\n");
        else
                reply("Align: ~ ignored.\n");
        break;

case SETCOM :
        if (!not)
                reply("Usage: setcom model x y z [ size [ count ] ]\n");
        else
                reply("Usage: ~ setcom model\n");
        break;

case VDWOPT :
        if (!not)
                reply("Usage: vdwopt [radii file] [density value] [nuc] [prot] [extend len]\n"
        else
                reply("Usage: ~ vdwopt [nuc] [prot] [extend]\n");
        break;

case ANGLE :
        if (!not)
                reply("Usage: angle angle_number atom1(#:@) atom2(#:@) atom3(#:@
```

```
		else
			reply("Usage: ~ angle angle_number [ , ... ]\n");
		break;

	case MATCH :
		if (!not)
			reply("Usage: match atom1(#:@) ... atom8(#:@)\n");
		else
			reply("Match: ~ does not make sense.\n");
		break;

	case SWAPAA :
		if (!not) {
			reply("Usage: swapaa new_residue_type residue(#:)\n")
		} else
			reply("Swapaa: ~ does not make sense.\n");
		break;

	case SWAPNUC :
		if (!not) {
			reply("Usage: swapna new_residue_type residue(#:)\n")
		} else
			reply("Swapna: ~ does not make sense.\n");
		break;

	case ADDGRP :
		if (!not) {
			reply("Usage: addgrp group,bond_length,bond_angle[,di
			reply("atom1(#:@) atom2(#:@) atom3(#:@)\n");
		} else
			reply("Addgrp: ~ does not make sense.\n");
		break;

	case ADDAA :
		if (!not) {
			reply("Usage: addaa residue_type,residue_seq[,conform
		} else
			reply("Addaa: ~ does not make sense.\n");
		break;

	case ADDNUC :
		if (!not) {
			reply("Usage: addnuc residue,bond_length,bond_angle[,
			reply("atom1(#:@) atom2(#:@) atom3(#:@)\n");
		} else
			reply("Addna: ~ does not make sense.\n");
		break;

	case DELGRP :
		if (!not)
			reply("Usage: delete new_residue_name atom(#:@)\n");
		else
			reply("Delete: Cannot undelete atoms.\n");
		break;

	case LINK :
		if (!not)
			reply("Usage: link residue(#:)\n");
		else
			reply("Usage: ~link residue(#:)\n");
		break;
	}
}

/*
 * null - do-nothing routine
 */
null()
{
	return;
}

/*
 * makebit - set the given bit on the given atom
 */
makebit(mol, res, at)
int	mol;
char	*res, *at;
{
	register		i;
	struct	spec_def	s;
	struct	speclist_def	sl[3];
	struct	wordrange_def	w[3];
	char	buffer[3][10];

#ifdef	DEBUG
	if (debug & D_EDIT)
		fprintf(stderr,
			"Bit at molecule %d residue %s atom %s.\n",
			mol, res, at);
#endif
	s.s_stat = 0;
	s.s_model = &sl[0];
	s.s_residue = &sl[1];
```

```
        s.s_umax = maxgu;
        s.s_espmin = mingesp;
        s.s_espmax = maxgesp;
        s.s_next = NIL;

        for (i = 0; i < 3; i++) {
                sl[i].sp_cnt = 1;
                sl[i].sp_range = &w[i];
                w[i].w_first = w[i].w_last = buffer[i];
                w[i].w_next = NIL;
        }

        (void) sprintf(buffer[0], "%d", mol);
        strcpy(buffer[1], res);
        strcpy(buffer[2], at);

#ifdef  DEBUG
        if (debug & D_EDIT)
                printspec(&s);
#endif
        setupbits();
        edit(&s);
}

#if     GRAFENG == PS300
/*
 * map_name -   Map a residue sequence number into a legal PS300 name
 */
char *
map_name(name)
register char   *name;
{
        register char   *cp;
        static char     buf[BUFSIZ];

        cp = buf;
        while (*name != '\0')
                if (isalnum(*name))
                        *cp++ = *name++;
                else {
                        *cp++ = '$';
                        name++;
                }
        *cp = '\0';
        return buf;
```

allocate.c

```
/* $Header: /usr/src/local/midas/src/editor/RCS/allocate.c,v 3.3 84/12/06 14:09:47 arno
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1983
 * Release 2.0                          13 May 1983
 * Release 2.1                          28 Jul 1983
 */

#ifndef  vms
#include <assert.h>
#else
#include "MIDAS$H:assert.h"
#endif
#include "editor.h"

/*
 * alloc - Allocate some space (bomb if none available)
 */
char    *
alloc(n, size)
int     n, size;
{
        char    *cp;

        cp = calloc((unsigned) n, (unsigned) size);
        assert(cp != NULL);
        return(cp);
}

/*
 * freespec - Free a specification list
 */
freespec(s)
struct  spec_def        *s;
{
        struct  spec_def        *s1, *s2;

        for (s1 = s; s1 != NULL; s1 = s2) {
                s2 = s1->s_next;
                if (!(s1->s_stat & DUPMOD) && s1->s_model != NULL)
                        freelist(s1->s_model);
                if (!(s1->s_stat & DUPRES) && s1->s_residue != NULL)
                        freelist(s1->s_residue);
                if (s1->s_atom != NULL)
                        freelist(s1->s_atom);
                free((char *) s1);
        }
}

/*
 * freelist - Free a specifier list
 */
freelist(sp)
struct  speclist_def    *sp;
{
        if (sp->sp_range != NULL)
                freerange(sp->sp_range);
        free((char *) sp);
}

/*
 * freerange - Free a word range
 */
freerange(w)
struct  wordrange_def   *w;
{
        struct  wordrange_def   *w1, *w2;

        for (w1 = w; w1 != NULL; w1 = w2) {
                w2 = w1->w_next;
                if (w1->w_last != all && w1->w_last != nothing)
                        if (w1->w_last != w1->w_first)
                                free(w1->w_last);
                if (w1->w_first != all && w1->w_first != nothing)
                        free(w1->w_first);
                free((char *) w1);
        }
}

/*
 * freetoken - Release space used by token
 */
freetoken(t)
struct  token_def       *t;
{
        struct  token_def       *t1;

        for (t1 = t; t1 != NULL; t1 = t) {
                t = t1->t_next;
                if (t1->t_value != NULL)
                        free(t1->t_value);
                free((char *) t1);
```

execute.c

```
/* $Header: execute.c,v 3.30 86/08/04 18:27:04 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          5 Jul 1983
 * Release 2.2                          28 Jul 1983
 * Release 2.3                          23 Aug 1983
 */

#include "editor.h"
#include <signal.h>
#ifndef   vms
extern char          *sys_errlist[];
#else
extern noshare char          *sys_errlist[];
#endif

/*
 * These are the special execution functions which affect only the
 * editor but not the database.
 */

/*
 * execalias - Add, list, or delete an alias
 */
execalias(t)
struct    token_def          *t;
{
          struct    token_def          *t1;
          int       i;

          if (notcom) {                                   /* Remove an alias */
                    if (t == NULL) {
                              reply("Alias name expected.\n");
                              return;
                    }
                    for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                              if (t1->t_type != NAME)
                                        continue;
                              for (i = 0; i < nalias; i++)
                                        if (strcmp(atab[i].a_name, t1->t_value) == 0)
                                                  break;
                              if (i >= nalias) {
                                        (void) sprintf(buf, "No such alias: '%s'\n",
                                                  t1->t_value);
                                        reply(buf);
                                        continue;
                              }
                              free(atab[i].a_name);
                              free(atab[i].a_line);
                              nalias--;
                              atab[i] = atab[nalias];
                    }
          } else {                                        /* Add an alias */
                    if (t == NULL) {
                              for (i = 0; i < nalias; i++) {
                                        (void) sprintf(buf, "'%s'   '%s'\n",
                                                  atab[i].a_name, atab[i].a_line);
                                        reply(buf);
                              }
                              return;
                    }

                    /* Is it possible to add it ? */
                    for (i = 0; i < nalias; i++)
                              if (strcmp(atab[i].a_name, t->t_value) == 0)
                                        break;
                    if (t->t_next == NULL) {
                              if (i < nalias)
                                        (void) sprintf(buf, "'%s'   '%s'\n",
                                                  atab[i].a_name, atab[i].a_line);
                              else
                                        (void) sprintf(buf, "No such alias: '%s'\n",
                                                  t->t_value);
                              reply(buf);
                              return;
                    }
                    if (i >= nalias) {
                              nalias++;
                              if (nalias >= MAXALIAS) {
                                        reply("Too many aliases.\n");
                                        nalias--;
                                        return;
                              }
                    }

                    /* Enter alias into table */
                    atab[i].a_name = t->t_value;
                    t->t_value = NULL;
                    t1 = t->t_next;
                    buf[0] = '\0';
```

```
                while (t1 != NULL) {
                        strcat(buf, t1->t_value);
                        strcat(buf, " ");
                        t1 = t1->t_next;
                }
                atab[i].a_line = alloc(strlen(buf)+1, sizeof (char));
                strcpy(atab[i].a_line, buf);
                (void) sprintf(buf, "aliased '%s' to '%s'\n", atab[i].a_name,
                        atab[i].a_line);
                reply(buf);
        }
        return;
}

/*
 * execread - Take input from a file instead of the keyboard
 */
execread(t)
struct  token_def       *t;
{
        struct  token_def       *t1;
        struct  token_def       savetoken;

        if (notcom) {
                reply("Cannot unread a file.\n");
                return;
        }
        savetoken = token;                      /* Preserve last token read */
        for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                if (t1->t_type != NAME)
                        continue;
                if (stptr+1 >= MAXSTACK) {
                        reply("Read nesting too deep.\n");
                        return;
                }
                stptr++;
                CURSTRM = fopen(t1->t_value, "r");
                if (CURSTRM == NULL) {
                        stptr--;
                        (void) sprintf(buf, "%s: cannot access.\n",
                                t1->t_value);
                        reply(buf);
                        continue;
                }
                CURNDX = -1;
                input();
        }
        token = savetoken;
        return;
}

/*
 * execopen - Open something
 */
execopen(t)
struct  token_def       *t;
{
        int     count;

        if (t == NULL) {
                openmodel(t);
                return;
        }
        count = strlen(t->t_value);
        if (strncmp(t->t_value, "surface", count) == 0)
                opensurf(t->t_next);
        else if (strncmp(t->t_value, "object", count) == 0)
                openobj(t->t_next);
        else if (strncmp(t->t_value, "model", count) == 0)
                openmodel(t->t_next);
        else
                openmodel(t);
#ifndef EDITONLY
#if     GRAFENG != PS300
        dostack();
#endif
#endif
        return;
}

/*
 * openmodel - Open a MIDAS coordinate database
 */
openmodel(t)
struct  token_def       *t;
{
#ifndef EDITONLY
        struct  input_def       in;
#endif
        struct  token_def       *t1;
        struct  chain_def       *ch;
        int     i, cnt;
        register                j;
```

```
        if (notcom) {                           /* Close the database */
                if (t == NULL) {
                        usage(OPEN, notcom);
                        return;
                }
                for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                        if (t1->t_type != NAME)
                                continue;
                        i = matchmol(t1);
                        if (i < 0 || i >= MAXMOD || mdb[i].m_fname == NULL) {
                                (void) sprintf(buf, "%s: no such open file.\n",
                                        t1->t_value);
                                reply(buf);
                                continue;
                        }
                        (void) mclose(mdb[i].m_fd);
                        if (mdb[i].s_fd >= 0) {
                                (void) mclose(mdb[i].s_fd);
                                (void) close(mdb[i].sd_fd);
                                mdb[i].s_fd = mdb[i].sd_fd = -1;
                        }
#ifndef EDITONLY
                        for (cnt = 0; cnt < MAXINROT; cnt++) {
                                if (rotation[cnt].r_type == INACTIVE)
                                        continue;
                                if (rotation[cnt].r_model != i)
                                        continue;
                                rotation[cnt].r_rotangle = 0;
                                unrot(cnt);
                        }
                        for (cnt = 0; cnt < MAXANGLE; cnt++) {
                                for (j = 0; j < angles[cnt].a_natom; j++) {
                                        if (angles[cnt].a_type == INACTIVE)
                                                continue;
                                        if (angles[cnt].a_model[j] != i)
                                                continue;
                                        unang(cnt);
                                        break;
                                }
                        }
                        for (cnt = 0; cnt < MAXDIST; cnt++) {
                                for (j = 0; j < 2; j++) {
                                        if (distance[cnt].d_type == INACTIV
                                                continue;
                                        if (distance[cnt].d_model[j] != i)
                                                continue;
                                        undist(cnt);
                                        break;
                                }
                        }
                        in.i_type = S_DELETE;
                        in.i_molnum = i;
                        process(&in, (char *) NULL);
                        anychg++;
#endif  EDITONLY
                        if (nbond[i] > 0) {
                                nbond[i] = 0;
                                free((char *) bonds[i]);
                        }
                        free(mdb[i].m_fname);
                        mdb[i].m_fname = NULL;
                        if (mdb[i].m_aname != NULL) {
                                free(mdb[i].m_aname);
                                mdb[i].m_aname = NULL;
                        }
                        mdb[i].m_flags = 0;
#ifndef EDITONLY
                        for (ch = mdb[i].m_chain; ch != NULL;
                                mdb[i].m_chain = ch) {
                                        ch = ch->ch_next;
                                        free((char *) mdb[i].m_chain);
                        }
                        mdb[i].m_chain = NULL;
                        mdb[i].m_nchain = 0;
#endif
                        (void) sprintf(buf, "Model %d closed.\n", i);
                        reply(buf);
                        nmol--;
                }
                return;
        } else {                                /* Open the database */
                if (t != NULL && strcmp(t->t_value, "#") == 0)
                        t = t->t_next;
                cnt = 0;
                for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                        cnt++;
#ifdef  DEBUG
                        if (debug & D_PARSE)
                                fprintf(stderr, "Arg %d = %s\n", cnt,
                                        t1->t_value);
#endif
                }
                if (cnt == 0) {
                        for (i = 0; i < MAXMOD; i++) {
```

```
                if (mdb[i].m_fname == NULL)
                        continue;
                (void) sprintf(buf, "Model %d %s\n", i,
                        mdb[i].m_fname);
                reply(buf);
                cnt++;
        }
        if (cnt == 0)
                reply("No models opened.\n");
        return;
}
if (cnt != 2 && cnt != 3) {
        usage(OPEN, notcom);
        return;
}
if (t->t_type != NAME) {
        usage(OPEN, notcom);
        return;
}
if (sscanf(t->t_value, "%d", &i) != 1 || i >= MAXMOD || i < 0) {
        usage(OPEN, notcom);
        return;
}
if (mdb[i].m_fname != NULL) {
        (void) sprintf(buf, "Model %d is already in use.\n", i);
        reply(buf);
        return;
}
t1 = t->t_next;
if (strlen(t1->t_value) >= MOLNMSZ) {
        (void) sprintf(buf,
                "pathname too long: \"%s\"\n%d character
                t1->t_value, MOLNMSZ - 1);
        reply(buf);
        return;
}
err_midas = 0;
mdb[i].m_fd = mopen(t1->t_value, "rw");
if (mdb[i].m_fd == -1) {
        cnt = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (mdb[i].m_fname == NULL)
                        continue;
                cnt++;
                if (mdb[i].s_fd >= 0)
                        cnt++;
        }
        if (cnt >= MAXSTREAM)
                (void) sprintf(buf,
                        "Only %d database allowed.\n",
                        MAXSTREAM);
        else if (err_midas) {
#ifndef   vms
                (void) sprintf(buf, "%s: %s.\n",
                        t1->t_value,
                        err_midas > 0 ?
                        sys_errlist[err_midas] :
                        midas_errlist[-err_midas]);
#else
                if (err_midas < 0)
                (void) sprintf(buf, "%s: %s.\n",
                        t1->t_value,
                        midas_errlist[-err_midas]);
                else {
                (void) sprintf(buf,
                        "%s: vms error (see CRT terminal).\n",
                        t1->t_value);
                    perror(t1->t_value);
                }
#endif
        } else
                (void) sprintf(buf, "%s: cannot access.\n",
                        t1->t_value);
        reply(buf);
        return;
    }
    mdb[i].s_fd = -1;
    mdb[i].m_flags = 0;
    mdb[i].m_fname = t1->t_value;
    t1->t_value = NULL;
#ifndef EDITONLY
    mdb[i].m_chain = NULL;
    mdb[i].m_nchain = 0;
#endif
    nmol++;
    if (cnt == 3) {
        t->t_next = t1->t_next;
        opensurf(t);
        t->t_next = t1;
    }
#ifndef   EDITONLY
    /* opensurf will "touch" the database if we are running as
     * the back end of MIDAS.  It will NOT touch the db if we
     * are running as midas.edit because that serves no useful
```

```
                * purpose since touch simply recalculates the global temp
                * factor range when running as midas.edit. */
               if (mdb[i].s_fd < 0)
#endif
                       touch(i);
               (void) sprintf(buf, "%s opened as model %d.\n",
                       mdb[i].m_fname, i);
               reply(buf);

               return;
       }
}

/*
 * opensurf - Associate a MIDAS surface database with a molecule
 */
opensurf(t)
struct  token_def       *t;
{
       struct  token_def       *t1;
       struct  srfhdr_def      shdr;
       int     i, cnt, nsurf;
       short   magic;

       if (notcom) {                           /* Get rid of surface */
               for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                       if (t1->t_type != NAME)
                               continue;
                       i = matchmol(t1);
                       if (i < 0 || i >= MAXMOD) {
                               (void) sprintf(buf, "There is no model '%s'.\n",
                                       t1->t_value);
                               reply(buf);
                       } else if (mdb[i].s_fd < 0) {
                               (void) sprintf(buf,
                                       "There is no surface for model '%s'.\
                                       t1->t_value);
                               reply(buf);
                       } else {
                               (void) mclose(mdb[i].s_fd);
                               (void) close(mdb[i].sd_fd);
                               mdb[i].s_fd = mdb[i].sd_fd = -1;
                               free(mdb[i].m_aname);
                               mdb[i].m_aname = NULL;
                               (void) sprintf(buf, "Surface %d closed.\n", i);
                               reply(buf);
                       }
               }
       } else {                                /* Add the surface file */
               if (strcmp(t->t_value, "#") == 0)
                       t = t->t_next;
               cnt = 0;
               for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                       cnt++;
#ifdef  DEBUG
                       if (debug & D_PARSE)
                               fprintf(stderr, "Arg %d = %s\n", cnt,
                                       t1->t_value);
#endif
               }
               if (cnt != 2) {
                       usage(OPEN, notcom);
                       return;
               }
               if (t->t_type != NAME)
                       return;
               i = matchmol(t);
               if (i < 0 || i >= MAXMOD || mdb[i].m_fname == NULL) {
                       (void) sprintf(buf, "There is no model '%s'.\n",
                               t->t_value);
                       reply(buf);
                       return;
               }
               if (mdb[i].s_fd >= 0) {
                       (void) sprintf(buf,
                               "There is already a surface for '%s'\n",
                               t->t_value);
                       reply(buf);
                       return;
               }
               if (mdb[i].m_flags & PSOB) {
                       (void) sprintf(buf,
                               "'%s' is a Picture System object.\n",
                               t->t_value);
                       reply(buf);
                       return;
               }
               t1 = t->t_next;
               err_midas = 0;
               mdb[i].s_fd = mopen(t1->t_value, "r");
               if (mdb[i].s_fd == -1) {
                       cnt = 0;
                       for (i = 0; i < MAXMOD; i++) {
                               if (mdb[i].m_fname == NULL)
```

```
                                continue;
                        cnt++;
                        if (mdb[i].s_fd >= 0)
                                cnt++;
                }
                if (cnt >= MAXSTREAM)
                        (void) sprintf(buf,
                                "Only %d open databases allowed.\
                                MAXSTREAM);
                else if (err_midas) {
#ifndef   vms
                        (void) sprintf(buf, "%s: %s.\n",
                                t1->t_value,
                                err_midas > 0 ?
                                sys_errlist[err_midas] :
                                midas_errlist[-err_midas]);
#else
                        if (err_midas < 0)
                        (void) sprintf(buf, "%s: %s.\n",
                                t1->t_value,
                                midas_errlist[-err_midas]);
                        else {
                        (void) sprintf(buf,
                                "%s: vms error (see CRT terminal).\
                                t1->t_value);
                            perror(t1->t_value);
                        }
#endif
                } else
                        (void) sprintf(buf, "%s: cannot access.\n",
                                t1->t_value);
                reply(buf);
                return;
        }
        mdb[i].m_aname = t1->t_value;
        t1->t_value = calloc((unsigned) strlen(mdb[i].m_aname)+5,
                sizeof (char));
        if (t1->t_value == NULL) {
                (void) mclose(mdb[i].s_fd);
                mdb[i].s_fd = -1;
                reply("Out of memory for surface name\n");
                return;
        }
        strcpy(t1->t_value, mdb[i].m_aname);
        strcat(t1->t_value, ".srf");
        if ((mdb[i].sd_fd = open(t1->t_value, 2)) == -1) {
                (void) mclose(mdb[i].s_fd);
                mdb[i].s_fd = -1;
                (void) sprintf(buf, "%s: cannot access.\n",
                        t1->t_value);
                reply(buf);
                return;
        }
        (void) read(mdb[i].sd_fd, (char *) &magic, sizeof magic);
        if (magic != SRF_MAGIC) {
                (void) close(mdb[i].sd_fd);
                (void) mclose(mdb[i].s_fd);
                mdb[i].s_fd = -1;
                (void) sprintf(buf, "%s: not a surface file\n",
                        t1->t_value);
                reply(buf);
                return;
        }
        (void) read(mdb[i].sd_fd, (char *) &shdr, sizeof shdr);
        if (shdr.sh_minesp != shdr.sh_maxesp) {
                mdb[i].m_espmin = shdr.sh_minesp;
                mdb[i].m_espmax = shdr.sh_maxesp;
        } else {
                mdb[i].m_espmin = shdr.sh_minesp - 1;
                mdb[i].m_espmax = shdr.sh_maxesp + 1;
        }
        t1->t_value = NULL;
        (void) sprintf(buf, "%s opened as surface %d.\n",
                mdb[i].m_aname, i);
        reply(buf);
#ifndef   EDITONLY
        touch(i);
#endif

        /* Update the global ESP range */
        nsurf = 0;
        for (cnt = 0; cnt < MAXMOD; cnt++)
                if (mdb[cnt].m_fname != NULL && mdb[cnt].s_fd >= 0)
                        nsurf++;
        if (nsurf == 1) {               /* Only one open ? */
                mingesp = mdb[i].m_espmin;
                maxgesp = mdb[i].m_espmax;
        } else {
                if (mingesp > mdb[i].m_espmin)
                        mingesp = mdb[i].m_espmin;
                if (maxgesp < mdb[i].m_espmax)
                        maxgesp = mdb[i].m_espmax;
        }
}
```

```
        return;
}

/*
 * openobj - Open a Picture System object as a model
 */
openobj(t)
struct  token_def       *t;
{
#ifndef EDITONLY
        struct  input_def       in;
#endif
        struct  token_def       *t1;
        int     i, cnt;

        if (notcom) {
                for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                        if (t1->t_type != NAME)
                                continue;
                        i = matchmol(t1);
                        if (i < 0 || i >= MAXMOD) {
                                (void) sprintf(buf, "There is no model '%s'.\n",
                                        t1->t_value);
                                reply(buf);
                        } if (!mdb[i].m_flags & PSOB) {
                                (void) sprintf(buf, "'%s' is not an object.\n",
                                        t1->t_value);
                                reply(buf);
                        } else {
                                free(mdb[i].m_fname);
                                mdb[i].m_fname = NULL;
                                mdb[i].m_flags = 0;
                                (void) sprintf(buf, "Object %d closed.\n", i);
                                reply(buf);
                                nmol--;
                        }
                }
        } else {
                if (strcmp(t->t_value, "#") == 0)
                        t = t->t_next;
                cnt = 0;
                for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                        cnt++;
#ifdef  DEBUG
                        if (debug & D_PARSE)
                                fprintf(stderr, "Arg %d = %s\n", cnt,
                                        t1->t_value);
#endif
                }
                if (cnt != 2) {
                        usage(OPEN, notcom);
                        return;
                }
                if (t->t_type != NAME)
                        return;
                i = matchmol(t);
                if (i < 0 || i >= MAXMOD) {
                        (void) sprintf(buf, "There is no model '%s'.\n",
                                t->t_value);
                        reply(buf);
                        return;
                }
                if (mdb[i].m_fname != NULL) {
                        (void) sprintf(buf, "Model %d is already in use.\n", i);
                        reply(buf);
                        return;
                }
                t1 = t->t_next;

#ifndef EDITONLY
                /* Create object for initial global rotation matrix */
                strcpy(in.i_name, "psh1");
                in.i_type = S_APPEND;
                in.i_molnum = i;
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = FALSE;
                in.i_firstmode = FM_NONE;
                process(&in, (char *) NULL);
                if (in.i_type) {
                        (void) sprintf(buf, "%s: cannot access.\n",
                                t1->t_value);
                        reply(buf);
                        return;
                }

                in.i_type = S_OBJECT;
                in.i_molnum = i;
                strcpy(in.i_name, "OBJ");
                in.i_size = strlen(t1->t_value);
                process(&in, t1->t_value);
                if (in.i_type) {
                        in.i_type = S_DELETE;
                        in.i_molnum = i;
```

```
                        t1->t_value);
                reply(buf);
                return;
        }
        mdb[i].m_active = in.i_ndata;
        mdb[i].m_com[0] = in.i_x;
        mdb[i].m_com[1] = in.i_y;
        mdb[i].m_com[2] = in.i_z;
        mdb[i].m_min[0] = in.i_x - in.i_size;
        mdb[i].m_min[1] = in.i_y - in.i_size;
        mdb[i].m_min[2] = in.i_z - in.i_size;
        mdb[i].m_max[0] = in.i_x + in.i_size;
        mdb[i].m_max[1] = in.i_y + in.i_size;
        mdb[i].m_max[2] = in.i_z + in.i_size;

        /* Create object for trailing pops */
        strcpy(in.i_name, "pop1");
        in.i_type = S_APPEND;
        in.i_molnum = i;
        in.i_size = 0;
        in.i_ndata = 0;
        in.i_newchain = FALSE;
        in.i_firstmode = FM_NONE;
        process(&in, (char *) NULL);
        if (in.i_type) {
                (void) sprintf(buf, "%s not opened.  Out of core.\n",
                        t1->t_value);
                reply(buf);
                return;
        }
        anychg++;
#endif  EDITONLY

        mdb[i].m_fname = t1->t_value;
        t1->t_value = NULL;
        mdb[i].m_fd = -1;
        mdb[i].s_fd = -1;
        mdb[i].m_flags = PSOB;
        mdb[i].m_chain = chalioc(1, sizeof (struct chain_def));
        mdb[i].m_chain->ch_rotf = -1;
        mdb[i].m_chain->ch_rotb = -1;
        mdb[i].m_chain->ch_start[0] = '\0';
        mdb[i].m_chain->ch_next = NULL;
        mdb[i].m_nchain = 1;
        nbond[i] = 0;
                (void) sprintf(buf, "%s opened as model %d.\n",
                        mdb[i].m_fname, i);
                reply(buf);
        }
        return;
}

/*
 * execsave - Save a MIDAS coordinate database
 */
execsave(t)
struct  token_def       *t;
{
        struct  token_def       *t1;
        int     i, cnt;
#ifndef EDITONLY
        int     cansave;
#endif
        register        k;

        if (notcom) {
                reply("Ok.  I won't save anything.\n");
                return;
        } else {
                cnt = 0;
                for (t1 = t; t1 != NULL; t1 = t1->t_next) {
                        cnt++;
#ifdef  DEBUG
                        if (debug & D_PARSE)
                                fprintf(stderr, "Arg %d = %s\n", cnt,
                                        t1->t_value);
#endif
                }
                /* Second argument is a delimiter */
                if (cnt != 1 && cnt != 2) {
                        reply("There must be one or two arguments.\n");
                        return;
                }
                if (t->t_type != NAME)
                        return;
                i = matchmol(t);
                if (i < 0 || i >= MAXMOD) {
                        (void) sprintf(buf, "There is no model '%s'.\n",
                                t->t_value);
                        reply(buf);
```

```
#ifndef	EDITONLY
		}
		cansave = TRUE;
		for (k = 0; k < MAXINROT; k++)
			if (rotation[k].r_type != INACTIVE &&
				rotation[k].r_model == i)
					cansave = FALSE;
		for (k = 0; k < MAXDIST; k++)
			if (distance[k].d_type & ACTIVE &&
				(distance[k].d_model[0] == i ||
				distance[k].d_model[1] == i))
					cansave = FALSE;
		if (!cansave) {
			reply("Can't save model with rotation or distance.\n");
			return;
		}
#endif	EDITONLY
		if (cnt == 2) {
			t1 = t->t_next;
			if (msave(mdb[i].m_fd, t1->t_value) == -1)
				(void) sprintf(buf, "Save failed\n");
			else
				(void) sprintf(buf, "'%s' saved as '%s'.\n",
					mdb[i].m_fname, t1->t_value);
		} else {
			if (msave(mdb[i].m_fd) == -1)
				(void) sprintf(buf, "Save failed\n");
			else
				(void) sprintf(buf, "'%s' saved.\n",
					mdb[i].m_fname);
		}
		reply(buf);
	}
	return;
}

/*
 * exechdir - Change directory
 */
exechdir(t)
struct	token_def		*t;
{
	if (t == NULL) {
		reply("Not enough arguments.\n");
		return;
	}
		reply("What do you mean not change directory ?\n");
		freetoken(t);
		return;
	}
	if (t->t_next != NULL) {
		reply("Too many arguments.\n");
		return;
	}
	if (chdir(t->t_value) == -1)
		reply("Changing directory failed.\n");
}

/*
 * execstop - Close all database and exit
 */
execstop(t)
struct	token_def		*t;
{

	if (t != NULL) {
		reply("Too many arguments.\n");
		return;
	}
	if (notcom) {
		reply("Ok, I won't stop.\n");
		return;
	}
	stop();
}

stop()
{
	register		i;

	(void) signal(SIGHUP, SIG_IGN);
	(void) signal(SIGINT, SIG_IGN);
	(void) signal(SIGTERM, SIG_IGN);
#ifndef	EDITONLY
#if	GRAFENG == PS2
	psfini();
#endif
#if	GRAFENG == MPS
	mpstop();
#endif
#if	GRAFENG == PS300
	psfini();
```

```
#if      GRAPENG == IRIS
#ifdef   DRAWPIC
        gexit();
        greset();
        textcolor(7);
        pagecolor(4);
#endif
#endif   IRIS
#endif   EDITONLY
        for (i = 0; i < MAXMOD; i++)
                if (mdb[i].m_fname != NULL) {
                        (void) mclose(mdb[i].m_fd);
                        if (mdb[i].s_fd > 0) {
                                (void) mclose(mdb[i].s_fd);
                                (void) close(mdb[i].sd_fd);
                        }
                        (void) sprintf(buf, "%s (%d) closed.\n",
                                mdb[i].m_fname, i);
                        reply(buf);
                }
#ifdef   DOSTATS
        cntinit(ERECFILE);
        for (i = 0; comtab[i].com_type != NOMORECOM; i++)
                cntincr(comtab[i].com_name, comtab[i].com_count);
        cntfini(ERECFILE);
#endif
        exit(0);
}

#ifndef  EDITONLY
/*
 * execsetc - Set the center of mass of a model
 */
execsetc(t)
struct   token_def       *t;
{
        register         i;
        register         struct   token_def       *t1;
        int      molno, count;
        float    data[4];

        if (t == NULL) {
                usage(SETCOM, notcom);
                return;
        }
        if (strcmp(t->t_value, "#") == 0)
if (molno < 0 || molno >= MAXMOD) {
        usage(SETCOM, notcom);
        return;
}
count = 0;
for (t1 = t; t1 != NULL; t1 = t1->t_next)
        count++;

switch (count) {

case 1 :
        if (!notcom) {
                usage(SETCOM, notcom);
                break;
        }
        mdb[molno].m_flags &= ~(SETCM | SETCNT);
        touch(molno);
        (void) sprintf(buf, "Unset model %d center of mass.\n",
                molno);
        break;

case 4 :
case 5 :
case 6 :
        if (notcom) {
                usage(SETCOM, notcom);
                break;
        }
        t1 = t->t_next;
        for (i = 0; i < 3; i++) {
                if (sscanf(t1->t_value, "%f", &data[i]) != 1)
                        break;
                t1 = t1->t_next;
        }
        if (i != 3) {
                usage(SETCOM, notcom);
                break;
        }
        for (i= 0; i < 3; i++)
                mdb[molno].m_com[i] = data[i];
        mdb[molno].m_flags |= SETCM;
        (void) sprintf(buf, "Model %d : (%.2f %.2f %.2f)",
                molno, data[0], data[1], data[2]);
        if (count > 4) {
                (void) sscanf(t1->t_value, "%f",
```

```
                        mdb[molno].m_flags |= SETRNG;
                        t1 = t1->t_next;
                        (void) sprintf(buf, "%s, %.2f Angstroms wide",
                                buf, mdb[molno].m_size);
                }
                if (count == 6) {
                        (void) sscanf(t1->t_value, "%d",
                                &mdb[molno].m_active);
                        mdb[molno].m_flags |= SETCNT;
                        (void) sprintf(buf, "%s, %d atoms", buf,
                                mdb[molno].m_active);
                }
                strcat(buf, "\n");
                reply(buf);
                break;

        default :
                usage(SETCOM, notcom);
                break;
        }
}

/*
 * execvopt - set vdw options
 */
execvopt(t)
struct  token_def       *t;
{
        register int        len;
        register double     dens;
        extern int          HYDROGEN, OLDVDW;
        extern double       vdw_extend, vdw_density;
        extern double       atof();

        if (t == NULL) {
                usage(VDWOPT, notcom);
                return;
        }
        while (t != NULL) {
                len = strlen(t->t_value);
                if (strncmp(t->t_value, "radii", len) == 0) {
                        t = t->t_next;
                        if (notcom || t == NULL) {
                                usage(VDWOPT, notcom);
                                return;
                        }
                }
                else if (strncmp(t->t_value, "density", len) == 0) {
                        t = t->t_next;
                        if (notcom || t == NULL) {
                                usage(VDWOPT, notcom);
                                return;
                        }
                        dens = atof(t->t_value);
                        if (dens <= 0) {
                                reply("Density must be greater than 0.\n");
                                return;
                        }
                        set_dense(dens);
                }
                else if (strncmp(t->t_value, "nuc", len) == 0)
                        HYDROGEN = !notcom;
                else if (strncmp(t->t_value, "prot", len) == 0)
                        HYDROGEN = !notcom;
                else if (strncmp(t->t_value, "oldvdw", len) == 0)
                        if (notcom)
                                OLDVDW = FALSE;
                        else
                                OLDVDW = TRUE;
                else if (strncmp(t->t_value, "extend", len) == 0) {
                        if (notcom)
                                vdw_extend = 0.0;
                        else {
                                t = t->t_next;
                                if (t == NULL) {
                                        usage(VDWOPT, notcom);
                                        return;
                                }
                                vdw_extend = atof(t->t_value);
                        }
                        set_dense(vdw_density);
                }
                else
                        usage(VDWOPT, notcom);
                t = t->t_next;
        }
}
#endif  EDITONLY

/*
 * execshell - Execute a shell command
 * Note that the output of the command is sent back as replies.
```

```
execshell(t)
struct	token_def		*t;
{
	struct	token_def		*t1;
	int	cnt;
	FILE	*pfd, *popen();

	cnt = 0;
	buf[0] = '\0';
	for (t1 = t; t1 != NULL; t1 = t1->t_next) {
		cnt += strlen(t1->t_value) + 1;
		if (cnt >= BUFLEN) {
			reply("Line too long.\n");
			return;
		}
		strcat(buf, t1->t_value);
		strcat(buf, " ");
	}
#ifdef	DEBUG
	if (debug & D_PARSE)
		fprintf(stderr, "Shell escape to execute:\n%s\n", buf);
#endif
	if ((pfd = popen(buf, "r")) == NULL) {
		reply("Unable to execute shell command.\n");
		return;
	}
	while (fgets(buf, BUFLEN, pfd) != NULL)
		fputs(buf, stderr);
	pclose(pfd);
}

#ifndef	EDITONLY
#if	GRAFENG == IRIS
/*
 * execmapcolors - execute a map colors.  This just updates the data
 * on the editor side.
 */
execmapcolors(t)
struct	token_def		*t;
{
	extern int		c_range, c_factor;

	c_range = atoi(t->t_value);
	c_factor = atoi(t->t_next->t_value);
	set_black();
}
```

```
/*
 * execgcom - executes graphics command
 */
execgcom(s)
struct	spec_def		*s;
{
#ifdef	DEBUG
	if (debug & D_EDIT) {
		fprintf(stderr, "(Graphics) %s%s\n", notcom ? "un" : "",
			comtab[curcom].com_name);
		printspec(s);
	}
#endif

	setupbits();
	edit(s);
}

/*
 * execscom - execute special command
 */
execscom(w, s)
struct	wordrange_def	*w;
struct	spec_def		*s;
{
	int	number, optnum;
	int	which;
	int	*ip;
	angle_t	*ap;
	double	fixrot();
#ifndef	EDITONLY
	extern	char	*cheatbuf;
	register		i;
#endif

#ifdef	DEBUG
	if (debug & D_EDIT) {
		fprintf(stderr, "(Special) %s%s\n", notcom ? "un" : "",
			comtab[curcom].com_name);
		fprintf(stderr, "\tWords ");
		printwords(w);
		fprintf(stderr, "\n");
		printspec(s);
	}
```

```
#endif
        switch (comnum) {

        case PCOLOR :
                if (notcom) {
                        reply("What do you mean 'not color'?\n");
                        break;
                }
                colornum(w, &number, &optnum);
                if (number < 0 || number > 63 || optnum < -1 || optnum > 63) {
                        reply("Color must be between 0 and 63 or red, blue, yell
                        reply("white, cyan, magenta.\n");
                        break;
                }
                which = colormode(w->w_next);
                if (which == 0) {
                        reply("Don't know what to color.\n");
                        break;
                }
                setupcolor(number, optnum, which);
                edit(s);
                break;

        case SURFACE :
                setupsurface();
                edit(s);
                break;

        case LINK :
                if (w != NULL) {
                        usage(LINK, notcom);
                        break;
                }
                setupgetres();
                prelink();
                edit(s);
                if (redochain() == -1) {
                        usage(LINK, notcom);
                        break;
                }
#ifndef EDITONLY
#if     GRAFENG != PS300
                dostack();

#endif
#endif
                break;
```

```
case GETCRD :
        if (notcom) {
                usage(GETCRD, notcom);
                notcom = !notcom;
        }
        setupcrd(1, 1);
        edit(s);
        if (chcoord() == -1)
                usage(GETCRD, notcom);
        break;

case ALIGN :
        if (notcom) {
                usage(ALIGN, notcom);
                notcom = !notcom;
        }
        setupcrd(-1, -2);
        edit(s);
        if (chmatch(s) == -1)
                usage(ALIGN, notcom);
        break;

case MATCH :
        if (notcom) {
                usage(MATCH, notcom);
                break;
        }
        setupcrd(2, 4);
        edit(s);
        if (chmatch(s) == -1)
                usage(MATCH, notcom);
        break;

case FINROT :
case BINROT :
        if (notcom) {
                if (w == NULL) {
                        usage(FINROT, notcom);
                        break;
                }
                ip = (int *) cheatbuf;
                if (*ip++ != 0) {
                        reply("Error in communication.\n");
                        break;
                }
                ap = (angle_t *) ip;
```

```
                        rotation[i].r_rotangle = *ap++;
                while (w != NULL) {
                        number = getnum(w);
                        w = w->w_next;
                        if (number < 0 || number >= MAXINROT) {
                                usage(FINROT, notcom);
                                continue;
                        }
                        if (rotation[number].r_type == INACTIVE)
                                reply("Rotation not active.\n");
                        else
                                unrot(number);
                }
        } else {
                number = getnum(w);
                if (number < 0 || number >= MAXINROT) {
                        for (i = 0; i < MAXINROT; i++)
                                if (rotation[i].r_type == INACTIVE) {
                                        number = i;
                                        break;
                                }
                        if (number < 0 || number >= MAXINROT) {
                                reply("No more rotations.\n");
                                break;
                        }
                }
                if (w != NULL && w->w_next != NULL) {
                        reply("One rotation at a time.\n");
                        break;
                }
                if (rotation[number].r_type != INACTIVE)
                        reply("Rotation already in use.\n");
                else
                        dorot(number, s);
        }
#if     GRAFENG != PS300
        dostack();
#endif
        break;

    case REVERSE :
    case FIXREV :
        if (w == NULL || notcom) {
                usage(REVERSE, notcom);
                break;
        }
                reply("Error in communication.\n");
                break;
        }
        ap = (angle_t *) ip;
        for (i = 0; i < MAXINROT; i++)
                rotation[i].r_rotangle = *ap++;
        while (w != NULL) {
                number = getnum(w);
                w = w->w_next;
                if (number < 0 || number >= MAXINROT) {
                        reply("Illegal rotation number.\n");
                        continue;
                }
                if (rotation[number].r_type == INACTIVE)
                        reply("Rotation not in use.\n");
                else {
                        if (comnum == FIXREV)
                                /*(void)*/fixrot(number);
                        doreverse(number);
                }
        }
#if     GRAFENG != PS300
        dostack();
#endif
        break;

    case DIST :
        if (notcom) {
                if (w == NULL) {
                        usage(DIST, notcom);
                        break;
                }
                while (w != NULL) {
                        number = getnum(w);
                        w = w->w_next;
                        if (number < 0 || number >= MAXDIST) {
                                usage(DIST, notcom);
                                continue;
                        }
                        if (distance[number].d_type == INACTIVE)
                                reply("Distance not active.\n");
                        else
                                undist(number);
                }
        } else {
```

```
                        number = getnum(w);
                        if (number < 0 || number >= MAXDIST) {
                                for (i = 0; i < MAXDIST; i++)
                                        if (distance[i].d_type ==
                                        INACTIVE) {
                                                number = i;
                                                break;
                                        }
                                if (number < 0 || number >= MAXDIST) {
                                        reply("No more distances.\n");
                                        break;
                                }
                        }
                        if (w != NULL && w->w_next != NULL) {
                                reply("One distance at a time.\n");
                                break;
                        }
                        if (distance[number].d_type != INACTIVE)
                                reply("Distance already in use.\n");
                        else
                                dodist(number, s);
                }
                break;

        case ANGLE :
                if (notcom) {
                        if (w == NULL) {
                                usage(ANGLE, notcom);
                                break;
                        }
                        while (w != NULL) {
                                number = getnum(w);
                                w = w->w_next;
                                if (number < 0 || number >= MAXANGLE) {
                                        usage(ANGLE, notcom);
                                        continue;
                                }
                                if (angles[number].a_type == INACTIVE)
                                        reply("Angle not active.\n");
                                else
                                        unang(number);
                        }
                } else {
                        number = getnum(w);
                        if (number < 0 || number >= MAXANGLE) {
                                for (i = 0; i < MAXDIST; i++)
                                        if (angles[i].a_type ==
                                                number = i;
                                                break;
                                        }
                                if (number < 0 || number >= MAXANGLE) {
                                        reply("No more angles.\n");
                                        break;
                                }
                        }
                        if (w != NULL && w->w_next != NULL) {
                                reply("One angle at a time.\n");
                                break;
                        }
                        if (angles[number].a_type != INACTIVE)
                                reply("Angle already in use.\n");
                        else
                                doang(number, s);
                }
                break;

        case REDRAW :
                if (notcom) {
                        reply("What do you mean 'not redraw'?\n");
                        break;
                }
                which = colormode(w);
                if (which == 0) {
                        reply("Don't know what to redraw.\n");
                        break;
                }
                setupredraw(which);
                edit(s);
#if      GRAFENG != PS300
                dostack();
#endif
                break;
#endif   EDITONLY

#if      GRAFENG == PS300
        case PICK:
                ps300pick(w, s);
                break;
#endif

#ifndef  EDITONLY
#if      GRAFENG == IRIS
        case CPK:
```

```
                break;
#endif
#endif

        case ADDGRP :
                if (notcom || preaddgrp(w) == -1) {
                        usage(ADDGRP, notcom);
                        break;
                }
                setupadd();
                edit(s);
                if (addgrp() == -1)
                        usage(ADDGRP, notcom);
                break;

        case SWAPAA :
                if (notcom || preswapres(w) == -1) {
                        usage(SWAPAA, notcom);
                        break;
                }
                setupgetres();
                edit(s);
                if (swapres() == -1)
                        usage(SWAPAA, notcom);
                break;

        case SWAPNUC :
                if (notcom || preswapres(w) == -1) {
                        usage(SWAPNUC, notcom);
                        break;
                }
                setupgetres();
                edit(s);
                if (swapres() == -1)
                        usage(SWAPAA, notcom);
                break;

        case ADDAA :
                if (notcom || preaddres(w) == -1) {
                        usage(ADDAA, notcom);
                        break;
                }
                setupgetres();
                edit(s);
                if (addres() == -1)
                        usage(ADDAA, notcom);
        case ADDNUC :
                if (notcom || preaddres(w) == -1) {
                        usage(ADDNUC, notcom);
                        break;
                }
                setupgetres();
                edit(s);
                if (addres() == -1)
                        usage(ADDNUC, notcom);
                break;

        case DELGRP :
                if (notcom || predelgrp(w) == -1) {
                        usage(DELGRP, notcom);
                        break;
                }
                setupadd();
                edit(s);
                if (delgrp() == -1)
                        usage(DELGRP, notcom);
                break;
        }
}

#ifndef EDITONLY
#if     GRAFENG == PS2 || GRAFENG == MPS
/*
 * execpick - execute the pick command
 */
execpick(t)
struct  token_def       *t;
{
        register        i;
        int     x, y, natom;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        char    atname[AT_NAME_SIZE + 1];
        struct  atom_def        *atomdata;
        struct  token_def       *t1;
        struct  input_def       in;

        i = 0;
        for (t1 = t; t1 != NULL; t1 = t1->t_next)
                i++;
        if (i != 2)
                return;
```

```
        (void) sscanf(t->t_value, "%d", &x);
        t = t->t_next;
        (void) sscanf(t->t_value, "%d", &y);
        in.i_type = S_IDENTRES;
        in.i_x = x;
        in.i_y = y;
        process(&in, (char *) NULL);
        if (in.i_type) {
                reply("No residue hit.\n");
                return;
        }
        (void) mseekr(mdb[in.i_molnum].m_fd, in.i_name, FBSEEK);
        natom = mreadr(mdb[in.i_molnum].m_fd, resseq, restype);
        (void) mreada(mdb[in.i_molnum].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[in.i_molnum].m_fd);
        i = drawtag(in.i_molnum, resseq, restype, atomdata, x, y);
        if (i < 0 || i >= natom) {
                reply("No atom hit.\n");
                return;
        }
        (void) matom(mdb[in.i_molnum].m_fd, i, atname);
        (void) sprintf(buf, "#%d:%s@%s\n", in.i_molnum, resseq, atname);
        reply(buf);
}
#endif  PS2 || MPS

#if     GRAFENG == PS300
static char     pickatom[AT_NAME_SIZE + 1];
static int      pickcolor;
static int      pickvec;
static int      newchain, newlink;
static int      firstatom;
static int      res_halfbond;

ps300pick(w, s)
struct  wordrange_def   *w;
struct  spec_def        *s;
{
        int     pickvisit(), pickagain();
        char    *cp;
        int     mfd, molnum;
        char    *resseq;
        struct specist_def      *slp;
        struct wordrange_def    *wp;
        struct input_def        in;

        /* Get the vector list type */
        if (w == NULL) {
                reply("No vector list description.\n");
                return;
        }
        cp = w->w_first;

        /* Get the vector number */
        w = w->w_next;
        if (w == NULL) {
                reply("No vector number.\n");
                return;
        }
        pickvec = atoi(w->w_first);

        /* Make sure we have an atom description */
        if (s == NULL) {
                reply("No atom description.\n");
                return;
        }

        /* Get the model number */
        slp = s->s_model;
        if (slp == NULL) {
                reply("No model description.\n");
                return;
        }
        wp = slp->sp_range;
        if (wp == NULL || wp->w_first == NULL) {
                reply("No model specifier description.\n");
                return;
        }
        molnum = atoi(wp->w_first);
        mfd = mdb[molnum].m_fd;

        /* Get the residue sequence */
        slp = s->s_residue;
        if (slp == NULL) {
                reply("No residue description.\n");
                return;
        }
        wp = slp->sp_range;
        if (wp == NULL || wp->w_first == NULL) {
                reply("No residue specifier description.\n");
                return;
        }
        resseq = wp->w_first;
```

```
	/* Get the halfbond mode and successor of this residue */
	in.i_type = S_FINDNEXT;
	in.i_molnum = molnum;
	strncpy(in.i_name, resseq, RES_SEQ_SIZE + 1);
	process(&in, (char *) NULL);
	res_halfbond = in.i_halfbond;

	/* Find the residue in the model database */
	if (strcmp(cp, "link") == 0) {
		/* If this is the front end, use the given residue sequence.
		 * Otherwise, use the successor. */
		if (in.i_type) {
			(void) sprintf(buf, "No successor to %s in model %d.\n",
				resseq, molnum);
			reply(buf);
			return;
		}
		if (pickvec == 1 || (res_halfbond && pickvec == 2))
			pickvec = MAXATOM;
		else {
			resseq = in.i_altname;
			pickvec = 1;
		}
	}
	if (mseekr(mfd, resseq, FBSEEK) == -1) {
		(void) sprintf(buf, "No residue %s in model %d.\n",
			resseq, molnum);
		reply(buf);
		return;
	}
	(void) mreada(mfd);

	/* Get the atom name and return it */
	newchain = newlink = FALSE;
	firstatom = TRUE;
	pickcolor = -1;
	mtrav(mfd, pickvisit, pickagain);
	psprintf("send '#%d:%s@%s' to <1>edit_input;\n", molnum,
		resseq, pickatom);
}

static
pickvisit(db, index, ischief, islink, nsons, firstime)
int	db;
int	index, ischief, islink, nsons, firstime;
{
	struct atom_def	data;

	if (pickvec <= 0)
		return;
	(void) mgeta(db, index, &data);
	if (data.status & STARTBIT && firstime)
		newchain = TRUE;
	if (!(data.status & EXISTBIT))
		return;
	if (data.status & SHOWBIT) {
		if ((data.status & BREAKBIT || newchain) && firstime) {
			newchain = FALSE;
		}
		else {
			if (pickcolor != data.color)
				if (pickcolor != -1 && res_halfbond)
					if (--pickvec)
						return;
		}
		pickcolor = data.color;
		pickvec--;
		(void) matom(db, index, pickatom);
	}
	firstatom = FALSE;
}

static
pickagain(db, index, ischief, islink, nsons)
int	db;
int	index, ischief, islink, nsons;
{
	struct atom_def	data;

	if (pickvec <= 0)
		return;
	if (islink)
		newlink = TRUE;
	(void) mgeta(db, index, &data);
	if (!(data.status & EXISTBIT))
		return;

	if ((data.status & SHOWBIT) && (nsons > 0 || newlink)) {
		if (res_halfbond && pickcolor != data.color)
			pickcolor = data.color;
		pickvec--;
		newlink = FALSE;
	}
}
```

```
#endif
#endif	EDITONLY

#ifdef	notdef
/*
 * execncom - execute internal command (e.g. open, cd)
 *	Used for debugging of editor.
 */
execncom(t)
struct	token_def		*t;
{
	if (debug & D_EDIT) {
		fprintf(stderr, "(No-edit) %s%s\n", notcom ? "un" : "",
			comtab[curcom].com_name);
		printtoken(t);
	}
	/* ARGSUSED */
}
#endif
```

rotdist.c

```
/* $Header: rotdist.c,v 3.14 86/10/15 10:27:29 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          22 Jul 1983
 */

#include "editor.h"
#ifdef  vms
#if     GRAFENG == PS300
#include "midas$h:fakegsr.h"
#endif
#endif

#define PI      3.14159

static  int     rotcomnum, distcomnum, angcomnum;

/*
 * unrot - remove rotation 'n'
 */
unrot(n)
int     n;
{
        register        i, j;
        int     savenum;
        double  fixrot(), t;
        int     mol;
        char    *rseq;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        struct  atom_def        *atomdata;
        struct  surf_def        *surfdata;

        if (rotation[n].r_type == INACTIVE) {
                reply("Rotation not active.\n");
                return;
        }
        t = fixrot(n);
        if (t == 0)
                (void) sprintf(buf, "Rotation %d removed.\n", n);
        else
                (void) sprintf(buf, "Rotation %d fixed at %.1f degrees.\n",
                        n, t);
        reply(buf);
#if     GRAFENG == PS300
        if (rotation[n].r_locname[0] != NULL) {
                free(rotation[n].r_locname[0]);
                rotation[n].r_locname[0] = NULL;
        }
        if (rotation[n].r_locname[1] != NULL) {
                free(rotation[n].r_locname[1]);
                rotation[n].r_locname[1] = NULL;
        }
#endif
        rotation[n].r_type = INACTIVE;
        nrot--;
        unrankrot(n);
        switch (rotation[n].r_natom) {

        case 2 :
                j = 1;
                break;

        case 4 :
                j = 2;
                break;
        }

        /* We must edit the affected residue since we want the matrices
         * to be correct.  If there is another rotation on the same atom
         * though, we do NOT want to turn off the rotation bit, therefore
         * we make the editor think that it should turn ON that bit instead
         * of turning it off.
         */
        i = -1;
        while ((i = findrot(rotation[n].r_model, rotation[n].r_resseq[j],
                rotation[n].r_atom[j], 0, i + 1)) >= 0)
                        if (i != n)
                                notcom = FALSE;
        savenum = curcom;
        curcom = rotcomnum;
        makebit(rotation[n].r_model, rotation[n].r_resseq[j],
                rotation[n].r_atom[j]);
        curcom = savenum;

        /* Redraw the residue */
        mol = rotation[n].r_model;
        rseq = rotation[n].r_resseq[j];
        (void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
        (void) mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
```

```
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#if     GRAFENG == PS300
        drawresidue(mol, resseq, restype, atomdata);
#endif  PS300
        drawbond(mol, resseq, restype, atomdata);
        drawlabel(mol, resseq, restype, atomdata);
        if (mdb[mol].s_fd >= 0 && mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                (void) mreada(mdb[mol].s_fd);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        } else
                surfdata = NULL;
        drawsurf(mol, resseq, restype, atomdata, surfdata);
}

/*
 * doreverse - reverse rotation 'n'
 */
doreverse(n)
int     n;
{
        register        i, j;
        int     mode;
        int     mol;
        char    *rseq;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        struct  atom_def        *atomdata;
        struct  surf_def        *surfdata;

        if (rotation[n].r_type == INACTIVE) {
                reply("Rotation not active.\n");
                return;
        }
        switch (rotation[n].r_natom) {

        case    2 :
                j = 1;
                break;

        case    4 :
                j = 2;
                break;
        }
        if (rotation[n].r_type & BACKWRD)
                mode = 1;
        else
                mode = 2;
        i = findrot(rotation[n].r_model, rotation[n].r_resseq[j],
                rotation[n].r_atom[j], mode, 0);
        if (i != -1) {
                (void) sprintf(buf,
                        "Rotation %d is already in that direction.\n", i);
                reply(buf);
                return;
        }
        unrankrot(n);
        rotation[n].r_type ^= BACKWRD;
        if (!rankrot(n)) {
                reply("Reversing will conflict with existing rotations.\n");
                rotation[n].r_type ^= BACKWRD;
                (void) rankrot(n);
                return;
        }
        rotation[n].r_type |= REMAKEMAT;
        makebit(rotation[n].r_model, rotation[n].r_resseq[j],
                rotation[n].r_atom[j]);

        /* Redraw the residue */
        mol = rotation[n].r_model;
        rseq = rotation[n].r_resseq[j];
        (void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
        (void) mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#if     GRAFENG == PS300
        drawresidue(mol, resseq, restype, atomdata);
#endif  PS300
        drawbond(mol, resseq, restype, atomdata);
        drawlabel(mol, resseq, restype, atomdata);
        if (mdb[mol].s_fd >= 0 && mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                (void) mreada(mdb[mol].s_fd);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        } else
                surfdata = NULL;
        drawsurf(mol, resseq, restype, atomdata, surfdata);
}

/*
 * dorot - add rotation 'n'
 */
dorot(n, s)
int     n;
struct  spec_def        *s;
{
        register        i, j;
```

```
        int     mol;
        char    *rseq;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        struct  atom_def        *atomdata;
        struct  surf_def        *surfdata;

        setuprot(n);
        edit(s);
        switch (rotation[n].r_natom) {

        case 2 :
                j = 1;
                break;

        case 4 :
                j = 2;
                break;

        default :
                reply("Wrong number of atoms specified.\n");
                /* Fall through */
        case -1 :
                reply("Rotation not incorporated.\n");
                return;
        }
        if (comnum == BINROT) {
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Backward rotation.\n");
#endif
                if ((i = findrot(rotation[n].r_model, rotation[n].r_resseq[j],
                        rotation[n].r_atom[j], 2, 0)) >= 0)
                                n = -1;
                if (n != -1)
                        rotation[n].r_type |= BACKWRD;
        } else {
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Forward rotation.\n");
#endif
                if ((i = findrot(rotation[n].r_model, rotation[n].r_resseq[j],
                        rotation[n].r_atom[j], 1, 0)) >= 0)
                                n = -1;
        }
        if (n == -1) {
                (void) sprintf(buf,
                        "Rotation %d already uses the pivot atom.\n", i);
                reply(buf);
                return;
        }
        nrot++;
        rotation[n].r_type |= (ACTIVE | REMAKEMAT);
        makebit(rotation[n].r_model, rotation[n].r_resseq[j],
                rotation[n].r_atom[j]);

        /* Figure out whether the rotation is mainchain */
        mol = rotation[n].r_model;
        rseq = rotation[n].r_resseq[j];
        (void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
        (void) mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
        i = mseeka(mdb[mol].m_fd, rotation[n].r_atom[j]);
        if (mchain(mdb[mol].m_fd, i))
                rotation[n].r_type |= MCHAIN;

        if (!rankrot(n)) {
                notcom = TRUE;
                nrot--;
                rotation[n].r_type = INACTIVE;
                i = -1;
                while ((i = findrot(rotation[n].r_model,
                        rotation[n].r_resseq[j], rotation[n].r_atom[j],
                        0, i + 1)) >= 0)
                                if (i != n)
                                        notcom = FALSE;
                makebit(rotation[n].r_model, rotation[n].r_resseq[j],
                        rotation[n].r_atom[j]);
                reply("Rotation is incompatible with existing ones.\n");
                return;
        }

        /* Redraw the residue */
        mol = rotation[n].r_model;
        rseq = rotation[n].r_resseq[j];
        (void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
        (void) mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#if     GRAFENG == PS300
        drawresidue(mol, resseq, restype, atomdata);
#endif  PS300
        drawbond(mol, resseq, restype, atomdata);
        drawlabel(mol, resseq, restype, atomdata);
```

```
        if (mdb[mol].s_fd >= 0 && mseekr(mdb[mol].s_fd, reseq, FBSEEK) == 0) {
                (void) mreada(mdb[mol].s_fd);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        } else
                surfdata = NULL;
        drawsurf(mol, reseq, restype, atomdata, surfdata);

        (void) sprintf(buf, "Rotation %d added.\n", n);
        reply(buf);

        return;
}

/*
 * makemat makes the matrices for the internal rotations
 */
makemats(n)
int     n;
{
        ps_t                    front[3], back[3];
        register int            i;
        double                  t;
        extern double           dihed();

#ifdef  DEBUG
        if (debug & D_PG)
                fprintf(stderr, "Remaking rotation %d matrices\n", n);
#endif  DEBUG
        switch (rotation[n].r_natom) {

        case 2 :
                if (rotation[n].r_type & BACKWRD)
                        for (i = 0; i < 3; i++) {
                                front[i] = MAPCRD(rotation[n].r_loc[1][i]);
                                back[i] = MAPCRD(rotation[n].r_loc[0][i]);
                        }
                else
                        for (i = 0; i < 3; i++) {
                                front[i] = MAPCRD(rotation[n].r_loc[0][i]);
                                back[i] = MAPCRD(rotation[n].r_loc[1][i]);
                        }
#if     GRAFENG == IRIS
                m_lookat(rotation[n].r_mat, rotation[n].r_invmat, front, back);
#else   IRIS
                lookat(rotation[n].r_mat, rotation[n].r_invmat, front, back);
#endif  IRIS
                rotation[n].r_initangle = 0;
                break;

        case 4 :
                if (rotation[n].r_type & BACKWRD)
                        for (i = 0; i < 3; i++) {
                                front[i] = MAPCRD(rotation[n].r_loc[2][i]);
                                back[i] = MAPCRD(rotation[n].r_loc[1][i]);
                        }
                else
                        for (i = 0; i < 3; i++) {
                                front[i] = MAPCRD(rotation[n].r_loc[1][i]);
                                back[i] = MAPCRD(rotation[n].r_loc[2][i]);
                        }
#if     GRAFENG == IRIS
                m_lookat(rotation[n].r_mat, rotation[n].r_invmat, front, back);
#else   IRIS
                lookat(rotation[n].r_mat, rotation[n].r_invmat, front, back);
#endif  IRIS
                t = dihed(rotation[n].r_loc[0], rotation[n].r_loc[1],
                        rotation[n].r_loc[2], rotation[n].r_loc[3]);
                if (t < 0)
                        t += PI * 2;
                rotation[n].r_initangle = MAPANGLE(t * (180.0 / PI));
                break;

        default :
                fprintf(stderr, "%d atoms in rotation %d\n",
                        rotation[n].r_natom, n);
                break;
        }
        rotation[n].r_type &= ~ REMAKEMAT;
}

/*
 * undist - remove a distance
 */
undist(n)
int     n;
{
        register                i, j;
        int             k;
        int             savenum;
        char            save[2];

        if (distance[n].d_type == INACTIVE) {
                reply("Distance not active.\n");
                return;
```

```
	}
	ndist--;
	distance[n].d_type = INACTIVE;
	(void) sprintf(buf, "Distance %d removed.\n", n);
	reply(buf);

	save[0] = save[1] = FALSE;
	for (k = 0; k < MAXINROT; k++) {
		if (k == n)
			continue;
		for (i = 0; i < 2; i++) {
			if (save[i])
				continue;
			for (j = 0; j < 2; j++) {
				if (distance[n].d_model[i] !=
					distance[k].d_model[j])
						continue;
				if (smatch(distance[n].d_resseq[i],
					distance[k].d_resseq[j]))
						continue;
				if (amatch(distance[n].d_atom[i],
					distance[k].d_atom[j]))
						continue;
				save[i] = TRUE;
			}
		}
	}
	savenum = curcom;
	curcom = distcomnum;
	for (i = 0; i < 2; i++)
		if (!save[i])
			makebit(distance[n].d_model[i], distance[n].d_resseq[i],
				distance[n].d_atom[i]);
#if	GRAFENG == PS300
	if (distance[n].d_vecname[0] != NULL)
		free(distance[n].d_vecname[0]);
	if (distance[n].d_vecname[0] != NULL)
		free(distance[n].d_vecname[1]);
	distance[n].d_vecname[0] = distance[n].d_vecname[1] = NULL;
	if (ndist == 0) {
		/* Normally drawpic() does this.  But if we just removed
		 * the last one, it won't know that it ought to do this. */
		reset_disttab();
		PSndBool(TRUE, 1, "distroute");
		PDisc("disttrigger", 2, 1, "distroute");
	}
#endif
		curcom = savenum;
}

/*
 * dodist - add a distance
 */
dodist(n, s)
int	n;
struct	spec_def		*s;
{
	setupdist(n);
	edit(s);
	if (distance[n].d_type == ACTIVE) {
		ndist++;
		makebit(distance[n].d_model[0], distance[n].d_resseq[0],
			distance[n].d_atom[0]);
		makebit(distance[n].d_model[1], distance[n].d_resseq[1],
			distance[n].d_atom[1]);
		(void) sprintf(buf, "Distance %d added.\n", n);
#if	GRAFENG == PS300
		if (ndist == 1) {
			PConnect("disttrigger", 2, 1, "distroute");
			PSndBool(TRUE, 1, "distroute");
		}
#endif
	} else {
		distance[n].d_type = INACTIVE;
		(void) sprintf(buf, "Wrong number of atoms specified.\n");
	}
	reply(buf);
}

/*
 * unang - remove an angle
 */
unang(n)
int	n;
{
	register		i, j;
	int	k;
	int	savenum;
	char	save[4];

	if (angles[n].a_type == INACTIVE) {
		reply("Angle not active.\n");
		return;
	}
```

```
		nangle--;
		angles[n].a_type = INACTIVE;
		(void) sprintf(buf, "Angle %d removed.\n", n);
		reply(buf);

		save[0] = save[1] = save[2] = save[3] = FALSE;
		for (k = 0; k < MAXANGLE; k++) {
			if (k == n)
				continue;
			for (i = 0; i < angles[n].a_natom; i++) {
				if (save[i])
					continue;
				for (j = 0; j < angles[k].a_natom; j++) {
					if (angles[n].a_model[i] !=
						angles[k].a_model[j])
							continue;
					if (smatch(angles[n].a_resseq[i],
						angles[k].a_resseq[j]))
							continue;
					if (amatch(angles[n].a_atom[i],
						angles[k].a_atom[j]))
							continue;
					save[i] = TRUE;
				}
			}
		}
		savenum = curcom;
		curcom = angcomnum;
		for (i = 0; i < angles[n].a_natom; i++)
			if (!save[i])
				makebit(angles[n].a_model[i], angles[n].a_resseq[i],
					angles[n].a_atom[i]);
		curcom = savenum;
}

/*
 * doang - add a angle
 */
doang(n, s)
int	n;
struct	spec_def		*s;
{
		setupang(n);
		edit(s);
		if (angles[n].a_type == ACTIVE) {
			nangle++;
			makebit(angles[n].a_model[0], angles[n].a_resseq[0],
				angles[n].a_atom[0]);
			makebit(angles[n].a_model[1], angles[n].a_resseq[1],
				angles[n].a_atom[1]);
			makebit(angles[n].a_model[2], angles[n].a_resseq[2],
				angles[n].a_atom[2]);
			if (angles[n].a_natom == 4)
				makebit(angles[n].a_model[3], angles[n].a_resseq[3],
					angles[n].a_atom[3]);
			(void) sprintf(buf, "Angle %d added.\n", n);
		} else {
			angles[n].a_type = INACTIVE;
			(void) sprintf(buf, "Wrong number of atoms specified.\n");
		}
		reply(buf);
}

/*
 * getcomnum - gets the command index associated with the commands
 */
getcomnum()
{
		register		i;

		for (i = 0; comtab[i].com_type != NOMORECOM; i++)
			switch (comtab[i].com_num) {

			case DIST :
				distcomnum = i;
				break;

			case FINROT :
				rotcomnum = i;
				break;

			case ANGLE :
				angcomnum = i;
				break;
			}
}
```

writeeo.c

```
/* $Header: writeeo.c,v 3.21 86/03/12 11:10:17 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "editor.h"
#ifdef  vms
#define index   strchr
#endif  vms

#if     GRAFENG == PS2 || GRAFENG == MPS
#define MAXVALUE        K32K
#else
#define MAXVALUE        1.0e38
#endif

/*
 * writeeo - writes information about molecules, rotations, distances,
 * etc. out on the standard output
 */
writeeo()
{
        register        i, j;
        float   range, t, com[3];
#ifdef  DEBUG
        int     count;
#endif
#ifdef  DRAWPIC
        ps_t    value;
#endif

#if     GRAFENG == IRIS
        gflush();
#ifdef  EMULATE
        end_egl();
#endif  EMULATE
#endif  IRIS

        if (verbose) {
#ifdef  DEBUG
                if (debug & D_LEX)
                        fprintf(stderr, "%d molecules\n", nmol);
                count = 0;
#endif
                printf("%d molecules\n", nmol);
#ifdef  DRAWPIC
#if     GRAFENG != IRIS
                wr = wu = wh = -MAXVALUE;
                wl = wd = wy = MAXVALUE;
#else   IRIS
                /*
                 * IRIS uses right handed coordinate system (rah! rah!)
                 */
                wr = wu = wy = -MAXVALUE;
                wl = wd = wh = MAXVALUE;
#endif  IRIS
#endif  DRAWPIC
                for (i = 0; i < MAXMOD; i++) {
                        if (mdb[i].m_fname == NULL) {
                                continue;
                        }
                        for (j = 0; j < 3; j++) {
                                if (mdb[i].m_flags & SETCM)
                                        com[j] = mdb[i].m_com[j];
                                else if (mdb[i].m_active > 0)
                                        com[j] = mdb[i].m_com[j]
                                                / mdb[i].m_active;
                                else
                                        com[j] = 0.;
                        }
                        if (mdb[i].m_flags & SETRNG)
                                range = mdb[i].m_size;
                        else {
                                range = 0;
                                for (j = 0; j < 3; j++) {
                                        t = mdb[i].m_max[j] - com[j];
                                        if (t > range)
                                                range = t;
                                        t = com[j] - mdb[i].m_min[j];
                                        if (t > range)
                                                range = t;
                                }
                        }
                        printf("%d %d %f %f %f %f\n", i, mdb[i].m_active,
                                com[0], com[1], com[2], range);
                        if (mdb[i].m_flags & PSOB)
                                printf("object %s\n", mdb[i].m_fname);
                        else if (mdb[i].s_fd >= 0)
                                printf("model %s %s\n", mdb[i].m_fname,
                                        mdb[i].m_aname);
```

```
                else
                        printf("model %s\n", mdb[i].m_fname);
#ifdef  DEBUG
                if (debug & LD_LEX)
                        fprintf(stderr, "%d %s %d %f %f %f %f\n", i,
                                mdb[i].m_fname, mdb[i].m_active, c
                                com[1], com[2], range);
                count++;
#endif
#ifdef  DRAWPIC
                value = MAPCRD(com[0] + range);
                if (value > wr)
                        wr = value;
                value = MAPCRD(com[0] - range);
                if (value < wl)
                        wl = value;
                value = MAPCRD(com[1] + range);
                if (value > wu)
                        wu = value;
                value = MAPCRD(com[1] - range);
                if (value < wd)
                        wd = value;
                value = MAPCRD(com[2] + range);
                if (value > wy)
                        wy = value;
                value = MAPCRD(com[2] - range);
                if (value < wh)
                        wh = value;
#endif
        }
#ifdef  DEBUG
        if (count != nmol)
                fprintf(stderr, "Editor: wrong number of molecules\n");
        count = 0;
        if (debug & D_LEX)
                fprintf(stderr, "%d rotations\n", nrot);
#endif
        /* WARNING : The last portion (starting with '?rotation')
         * of this line will be used by the interactive module
         * as a command to the editor when it restarts from
         * a session file.  Therefore, the format is EXTREMELY
         * important */
        printf("%d rotations\n", nrot);
        for (i = 0; i < MAXINROT; i++) {
                if (rotation[i].r_type == INACTIVE) {
                        continue;
                }
#define FORMAT "%d %.2f %s %d #%d :%s @%s"
#ifdef  DEBUG
                count++;
                if (debug & D_LEX)
                        fprintf(stderr, FORMAT, i,
                                (double) UNMAPANGLE(rotation[i].r_initangle),
                                (rotation[i].r_type & BACKWRD) ?
                                "brotation" : "frotation", i,
                                rotation[i].r_model,
                                rotation[i].r_resseq[0],
                                rotation[i].r_atom[0]);
#endif
                printf(FORMAT, i,
                        (double) UNMAPANGLE(rotation[i].r_initangle),
                        (rotation[i].r_type & BACKWRD) ?
                        "brotation" : "frotation", i,
                        rotation[i].r_model, rotation[i].r_resseq[0],
                        rotation[i].r_atom[0]);
#undef  FORMAT
                for (j = 1; j < rotation[i].r_natom; j++) {
                        if (smatch(rotation[i].r_resseq[j - 1],
                           rotation[i].r_resseq[j])) {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, " :%s",
                                           rotation[i].r_resseq[j]);
#endif  DEBUG
                                printf(" :%s", rotation[i].r_resseq[j]);
                        }
#ifdef  DEBUG
                        if (debug & D_LEX)
                                fprintf(stderr, " @%s",
                                   rotation[i].r_atom[j]);
#endif  DEBUG
                        printf(" @%s", rotation[i].r_atom[j]);
                }
#ifdef  DEBUG
                if (debug & D_LEX)
                        fprintf(stderr, "\n");
#endif  DEBUG
                printf("\n");
        }
#ifdef  DEBUG
        if (count != nrot)
                fprintf(stderr, "Editor: wrong number of rotations\n");
        count = 0;
        if (debug & D_LEX)
```

```
                        fprintf(stderr, "%d distances\n", ndist);
#endif  DEBUG
                /* WARNING : The last portion (starting with 'distance')
                 * of this line will be used by the interactive module
                 * as a command to the editor when it restarts from
                 * a session file.  Therefore, the format is EXTREMELY
                 * important */
                printf("%d distances\n", ndist);
                for (i = 0; i < MAXDIST; i++) {
                        if (distance[i].d_type == INACTIVE)
                                continue;
#ifdef  DEBUG
                        count++;
#endif  DEBUG

#if     GRAFENG != IRIS
                        if (distance[i].d_type & HOLDALL) {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, "%d 0 0 ", i);
#endif  DEBUG
                                printf("%d 0 0\n", i);
                        }
                        else {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, "%d %u %u ", i,
                                                distance[i].d_psloc[0],
                                                distance[i].d_psloc[1]);
#endif  DEBUG
                                printf("%d %u %u\n", i, distance[i].d_psloc[0],
                                        distance[i].d_psloc[1]);
                        }
#ifdef  DEBUG
                        if (debug & LD_LEX)
                                fprintf(stderr, "oblocs = %u %u\n",
                                        distance[i].d_obloc[0],
                                        distance[i].d_obloc[1]);
#endif  DEBUG
#else   IRIS
#ifdef  DEBUG
                        if (debug & D_LEX)
                                fprintf(stderr, "%d\n", i);
#endif  DEBUG
                        printf("%d\n", i);

#ifdef  DEBUG
                        if (debug & D_LEX) {
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        distance[i].d_x[0], distance[i].d_y[0],
                                        distance[i].d_z[0]);
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        distance[i].d_x[1], distance[i].d_y[1],
                                        distance[i].d_z[1]);
                        }
#endif  DEBUG
                        printf("%.3f %.3f %.3f\n", distance[i].d_x[0],
                            distance[i].d_y[0], distance[i].d_z[0]);
                        printf("%.3f %.3f %.3f\n", distance[i].d_x[1],
                            distance[i].d_y[1], distance[i].d_z[1]);
#endif  IRIS

#ifdef  DEBUG
                        if (debug & D_LEX) {
                                fprintf(stderr, "distance %d # %d : %s @ %s ",
                                        i, distance[i].d_model[0],
                                        distance[i].d_resseq[0],
                                        distance[i].d_atom[0]);
                                fprintf(stderr, "# %d : %s @ %s\n",
                                        distance[i].d_model[1],
                                        distance[i].d_resseq[1],
                                        distance[i].d_atom[1]);
                        }
#endif  DEBUG
                        printf("distance %d # %d : %s @ %s ", i,
                                distance[i].d_model[0], distance[i].d_resseq[0],
                                distance[i].d_atom[0]);
                        printf("# %d : %s @ %s\n", distance[i].d_model[1],
                                distance[i].d_resseq[1], distance[i].d_atom[1]);
                }
#ifdef  DEBUG
                if (count != ndist)
                        fprintf(stderr, "Editor: wrong number of distances\n");
                count = 0;
                if (debug & D_LEX)
                        fprintf(stderr, "%d angles\n", nangle);
#endif  DEBUG
                /* WARNING : The last portion (starting with 'angle')
                 * of this line will be used by the interactive module
                 * as a command to the editor when it restarts from
                 * a session file.  Therefore, the format is EXTREMELY
                 * important */
                printf("%d angles\n", nangle);
                for (i = 0; i < MAXANGLE; i++) {
```

```
                        if (angles[i].a_type == INACTIVE)
                                continue;
#ifdef  DEBUG
                        count++;
#endif  DEBUG
#if     GRAFENG != IRIS
                        if (angles[i].a_type & HOLDALL) {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, "%d 0 0 0 ", i);
#endif  DEBUG
                                printf("%d 0 0 0 ", i);
                        }
                        else {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, "%d %u %u %u ", i,
                                                angles[i].a_psloc[0],
                                                angles[i].a_psloc[1],
                                                angles[i].a_psloc[2]);
#endif  DEBUG
                                printf("%d %u %u %u ", i,
                                        angles[i].a_psloc[0],
                                        angles[i].a_psloc[1],
                                        angles[i].a_psloc[2]);
                        }
#ifdef  DEBUG
                        if (debug & LD_LEX)
                                fprintf(stderr, "(oblocs = %u %u %u) ",
                                        angles[i].a_obloc[0],
                                        angles[i].a_obloc[1],
                                        angles[i].a_obloc[2]);
#endif  DEBUG
#ifdef  DEBUG
                        if (debug & D_LEX)
                                fprintf(stderr, "%u\nangle %d",
                                        (angles[i].a_natom == 4) ?
                                        angles[i].a_psloc[3] : 0, i);
#endif  DEBUG
                        printf("%u\n", (angles[i].a_natom == 4) ?
                                angles[i].a_psloc[3] : 0);
#else   IRIS
#ifdef  DEBUG
                        if (debug & D_LEX)
                                fprintf(stderr, "%d\n", i);
#endif  DEBUG
                        printf("%d\n", i);
#ifdef  DEBUG
                        if (debug & D_LEX) {
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        angles[i].a_x[0], angles[i].a_y[0],
                                        angles[i].a_z[0]);
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        angles[i].a_x[1], angles[i].a_y[1],
                                        angles[i].a_z[1]);
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        angles[i].a_x[2], angles[i].a_y[2],
                                        angles[i].a_z[2]);
                                fprintf(stderr, "%.3f %.3f %.3f\n",
                                        angles[i].a_x[3], angles[i].a_y[3],
                                        angles[i].a_z[3]);
                        }
#endif  DEBUG
                        printf("%.3f %.3f %.3f\n", angles[i].a_x[0],
                            angles[i].a_y[0], angles[i].a_z[0]);
                        printf("%.3f %.3f %.3f\n", angles[i].a_x[1],
                            angles[i].a_y[1], angles[i].a_z[1]);
                        printf("%.3f %.3f %.3f\n", angles[i].a_x[2],
                            angles[i].a_y[2], angles[i].a_z[2]);
                        if (angles[i].a_natom == 4)
                                printf("%.3f %.3f %.3f\n", angles[i].a_x[3],
                                    angles[i].a_y[3], angles[i].a_z[3]);
                        else
                                printf("0.0 0.0 0.0\n");
#endif  IRIS
#ifdef  DEBUG
                        if (debug & D_LEX)
                                fprintf(stderr, "angle %d", i);
#endif  DEBUG
                        printf("angle %d", i);
                        for (j = 0; j < angles[i].a_natom; j++) {
#ifdef  DEBUG
                                if (debug & D_LEX)
                                        fprintf(stderr, " #%d:%s@%s",
                                                angles[i].a_model[j],
                                                angles[i].a_resseq[j],
                                                angles[i].a_atom[j]);
#endif  DEBUG
                                printf(" #%d:%s@%s",
                                        angles[i].a_model[j],
                                        angles[i].a_resseq[j],
                                        angles[i].a_atom[j]);
                        }
```

writeeo.c

```
#ifdef	DEBUG
			if (debug & D_LEX)
				fprintf(stderr, "\n");
#endif	DEBUG
			printf("\n");
		}
#ifdef	DEBUG
		if (count != nangle)
			fprintf(stderr, "Editor: wrong number of angles\n");
		if (debug & D_LEX)
			fprintf(stderr, "%d replies\n", nreplies);
#endif	DEBUG
		printf("%d replies\n", nreplies);
		for (i = 0; i < nreplies; i++) {
			fputs(repbuf[i], stdout);
#ifdef	DEBUG
			if (debug & LD_LEX)
				fputs(repbuf[i], stderr);
#endif	DEBUG
		}
		fputs("SYNC\n", stdout);
		(void) fflush(stdout);
	}
}
```

unused.c

```
/* $Header: unused.c,v 3.3 86/06/03 14:23:51 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          28 Jul 1983
 */

#include "editor.h"
#include <ctype.h>

struct  unused_def      {
        char    *u_name;
        struct  unused_def      *u_next;
};

static  struct  unused_def      *molname, *resname, *atmname;

static
do_insert(q, name)
struct  unused_def      **q;
char    *name;
{
        register        struct  unused_def      *p;

        for (p = *q; p != NULL; p = p->u_next)
                if (p->u_name == name)
                        return;
        if (p == NULL) {
                p = (struct unused_def *) alloc(1, sizeof (struct unused_def));
                p->u_next = *q;
                p->u_name = name;
                *q = p;
        }
}

static
remove(q, name)
struct  unused_def      **q;
char    *name;
{
        register        struct  unused_def      *p;

        for (p = *q; p != NULL; p = p->u_next)
                if (p->u_name == name) {
                        *q = p->u_next;
                        free((char *) p);
                        return;
                } else
                        q = &p->u_next;
}

insmol(name)
char    *name;
{
        do_insert(&molname, name);
}

insres(name)
char    *name;
{
        do_insert(&resname, name);
}

insatm(name)
char    *name;
{
        do_insert(&atmname, name);
}

delmol(name)
char    *name;
{
        remove(&molname, name);
}

delres(name)
char    *name;
{
        remove(&resname, name);
}

delatm(name)
char    *name;
{
        remove(&atmname, name);
}

prunused()
{
        register        struct  unused_def      *p;
        register        i;
```

unused.c

```
        struct  token_def       t;
        struct  unused_def      *p1, **save;
        char    lbuf[80];

        buf[0] = '\0';
        t.t_type = NAME;
        save = &molname;
        for (p = molname; p != NULL; p = p1) {
                p1 = p->u_next;
                t.t_value = p->u_name;
                i = matchmol(&t);
                if (i < 0 || i >= MAXMOD) {
                        save = &p->u_next;
                        continue;
                }
                if (mdb[i].m_fname != NULL && mdb[i].m_flags & PSOB) {
                        if (buf[0] != '\0')
                                strcat(buf, ",");
                        strcat(buf, p->u_name);
                        free((char *) p);
                        *save = p1;
                }
        }
        if (buf[0] != '\0') {
                (void) sprintf(lbuf, "Objects %s cannot be edited.\n", buf);
                reply(lbuf);
        }
        uprint(&molname, "Models");
        uprint(&resname, "Residues");
        uprint(&atmname, "Atoms");
}

uprint(q, msg)
struct  unused_def      **q;
char    *msg;
{
        register        struct  unused_def      *p, *p1;
        register        char    *cp;
        char    lbuf[80];

        buf[0] = '\0';
        for (p = *q; p != NULL; p = p1) {
                if (strcmp(p->u_name, all)) {
                        if (buf[0] != '\0')
                                strcat(buf, ",");
                        for (cp = p->u_name; *cp; cp++)
                                if (!isprint(*cp))
                                        *cp = '?';
                        strcat(buf, p->u_name);
                }
                if (strlen(buf) > sizeof lbuf - 22) {
                        /* 22 = " not found.\n" + "Residues" + " " */
                        (void) sprintf(lbuf, "%s %s not found.\n", msg, buf);
                        reply(lbuf);
                        buf[0] = '\0';
                }
                p1 = p->u_next;
                free((char *) p);
        }
        if (buf[0] != '\0') {
                (void) sprintf(lbuf, "%s %s not found.\n", msg, buf);
                reply(lbuf);
        }
        *q = NULL;
}
```

rankrot.c

```
/* $Header: /usr/src/local/midas/src/editor/RCS/rankrot.c,v 3.3 85/04/04 22:17:05 tef E
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "editor.h"

/*
 * rankrot - rank the rotation in terms of whether it came first
 *          also check whether it collides with existing rotation
 */
rankrot(n)
int     n;
{
        struct  chain_def       *ch, *findchain();
        register                i, count;
        int     statf, statb;
        int     *last;
        int     *rotf, *rotb;

        rotation[n].r_linkf = -1;
        rotation[n].r_linkb = -1;

        ch = findchain(n);
        rotf = &ch->ch_rotf;
        rotb = &ch->ch_rotb;

        if (rotation[n].r_type & MCHAIN) {
                if (rotation[n].r_type & BACKWRD) {

                        /* Backward Mainchain Rotation */

                        if (*rotf != -1 && comprot(n, *rotf) > 0)
                                return(FALSE);
                        last = rotb;
                        count = 0;
                        for (i = *rotb; i != -1; i = rotation[i].r_linkb) {
                                if (++count >= MAXMSTCK) {
                                        reply("Too many nested rotations.\n
                                        return(FALSE);
                                }
                                if (comprot(n, i) > 0) {
                                        rotation[n].r_linkb = i;
                                        *last = n;
                                        return(TRUE);
                                }
                                last = &rotation[i].r_linkb;
                        }
                        *last = n;
#if      GRAFENG == PS300
                        if (*rotb == n) {
                                struct  input_def       in;

                                /* This is the residue where forward and
                                 * backward residues meet */
                                in.i_type = S_BACKWARD;
                                in.i_molnum = rotation[n].r_model;
                                switch (rotation[n].r_natom) {

                                case 2 :
                                        strcpy(in.i_name,
                                                rotation[n].r_resseq[1]);
                                        break;

                                case 4 :
                                        strcpy(in.i_name,
                                                rotation[n].r_resseq[2]);
                                        break;
                                }
                                strcpy(in.i_altname, ch->ch_start);
                                process(&in, (char *) NULL);
                                makechains(in.i_molnum);
                        }
#endif   PS300
                        return(TRUE);
                } else {

                        /* Forward Mainchain Rotation */

                        if (*rotb != -1 && comprot(n, *rotb) < 0)
                                return(FALSE);
                        last = rotf;
                        count = 0;
                        for (i = *rotf; i != -1; i = rotation[i].r_linkf) {
                                if (++count >= MAXMSTCK) {
                                        reply("Too many nested rotations.\n");
                                        return(FALSE);
                                }
                                if (comprot(n, i) < 0) {
                                        rotation[n].r_linkf = i;
```

```
                                *last = n;
                                return(TRUE);
                        }
                        last = &rotation[i].r_linkf;
                }
                *last = n;
                return(TRUE);
        }
} else {
        if (rotation[n].r_type & BACKWRD) {

                /* Backward Sidechain Rotation */

                for (i = 0; i < MAXINROT; i++) {
                        if (rotation[i].r_type == INACTIVE || i == n)
                                continue;
                        if (rotation[i].r_type & (MCHAIN|BACKWRD))
                                continue;
                        if (comprot(n, i) == 2)
                                return(FALSE);
                }
                count = 0;
                for (i = *rotf; i != -1; i = rotation[i].r_linkf)
                        if (++count >= MAXMSTCK) {
                                reply("Too many nested rotations.\n
                                return(FALSE);
                        }
                count = 0;
                for (i = *rotb; i != -1; i = rotation[i].r_linkb)
                        if (++count >= MAXMSTCK) {
                                reply("Too many nested rotations.\n
                                return(FALSE);
                        }
                if (*rotb != -1 && (statb = comprot(n, *rotb)) == -1)
                        return(FALSE);
                if (*rotf != -1 && (statf = comprot(n, *rotf)) == 1)
                        return(FALSE);
                if (*rotb == -1) {
                        *rotb = n;
                } else {
                        if (statb > 0) {
                                rotation[n].r_linkb = *rotb;
                                *rotb = n;
                        } else {
                                for (i = *rotb; i != -1;
                                   i = rotation[i].r_linkb) {
                                        if (comprot(n, i) < 0) {
                                                rotation[n].r_linkb =
                                                   rotation[i].r_linkb;
                                                rotation[i].r_linkb = n;
                                                break;
                                        }
                                }
                        }
                }
                if (*rotf == -1) {
                        *rotf = n;
                } else {
                        if (statf < 0) {
                                rotation[n].r_linkf = *rotf;
                                *rotf = n;
                        } else {
                                for (i = *rotf; i != -1;
                                   i = rotation[i].r_linkf) {
                                        if (comprot(n, i) > 0) {
                                                rotation[n].r_linkf =
                                                   rotation[i].r_linkf;
                                                rotation[i].r_linkf = n;
                                                break;
                                        }
                                }
                        }
                }
#if     GRAFENG == PS300
                if (*rotb == n) {
                        struct  input_def       in;

                        /* This is the residue where forward and
                         * backward residues meet */
                        in.i_type = S_BACKWARD;
                        in.i_molnum = rotation[n].r_model;
                        switch (rotation[n].r_natom) {

                        case 2 :
                                strcpy(in.i_name,
                                        rotation[n].r_resseq[1]);
                                break;

                        case 4 :
                                strcpy(in.i_name,
                                        rotation[n].r_resseq[2]);
                                break;
                        }
                        strcpy(in.i_altname, ch->ch_start);
```

```
                        process(&in, (char *) NULL);
                        makechains(in.i_molnum);
                }
#endif  PS300
                return(TRUE);
        } else {

                /* Forward Sidechain Rotation */

                count = 0;
                for (i = 0; i < MAXINROT; i++) {
                        if (rotation[i].r_type == INACTIVE || i == n)
                                continue;
                        if (rotation[i].r_type & (MCHAIN|BACKWRD))
                                continue;
                        statb = comprot(n, i);
                        if (statb == 2 || statb == -2)
                                count++;
                }
                if (count >= MAXMSTCK) {
                        reply("Too many nested rotations.\n");
                        return(FALSE);
                }
                if (*rotb == -1)
                        return(TRUE);
                if (rotation[*rotb].r_type & MCHAIN)
                        return(TRUE);
                if (comprot(n, *rotb) != -2)
                        return(TRUE);
                return(FALSE);
        }
    }
    /* NOTREACHED */
}

static  char    *at1, *at2;
static  int     compres;
static  char    truename[AT_NAME_SIZE+1];

/*
 * comprot - compare locations of the rotation pivot atoms
 *      returns  0 if the same
 *                1 if first one in back (different residue)
 *                2 if first one in back (same residue)
 *               -1 if first one in front (different residue)
 *               -2 if first one in front (same residue)
 *                3 if not the same model
 */
comprot(one, two)
int     one, two;
{
        register        char    *res1, *res2;
        char    resseq[RES_SEQ_SIZE+1];
        char    restype[RES_TYPE_SIZE+1];
        int     fd;
        int     cmpr(), null();

        fd = mdb[rotation[one].r_model].m_fd;
        if (rotation[one].r_model != rotation[two].r_model)
                return(3);
        switch (rotation[one].r_natom) {

        case 2 :
                res1 = rotation[one].r_resseq[1];
                at1 = rotation[one].r_atom[1];
                break;

        case 4 :
                res1 = rotation[one].r_resseq[2];
                at1 = rotation[one].r_atom[2];
                break;
        }
        switch (rotation[two].r_natom) {

        case 2 :
                res2 = rotation[two].r_resseq[1];
                at2 = rotation[two].r_atom[1];
                break;

        case 4 :
                res2 = rotation[two].r_resseq[2];
                at2 = rotation[two].r_atom[2];
                break;
        }
        if (smatch(res1, res2) == 0) {
                if (amatch(at1, at2) == 0)
                        return(0);
                (void) mseekr(fd, res1, FBSEEK);
                (void) mreadr(fd, resseq, restype);
                compres = 0;
                (void) mtrav(fd, cmpr, null);
                if (compres == 1)
                        return(-2);
                else
```

```
                        return(2);
        } else {
                (void) mseekr(fd, res1, FBSEEK);
                while (mreadr(fd, reseeq, restype) > 0) {
                        if (smatch(reseeq, res2) == 0)
                                return(-1);
                }
                return(1);
        }
        /* NOTREACHED */
}

cmpr(mfd, index, ischief, islinkage, nson, firstime)
int     mfd;
int     index;
int     ischief, islinkage, nson, firstime;
{
        if (!firstime || compres != 0)
                return;
        (void) matom(mfd, index, truename);
        if (amatch(truename, at1) == 0)
                compres = 1;
        else if (amatch(truename, at2) == 0)
                compres = 2;
        /* ARGSUSED */
}

/*
 * findchain - find the chain in which this rotation is occurring
 */
struct  chain_def       *
findchain(n)
int     n;
{
        register        mol;
        register        char    *res;
        struct  chain_def       *ch, *prevch;

        mol = rotation[n].r_model;
        prevch = mdb[mol].m_chain;
        switch (rotation[n].r_natom) {

        case 2 :
                res = rotation[n].r_reseeq[1];
                break;

        case 4 :
                res = rotation[n].r_reseeq[2];
                break;
        }
        for (ch = prevch->ch_next; ch != NULL; ch = ch->ch_next)
                if (infront(mdb[mol].m_fd, res, ch->ch_start))
                        break;
                else
                        prevch = ch;
        return(prevch);
}

/*
 * infront - returns TRUE if res1 is in front of res2
 */
infront(fd, res1, res2)
int     fd;
char    *res1, *res2;
{
        switch (mrconn(fd, res1, res2)) {

        default :
                fprintf(stderr, "Impossible condition in infront()\n");
                abort();

        case 2 :
        case 3 :
        case 5 :
                return(FALSE);

        case 1 :
        case 4 :
                return(TRUE);
        }
}

/*
 * unrankrot - removes the rotation from the ranking list
 */
unrankrot(n)
int     n;
{
        register        i;
        struct  chain_def       *ch;
        int     *last;
#if     GRAFENG == PS300
        int     remakech;
#endif
```

```
        ch = findchain(n);

#if     GRAFENG == PS300
        if (ch->ch_rotb == n) {
                struct  input_def       in;

                in.i_type = S_FORWARD;
                in.i_molnum = rotation[n].r_model;
                switch (rotation[n].r_natom) {

                case 2 :
                        strcpy(in.i_name, rotation[n].r_resseq[1]);
                        break;

                case 4 :
                        strcpy(in.i_name, rotation[n].r_resseq[2]);
                        break;
                }
                strcpy(in.i_altname, ch->ch_start);
                process(&in, (char *) NULL);
                remakech = in.i_molnum;
        }
        else
                remakech = -1;
#endif  PS300

        last = &ch->ch_rotf;
        for (i = ch->ch_rotf; i != -1; i = rotation[i].r_linkf) {
                if (i == n) {
                        *last = rotation[i].r_linkf;
                        break;
                } else
                        last = &rotation[i].r_linkf;
        }

        last = &ch->ch_rotb;
        for (i = ch->ch_rotb; i != -1; i = rotation[i].r_linkb) {
                if (i == n) {
                        *last = rotation[i].r_linkb;
                        break;
                } else
                        last = &rotation[i].r_linkb;
        }

#if     GRAFENG == PS300
        if (ch->ch_rotb != -1) {
                struct  input_def       in;

                i = ch->ch_rotb;
                in.i_type = S_BACKWARD;
                in.i_molnum = rotation[i].r_model;
                switch (rotation[i].r_natom) {

                case 2 :
                        strcpy(in.i_name, rotation[i].r_resseq[1]);
                        break;

                case 4 :
                        strcpy(in.i_name, rotation[i].r_resseq[2]);
                        break;
                }
                strcpy(in.i_altname, ch->ch_start);
                process(&in, (char *) NULL);
        }
        if (remakech != -1)
                makechains(remakech);
#endif  PS300
}
```

touch.c

```
/* $Header: touch.c,v 3.17 86/09/22 15:35:52 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "editor.h"

#ifdef    vms
#if GRAFENG == PS300
#include "MIDAS$H:fakegsr.h"
#endif
#endif
#define   EFMT    "Error in database %d.  No such atom '%s' '%s'\n"

#ifndef   EDITONLY
static    int     firstouch;
#endif
static    int     initial, molnum;
static    char    *rseq, *rtype;
static    float   tfmin, tfmax;
static    struct  atom_def        *atoms;

#if       GRAFENG == PS300
char      *map_name();
#endif

/*
 * touch - compute the bounds and center of mass of the given model
 */
touch(mol)
int       mol;
{
          char    seq[RES_SEQ_SIZE+1], type[RES_TYPE_SIZE+1];
#ifndef   EDITONLY
          struct  surf_def        *surfdat;
          struct  input_def       in;
#endif
#if       GRAFENG == PS300
          char    *mapped_name;
          char    namebuf[128];
#endif
          int     tvisit(), null();

          /* Initialize variables */
          molnum = mol;
          initial = TRUE;
#ifndef   EDITONLY
          mdb[mol].m_com[0] = mdb[mol].m_com[1] = mdb[mol].m_com[2] = 0;
          mdb[mol].m_active = 0;
          firstouch = (mdb[mol].m_chain == NULL);
#if       GRAFENG == PS300
          if (firstouch)
                    POptStru();
#endif    PS300
#endif    EDITONLY

          /* Start at the beginning and go to the end */
          (void) mseekr(mdb[mol].m_fd, all, FLSEEK);
          while (mreadr(mdb[mol].m_fd, seq, type) > 0) {
                    if (tmatch(type, "BOND") == 0)
                              continue;

#if       GRAFENG == PS300
                    mapped_name = map_name(seq);
                    if (firstouch) {
                              (void) sprintf(namebuf, "m%dr%s", mol, mapped_name);
                              nullnode(namebuf);
                    }
#endif
                    /* Compute the center of mass */
                    (void) mreada(mdb[mol].m_fd);
                    atoms = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                    rseq = seq;
                    rtype = type;
                    (void) mtrav(mdb[mol].m_fd, tvisit, null);

#ifndef   EDITONLY
                    /* Redraw the molecule */
                    if (mdb[mol].s_fd >= 0 &&
                              mseekr(mdb[mol].s_fd, seq, FBSEEK) == 0) {
                                        (void) mreada(mdb[mol].s_fd);
                                        surfdat = (struct surf_def *)
                                                  mdatptr(mdb[mol].s_fd);
                    } else
                              surfdat = NULL;
#if       GRAFENG == PS300
                    if (firstouch) {
                              /*
                               * We put in these "empty" declarations to fool
                               * the PS300 optimizer into believing that these
```

```
                        * nodes are data nodes and therefore do not
                        * need to be surrounded by state saves and restores.
                        * This is a very important optimization because
                        * it saves as many state save/restore pairs as
                        * there are residues.
                        */
                        (void) sprintf(namebuf, "m%dr%sprev", mol, mapped_na
                        nullnode(namebuf);
                        (void) sprintf(namebuf, "m%dr%snext", mol, mapped_na
                        nullnode(namebuf);
                }
                drawresidue(mol, seq, type, atoms);
#endif  PS300
                drawbond(mol, seq, type, atoms);
                drawlabel(mol, seq, type, atoms);
                drawsurf(mol, seq, type, atoms, surfdat);
#if     GRAFENG == PS300
                if (firstouch) {
                        (void) sprintf(namebuf, "m%dr%sprev", mol, mapped_na
                        PNameNil(namebuf);
                        (void) sprintf(namebuf, "m%dr%snext", mol, mapped_na
                        PNameNil(namebuf);
                }
#endif
#endif  EDITONLY
        } /* end while(mreadr()) */

#ifndef EDITONLY
        if (firstouch) {
                (void) sprintf(in.i_name, "pop%d", mdb[mol].m_nchain);
                in.i_type = S_APPEND;
                in.i_molnum = mol;
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = FALSE;
                in.i_firstmode = FM_NONE;
                process(&in, (char *) NULL);
                getbond(mol);
#if     GRAFENG == PS300
                PEndOpt();
#endif
        }

#ifdef  DEBUG
        if (debug & LD_PG) {
                fprintf(stderr, "Range = (%f,%f,%f)-(%f,%f,%f)\n",
                        mdb[molnum].m_min[0],mdb[molnum].m_min[1],
                        mdb[molnum].m_min[2],mdb[molnum].m_max[0],
                        mdb[molnum].m_max[1],mdb[molnum].m_max[2]);
        }
#endif
#endif  EDITONLY

        /* Update the global temperature factor range */
        if (nmol == 1) {
                mingtf = tfmin;
                maxgtf = tfmax;
        } else {
                if (mingtf > tfmin)
                        mingtf = tfmin;
                if (maxgtf < tfmax)
                        maxgtf = tfmax;
        }
        if (mingtf == maxgtf) {
                mingtf -= 1;
                maxgtf += 1;
        }
#ifndef EDITONLY
#if     GRAFENG == PS300
        makechains(mol);
#endif
#endif

        return;
}

/*
 * tvisit - record the bounds and increments the rotational moment of
 * of the given model
 */
tvisit(db, index, ischief, islink, nsons, firstime)
int     db;
int     index, ischief, islink, nsons, firstime;
{
#ifndef EDITONLY
        struct  chain_def       *ch, **chptr;
        struct  input_def       in;

        /* Produce the chain information */
        if (atoms[index].status & STARTBIT && firstime && firstouch) {
                chptr = &mdb[molnum].m_chain;
                for (ch = mdb[molnum].m_chain; ch != NULL; ch = ch->ch_next)
                        chptr = &ch->ch_next;

                if (mdb[molnum].m_nchain > 0) {
```

```
                        (void) sprintf(in.i_name, "pop%d",
                                mdb[molnum].m_nchain);
                        in.i_type = S_APPEND;
                        in.i_molnum = molnum;
                        in.i_size = 0;
                        in.i_ndata = 0;
                        in.i_newchain = FALSE;
                        in.i_firstmode = FM_NONE;
                        process(&in, (char *) NULL);
                }

                mdb[molnum].m_nchain++;
                (void) sprintf(in.i_name, "psh%d", mdb[molnum].m_nchain);
                in.i_type = S_APPEND;
                in.i_molnum = molnum;
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = TRUE;
                in.i_firstmode = FM_NONE;
                process(&in, (char *) NULL);

                ch = challoc(1, sizeof (struct chain_def));
                ch->ch_rotf = -1;
                ch->ch_rotb = -1;
                ch->ch_start[RES_SEQ_SIZE] = '\0';
                strncpy(ch->ch_start, rseq, RES_SEQ_SIZE);
                ch->ch_next = NULL;
                *chptr = ch;
        }

        if (ischief && firstime) {
                strcpy(in.i_name, rseq);
                in.i_type = S_APPEND;
                in.i_molnum = molnum;
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = FALSE;
                in.i_firstmode = FM_NONE;
                process(&in, (char *) NULL);
        }
#endif  EDITONLY

        /* Skip atom if it does not exist or has been visited before */
        if (!(atoms[index].status & EXISTBIT) || !firstime)
                return;

        if (initial) {
                tfmax = tfmin = atoms[index].tempfac;
#ifdef  EDITONLY
                initial = FALSE;
#endif
        } else {
                if (tfmin > atoms[index].tempfac)
                        tfmin = atoms[index].tempfac;
                else if (tfmax < atoms[index].tempfac)
                        tfmax = atoms[index].tempfac;
        }

#ifndef EDITONLY
        if (!(atoms[index].status & ONBITS))
                return;

        /* Check the bounds */
        if (initial) {
                initial = FALSE;
                mdb[molnum].m_max[0] = mdb[molnum].m_min[0] = atoms[index].x;
                mdb[molnum].m_max[1] = mdb[molnum].m_min[1] = atoms[index].y;
                mdb[molnum].m_max[2] = mdb[molnum].m_min[2] = atoms[index].z;
        } else {
                if (mdb[molnum].m_max[0] < atoms[index].x)
                        mdb[molnum].m_max[0] = atoms[index].x;
                else if (mdb[molnum].m_min[0] > atoms[index].x)
                        mdb[molnum].m_min[0] = atoms[index].x;
                if (mdb[molnum].m_max[1] < atoms[index].y)
                        mdb[molnum].m_max[1] = atoms[index].y;
                else if (mdb[molnum].m_min[1] > atoms[index].y)
                        mdb[molnum].m_min[1] = atoms[index].y;
                if (mdb[molnum].m_max[2] < atoms[index].z)
                        mdb[molnum].m_max[2] = atoms[index].z;
                else if (mdb[molnum].m_min[2] > atoms[index].z)
                        mdb[molnum].m_min[2] = atoms[index].z;
        }

        /* Accumulate the center of mass information */
        mdb[molnum].m_com[0] += atoms[index].x;
        mdb[molnum].m_com[1] += atoms[index].y;
        mdb[molnum].m_com[2] += atoms[index].z;
        mdb[molnum].m_active++;
#endif
        /* ARGSUSED */
}

#ifndef EDITONLY
/*
```

touch.c

```
 * getbond - Get the coordinates of the disulfide/hydrogen bonded atoms
 */
getbond(mol)
int     mol;
{
        int     n;
        char    seq[RES_SEQ_SIZE+1], type[RES_TYPE_SIZE+1];
        struct  bond_def        *bondat;
        struct  input_def       in;
        register        i, j;

        if (mseekr(mdb[mol].m_fd, "BOND", FLSEEK) == -1)
                return;

        nbond[mol] = mreadr(mdb[mol].m_fd, seq, type);
        if (nbond[mol] <= 0)
                return;

        bonds[mol] = (struct bc_def *) alloc(nbond[mol],
                sizeof (struct bc_def));
        (void) mreada(mdb[mol].m_fd);
        bondat = (struct bond_def *) mdatptr(mdb[mol].m_fd);
        for (i = 0; i < nbond[mol]; i++)
                for (j = 0; j < 2; j++) {
                        strncpy(bonds[mol][i].b_res[j], bondat[i].b_resseq[j],
                                RES_SEQ_SIZE);
                        strncpy(bonds[mol][i].b_atom[j], bondat[i].b_atname[j],
                                AT_NAME_SIZE);
                }

        for (i = 0; i < nbond[mol]; i++) {
                for (j = 0; j < 2; j++) {
                        (void) mseekr(mdb[mol].m_fd, bonds[mol][i].b_res[j],
                                FBSEEK);
                        (void) mreada(mdb[mol].m_fd);
                        atoms = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                        n = mseeka(mdb[mol].m_fd, bonds[mol][i].b_atom[j]);
                        if (n == -1) {
                                fprintf(stderr, EFMT, mol,
                                        bonds[mol][i].b_res[j],
                                        bonds[mol][i].b_atom[j]);
                                bonds[mol][i].b_disp[j] = 0;
                                continue;
                        }
                        bonds[mol][i].b_x[j] = atoms[n].x;
                        bonds[mol][i].b_y[j] = atoms[n].y;
                        bonds[mol][i].b_z[j] = atoms[n].z;
                        bonds[mol][i].b_disp[j] = (atoms[n].status & SHOWBIT);
                        bonds[mol][i].b_color = atoms[n].color;
                }
        }

        strcpy(in.i_name, "bond");
        in.i_type = S_APPEND;
        in.i_molnum = mol;
        in.i_size = 0;
        in.i_ndata = 0;
        in.i_newchain = TRUE;
        in.i_firstmode = FM_NONE;
        process(&in, (char *) NULL);
}
#endif  EDITONLY
```

chgstruct.c

```
/* $Header: chgstruct.c,v 3.16 86/07/24 12:45:17 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          5 Jul 1983
 * Release 2.1                          23 Aug 1983
 */

#include <ctype.h>
#include "editor.h"

#ifndef  vms
#include <exit.h>
#else
#include "midas$h:exit.h"
#endif
#define  IFILE              TMPDIR(Madd)
#define  OFILE              TMPDIR(Mnew)

#define  TMPTPL             "MIDAS"
#define  ANYBIT             (DISTBIT | ANGLEBIT | ROTBIT | BONDBIT)

#define  C_DEFAULT          0
#define  C_TYPE             1
#define  C_ANGLES           2

static   struct   {
         int      _molnum;
         char     _resseq[RES_SEQ_SIZE + 1];
         char     _restype[RES_TYPE_SIZE + 1];
         char     _atname[AT_NAME_SIZE + 1];
         struct   atom_def          _data;
}        addatoms[3];                              /* Atom table */

static   float    bond_length, bond_angle, dihedral_angle;
static   char     group[10], newgrp[10], newseq[10];
static   char     conformation[20];
static   int      conf_type;
static   int      nchosen;
static   int      setstart;
FILE     *popen();

char     *quote_str();

/*
 * Functions which handle the addgrp command
 */

/*
 * preaddgrp - sets up the necessary information for adding a new group
 * and cleans up the atom table in preparation for editing.
 */
preaddgrp(words)
struct   wordrange_def     *words;
{
         register          i;
         register          struct    wordrange_def     *w;

         i = 0;
         for (w = words; w != NULL; w = w->w_next)
                  i++;
         if (i < 3 || i > 5)
                  return(-1);
         w = words;
         strcpy(group, w->w_first);
         w = w->w_next;
         if (sscanf(w->w_first, "%f", &bond_length) != 1)
                  return(-1);
         w = w->w_next;
         if (sscanf(w->w_first, "%f", &bond_angle) != 1)
                  return(-1);
         w = w->w_next;
         if (w != NULL) {
                  if (isdigit(w->w_first[0])) {
                           if (sscanf(w->w_first, "%f", &dihedral_angle) != 1)
                                    return(-1);
                           w = w->w_next;
                  } else
                           dihedral_angle = 0;
                  if (w != NULL)
                           strcpy(newgrp, w->w_first);
                  else
                           newgrp[0] = '\0';
         } else {
                  dihedral_angle = 0;
                  newgrp[0] = '\0';
         }
         nchosen = 0;
         return(0);
}

/*
 * addgrp - checks the atom table for the appropriate number of atoms
```

chgstruct.c

```
 * present, prints the selected residue into a temporary file, calls
 * an external program to perform magic (putting the new group on), and
 * reconstructs the selected residue from the output of the program.
 */
addgrp()
{
        char    buf[512];
        FILE    *fd;
        int     val;
        int     atnum;
        register        char    *cp;
        char    qbuf[9][20];

        if (nchosen != 3)
                return(-1);
        fd = fopen(IFILE, "w");
        if (fd == NULL) {
                perror(IFILE);
                return(-1);
        }
        atnum = dumpres(addatoms[0]._molnum, addatoms[0]._resseq, fd, TRUE);
        if (smatch(addatoms[1]._resseq, addatoms[0]._resseq))
                printpdb(fd, atnum++, addatoms[1]._atname,
                        addatoms[1]._restype, addatoms[1]._resseq,
                        &addatoms[1]._data);
        if (smatch(addatoms[2]._resseq, addatoms[0]._resseq))
                printpdb(fd, atnum++, addatoms[2]._atname,
                        addatoms[2]._restype, addatoms[1]._resseq,
                        &addatoms[2]._data);
        (void) fclose(fd);

        if (newgrp[0] == '\0')
                strcpy(newgrp, addatoms[0]._restype);
        for (cp = newgrp; *cp != '\0'; cp++)
                if (islower(*cp))
                        *cp = toupper(*cp);

        (void) sprintf(buf, "%s -i %s -o %s -r1 %s -r2 %s -r3 %s -1 %s -2 %s -3 %s -
                P_ADDGRP, IFILE, OFILE,
                quote_str(addatoms[0]._resseq, qbuf[0]),
                quote_str(addatoms[1]._resseq, qbuf[1]),
                quote_str(addatoms[2]._resseq, qbuf[2]),
                quote_str(addatoms[0]._atname, qbuf[3]),
                quote_str(addatoms[1]._atname, qbuf[4]),
                quote_str(addatoms[2]._atname, qbuf[5]),
                quote_str(group, qbuf[6]), quote_str(TMPTPL, qbuf[7]),
                quote_str(newgrp, qbuf[8]), bond_length, bond_angle,
                dihedral_angle);

        fd = popen(buf, "r");
        if (fd == NULL) {
                reply("Cannot fork add.\n");
                (void) unlink(IFILE);
                return(-1);
        }
        val = 0;
        while (fgets(buf, sizeof buf, fd) != NULL) {
                reply(buf);
                val++;
        }
        if (pclose(fd) != E_NORMAL) {
                val++;
                reply("Add group terminated abnormally.\n");
        }
        (void) unlink(IFILE);
        if (val > 0) {
                (void) unlink(OFILE);
                return(0);
        }

        fd = fopen(OFILE, "r");
        if (fd == NULL) {
                perror(OFILE);
                return(-1);
        }
        val = buildres(addatoms[0]._molnum, addatoms[0]._resseq, TMPTPL,
                fd, TRUE);
        (void) fclose(fd);
        (void) unlink(OFILE);
        return(val);
}

/*
 * Functions which handle the swapaa and swapna commands
 */

/*
 * preswapres - sets up the new residue type and cleans up the atom table.
 */
preswapres(words)
struct  wordrange_def   *words;
{
        register        char    *cp;
```

```
        if (words == NULL || words->w_next != NULL)
                return(-1);
        strcpy(newgrp, words->w_first);
        for (cp = newgrp; *cp != '\0'; cp++)
                if (islower(*cp))
                        *cp = toupper(*cp);
        nchosen = 0;
        return(0);
}

/*
 * swapres - checks to find that only one residue is selected, prints
 * the selected residue, calls an external program to perform the replacement,
 * and reconstructs the selected residue.
 */
swapres()
{
        FILE    *fd;
        int     val;
        char    buf[512];
        char    qbuf[2][20];

        if (nchosen != 1)
                return(-1);
        fd = fopen(IFILE, "w");
        if (fd == NULL) {
                perror(IFILE);
                return(-1);
        }
        (void) dumpres(addatoms[0]._molnum, addatoms[0]._resseq, fd, TRUE);
        (void) fclose(fd);

        (void) sprintf(buf, "%s -i %s -o %s -r %s -s %s",
                (comnum == SWAPAA) ? P_SWAPAA : P_SWAPNUC,
                IFILE, OFILE, quote_str(addatoms[0]._resseq, qbuf[0]),
                quote_str(newgrp, qbuf[1]));

        fd = popen(buf, "r");
        if (fd == NULL) {
                reply("Cannot fork swap.\n");
                (void) unlink(IFILE);
                return(-1);
        }
        val = 0;
        while (fgets(buf, sizeof buf, fd) != NULL) {
                reply(buf);
                val++;
        }
        if (pclose(fd) != E_NORMAL) {
                val++;
                reply("Swap residue terminated abnormally.\n");
        }
        (void) unlink(IFILE);
        if (val > 0) {
                (void) unlink(OFILE);
                return(0);
        }

        fd = fopen(OFILE, "r");
        if (fd == NULL) {
                perror(OFILE);
                return(-1);
        }
        val = buildres(addatoms[0]._molnum, addatoms[0]._resseq, newgrp,
                fd, TRUE);
        (void) fclose(fd);
        (void) unlink(OFILE);
        return(val);
}

/*
 * Functions to handle the addaa (and when Paul gets addnuc ready, addnuc)
 * command(s).
 */

/*
 * preaddres - sets up the residue to be added and cleans up the table.
 */
preaddres(words)
struct  wordrange_def   *words;
{
        register        char    *cp;

        if (words == NULL)
                return(-1);
        strcpy(newgrp, words->w_first);
        for (cp = newgrp; *cp != '\0'; cp++)
                if (islower(*cp))
                        *cp = toupper(*cp);

        words = words->w_next;
        if (words == NULL)
                return(-1);
        strcpy(newseq, words->w_first);
```

chgstruct.c

```
	for (cp = newseq; *cp != '\0'; cp++)
		if (islower(*cp))
			*cp = toupper(*cp);

	nchosen = 0;

	words = words->w_next;
	if (words == NULL)
		conf_type = C_DEFAULT;
	else if (isalpha(words->w_first[0])) {
		conf_type = C_TYPE;
		strcpy(conformation, words->w_first);
		for (cp = conformation; *cp != '\0'; cp++)
			if (islower(*cp))
				*cp = toupper(*cp);
	} else if (isdigit(words->w_first[0])) {
		conf_type = C_ANGLES;
		strcpy(conformation, words->w_first);
		words = words->w_next;
		if (words == NULL || !isdigit(words->w_first[0]))
			return(-1);
		strcat(conformation, " ");
		strcat(conformation, words->w_first);
	} else
		return(-1);
	return(0);
}

/*
 * addres - makes sure only one residue is selected and gets an external
 * program to create coordinates of the new residue.  Creates a node for
 * this new residue in the storage module.  Finally, reconstruct the
 * residue.
 */
addres()
{
	FILE	*fd;
	int	val;
	char	buf[512];
	char	resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
	char	qbuf[3][20];
#ifndef	EDITONLY
	struct	input_def		in;
#endif

	if (nchosen != 1)
		return(-1);

if (mseekr(mdb[addatoms[0]._molnum].m_fd, newseq, FBSEEK) != -1) {
		(void) sprintf(buf, "Residue %s already exists.\n", newseq);
		reply(buf);
		return(-1);
}
fd = fopen(IFILE, "w");
if (fd == NULL) {
		perror(IFILE);
		return(-1);
}
(void) dumpres(addatoms[0]._molnum, addatoms[0]._resseq, fd, FALSE);
(void) fclose(fd);

(void) sprintf(buf, "%s -i %s -o %s -r %s -s %s -t %s",
		(comnum == ADDAA) ? P_ADDAA : P_ADDNUC,
		IFILE, OFILE, quote_str(addatoms[0]._resseq, qbuf[0]),
		quote_str(newseq, qbuf[1]), quote_str(newgrp, qbuf[2]));

switch (conf_type) {

case C_DEFAULT :
		break;

case C_TYPE :
case C_ANGLES :
		strcat(buf, " -c ");
		strcat(buf, quote_str(conformation, qbuf[0]));
		break;
}

fd = popen(buf, "r");
if (fd == NULL) {
		reply("Cannot fork swap.\n");
		(void) unlink(IFILE);
		return(-1);
}
val = 0;
while (fgets(buf, sizeof buf, fd) != NULL) {
		reply(buf);
		val++;
}
if (pclose(fd) != E_NORMAL) {
		val++;
		reply("Add residue terminated abnormally.\n");
}
(void) unlink(IFILE);
if (val > 0) {
```

```
                (void) unlink(OFILE);
                return(0);
        }

        fd = fopen(OFILE, "r");
        if (fd == NULL) {
                perror(OFILE);
                return(-1);
        }
#ifndef EDITONLY
        strncpy(in.i_name, newseq, RES_SEQ_SIZE);
        strncpy(in.i_altname, addatoms[0]._resseq, RES_SEQ_SIZE);
        in.i_type = S_ADD;
        in.i_molnum = addatoms[0]._molnum;
        in.i_size = 0;
        in.i_ndata = 0;
        in.i_newchain = FALSE;
        in.i_firstmode = FM_NONE;
        in.i_color = -1;
        in.i_lcolor = -1;
        process(&in, (char *) NULL);
#endif EDITONLY
        val = buildres(addatoms[0]._molnum, newseq, newgrp, fd, TRUE);

        /* Rebuild the residue we added to since we might delete an OXT */
        (void) mseekr(mdb[addatoms[0]._molnum].m_fd,
                addatoms[0]._resseq, FBSEEK);
        (void) mreadr(mdb[addatoms[0]._molnum].m_fd, resseq, restype);
        (void) fseek(fd, 0L, 0);
        (void) buildres(addatoms[0]._molnum, resseq, restype, fd, FALSE);

        (void) fclose(fd);
        (void) unlink(OFILE);
        return(val);
}

/*
 * Functions to handle the delete command
 */

/*
 * predelgrp - setup the new group name after deletion
 */
predelgrp(words)
struct  wordrange_def       *words;
{
        register            char        *cp;

        if (words == NULL || words->w_next != NULL)
                return(-1);
        strcpy(newgrp, words->w_first);
        for (cp = newgrp; *cp != '\0'; cp++)
                if (islower(*cp))
                        *cp = toupper(*cp);
        nchosen = 0;
        return(0);
}

/*
 * delgrp - call an external program to create the new template and
 * reconstruct the residue.
 */
delgrp()
{
        FILE    *fd, *popen();
        char    buf[512];
        int     val;
        char    qbuf[4][20];

        if (nchosen != 1)
                return(-1);

        (void) sprintf(buf, "%s -r %s -n %s -a %s -f %s", P_DELETE,
                quote_str(addatoms[0]._restype, qbuf[0]),
                quote_str(newgrp, qbuf[1]),
                quote_str(addatoms[0]._atname, qbuf[2]),
                quote_str(TMPTPL, qbuf[3]));

        fd = popen(buf, "r");
        if (fd == NULL) {
                reply("Cannot fork delete.\n");
                return(-1);
        }
        val = 0;
        while (fgets(buf, sizeof buf, fd) != NULL) {
                reply(buf);
                val++;
        }
        if (pclose(fd) != E_NORMAL) {
                val++;
                reply("Delete group terminated abnormally.\n");
        }
        if (val > 0)
                return(0);
```

```
	fd = fopen(IFILE, "w");
	if (fd == NULL) {
		perror(IFILE);
		return(-1);
	}
	(void) dumpres(addatoms[0]._molnum, addatoms[0]._resseq, fd, TRUE);

	fd = freopen(IFILE, "r", fd);
	if (fd == NULL) {
		perror(IFILE);
		return(-1);
	}
	val = buildres(addatoms[0]._molnum, addatoms[0]._resseq, TMPTPL,
		fd, FALSE);
	(void) fclose(fd);
	(void) unlink(IFILE);
	return(val);
}

/*
 * Functions to handle the link command
 */

/*
 * prelink - clean up the atom table.
 */
prelink()
{
	nchosen = 0;
}

/*
 * redochain - reorganize the storage module structure of the model
 */
redochain()
{
	register	i;
	char	atname[AT_NAME_SIZE + 1];
	char	resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
	int	mol, chief, count;
	struct	atom_def	atdata;
#ifndef	EDITONLY
	struct	atom_def	*atomdata;
	struct	surf_def	*surfdata;
	struct	chain_def	*ch, *prevch;
	struct	input_def	in;
#endif

	if (nchosen != 1)
		return(-1);
	mol = addatoms[0]._molnum;

#ifndef	EDITONLY
	/* Locate the chain that the residue is in */
	count = 1;
	prevch = mdb[mol].m_chain;
	for (ch = prevch->ch_next; ch != NULL; ch = ch->ch_next)
		if (infront(mdb[mol].m_fd, addatoms[0]._resseq, ch->ch_start))
			break;
		else {
			prevch = ch;
			count++;
		}
#endif

	/* Make sure it's OK to do the operation */
	(void) mseekr(mdb[mol].m_fd, addatoms[0]._resseq, FBSEEK);
	(void) mreadr(mdb[mol].m_fd, resseq, restype);
	(void) mreada(mdb[mol].m_fd);
	chief = mchief(mdb[mol].m_fd);
	(void) mgeta(mdb[mol].m_fd, chief, (char *) &atdata);
	if (notcom) {
		if (atdata.status & STARTBIT) {
			reply("Residue already starts a chain.\n");
			return(-1);
		}
	}
	else {
		if (!(atdata.status & STARTBIT)) {
			reply("Residue does not start a chain.\n");
			return(-1);
		}
#ifndef	EDITONLY
		if (prevch == mdb[mol].m_chain) {
			reply("Cannot link first chain in model.\n");
			return(-1);
		}
		for (ch = mdb[mol].m_chain; ch != NULL; ch = ch->ch_next)
			if (ch->ch_next == prevch)
				break;
		if (ch->ch_rotf != -1 && prevch->ch_rotb != -1) {
			reply("Link would result in incompatible rotations.\n");
			return(-1);
```

```
                }
#endif
        }
        (void) matom(mdb[mol].m_fd, chief, atname);
        makebit(addatoms[0]._molnum, addatoms[0]._resseq, atname);

#ifndef EDITONLY
        /* Remove all rotations so that they get ranked later */
        for (i = 0; i < MAXINROT; i++)
                if (rotation[i].r_type != INACTIVE)
                        unrankrot(i);

        if (!notcom) {
                /* Remove the pop and push preceding this chain */
                in.i_molnum = mol;
                (void) sprintf(in.i_name, "pop%d", count - 1);
                in.i_type = S_REMOVE;
                process(&in, (char *) NULL);
                (void) sprintf(in.i_name, "psh%d", count);
                in.i_type = S_REMOVE;
                process(&in, (char *) NULL);

                /* Rename everything beneath this chain to take up the slack */
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = FALSE;
                in.i_firstmode = FM_NONE;
                in.i_color = -1;
                in.i_lcolor = -1;
                for (i = count; i <= mdb[mol].m_nchain; i++) {
                        (void) sprintf(in.i_name, "pop%d", i - 1);
                        (void) sprintf(in.i_altname, "pop%d", i);
                        in.i_type = S_RENAME;
                        process(&in, (char *) NULL);
                }
                for (i = count + 1; i <= mdb[mol].m_nchain; i++) {
                        (void) sprintf(in.i_name, "psh%d", i - 1);
                        (void) sprintf(in.i_altname, "psh%d", i);
                        in.i_type = S_RENAME;
                        process(&in, (char *) NULL);
                }
                for (ch = mdb[mol].m_chain; ch != NULL; ch = ch->ch_next)
                        if (ch->ch_next == prevch)
                                break;
                ch->ch_next = prevch->ch_next;
                free((char *) prevch);
                mdb[mol].m_nchain--;
        }
        else {
                /* Rename every push and pop node beneath this chain
                 * to make room for the new chain */
                in.i_size = 0;
                in.i_ndata = 0;
                in.i_newchain = FALSE;
                in.i_firstmode = FM_NONE;
                in.i_color = -1;
                in.i_lcolor = -1;
                in.i_molnum = mol;
                for (i = mdb[mol].m_nchain; i > count; i--) {
                        (void) sprintf(in.i_name, "psh%d", i + 1);
                        (void) sprintf(in.i_altname, "psh%d", i);
                        in.i_type = S_RENAME;
                        process(&in, (char *) NULL);
                }
                for (i = mdb[mol].m_nchain; i >= count; i--) {
                        (void) sprintf(in.i_name, "pop%d", i + 1);
                        (void) sprintf(in.i_altname, "pop%d", i);
                        in.i_type = S_RENAME;
                        process(&in, (char *) NULL);
                }

                /* Insert the pop and push preceding this chain */
                strcpy(in.i_altname, addatoms[0]._resseq);
                (void) sprintf(in.i_name, "pop%d", count);
                in.i_type = S_INSERT;
                process(&in, (char *) NULL);
                (void) sprintf(in.i_name, "psh%d", count + 1);
                in.i_newchain = TRUE;
                in.i_type = S_INSERT;
                process(&in, (char *) NULL);
                ch = challoc(1, sizeof (struct chain_def));
                ch->ch_rotf = -1;
                ch->ch_rotb = -1;
                ch->ch_start[RES_SEQ_SIZE] = '\0';
                strncpy(ch->ch_start, addatoms[0]._resseq, RES_SEQ_SIZE);
                ch->ch_next = prevch->ch_next;
                prevch->ch_next = ch;
                mdb[mol].m_nchain++;
        }

        /* Rerank all rotations so that they get chained properly */
        for (i = 0; i < MAXINROT; i++)
                if (rotation[i].r_type != INACTIVE)
                        (void) rankrot(i);
```

```
        /* Redraw the residue */
        (void) mseekr(mdb[mol].m_fd, addatoms[0]._resseq, FBSEEK);
        (void) mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#if     GRAFENG == PS300
        drawresidue(mol, resseq, restype, atomdata);
#endif
        drawbond(mol, resseq, restype, atomdata);
        drawlabel(mol, resseq, restype, atomdata);
        if (mdb[mol].s_fd >= 0 && mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                (void) mreada(mdb[mol].s_fd);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        }
        else
                surfdata = NULL;
        drawsurf(mol, resseq, restype, atomdata, surfdata);

#if     GRAFENG == PS300
        makechains(mol);
#endif
#endif  EDITONLY
        return(0);
}

/*
 * remcalc - remove calculations involving the given residue (including
 * distance and angle calculations and internal rotations).
 */
remcalc(mol, rseq)
int     mol;
char    *rseq;
{
        int     found;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        struct  bond_def        bondata;
        register                cnt, i, j;

#ifndef EDITONLY
        for (cnt = 0; cnt < MAXINROT; cnt++) {
                if (rotation[cnt].r_type == INACTIVE)
                        continue;
                if (rotation[cnt].r_model != mol)
                        continue;
                for (i = 0; i < rotation[cnt].r_natom; i++) {
                        if (smatch(rotation[cnt].r_resseq[i], rseq))
                                continue;
                        rotation[cnt].r_rotangle = 0;
                        unrot(cnt);
                        break;
                }
        }
        for (cnt = 0; cnt < MAXANGLE; cnt++) {
                if (angles[cnt].a_type == INACTIVE)
                        continue;
                for (i = 0; i < angles[cnt].a_natom; i++) {
                        if (angles[cnt].a_model[i] != mol)
                                continue;
                        if (smatch(angles[cnt].a_resseq[i], rseq))
                                continue;
                        unang(cnt);
                        break;
                }
        }
        for (cnt = 0; cnt < MAXDIST; cnt++) {
                if (distance[cnt].d_type == INACTIVE)
                        continue;
                for (i = 0; i < 2; i++) {
                        if (distance[cnt].d_model[i] != mol)
                                continue;
                        if (smatch(distance[cnt].d_resseq[i], rseq))
                                continue;
                        undist(cnt);
                        break;
                }
        }
#endif  EDITONLY
        cnt = 0;
        for (i = 0; i < nbond[mol]; i++) {
                found = FALSE;
                for (j = 0; j < 2; j++) {
                        if (smatch(bonds[mol][i].b_res[j], rseq))
                                continue;
                        found = TRUE;
                        break;
                }
                if (!found) {
                        if (i != cnt)
                                bonds[mol][cnt] = bonds[mol][i];
                        cnt++;
                }
        }
        if (cnt != nbond[mol]) {
```

```
                if (cnt == 0)
                        free((char *) bonds[mol]);
                (void) mseekr(mdb[mol].m_fd, "BOND", FLSEEK);
                (void) mreadr(mdb[mol].m_fd, resseq, restype);
                (void) mseekr(mdb[mol].m_fd, resseq, FBSEEK);
                (void) mwrtr(mdb[mol].m_fd, resseq, restype, cnt,
                        sizeof (struct bond_def));
                for (i = 0; i < cnt; i++) {
                        strncpy(bondata.b_resseq[0], bonds[mol][i].b_res[0],
                                RES_SEQ_SIZE);
                        strncpy(bondata.b_resseq[1], bonds[mol][i].b_res[1],
                                RES_SEQ_SIZE);
                        strncpy(bondata.b_atname[0], bonds[mol][i].b_atom[0],
                                AT_NAME_SIZE);
                        strncpy(bondata.b_atname[1], bonds[mol][i].b_atom[1],
                                AT_NAME_SIZE);
                        (void) mputa(mdb[mol].m_fd, i, (char *) &bondata);
                }
                (void) mwrta(mdb[mol].m_fd);
                nbond[mol] = cnt;
        }
}

/*
 * Functions to print and read specified residues.
 */

/*
 * dumpres - dump the given residue onto the given file stream.
 * If flag is TRUE, then remove any visible, printed atom from
 * the center of mass calculation.
 */
dumpres(mol, rseq, fd, flag)
char    *rseq;
FILE    *fd;
int     flag;
{
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        int     natom, chief;
        char    atname[AT_NAME_SIZE + 1];
        struct  atom_def        at;
        register        i, atnum;

        (void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
        natom = mreadr(mdb[mol].m_fd, resseq, restype);
        (void) mreada(mdb[mol].m_fd);
        atnum = 1;
        chief = mchief(mdb[mol].m_fd);
        for (i = 0; i < natom; i++) {
                (void) mgeta(mdb[mol].m_fd, i, (char *) &at);
                if (!(at.status & EXISTBIT))
                        continue;
                (void) matom(mdb[mol].m_fd, i, atname);
                printpdb(fd, atnum++, atname, restype, resseq, &at);
                if (i == chief) {
                        if (at.status & STARTBIT)
                                setstart = TRUE;
                        else
                                setstart = FALSE;
                }
                if (flag && at.status & SHOWBIT) {
                        mdb[mol].m_active--;
                        mdb[mol].m_com[0] -= at.x;
                        mdb[mol].m_com[1] -= at.y;
                        mdb[mol].m_com[2] -= at.z;
                }
        }
        if (flag)
                remcalc(mol, rseq);
        return(atnum);
}

/*
 * buildres - construct the given residue from the file.  If extra atoms
 * are there, report them (unless report is FALSE).
 */
buildres(mol, rseq, rtype, fd, report)
int     mol;
char    *rseq, *rtype;
FILE    *fd;
int     report;
{
        register        i;
        char    resseq[RES_SEQ_SIZE + 1], restype[RES_TYPE_SIZE + 1];
        char    realtype[RES_TYPE_SIZE + 1];
        int     natom, atnum, chief;
        char    atname[AT_NAME_SIZE + 1];
        char    buf1[80], buf2[80];
        struct  atom_def        at;
#ifndef EDITONLY
        struct  atom_def        *atomdata;
        struct  surf_def        *surfdata;
#endif
```

```
(void) mseekr(mdb[mol].m_fd, rseq, FBSEEK);
natom = mwrtr(mdb[mol].m_fd, rseq, rtype, -1, sizeof (struct atom_def));
if (natom == -1) {
        (void) sprintf(buf1, "%s: unable to access residue.\n", rtype);
        reply(buf1);
        return(-1);
}

at.status = 0;
for (i = 0; i < natom; i++)
        (void) mputa(mdb[mol].m_fd, i, (char *) &at);

buf1[0] = '\0';
while (readpdb(fd, &atnum, atname, restype, resseq, &at) == 0) {
        if (smatch(resseq, rseq))
                continue;
        i = mseeka(mdb[mol].m_fd, atname);
        if (i == -1) {
                if (report) {
                        if (buf1[0] != '\0')
                                strcat(buf1, ", ");
                        strcat(buf1, atname);
                        if (strlen(buf1) > 40) {
                                strcpy(buf2, "Unknown atoms: ");
                                strcat(buf2, buf1);
                                strcat(buf2, "\n");
                                reply(buf2);
                                buf1[0] = '\0';
                        }
                } else
                        continue;
        }
        mdb[mol].m_active++;
        mdb[mol].m_com[0] += at.x;
        mdb[mol].m_com[1] += at.y;
        mdb[mol].m_com[2] += at.z;
        (void) mputa(mdb[mol].m_fd, i, (char *) &at);
        strcpy(realtype, restype);
}

if (setstart) {
        chief = mchief(mdb[mol].m_fd);
        (void) mgeta(mdb[mol].m_fd, chief, (char *) &at);
        at.status |= STARTBIT;
        (void) mputa(mdb[mol].m_fd, chief, (char *) &at);
}
        (void) mwrta(mdb[mol].m_fd);

#ifndef EDITONLY
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
#if     GRAFENG == PS300
        drawresidue(mol, rseq, realtype, atomdata);
#endif
        drawbond(mol, rseq, realtype, atomdata);
        drawlabel(mol, rseq, realtype, atomdata);
        if (mdb[mol].s_fd >= 0 && mseekr(mdb[mol].s_fd, rseq, FBSEEK) == 0) {
                (void) mreada(mdb[mol].s_fd);
                surfdata = (struct surf_def *) mdatptr(mdb[mol].s_fd);
        } else
                surfdata = NULL;
        drawsurf(mol, rseq, realtype, atomdata, surfdata);
#endif  EDITONLY

        if (report && buf1[0] != '\0') {
                strcpy(buf2, "Unknown atoms: ");
                strcat(buf2, buf1);
                strcat(buf2, "\n");
                reply(buf2);
        }

        /* Update the global temperature factor range.
         * The default temperature factor from readpdb is 0;
         * so we check against 0 */
        if (mingtf > 0)
                mingtf = 0;
        if (maxgtf < 0)
                maxgtf = 0;
        return(0);
}

/*
 * Functions called by editing routines.  These functions are in this
 * module because they fill in data structures which are only visible
 * in this module.
 */

/*
 * insertadd - insert selected atom into the atom table.
 */
insertadd(mol, rseq, rtype, data, aname)
int     mol;
char    *rseq, *rtype;
struct  atom_def        *data;
```

chgstruct.c

```
char	*aname;
{
	register		i;

	i = nchosen++;
	if (nchosen > 3)
		return;
	addatoms[i]._molnum = mol;
	strcpy(addatoms[i]._resseq, rseq);
	strcpy(addatoms[i]._restype, rtype);
	strcpy(addatoms[i]._atname, aname);
	addatoms[i]._data = *data;
}

/*
 * insertgetres - put the selected atom into the table.  If there is
 * already an atom, then the two must be in the same residue.
 */
insertgetres(mol, rseq, rtype, data)
int	mol;
char	*rseq, *rtype;
struct	atom_def		*data;
{
	if (nchosen == 0) {
		addatoms[0]._molnum = mol;
		strcpy(addatoms[0]._resseq, rseq);
		strcpy(addatoms[0]._restype, rtype);
		addatoms[0]._data = *data;
		nchosen = 1;
	} else if (nchosen == 1) {
		if (smatch(addatoms[0]._resseq, rseq))
			nchosen++;
	}
}

char *
quote_str(str, buf)
register char	*str;
char		*buf;
{
	register char	*bp;

	bp = buf;
	while (*str) {
		if (!isalnum(*str))
			*bp++ = '\';
		*bp++ = *str++;
	}
	*bp = '\0';
	return buf;
}
```

pdb.c

```
/* $Header: /usr/src/local/midas/src/editor/RCS/pdb.c,v 3.2 83/12/21 11:54:06 conrad
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          6 Jul 1983
 */

#include <ctype.h>
#include "editor.h"

#define  MAXLINE 80
#define  ATNUM  6
#define  ATNAME 12
#define  RESNAME 17
#define  RESEQ  22
#define  COORD  30
#define  TEMPFAC 60

/*
 * readpdb - read in the next atom record in the given file
 */
readpdb(infd, atnmb, atname, restype, reseq, atdatptr)
FILE    *infd;
int     *atnmb;
char    atname[AT_NAME_SIZE+1];
char    restype[RES_TYPE_SIZE+1];
char    reseq[RES_SEQ_SIZE+1];
struct  atom_def            *atdatptr;
{
        char    line[MAXLINE];
        register            char    *c;

restart:
        if(fgets(line, MAXLINE, infd) == NULL)
                return(-1);

        for (c = &line[0]; *c != '\0'; c++){
                if(islower(*c))
                        *c = toupper(*c);
        }

        if(!strncmp(line, "ATOM  ", 6) && !strncmp(line, "HETATM", 6))
                goto restart;

        (void) sscanf(&line[ATNUM], "%5d", atnmb);
        (void) sscanf(&line[ATNAME], "%4s", atname);
        (void) sscanf(&line[RESNAME], "%3s", restype);
        (void) sscanf(&line[RESEQ], "%4s", reseq);
        (void) sscanf(&line[COORD], "%f %f %f", &atdatptr->x, &atdatptr->y,
                &atdatptr->z);
        (void) sscanf(&line[TEMPFAC], "%f", &atdatptr->tempfac);

        atdatptr->status = EXISTBIT | SHOWBIT;
        atdatptr->color = atdatptr->lcolor = atdatptr->scolor = 0;
        atdatptr->tempfac = 0;

        return(0);

}

/*
 * printpdb - print atom information into given file
 */
printpdb(outfd, atnmb, atname, restype, reseq, atdatptr)
FILE    *outfd;
char    atname[AT_NAME_SIZE+1];
char    restype[RES_TYPE_SIZE+1];
char    reseq[RES_SEQ_SIZE+1];
int     atnmb;
struct  atom_def            *atdatptr;
{

#define OUT "ATOM %6d %-4s %3s  %4s    %8.3f%8.3f%8.3f %11.2f\n"

        fprintf(outfd, OUT, atnmb, atname, restype, reseq, atdatptr->x,
                atdatptr->y, atdatptr->z, atdatptr->tempfac);
}
```

store.c

```
/* $Header: store.c,v 3.38 86/09/22 15:35:46 conrad Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          29 Jul 1983
 * Release 2.2                          18 Aug 1983
 * Release 2.3                          23 Aug 1983
 * Release 2.4                          26 Sep 1983
 */

/*
 * MIDAS-STORE is a program which keeps track of PS2 objects for
 * the MIDAS editor.  This program is only necessary because one
 * process can only have 64K bytes of data (on the 11/70).
 */
#include <signal.h>

#ifndef  vms
#include "grafeng.h"
#else
#include "MIDAS$H:machdef.h"
#endif

#if      GRAFENG == IRIS
#include <device.h>
#undef   REDRAW
#endif

#include "editor.h"

#if      GRAFENG == PS2
#include <ps2.h>
#endif   PS2

#if      GRAFENG == IRIS

#include <irisdefs.h>
#undef   REDRAW/* This is defined in device.h, but not used by us */

#ifndef  ONEPROC
#define  ONEPROC/* IRIS uses terminal memory to store objects */
#endif   ONEPROC
#endif   IRIS

#if      GRAFENG == PS300
#ifndef  vms
#include <ps300/gsr.h>
#else    vms
#include "MIDAS$H:fakegsr.h"
#endif   vms
#endif   PS300

#ifndef  ONEPROC
char     id[]    =       "MIDAS STORAGE MODULE";
#endif   ONEPROC

/* debugging control */
/* #define        DEBUG           /* enable assertion testing of OB objects */
/* #define        ECHOCOM         /* echo commands as they are received */
/* #define        BIGDEBUG        /* enable assertion testing of memory arena */
/* #define        HUGEDEBUG       /* create UDL on disk instead of in PS memory */
/* #define        DUMPTAB         /* dump the hash table on each command */

#define  HASHSIZE        59

#if      GRAFENG == MPS
#define  MINSIZE         (5*sizeof (ps_t))       /* setbase in every res */
#else    MPS
#define  MINSIZE         (2*sizeof (ps_t))
#endif   MPS

#define  HWSIZE          250
#define  SD_FORWARD      01
#define  SD_BACKWARD     02
#ifndef  SENTINEL
#define  SENTINEL        0
#endif   SENTINEL

#define  PSSIZE(size)    ((size / sizeof (ps_t)) - 2)

#if      GRAFENG == PS2
#define  PSMEMSZ         MAXPSMEM
#endif   PS2

#if      GRAFENG == MPS
extern   int extdmem;
#define  PSMEMSZ         (extdmem ? MAXPSMEM:NRMEMSZ)
#endif   MPS

#if      GRAFENG == PS300
char             *map_name();
```

store.c

```
extern int          ps300cm[];
#endif  PS300

#if     GRAFENG != IRIS
#define TOOMUCH(add, cur)       (PSSIZE(add) + cur >= PSMEMSZ)
#endif  IRIS

struct  hash_def            {
        struct  hash_def        *h_next;        /* Next item in hash chain
        struct  hash_def        *h_flink;       /* Next item in chain */
        struct  hash_def        *h_blink;       /* Last item in chain */
        char    h_name[RES_SEQ_SIZE+1];         /* Name of this entry */
        ps_t    h_x, h_y, h_z;                  /* Coordinate of first atom
        ps_t    h_lx, h_ly, h_lz;               /* Coordinate of last atom
        char    h_brkchain;                     /* Residue breaks chain */
        char    h_firstmode;                    /* First mode of the res */
        char    h_color, h_lcolor;              /* first and last BOND colo
        char    h_newchain;                     /* Residue starts new chai
        char    h_direction;                    /* Which way does residu
        int     h_ndata;                        /* Number of data */
        int     h_level;                        /* Levels of back rotations
#if     GRAFENG == PS2 || GRAFENG == MPS
        char    *h_bdata;                       /* Pointer to bond data */
        int     h_bsize;                        /* Size of bond data */
        char    *h_sdata;                       /* Pointer to surface data *
        int     h_ssize;                        /* Size of surface data */
        char    *h_ldata;                       /* Pointer to label data */
        int     h_lsize;                        /* Size of label data */
#endif
#if     GRAFENG == PS300
        char    h_halfbond;                     /* In halfbond mode ? */
#endif
#ifdef  BIGDEBUG
        char    *h_dbdata, *h_dldata, *h_dsdata;
#endif  BIGDEBUG
};

struct  mol_def  {
        int     inuse;                          /* Model in use ? */
        struct  hash_def        *hashtab[HASHSIZE];
        struct  hash_def        *m_first, *m_last;
};

struct  mol_def  moltab[MAXMOD];
static  int      toomuch;
int     halfbond;

#if     GRAFENG == PS2
int     _hstkcnt;               /* Bug in -lg. Must supply this symbol since
                                 * pemem expects it to be defined but it is
                                 * defined only if bldcon() is loaded. */
#endif  PS2

struct  hash_def        *newbucket(), *findbucket();

#ifndef  ONEPROC
main()
{
        register        i;
        struct  input_def       input;
        int     stop();

        (void) signal(SIGINT, SIG_IGN);
        (void) signal(SIGTERM, stop);
        for (i = 0; i < MAXMOD; i++)
                moltab[i].inuse = FALSE;
#if     GRAFENG == PS2 || GRAFENG == MPS
        setupps();
        getps();
#endif  GRAFENG == PS2 || GRAFENG == MPS
#if     GRAFENG == PS2
        BUFRPS(150);
#endif  PS2
        toomuch = FALSE;
        input.i_type = FALSE;

        do {
                (void) write(1, (char *) &input, sizeof input);
                (void) read(0, (char *) &input, sizeof input);
                if (input.i_type == S_QUIT)
                        break;
                process(&input);
        } while (TRUE);
        stop();
        /* NOTREACHED */
}

stop()
{
        exit(0);
}
#endif  ONEPROC

#ifdef  ONEPROC
```

store.c

```
process(in, newptr)
register        struct    input_def          *in;
char    *newptr;
#else   ONEPROC
process(in)
register struct input_def               *in;
#endif  ONEPROC
{
        register struct hash_def                *p, *new;
#if     GRAFENG != PS300
#if     GRAFENG != IRIS
        register int                            index;
        psaddr_t                                cnt, sentinel;
        ps_t                                    objbuf[5];
#endif  IRIS
#endif  PS300
#ifndef ONEPROC
        register int                            i;
        register char                           *addr;
        register char                           *newptr;
        char                                    inbuf[PIPESIZE];
#endif  ONEPROC
#ifdef  BIGDEBUG
        register char                           *d_newptr;
#endif  BIGDEBUG
        extern struct hash_def                  *getptr();

        toomuch = FALSE;
#ifdef  ECHOCOM
        switch (in->i_type) {

        case S_QUIT :
                fputs("S:quit\n", stderr);
                break;

        case S_APPEND :
                fprintf(stderr, "S:append '%s'\n", in->i_name);
                break;

        case S_SREPLACE :
                fprintf(stderr, "S:sreplace '%s'\n", in->i_name);
                break;

        case S_BREPLACE :
                fprintf(stderr, "S:breplace '%s'\n", in->i_name);
                break;

        case S_DELETE :
                fprintf(stderr, "S:delete %d\n", in->i_molnum);
                break;

        case S_REPLACE :
                fprintf(stderr, "S:replace '%s'\n", in->i_name);
                break;

        case S_SCHDIR :
                fputs("S:chdir\n", stderr);
                break;

        case S_LREPLACE :
                fprintf(stderr, "S:lreplace '%s'\n", in->i_name);
                break;

        case S_HLFBND :
                fputs("S:halfbond\n", stderr);
                break;

        case S_ADD :
                fprintf(stderr, "S:add '%s' '%s'\n", in->i_name,
                        in->i_altname);
                break;

        case S_INSERT :
                fprintf(stderr, "S:insert '%s' '%s'\n", in->i_name,
                        in->i_altname);
                break;

        case S_RENAME :
                fprintf(stderr, "S:rename '%s' '%s'\n", in->i_name,
                        in->i_altname);
                break;

        case S_REMOVE :
                fprintf(stderr, "S:remove '%s'\n", in->i_name);
                break;

#if     GRAFENG != PS300
        case S_DRAW :
                fputs("S:draw\n", stderr);
                break;

        case S_OBJECT :
                fputs("S:object\n", stderr);
                break;
```

```
#if     GRAFENG != IRIS
        case S_IDENTRES :
                fputs("S:identres\n", stderr);
                break;

        case S_IDENTATM :
                fputs("S:identatm\n", stderr);
                break;
#endif
#endif  PS300
#if     GRAFENG == PS300
        case S_BACKWARD :
                fprintf(stderr, "S:backward '%s' '%s'\n", in->i_name,
                        in->i_altname);
                break;

        case S_FORWARD :
                fprintf(stderr, "S:forward '%s' '%s'\n", in->i_name,
                        in->i_altname);
                break;

        case S_GETLEVEL :
                fprintf(stderr, "S:getlevel '%s'\n", in->i_name);
                break;

        case S_SETLEVEL :
                fprintf(stderr, "S:setlevel '%s' '%d'\n",
                        in->i_name, in->i_ndata);
                break;

        case S_FINDNEXT :
                fprintf(stderr, "S:findnext %d '%s'\n",
                        in->i_molnum, in->i_name);
#endif  PS300

        default :
                fprintf(stderr, "S:unknown command = %d\n", in->i_type);
                break;
        }
#endif  ECHOCOM

        switch (in->i_type) {

        case S_QUIT :
                stop();

        case S_APPEND :
        case S_BREPLACE :
        case S_SREPLACE :
        case S_LREPLACE :
        case S_REPLACE :
        case S_ADD :
        case S_INSERT :
                p = getptr(in);
                if (p == NULL) {
#ifndef ONEPROC
                        clearbuf(in->i_size);
#else   ONEPROC
                        if (newptr != NULL)
                                free(newptr);
#endif  ONEPROC
                        toomuch = TRUE;
                        break;
                }
                moltab[in->i_molnum].inuse = TRUE;

#ifndef ONEPROC
                if (in->i_size > 0) {
                        newptr = malloc((unsigned) in->i_size
                                * (sizeof (char)));
                        if (newptr == NULL) {
                                clearbuf(in->i_size);
                                toomuch = TRUE;
                                break;
                        }
                        i = in->i_size;
                        addr = newptr;
                        while (i > PIPESIZE) {
                                (void) read(0, addr, PIPESIZE);
                                addr += PIPESIZE;
                                i -= PIPESIZE;
                        }
                        if (i > 0)
                                (void) read(0, addr, i);
                } else {
                        newptr = NULL;
                        in->i_size = 0;
                }
#endif  ONEPROC

#ifdef  DEBUG
                if (newptr != NULL)
                        checkob(in->i_name, (ps_t *) newptr,
```

```
                                        in->i_size);
#endif  DEBUG
#ifdef  BIGDEBUG
                d_newptr = malloc((unsigned) in->i_size * sizeof (char));
                blkcpy(newptr, d_newptr, in->i_size);
#endif  BIGDEBUG
#if GRAFENG == PS2 || GRAFENG == MPS
                if (in->i_type == S_SREPLACE || in->i_type == S_REPLACE) {
                        if (p->h_sdata != NULL && p->h_sdata != p->h_bdata
                          && p->h_sdata != p->h_ldata)
                                free(p->h_sdata);
                        p->h_sdata = newptr;
                        p->h_ssize = in->i_size;
#ifdef  BIGDEBUG
                        if (p->h_dsdata != NULL && p->h_dsdata != p->h_dbdat
                          && p->h_dsdata != p->h_dldata)
                                free(p->h_dsdata);
                        p->h_dsdata = d_newptr;
#endif  BIGDEBUG
                }
                if (in->i_type == S_LREPLACE || in->i_type == S_REPLACE) {
                        if (p->h_ldata != NULL && p->h_ldata != p->h_bdata
                          && p->h_ldata != p->h_sdata)
                                free(p->h_ldata);
                        p->h_ldata = newptr;
                        p->h_lsize = in->i_size;
#ifdef  BIGDEBUG
                        if (p->h_dldata != NULL && p->h_dldata != p->h_dbdata
                          && p->h_dldata != p->h_dsdata)
                                free(p->h_dldata);
                        p->h_dldata = d_newptr;
#endif  BIGDEBUG
                }
                if (in->i_type == S_BREPLACE || in->i_type == S_REPLACE) {
                        if (p->h_bdata != NULL && p->h_bdata != p->h_sdata
                          && p->h_bdata != p->h_ldata)
                                free(p->h_bdata);
                        p->h_bdata = newptr;
                        p->h_bsize = in->i_size;
#ifdef  BIGDEBUG
                        if (p->h_dbdata != NULL && p->h_dbdata != p->h_dsdat
                          && p->h_dbdata != p->h_dldata)
                                free(p->h_dbdata);
                        p->h_dbdata = d_newptr;
#endif  BIGDEBUG
                        p->h_ndata = in->i_ndata;
                        p->h_brkchain = in->i_newchain;
                        p->h_firstmode = in->i_firstmode;
                        p->h_x = in->i_x;
                        p->h_y = in->i_y;
                        p->h_z = in->i_z;
                        p->h_lx = in->i_lx;
                        p->h_ly = in->i_ly;
                        p->h_lz = in->i_lz;
                        p->h_color = in->i_color;
                        p->h_lcolor = in->i_lcolor;
                }
#else GRAFENG == PS2 || GRAFENG == MPS
                if (in->i_type == S_BREPLACE) {
                        p->h_brkchain = in->i_newchain;
                        p->h_firstmode = in->i_firstmode;
                        p->h_x = in->i_x;
                        p->h_y = in->i_y;
                        p->h_z = in->i_z;
                        p->h_lx = in->i_lx;
                        p->h_ly = in->i_ly;
                        p->h_lz = in->i_lz;
                        p->h_color = in->i_color;
                        p->h_lcolor = in->i_lcolor;
#if     GRAFENG == PS300
                        p->h_halfbond = in->i_halfbond;
                        fix_links(p, in->i_molnum);
#endif  PS300
                }
                if (in->i_type == S_APPEND ||
                  in->i_type == S_ADD ||
                  in->i_type == S_INSERT) {
                        p->h_newchain = in->i_newchain;
#if     GRAFENG == PS300
                        if (!p->h_newchain) {
                                new = p->h_blink;
                                if (new == NULL) {
                                        fprintf(stderr, "Impcond1 '%s'\n",
                                                in->i_name);
                                        break;
                                }
                                if (new->h_direction & SD_FORWARD)
                                        setdir(in->i_molnum, p, SD_FORWARD);
                                else
                                        setdir(in->i_molnum, p, SD_BACKWARD);
                        }
                        else
                                setdir(in->i_molnum, p, SD_FORWARD);
#endif  PS300
```

```
                }
#endif GRAFENG == PS2 || GRAFENG == MPS
                break;

        case S_DELETE :
                do_delobj(in->i_molnum);
                break;

        case S_SCHDIR :
                if (in->i_size <= 0)
                        break;
#ifndef ONEPROC
                (void) read(0, inbuf, in->i_size);
                inbuf[in->i_size] = '\0';
                newptr = inbuf;
#endif  ONEPROC
                (void) chdir(newptr);
                break;

        case S_HLFBND :
                halfbond = !halfbond;
                break;

        case S_RENAME :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                (void) delbucket(in->i_molnum, p);
                new = newbucket(in->i_molnum, in->i_name, p);
                break;

        case S_REMOVE :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                (void) delbucket(in->i_molnum, p);
                new = p->h_blink;
                if (new == NULL)
                        moltab[in->i_molnum].m_first = p->h_flink;
                else {
                        new->h_flink = p->h_flink;
#if     GRAFENG == PS300
                        pprintf("m%dr%snext := ", in->i_molnum,
                                        map_name(new->h_name));
#endif  PS300
                }
                new = p->h_flink;
                if (new != NULL) {
                        new->h_blink = p->h_blink;
#if     GRAFENG == PS300
                        pprintf("inst m%dr%s;\n", in->i_molnum,
                                        map_name(new->h_name));
#endif  PS300
                }
#if     GRAFENG == PS300
                else
                        pprintf("nil;\n");
#endif  PS300
                free((char *) p);
                break;

#if     GRAFENG != PS300
        case S_DRAW :
#ifndef ONEPROC
                (void) read(0, (char *) mdb,
                        sizeof (struct mdbs_def) * MAXMOD);
                (void) read(0, (char *) distance,
                        sizeof (struct dist_def) * MAXDIST);
                (void) read(0, (char *) angles,
                        sizeof(struct angle_def) * MAXANGLE);

#if     GRAFENG == PS2 || GRAFENG == MPS
                getps();
#endif  GRAFENG == PS2 || GRAFENG == MPS
#endif  ONEPROC
                do_draw();              /* output host objects to PS memory */
#ifndef ONEPROC
#if     GRAFENG == PS2 || GRAFENG == MPS
                putps();
#endif  GRAFENG == PS2 || GRAFENG == MPS

                (void) write(1, (char *) mdb,
                        sizeof (struct mdbs_def) * MAXMOD);
                (void) write(1, (char *) distance,
                        sizeof (struct dist_def) * MAXDIST);
                (void) write(1, (char *) angles,
                        sizeof(struct angle_def) * MAXANGLE);
#endif  ONEPROC
                break;
```

store.c

```
#if     GRAFENG != IRIS
        case S_OBJECT :
                /* read in a user OB (behaves logically as surf data) */
                in->i_type = S_APPEND;
                p = getptr(in);
                if (p == NULL) {
#ifndef ONEPROC
                        clearbuf(in->i_size);
#endif  ONEPROC
                        toomuch = TRUE;
                        break;
                }
                moltab[in->i_molnum].inuse = TRUE;

                if (in->i_size <= 0) {
                        toomuch = TRUE;
                        do_delobj(in->i_molnum);
                        break;
                }
#ifndef ONEPROC
                (void) read(0, inbuf, in->i_size);
                inbuf[in->i_size] = '\0';
                newptr = inbuf;
#endif  ONEPROC
                index = open(newptr, 0);
                if (index < 0) {
                        toomuch = TRUE;
                        do_delobj(in->i_molnum);
                        break;
                }
                (void) read(index, (char *) &cnt, sizeof cnt);
                if (lseek(index, (long) ((cnt - 1) * 2L), 0) == -1) {
                        toomuch = TRUE;
                        do_delobj(in->i_molnum);
                        break;
                }
                (void) read(index, (char *) &sentinel, sizeof sentinel);
                if (sentinel != SENTINEL) {
                        toomuch = TRUE;
                        do_delobj(in->i_molnum);
                        break;
                }
                p->h_ssize = cnt * sizeof (ps_t);
                p->h_sdata = malloc((unsigned) p->h_ssize);
                if (p->h_sdata == NULL) {
                        toomuch = TRUE;
                        do_delobj(in->i_molnum);
                        break;
                }
                (void) lseek(index, 0L, 0);
                (void) read(index, p->h_sdata, p->h_ssize);
                if (read(index, (char *) objbuf,
                   sizeof objbuf) != sizeof objbuf) {
                        in->i_ndata = 0;
                        in->i_size = 0;
                        in->i_x = 0;
                        in->i_y = 0;
                        in->i_z = 0;
                } else {
                        in->i_size = objbuf[0];
                        in->i_size = objbuf[1];
                        in->i_x = objbuf[2];
                        in->i_y = objbuf[3];
                        in->i_z = objbuf[4];
                }
                (void) close(index);
                break;

        case S_IDENTRES :
                toomuch = identres(in->i_x, in->i_y, in);
                break;

        case S_IDENTATM :
                p = getptr(in);
                if (p == NULL) {
#ifndef ONEPROC
                        clearbuf(in->i_size);
#else   ONEPROC
                        if (newptr != NULL)
                                free(newptr);
#endif  ONEPROC
                        toomuch = 0;
                        break;
                }
#ifndef ONEPROC
                newptr = malloc((unsigned) in->i_size * (sizeof (char)));
                if (newptr == NULL) {
                        clearbuf(in->i_size);
                        toomuch = 0;
                        break;
                }
                i = in->i_size;
                addr = newptr;
                while (i > PIPESIZE) {
```

```
                        (void) read(0, addr, PIPESIZE);
                        addr += PIPESIZE;
                        i -= PIPESIZE;
                }
                if (i > 0)
                        (void) read(0, addr, i);
#endif  ONEPROC
                /* newptr is a special residue with tagged atoms */
                toomuch = identatm(in->i_x, in->i_y, in->i_molnum,
                        p, newptr, in->i_size);
                if (newptr != NULL)
                        free(newptr);
                break;
#endif  IRIS
#endif  PS300

#if     GRAFENG == PS300
        case S_GETLEVEL :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        in->i_ndata = 0;
                        break;
                }
                in->i_ndata = p->h_level;
                break;

        case S_SETLEVEL :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                p->h_level = in->i_ndata;
                break;

        case S_FORWARD :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                setdir(in->i_molnum, p, SD_FORWARD);
                while (smatch(p->h_name, in->i_altname)) {
                        p = p->h_blink;
                        if (p == NULL)
                                break;
                        setdir(in->i_molnum, p, SD_FORWARD);
                }
                break;

        case S_BACKWARD :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                while (smatch(p->h_name, in->i_name)) {
                        setdir(in->i_molnum, p, SD_BACKWARD);
                        p = p->h_flink;
#ifdef  DEBUG
                        if (p == NULL) {
                                fprintf(stderr, "Back '%s' to '%s' failed\n");
                                abort();
                        }
#endif  DEBUG
                }
                setdir(in->i_molnum, p, SD_FORWARD | SD_BACKWARD);
                break;

        case S_FINDNEXT :
                p = getptr(in);
                if (p == NULL) {
                        toomuch = TRUE;
                        break;
                }
                in->i_halfbond = p->h_halfbond;
                while (p != NULL) {
                        if (p->h_firstmode != FM_NONE)
                                break;
                        p = p->h_flink;
                }
                if (p == NULL || p->h_firstmode == FM_MOVE) {
                        toomuch = TRUE;
                        break;
                }
                (void) strncpy(in->i_altname, p->h_name, RES_SEQ_SIZE + 1);
                break;
#endif  PS300
        }       /* end switch */

        in->i_type = toomuch;
#ifndef ONEPROC
#if     GRAFENG == PS2
```

```
        _flwrbuf();
#endif  PS2
#endif  ONEPROC
#ifdef  DUMPTAB
        printtab();
#endif  DUMPTAB
        return;  /* back to main() or drawpic.c */
}

static  struct  hash_def        *
getptr(in)
register        struct  input_def       *in;
{
        register        struct  hash_def        *p, *new;

        switch (in->i_type) {

        case S_APPEND :
                p = findbucket(in->i_molnum, in->i_name);
                if (p != NULL)
                        return(NULL);
                new = newbucket(in->i_molnum, in->i_name,
                        (struct hash_def *) NULL);
                if (new == NULL)
                        return(NULL);
                new->h_blink = moltab[in->i_molnum].m_last;
                if (moltab[in->i_molnum].m_first == NULL)
                        moltab[in->i_molnum].m_first = new;
                if (moltab[in->i_molnum].m_last != NULL)
                        moltab[in->i_molnum].m_last->h_flink = new;
                moltab[in->i_molnum].m_last = new;
                return(new);

        case S_SREPLACE :
        case S_BREPLACE :
        case S_LREPLACE :
        case S_REPLACE :
        case S_IDENTATM :
        case S_REMOVE :
        case S_GETLEVEL :
        case S_SETLEVEL :
        case S_FORWARD :
        case S_FINDNEXT :
                return(findbucket(in->i_molnum, in->i_name));

        case S_BACKWARD :
        case S_RENAME :
                return(findbucket(in->i_molnum, in->i_altname));

        case S_ADD :
        case S_INSERT :
                p = findbucket(in->i_molnum, in->i_altname);
                if (p == NULL)
                        return(NULL);
                if (in->i_type == S_INSERT) {
                        p = p->h_blink;
                        if (p == NULL)
                                return(NULL);
                }
                new = newbucket(in->i_molnum, in->i_name,
                        (struct hash_def *) NULL);
                if (new == NULL)
                        return(NULL);
                new->h_flink = p->h_flink;
                new->h_blink = p;
                p->h_flink = new;
                if (new->h_flink != NULL)
                        new->h_flink->h_blink = new;
                if (moltab[in->i_molnum].m_last == p)
                        moltab[in->i_molnum].m_last = new;
                return(new);

        default :
                fprintf(stderr, "Store: getptr called with %d\n", in->i_type);
                return(NULL);
        }
}

static
struct  hash_def        *
newbucket(molnum, name, new)
int     molnum;
char    *name;
struct  hash_def        *new;
{
        int     index;

        if (new == NULL) {
                new = (struct hash_def *) malloc(sizeof (struct hash_def));
                if (new == NULL) {
                        fprintf(stderr, "Newbucket: out of memory\n");
                        return(NULL);
                }
                initbucket(new);
```

```
	}
	index = hash(name);
	strncpy(new->h_name, name, RES_SEQ_SIZE);
	new->h_name[RES_SEQ_SIZE] = '\0';
	new->h_next = moltab[molnum].hashtab[index];
	moltab[molnum].hashtab[index] = new;
	return(new);
}

static
struct	hash_def		*
findbucket(molnum, name)
int	molnum;
char	*name;
{
	int		index;
	register		struct	hash_def		*p;

	index = hash(name);
	if (moltab[molnum].hashtab[index] == NULL)
		return(NULL);

	for (p = moltab[molnum].hashtab[index]; p != NULL; p = p->h_next)
		if (strncmp(name, p->h_name, RES_SEQ_SIZE) == 0)
			break;
	return(p);
}

static
delbucket(molnum, bucket)
int	molnum;
struct	hash_def		*bucket;
{
	int		index;
	register		struct	hash_def		**p;

	index = hash(bucket->h_name);
	if (moltab[molnum].hashtab[index] == NULL)
		return(-1);

	for (p = &moltab[molnum].hashtab[index]; *p != NULL;
	   p = &((*p)->h_next) )
		if (*p == bucket) {
			*p = bucket->h_next;
			return(0);
		}
	return(-1);
}

static
initbucket(new)
register		struct	hash_def		*new;
{
	new->h_flink = new->h_blink = new->h_next = NULL;
	new->h_firstmode = FM_NONE;
	new->h_brkchain = 0;
	new->h_newchain = 0;
	new->h_direction = 0;
	new->h_color = -1;
	new->h_lcolor = -1;
	new->h_level = 0;
	new->h_ndata = 0;
#if	GRAFENG == PS2 || GRAFENG == MPS
	new->h_bdata = new->h_ldata = new->h_sdata = NULL;
	new->h_bsize = new->h_lsize = new->h_ssize = 0;
#endif	PS300
#ifdef	BIGDEBUG
	new->h_dsdata = new->h_dbdata = new->h_dldata = NULL;
#endif	BIGDEBUG
}

static
hash(s)
register		char	*s;
{
	register		index;
	register		count;

	count = index = 0;
	while (*s != '\0' && count < RES_SEQ_SIZE) {
		index += *s++;
		count++;
	}
	if (index < 0)
		index = -index;
	index %= HASHSIZE;
	return(index);
}

#if	GRAFENG == PS300
static
setdir(molnum, p, newdir)
int	molnum;
register		struct	hash_def		*p;
```

```
int	newdir;
{
	register	struct	hash_def	*q;

	q = p->h_blink;
	if (!p->h_newchain && q != NULL && q->h_direction & SD_FORWARD) {
		/*
		 * two psprintf's since map_name returns pointer to static
		 * buffer.
		 */
		psprintf("m%dr%snext := ", molnum, map_name(q->h_name));
		psprintf("inst m%dr%s;\n", molnum, map_name(p->h_name));
	}

	if (newdir & SD_BACKWARD && q != NULL) {
		psprintf("m%dr%sprev := ", molnum, map_name(p->h_name));
		psprintf("inst m%dr%s;\n", molnum, map_name(q->h_name));
	}
	else if (p->h_direction & SD_BACKWARD)
		psprintf("m%dr%sprev := nil;\n", molnum, map_name(p->h_name)

	q = p->h_flink;
	if (newdir & SD_FORWARD && q != NULL && !q->h_newchain) {
		psprintf("m%dr%snext := ", molnum, map_name(p->h_name));
		psprintf("inst m%dr%s;\n", molnum, map_name(q->h_name));
	}
	else if (p->h_direction & SD_FORWARD)
		psprintf("m%dr%snext := nil;\n", molnum, map_name(p->h_name)

	if (q != NULL && !q->h_newchain && q->h_direction & SD_BACKWARD) {
		psprintf("m%dr%sprev := ", molnum, map_name(q->h_name));
		psprintf("inst m%dr%s;\n", molnum, map_name(p->h_name));
	}

	p->h_direction = newdir;
}

static
drawlink(name, from, to)
char	*name;
register struct hash_def		*from, *to;
{
	ps_t	midx, midy, midz;
	static P_VectorType linkdat[2];

	if (from->h_halfbond) {
		midx = (from->h_lx + to->h_x) / 2;
		midy = (from->h_ly + to->h_y) / 2;
		midz = (from->h_lz + to->h_z) / 2;
	}
	MVecBegn(name, P_Item, 0);
	linkdat[0].Draw = FALSE;
	linkdat[0].V4[0] = from->h_lx;
	linkdat[0].V4[1] = from->h_ly;
	linkdat[0].V4[2] = from->h_lz;
	if (from->h_halfbond) {
		linkdat[0].V4[3] = ps300cm[from->h_color];
		linkdat[1].Draw = TRUE;
		linkdat[1].V4[0] = midx;
		linkdat[1].V4[1] = midy;
		linkdat[1].V4[2] = midz;
		linkdat[1].V4[3] = ps300cm[from->h_color];
		MVecList(2, linkdat);
		linkdat[0].Draw = TRUE;
		linkdat[0].V4[0] = to->h_x;
		linkdat[0].V4[1] = to->h_y;
		linkdat[0].V4[2] = to->h_z;
		linkdat[0].V4[3] = ps300cm[to->h_color];
		MVecList(1, linkdat);
	}
	else {
		linkdat[0].V4[3] = ps300cm[to->h_color];
		linkdat[1].Draw = TRUE;
		linkdat[1].V4[0] = to->h_x;
		linkdat[1].V4[1] = to->h_y;
		linkdat[1].V4[2] = to->h_z;
		linkdat[1].V4[3] = ps300cm[to->h_color];
		MVecList(2, linkdat);
	}
	MVecEnd();
}
#endif	PS300

#if	GRAFENG != PS300
static
do_draw()
{
	register struct hash_def		*p;
	register int			i, j, n;
	register int			curcolor = -1;
	register ps_t			x, y, z;
#if	GRAFENG != IRIS
	unsigned int			loc;
	int				counter = 0;
```

```
#endif  IRIS

#ifdef  HUGEDEBUG
        int     save;
        extern  _memfid;

        save = _memfid;
        (void) close(creat(TMPDIR(psmem.sim), 0666));
        _memfid = open(TMPDIR(psmem.sim), 2);
        fprintf(stderr, "psmem.sim opened as %d\n", _memfid);
#endif  HUGEDEBUG

        flushcrd();             /* should be nothing there */
#if     GRAFENG == PS2
        makeps('pc', &counter);
#endif  PS2
#if     GRAFENG == MPS
        udelet(&'bd');          /* order important for efficiency reasons */
        udelet(&'pc');
        ubegin(&'pc', &counter);
#endif  MPS
#if     GRAFENG == IRIS
#ifdef  DRAWPIC
        makeobj(O_ALL);
        ir_color(BLACK);
        clear();
        callobj(O_BOND);
        callobj(O_SSBOND);
        callobj(O_SURFACE);
        callobj(O_LABEL);
        closeobj();
#endif  DRAWPIC
#endif  IRIS

        /* Draw the surface */
#if     GRAFENG == PS2
        speed(0);               /* Full line generator speed (OK for dots per E&S) */
        fixlg(DOTTED);
#endif  PS2
#if     GRAFENG == IRIS
        makeobj(O_SURFACE);
#endif  IRIS
#if     GRAFENG == MPS
        /*
         * HELP NEEDED HERE SOMEDAY!
         * See comment in drawpic.c, routine drawsurf()
         */
#endif  MPS
        for (i = 0; i < MAXMOD; i++) {
                if (toomuch)
                        break;
                if (!moltab[i].inuse)
                        continue;
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink) {
#ifdef  BIGDEBUG
                        compob(p->h_sdata, p->h_dsdata, p->h_ssize,
                                p->h_name, "Surface");
#endif  BIGDEBUG
#if     GRAFENG != IRIS
                        if (p->h_ssize > MINSIZE && p->h_sdata != NULL) {
                                if (TOOMUCH(p->h_ssize, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                                        fprintf(stderr, "!%d %d (%d) >? %u!\n",
                                                PSSIZE(p->h_ssize), counter,
                                                PSSIZE(p->h_ssize) + counter,
                                                PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                                        toomuch = TRUE;
                                        break;
                                }
                                drawob((ps_t *) p->h_sdata);
                        }
#else   IRIS
                        if (is_validobj(i, p->h_name))
                                callobj(numobj(i, p->h_name, O_SURFACE));
#endif  IRIS
                }
        }
#if     GRAFENG == PS2
        if (!toomuch) {
                fixlg(!DOTTED);
                speed(1);               /* back to 1/2 speed for labels */
        }
#endif  PS2
#if     GRAFENG == IRIS
        closeobj();
#endif  IRIS

        /* Draw the label */
#if     GRAFENG == IRIS
        makeobj(O_LABEL);
        font(SMALLFONT);
```

store.c

```
#endif  IRIS
        for (i = 0; i < MAXMOD; i++) {
                if (toomuch)
                        break;
                if (!moltab[i].inuse)
                        continue;
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink) {
#ifdef  BIGDEBUG
                        compob(p->h_ldata, p->h_dldata, p->h_lsize,
                                p->h_name, "Label");
#endif  BIGDEBUG
#if     GRAFENG != IRIS
                        if (p->h_lsize > MINSIZE && p->h_ldata != NULL) {
                                if (TOOMUCH(p->h_lsize, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                                        fprintf(stderr, "!%d %d (%d) >? %u!\
                                                PSSIZE(p->h_lsize), coun
                                                PSSIZE(p->h_lsize) + cou
                                                PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                                        toomuch = TRUE;
                                        break;
                                }
                                drawob((ps_t *) p->h_ldata);
                        }
#else   IRIS
                        if (is_validobj(i, p->h_name))
                                callobj(numobj(i, p->h_name, O_LABEL));
#endif  IRIS
                }
        }
#if     GRAFENG == IRIS
        closeobj();
#endif  IRIS

        /* Draw the bonds */
#if     GRAFENG == PS2
        entryps('bd');
#endif  PS2
#if     GRAFENG == MPS
        uend();
        if (toomuch) {
                udelet(&'pc');
                ubegin(&'pc', &counter);
                ttran(&0, &0, &0);
                uend();
        }
        ubegin(&'bd', &counter);
        setbase();          /* for the savecrd()'s below */
#endif  MPS
#if     GRAFENG == IRIS
        makeobj(O_BOND);
#else
        loc = 0;
#endif

        n = FALSE;
        x = y = z = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (toomuch)
                        break;
                if (!moltab[i].inuse)
                        continue;
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink) {
#ifdef  BIGDEBUG
                        compob(p->h_bdata, p->h_dbdata, p->h_bsize,
                                p->h_name, "Bond");
#endif
                        if (p->h_brkchain)
                                n = TRUE;
#if     GRAFENG != IRIS
                        if (p->h_bsize > MINSIZE && p->h_bdata != NULL)
#else
                        if (is_validobj(i, p->h_name))
#endif
                        {
                                if (n && p->h_firstmode != FM_NONE) {
                                        /* Can't use moveto/lineto since they
                                         * change the base registers and make
                                         * the draw3d in the objects fail
                                         */
                                        savecrd(MOVETO, p->h_x, p->h_y, p->h_z);
                                        flushcrd();
#if     GRAFENG != IRIS
                                        loc += ITEMSIZE;
#endif
                                        n = FALSE;
                                } else if (halfbond && p->h_firstmode == FM_LINE
                                  && curcolor != p->h_color) {
                                        x = (x + p->h_x) / 2;
                                        y = (y + p->h_y) / 2;
                                        z = (z + p->h_z) / 2;
                                        savecrd(LINETO, x, y, z);
```

store.c

```
                        flushcrd();
#if     GRAFENG != IRIS
                        loc += ITEMSIZE;
#endif
                }
#if     GRAFENG != IRIS
                if (TOOMUCH(p->h_bsize, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                        fprintf(stderr, "!%d %d (%d) >? %u!\
                                PSSIZE(p->h_bsize), cou
                                PSSIZE(p->h_bsize) + co
                                PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                        toomuch = TRUE;
                        break;
                }
                /* No buffering so no flushcrd() */
                drawob((ps_t *) p->h_bdata);
#else   IRIS
                callobj(numobj(i, p->h_name, O_BOND));
#endif  IRIS
#if     GRAFENG == MPS
                resetbase();
#endif
                x = p->h_lx;
                y = p->h_ly;
                z = p->h_lz;
                curcolor = p->h_lcolor;
        }

        for (j = 0; j < MAXDIST; j++) {
                if (distance[j].d_type == INACTIVE)
                        continue;
                if (distance[j].d_type & HOLDALL)
                        continue;
#if     GRAFENG != IRIS
                fitdist(j, i, p->h_name, loc);
#endif
        }

        for (j = 0; j < MAXANGLE; j++) {
                if (angles[j].a_type == INACTIVE)
                        continue;
                if (angles[j].a_type & HOLDALL)
                        continue;
#if     GRAFENG != IRIS
                        fitangle(j, i, p->h_name, loc);
#endif
                }
#if     GRAFENG != IRIS
                loc += p->h_ndata;
#endif
        }
}
#if     GRAFENG == PS2
moveto(0, 0, 0);        /* So deflection amp doesn't work so hard */
huesat(WHITE);          /* So cursor stays white */
stopps();
if (toomuch) {
        makeps('pc', &counter);
        entryps('bd');
        moveto(0, 0, 0);
        stopps();
}
#endif  PS2
#if     GRAFENG == IRIS
closeobj();
#endif  IRIS
#if     GRAFENG == MPS
damove(&0, &0, &0);
lcolor(&WHITE);
uend();
if (toomuch) {
        udelet(&'bd');
        ubegin(&'bd', &counter);
        damove(&0, &0, &0);
        uend();
}
#endif  MPS

#if     GRAFENG == IRIS
makeobj(O_PNTS);
#ifdef  DEBUG
ir_color(GREEN);
#endif
for (i = 0; i < MAXMOD; i++) {
        if (!moltab[i].inuse)
                continue;
        initnames();
        loadname((short) i);
        pushname(0);    /* dummy so that drawbond() can just loadname */
```

```
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink)
                        if (is_validobj(i, p->h_name))
                                callobj(numobj(i, p->h_name, O_PNTS));
                popname();
        }
        closeobj();
#endif  IRIS

#ifdef  HUGEDEBUG
        (void) close(_memfid);
        _memfid = save;
#endif  HUGEDEBUG
}

#if     GRAFENG != IRIS
static
fitdist(n, model, resseq, loc)
register        n;
int     model;
char    *resseq;
unsigned        loc;
{
        register        i;

        for (i = 0; i < 2; i++) {
                if (distance[n].d_model[i] == model &&
                    !smatch(resseq, distance[n].d_resseq[i]))
                        distance[n].d_psloc[i] = loc + distance[n].d_obloc[i];
        }
}

static
fitangle(n, model, resseq, loc)
register        n;
int     model;
char    *resseq;
unsigned        loc;
{
        register        i;

        for (i = 0; i < angles[n].a_natom; i++) {
                if (angles[n].a_model[i] == model &&
                    !smatch(resseq, angles[n].a_resseq[i]))
                        angles[n].a_psloc[i] = loc + angles[n].a_obloc[i];
        }
}
#endif  IRIS

#endif  PS300

static
do_delobj(molnum)
int     molnum;
{
        register struct hash_def        *p, *new;
        register int                    i;
#if     GRAFENG == PS300
        char                            buf[80];
#endif  PS300

        if (!moltab[molnum].inuse)
                return;
#if     GRAFENG == PS300
        psprintf("delete m%d";\noptimize memory;\n", molnum);
#endif  PS300
        for (i = 0; i < HASHSIZE; i++) {
                for (p = moltab[molnum].hashtab[i]; p != NULL; p = new) {
                        new = p->h_next;
#if     GRAFENG == IRIS
                        if (p->h_name[0] == 'p')
                                delobj(numobj(molnum, p->h_name, 0));
                        else if (p->h_name[0] == 'b') {
                                delobj(numobj(molnum, p->h_name, O_BOND));
                                delobj(numobj(molnum, p->h_name, O_PNTS));
                        }
                        else {
                                delobj(numobj(molnum, p->h_name, O_LABEL));
                                delobj(numobj(molnum, p->h_name, O_BOND));
                                delobj(numobj(molnum, p->h_name, O_PNTS));
                                delobj(numobj(molnum, p->h_name, O_SURFACE));
                        }
#endif  IRIS
#if GRAFENG == PS2 || GRAFENG == MPS
                        if (p->h_bdata != NULL)
                                free(p->h_bdata);
                        if (p->h_sdata != NULL && p->h_sdata != p->h_bdata)
                                free(p->h_sdata);
                        if (p->h_ldata != NULL && p->h_ldata != p->h_bdata &&
                        p->h_ldata != p->h_sdata)
                                free(p->h_ldata);
#endif GRAFENG == PS2 || GRAFENG == MPS
                        free((char *) p);
                }
                moltab[molnum].hashtab[i] = NULL;
        }
```

```
        moltab[molnum].m_first = moltab[molnum].m_last = NULL;
        moltab[molnum].inuse = FALSE;
        return;
}

#ifndef ONEPROC
static
clearbuf(size)
register        size;
{
        char    inbuf[PIPESIZE];

        while (size > PIPESIZE) {
                (void) read(0, inbuf, PIPESIZE);
                size -= PIPESIZE;
        }
        if (size > 0)
                (void) read(0, inbuf, size);
}
#endif  ONEPROC

#if     GRAFENG == PS2 || GRAFENG == MPS
static
identres(hx, hy, in)
ps_t    hx, hy;
struct  input_def       *in;
{
        register        struct  hash_def        *p;
        register        i;
        ps_t    matrix[4][4];
        int     counter = 0;
        int     n, done;
        int     rescnt;

#if     GRAFENG == PS2
        mdtype('pc', 'bd', &counter);
#endif  PS2
#if     GRAFENG == MPS
        udelet(&'bd');
        ubegin(&'bd', &counter);
#endif  MPS
        done = n = FALSE;
        for (i = 0; i < MAXMOD; i++) {
                if (done)
                        break;
                if (!moltab[i].inuse)
                        continue;
                rescnt = 0;
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink) {
                        if (done)
                                break;
                        if (p->h_brkchain)
                                n = TRUE;
                        if (p->h_bsize > MINSIZE && p->h_bdata != NULL) {
#if     GRAFENG == PS2
                                hittag(rescnt * 10 + i + 1);
#endif  PS2
#if     GRAFENG == MPS
                                int     udiname;

                                udiname = rescnt * 10 + i + 1;
                                hname(&udiname);
#endif  MPS
                                if (n && p->h_firstmode != FM_NONE) {
#if     GRAFENG == PS2
                                        savecrd(MOVETO, p->h_x, p->h_y, p->h_z);
                                        flushcrd();
#endif  PS2
#if     GRAFENG == MPS
                                        damove(&p->h_x, &p->h_y, &p->h_z);
#endif  MPS
                                        n = FALSE;
                                }
                                if (TOOMUCH(p->h_bsize, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                                        fprintf(stderr, "!%d %d (%d) >? %u!\n",
                                                PSSIZE(p->h_bsize), counter,
                                                PSSIZE(p->h_bsize) + counter,
                                                PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                                        done = TRUE;
                                        break;
                                }
                                drawob((ps_t *) p->h_bdata);
                        }
                        rescnt++;
                }
        }
#if     GRAFENG == PS2
        stoppe();
#endif  PS2
#if     GRAFENG == MPS
```

```
        uend();
#endif  MPS

        if (done) {
                remakebond();
                return(1);
        }
#if     GRAFENG == PS2
        bldcon(STOREMAT, matrix);
        hitset(hx, hy, HWSIZE);
        drawps('bd');
        rescnt = hitclr();
        bldcon(INITIALIZE);
        bldcon(LOADMAT, matrix);
#endif  PS2
#if     GRAFENG == MPS
        hbegin(&hx, &hy, &HWSIZE);
        udata(&'bd', psobs);
        hget(&i, &rescnt);
        if (i == 0)
                rescnt = 0;
        hend();
#endif  MPS
        if (rescnt == 0) {
                remakebond();
                return(1);
        }
        in->i_molnum = rescnt % 10 - 1;
        rescnt /= 10;
        for (p = moltab[in->i_molnum].m_first; p != NULL; p = p->h_flink)
                if (rescnt-- == 0) {
                        strcpy(in->i_name, p->h_name);
                        break;
                }
        return(0);
}

static
identatm(hx, hy, molnum, res, newptr, newsize)
ps_t    hx, hy;
int     molnum;
struct  hash_def        *res;
char    *newptr;
int     newsize;
{
        register        struct  hash_def        *p;
        register        i, size;
        ps_t    *ob;
        ps_t    matrix[4][4];
        int     counter = 0;
        int     n, done;

#if     GRAFENG == PS2
        mdfyps('pc', 'bd', &counter);
#endif  PS2
#if     GRAFENG == MPS
        udelet(&'bd');
        ubegin(&'bd', &counter);
#endif  MPS
        done = n = FALSE;
        for (p = moltab[molnum].m_first; p != NULL; p = p->h_flink) {
                if (done)
                        break;
                if (p->h_brkchain)
                        n = TRUE;
                if (p->h_bsize > MINSIZE && p->h_bdata != NULL) {
                        if (n && p->h_firstmode != FM_NONE) {
#if     GRAFENG == PS2
                                savecrd(MOVETO, p->h_x, p->h_y, p->h_z);
                                flushcrd();
#endif  PS2
#if     GRAFENG == MPS
                                damove(&p->h_x, &p->h_y, &p->h_z);
#endif  MPS
                                n = FALSE;
                        }
                        if (p == res) {
                                ob = (ps_t *) newptr;           /* special OB w/tags */
                                size = newsize;
                        } else {
                                ob = (ps_t *) p->h_bdata;
                                size = p->h_bsize;
                        }
                        if (TOOMUCH(size, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                                fprintf(stderr, "!%d %d (%d) >? %u!\n",
                                        PSSIZE(size), counter,
                                        PSSIZE(size) + counter, PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                                done = TRUE;
                                break;
                        }
```

```
                        drawob(ob);
                }
        }
#if     GRAFENG == PS2
        stopps();
#endif  PS2
#if     GRAFENG == MPS
        uend();
#endif  MPS

        if (done) {
                remakebond();
                return(0);
        }
#if     GRAFENG == PS2
        bldcon(STOREMAT, matrix);
        hitset(hx, hy, HWSIZE);
        drawps('bd');
        i = hitclr();
        bldcon(INITIALIZE);
        bldcon(LOADMAT, matrix);
#endif  PS2
#if     GRAFENG == MPS
        hbegin(&hx, &hy, &HWSIZE);
        udata(&'bd', psobs);                    /* have MAP check for hits */
        hget(&n, &i);
        if (n == 0)
                i = 0;
        hend();
#endif  MPS
        remakebond();
        return(i);
}

static
remakebond()
{
        register        struct  hash_def        *p;
        register        i;
        int     n, done;
        ps_t    x, y, z;
        int     counter = 0;

#if     GRAFENG == PS2
        mdfyps('pc', 'bd', &counter);
#endif  PS2
#if     GRAFENG == MPS
        udelet(&'bd');
        ubegin(&'bd', &counter);
#endif  MPS
        done = n = FALSE;
        x = y = z = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (done)
                        break;
                if (!moltab[i].inuse)
                        continue;
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink) {
                        if (p->h_brkchain)
                                n = TRUE;
                        if (p->h_bsize > MINSIZE && p->h_bdata != NULL) {
                                if (n && p->h_firstmode != FM_NONE) {
#if     GRAFENG == PS2
                                        savecrd(MOVETO, p->h_x, p->h_y, p->h_z);
                                        flushcrd();
#endif  PS2
#if     GRAFENG == MPS
                                        damove(&p->h_x, &p->h_y, &p->h_z);
#endif  MPS
                                        n = FALSE;
                                } else if (halfbond
                                  && p->h_firstmode == FM_LINE) {
                                        x = (x + p->h_x) / 2;
                                        y = (y + p->h_y) / 2;
                                        z = (z + p->h_z) / 2;
#if     GRAFENG == PS2
                                        savecrd(LINETO, x, y, z);
                                        flushcrd();
#endif  PS2
#if     GRAFENG == MPS
                                        daline(&x, &y, &z);
#endif  MPS
                                }
                                if (TOOMUCH(p->h_bsize, counter)) {
#ifndef SYSTEM
#ifdef  DEBUG
                                        fprintf(stderr, "!%d %d (%d) >? %u!\n",
                                                PSSIZE(p->h_bsize), counter,
                                                PSSIZE(p->h_bsize) + counter,
                                                PSMEMSZ);
#endif  DEBUG
#endif  SYSTEM
                                        done = TRUE;
                                        break;
```

```
                                        }
                                        drawob((ps_t *) p->h_bdata);
                                        x = p->h_lx;
                                        y = p->h_ly;
                                        z = p->h_lz;
                                }
                        }
                }
#if     GRAFENG == PS2
                moveto(0, 0, 0);
                huesat(WHITE);
                stopps();
                if (done) {
                        mdfyps('pc', 'bd', &counter);
                        tran(0, 0, 0);
                        stopps();
                }
#endif  PS2
#if     GRAFENG == MPS
                damove(&0, &0, &0);
                lcolor(&WHITE);
                uend();
                if (done) {
                        udelet(&'bd');
                        ubegin(&'bd', &counter);
                        ttran(&0, &0, &0);
                        uend();
                }
#endif  MPS
}
#endif  GRAFENG == PS2 || GRAFENG == MPS

#if     GRAFENG == PS300
fix_links(p, molnum)
register struct hash_def        *p;
int                             molnum;
{
        register struct hash_def        *prev, *next;
        char                            gname[BUFSIZ];

        /*
         * Find the previous residue within the current chain which
         * is not empty.
         */
        for (prev = p->h_blink; prev != NULL; prev = prev->h_blink)
                if (prev->h_brkchain || prev->h_firstmode != FM_NONE)
                        break;
        if (prev != NULL && prev->h_firstmode == FM_NONE)
                prev = NULL;

        /*
         * Find the next residue within the current chain which is
         * not empty.
         */
        for (next = p->h_flink; next != NULL; next = next->h_flink)
                if (next->h_brkchain || next->h_firstmode != FM_NONE)
                        break;
        if (next != NULL && next->h_brkchain)
                next = NULL;

        /*
         * Create the link for this residue.
         */
        (void) sprintf(gname, "m%dr%slink", molnum, map_name(p->h_name));
        if (prev != NULL && !p->h_brkchain && p->h_firstmode == FM_LINE)
                drawlink(gname, prev, p);
        else {
                nullnode(gname);
        }

        /*
         * Fix up anything after this one
         */
        if (next == NULL)
                return;
        (void) sprintf(gname, "m%dr%slink", molnum, map_name(next->h_name));
        if (!p->h_brkchain && p->h_firstmode == FM_NONE)
                p = prev;
        if (p != NULL && p->h_firstmode != FM_NONE)
                drawlink(gname, p, next);
        else {
                nullnode(gname);
        }
}
#endif  PS300

#if     GRAFENG == MPS
resetbase()
{
        /*
         * The MPS library tries to protect
         * us from doing stupid things like
         * doing draws in OFFSET mode when the
         * MAP base register is in an undefined
```

```
         * state.  Unfortunately, you can't
         * find out what the real state of the
         * base register is, and so the software
         * does the best it can which means
         * declaring the base reg "undefined"
         * after things like drawob(). This
         * can waste lots of MPS memory by
         * requiring dbase() calls after every
         * drawob() call.
         *
         * resetbase() essentially says "heh,
         * we know what we're doing, the base
         * register really is still valid and
         * you don't need to do a dbase() call"
         */

        extern struct u0com u0com;

        u0com.gbase = 0;            /* base reg valid for all draw types */
}
#endif

#ifdef  DEBUG
static
checkob(name, ob, size)
char    *name;
ps_t    *ob;
int     size;
{
        register        ps_t    *fromp;
        register        int     count;

        fromp = ob + 1;
        count = PSSIZE(size);
        if (*(fromp + count) != SENTINEL) {
                fprintf(stderr, "STORE: bad ob array for residue %s\n", name);
                abort();
        }
}
#endif  DEBUG

#ifdef  BIGDEBUG
compob(inuse, backup, size, name, type)
register        char    *inuse, *backup;
register        size;
char    *name, *type;
{
#define ALIGN_T         int
#define ASIZE           sizeof (ALIGN_T)
#define align(x)        ((((x) + ASIZE + ASIZE - 1) / ASIZE) * ASIZE)

#ifndef vms             /* depends on malloc internals */
        if (*(((int *) inuse) - 1) != align(size))
                fprintf(stderr, "orig %s '%s' looks suspicious (%d != %d)\n",
                        type, name, *(((int *) inuse) - 1), align(size));
        if (*(((int *) backup) - 1) != align(size))
                fprintf(stderr, "back %s '%s' looks suspicious (%d != %d)\n",
                        type, name, *(((int *) backup) - 1), align(size));
#endif
        while (size--)
                if (*inuse++ != *backup++) {
                        fprintf(stderr, "S:verify %s '%s' failed.\n",
                                type, name);
                        return;
                }
}
#endif  BIGDEBUG

#ifdef  DUMPTAB
printtab()
{
        register        struct  hash_def        *p;
        register        i;

        for (i = 0; i < MAXMOD; i++) {
                if (!moltab[i].inuse)
                        continue;
#ifdef  BIGDEBUG
                fprintf(stderr, "%6d _addr_ flink_ blink_ _next_\n");
#endif  BIGDEBUG
                for (p = moltab[i].m_first; p != NULL; p = p->h_flink)
#ifdef  BIGDEBUG
                        fprintf(stderr, "%6s %6x %6x %6x %6x\n",
                                p->h_name, p, p->h_flink, p->h_blink,
                                p->h_next);
#else   BIGDEBUG
                        fprintf(stderr, "%s ", p->h_name);
#endif  BIGDEBUG
                }
        fprintf(stderr, "\n");
}
#endif  DUMPTAB

#ifdef  DEBUG
```

store.c

```
prentry(p)
register        struct    hash_def        *p;
{
        if (p == NULL) {
                fprintf(stderr, "NULL\n");
                return;
        }
        fprintf(stderr, "%s:", p->h_name);
        fprintf(stderr, "brkchain=%d firstmode=%d newchain=%d direction=%o\n",
                p->h_brkchain, p->h_firstmode, p->h_newchain, p->h_direction);
}
#endif  DEBUG

#ifndef EDITONLY
#if     GRAFENG == IRIS

static  char    restype[RES_TYPE_SIZE+1];

static short    lastrad;

static int      lastcolor;

extern char     *rseq, *rtype;

extern int      molnum, mat_stack;

extern struct atom_def  *atoms;

makecpk()
{
        register int    mol;
        char            seq[RES_SEQ_SIZE+1];
        char            buf[20];
        int             cpkvisit(), cpkagain();

        makeobj(O_CPK);
        lastcolor = -1;
        lastrad = -1;
        for (mol = 0; mol < MAXMOD; mol++) {
                if (!moltab[mol].inuse)
                        continue;

                /* Start at the beginning and go to the end */
                molnum = mol;
                mdb[mol].m_nchain = 0;
                (void) mseekr(mdb[mol].m_fd, all, FLSEEK);
                while (mreadr(mdb[mol].m_fd, seq, restype) > 0) {
                        if (tmatch(restype, "BOND") == 0)
                                continue;

                        /* Set up global variables */
                        molnum = mol;
                        rseq = seq;
                        rtype = restype;
                        mat_stack = 0;

                        (void) mreada(mdb[mol].m_fd);
                        atoms = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                        (void) mtrav(mdb[mol].m_fd, cpkvisit, cpkagain);
                }
                sprintf(buf, "pop%d", mdb[mol].m_nchain);
                callobj(numobj(mol, buf, 0));
        }
        closeobj();
}

static
cpkvisit(db, index, ischeif, islink, nsons, firstime)
int     db;
int     index, ischeif, islink, nsons, firstime;
{
        register struct atom_def        *data;
        register ps_t                   x, y, z;
        register short                  rad;
        char                            atname[AT_NAME_SIZE+1];
        char                            buf[20];
        extern double                   getradius();

        if (!firstime)
                return;

        data = &atoms[index];

        if (data->status & STARTBIT) {
                if (mdb[molnum].m_nchain > 0) {
                        sprintf(buf, "pop%d", mdb[molnum].m_nchain);
                        callobj(numobj(molnum, buf, O_BOND));
                }
                mdb[molnum].m_nchain++;
                sprintf(buf, "psh%d", mdb[molnum].m_nchain);
                callobj(numobj(molnum, buf, O_BOND));
        }

        if (!(data->status & EXISTBIT))
```

```
                return;

        if (data->status & SHOWBIT) {
                matom(db, index, atname);
                x = MAPCRD(data->x);
                y = MAPCRD(data->y);
                z = MAPCRD(data->z);
                rad = getradius(atname, restype) * 1000;

                if (data->color != lastcolor) {
                        lastcolor = data->color;
                        color(lastcolor);
                }
                if (rad != lastrad) {
                        lastrad = rad;
                        loadname(rad);
                }
                pnt(x, y, z);
                xfpt(x, y, z);
        }

        /* Check for rotations */
        if (data->status & ROTBIT) {
                pushrot(db, index);
                if (nsons == 0)
                        poprot(db, index);
        }
}

/* ARGSUSED */
static
cpkagain(db, index, ischief, islink, nsons)
int     db;
int     index, ischief, islink, nsons;
{
        register struct atom_def                *data;

        data = &atoms[index];
        if (!(data->status & EXISTBIT))
                return;

        /* Check rotation (if there are no more branches) */
        if (data->status & ROTBIT && nsons == 0)
                poprot(db, index);
}
#endif  IRIS
#endif  EDITONLY
```

distance.i

```
/* $Header: /usr/src/local/midas/src/editor/RCS/distance.i,v 3.1 83/11/23 15:09:59 bin
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 */

static  int     distnum;
static  int     dnatom;

/*
 * setupdist - check for distance specification validity
 */
setupdist(n)
int     n;
{
        distance[n].d_type = INACTIVE;
        distnum = n;
        dnatom = 0;
        editres = selres;
        prepres = NULL;
        realedit = editdist;
}

/*
 * editdist - set up distance information and check for validity
 */
editdist(selected, index, atname)
int     selected, index;
char    *atname;
{
        register        i;

        if (!selected || dnatom > 2)
                return;
        i = dnatom++;
        switch (i) {

        case 1 :
                distance[distnum].d_type = ACTIVE;
                /* Fall through */
        case 0 :
                distance[distnum].d_model[i] = mol;
                strcpy(distance[distnum].d_resseq[i], resseq);
                strcpy(distance[distnum].d_atom[i], atname);
                break;
        case 2 :
                distance[distnum].d_type = INACTIVE;
                break;
        }
        /* ARGSUSED */
}
```

coord.l

```
/* $Header: /usr/src/local/midas/src/editor/RCS/coord.l,v 3.2 84/07/18 13:13:09 arnold
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 */

/*
 * To superimpose two molecules, the interactive program needs to
 * know the coordinates of various atoms.  These routines put
 * coordinate data in the 'cheating buffer' so that midas can
 * get at them
 */

static  int     cmol;
static  int     ccnt;
static  int     cnatm;
static  int     curmol;
static  char    *cptr;          /* ptr to next avail storage loc */
static  int     bufleft;
extern  char    *cheatbuf;

/*
 * setupcrd - sets up the variables needed by stcoords
 */
setupcrd(nmol, natm)
int     nmol, natm;
{
        register        int     *i;
        int     selres(), stcoord();

        bufleft = CHEATSZ / sizeof (int);
        i = (int *) cheatbuf;
        *i++ = nmol;
        cmol = 0;
        if (natm < 0)
                ccnt = 0;
        else
                ccnt = natm;
        curmol = -1;
        cnatm = natm;
        cptr = (char *) i;
        editres = selres;
        prepres = NULL;
        realedit = stcoord;
}

/*
 * stcoord - stores away the coordinate and set cmol to -1 on error
 */
stcoord(selected, index, atname)
int     selected, index;
char    *atname;
{
        register        float   *f;
        register        int     *i;

        if (!selected || cmol == -1)
                return;
        if (mol != curmol) {
                if (cnatm > 0 && ccnt != cnatm) {
                        cmol = -1;
                        return;
                }
                bufleft -= sizeof (int);
                if (bufleft < 0) {
                        cmol = -1;
                        return;
                }
                i = (int *) cptr;
                *i++ = mol;
                cptr = (char *) i;
                curmol = mol;
                cmol++;
                if (cnatm > 0)
                        ccnt = 0;
        }
        bufleft -= 3 * sizeof (float);
        if (bufleft < 0) {
                cmol = -1;
                reply("Communications buffer overflow.\n");
                return;
        }
        f = (float *) cptr;
        *f++ = atomdata[index].x;
        *f++ = atomdata[index].y;
        *f++ = atomdata[index].z;
        cptr = (char *) f;
        ccnt++;
        /* ARGSUSED */
}

/*
```

coord.i

```
 * chcoord - checks the coordinates to make sure everything's OK
 */
chcoord()
{
        if (ccnt == -cnatm) {
                *((int *) cheatbuf) = cmol;
                return(0);
        }
        if (cmol != *((int *) cheatbuf) || cnatm != ccnt) {
                *((int *) cheatbuf) = 0;
                return(-1);
        }
        return(0);
}

/*
 * chmatch - checks the coordinates to make sure everything's OK
 */
chmatch(s)
struct  spec_def            *s;
{
        struct  token_def           t;
        int     molno;
        int     *i;

        if (chcoord() == -1)
                return(-1);
        i = (int *) cheatbuf;
        t.t_value = s->s_model->sp_range->w_first;
        molno = matchmol(&t);
        if (molno != *(i+1))
                *i = - *i;
        return(0);
}
```

fixrot.i

```
/* $Header: fixrot.i,v 3.10 86/01/15 10:34:53 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                  13 May 1983
 * Release 2.1                  28 Jul 1983
 * Release 2.2                  23 Aug 1983
 * Release 2.3                  12 Sep 1983
 */

#define BMAIN   01              /* Backward mainchain */
#define FMAIN   02              /* Forward mainchain */
#define BSIDE   04              /* Backward sidechain */
#define FSIDE   010             /* Forward sidechain */
#define ALLFIX  0               /* Entire residue affected */

#define PI      3.14159

static  char    fixmode;        /* Mode of operation in dofix */
static  char    curon;          /* Current atom needs fixing ? */
static  int     aindex;         /* Index of atom with ROTBIT on */
static  char    *chfirst, *chlast;      /* Residue names of this/next chain
static  ps_t    fixmat[4][4];   /* Fixing rotation matrix */

/*
 * fixrot - fixes a rotation in the new orientation
 */
double
fixrot(n)
int     n;
{
        double  t;
        angle_t deg;
#if     GRAFENG == PS300
        ps_t    mat[4][4];
#endif
        register        char    *res, *at;
        struct  chain_def       *ch, *findchain();
        int     fixvisit(), fixagain();

        t = UNMAPANGLE(rotation[n].r_rotangle);
        if (t < 0.05 && t > -0.05)
                return(0.0);

        deg = rotation[n].r_initangle + rotation[n].r_rotangle;
        t = UNMAPANGLE(deg);

        if (t < 0)
                t += 360.;

#if     GRAFENG == PS2
        push();
        bldcon(LOADMAT, rotation[n].r_invmat);
        rot(rotation[n].r_rotangle, ZAXIS);
        bldcon(CONCATMAT, rotation[n].r_mat);
        bldcon(STOREMAT, fixmat);
        pop();
        errcheck(fixmat, 1);
#endif  PS2
#if     GRAFENG == MPS
        tpush();
        tset(rotation[n].r_invmat);
        trotz(rotation[n].r_rotangle);
        tcon(rotation[n].r_mat);
        tget(fixmat);
        tpop();
        errcheck(fixmat, 1);
#endif  MPS
#if GRAFENG == IRIS
        pushmatrix();
        loadmatrix(rotation[n].r_invmat);
        rotate(rotation[n].r_rotangle, 'z');
        multmatrix(rotation[n].r_mat);
        getmatrix(fixmat);
        popmatrix();
#endif
#if     GRAFENG == PS300
        getrot(fixmat, UNMAPANGLE(rotation[n].r_rotangle) / 180.0 * PI, ZAXIS);
        multmat(rotation[n].r_invmat, fixmat, mat, 1.0);
        multmat(mat, rotation[n].r_mat, fixmat, 1.0);
#endif

        mol = rotation[n].r_model;

        switch (rotation[n].r_natom) {

        case 2 :
                res = rotation[n].r_reseq[1];
                at = rotation[n].r_atom[1];
                break;

        case 4 :
                res = rotation[n].r_reseq[2];
                at = rotation[n].r_atom[2];
```

```
                break;

        default :
                fprintf(stderr, "Impossible condition in fixrot\n");
                return(-1.0);
        }
        ch = findchain(n);
        chfirst = ch->ch_start;
        if (ch->ch_next != NULL)
                chlast = ch->ch_next->ch_start;
        else
                chlast = NULL;

        (void) mseekr(mdb[mol].m_fd, res,
                (rotation[n].r_type & BACKWRD) ? BBSEEK : FBSEEK);
        natom = mreadr(mdb[mol].m_fd, resseq, restype);
#ifdef  DEBUG
        if (debug & D_EDIT)
                fprintf(stderr, "Fixing residue %s\n", resseq);
#endif  DEBUG
        (void) mreada(mdb[mol].m_fd);
        atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
        aindex = mseeka(mdb[mol].m_fd, at);

        if (rotation[n].r_type & MCHAIN) {
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "%s mainchain\n",
                                (rotation[n].r_type & BACKWRD) ?
                                "Backward" : "Forward");
#endif  DEBUG
                fixmode = (rotation[n].r_type & BACKWRD) ? BMAIN : FMAIN;
                (void) mtrav(mdb[mol].m_fd, fixvisit, fixagain);
                (void) mwrta(mdb[mol].m_fd);
                if (rotation[n].r_type & BACKWRD) {
                        if (strcmp(res, chfirst) != 0)
                                allfix(CBLSEEK);
                }
                else
                        allfix(CFLSEEK);
        }
        else {
                if (rotation[n].r_type & BACKWRD) {
#ifdef  DEBUG
                        if (debug & D_EDIT)
                                fprintf(stderr, "Backward sidechain\n");
#endif  DEBUG
                        fixmode = BSIDE;
                        (void) mtrav(mdb[mol].m_fd, fixvisit, fixagain);
                        (void) mwrta(mdb[mol].m_fd);
                        if (strcmp(res, chfirst) != 0) {
                                allfix(CBLSEEK);
                                (void) mseekr(mdb[mol].m_fd, res, FBSEEK);
                                natom = mreadr(mdb[mol].m_fd, resseq, restype);
                        }
                        allfix(CFLSEEK);
                } else {
#ifdef  DEBUG
                        if (debug & D_EDIT)
                                fprintf(stderr, "Forward sidechain\n");
#endif  DEBUG
                        fixmode = FSIDE;
                        (void) mtrav(mdb[mol].m_fd, fixvisit, fixagain);
                        (void) mwrta(mdb[mol].m_fd);
                }
        }
        return(t);
}

/*
 * allfix - fix everything from here to the end
 */
allfix(mode)
int     mode;
{
        int     fixvisit(), fixagain();

        fixmode = ALLFIX;
        while (mseekr(mdb[mol].m_fd, "*", mode) == 0) {
                if (mreadr(mdb[mol].m_fd, resseq, restype) <= 0)
                        break;
                if (tmatch(restype, "BOND") == 0)
                        continue;
                if (chlast != NULL && smatch(resseq, chlast) == 0)
                        break;
#ifdef  DEBUG
                if (debug & D_EDIT)
                        fprintf(stderr, "Fixing residue %s\n", resseq);
#endif  DEBUG
                (void) mreada(mdb[mol].m_fd);
                atomdata = (struct atom_def *) mdatptr(mdb[mol].m_fd);
                (void) mtrav(mdb[mol].m_fd, fixvisit, fixagain);
                (void) mwrta(mdb[mol].m_fd);
                if (mdb[mol].s_fd >= 0 &&
```

```
                        mseekr(mdb[mol].s_fd, resseq, FBSEEK) == 0) {
                                (void) mreada(mdb[mol].s_fd);
                                surfdata = (struct surf_def *)
                                        mdatptr(mdb[mol].s_fd);
                } else
                        surfdata = NULL;
                drawbond(mol, resseq, restype, atomdata);
                drawlabel(mol, resseq, restype, atomdata);
                drawsurf(mol, resseq, restype, atomdata, surfdata);
                if (smatch(resseq, chfirst) == 0)
                        break;
        }
}

/*
 * fixvisit - visiting routine for fixing rotations
 */
static
fixvisit(mfd, index, ischief, islinkage, nson, firstime)
int     mfd;
int     index;
int     ischief, islinkage, nson, firstime;
{
        ps_t    in[4], out[4];
#ifdef  DEBUG
        char    atname[AT_NAME_SIZE + 1];
#endif  DEBUG

        if (!firstime)
                return;
        if (ischief) {
                switch (fixmode) {

                case BMAIN :
                case BSIDE :
                case ALLFIX :
                        curon = TRUE;
                        break;

                case FMAIN :
                case FSIDE :
                        curon = FALSE;
                        break;
                }
        }
#ifdef  DEBUG
        if (debug & D_EDIT) {
                (void) matom(mfd, index, atname);
                fprintf(stderr, "%s%s fixed.\n",
                        atname, curon ? "" : " not");
        }
#endif  DEBUG
        if (!(atomdata[index].status & EXISTBIT))
                return;
        if (curon) {
                in[0] = MAPCRD(atomdata[index].x);
                in[1] = MAPCRD(atomdata[index].y);
                in[2] = MAPCRD(atomdata[index].z);
                in[3] = fixmat[3][3];
                if (atomdata[index].status & ONBITS) {
                        mdb[mol].m_com[0] -= atomdata[index].x;
                        mdb[mol].m_com[1] -= atomdata[index].y;
                        mdb[mol].m_com[2] -= atomdata[index].z;
                }
                transform(fixmat, in, out, fixmat[3][3]);
                atomdata[index].x = UNMAPCRD(out[0]);
                atomdata[index].y = UNMAPCRD(out[1]);
                atomdata[index].z = UNMAPCRD(out[2]);
                if (atomdata[index].status & ONBITS) {
                        mdb[mol].m_com[0] += atomdata[index].x;
                        mdb[mol].m_com[1] += atomdata[index].y;
                        mdb[mol].m_com[2] += atomdata[index].z;
                }
                checkrot(mfd, index, atomdata[index].x,
                        atomdata[index].y, atomdata[index].z);
        }
        if (index == aindex) {
                switch (fixmode) {

                case BMAIN :
                case BSIDE :
                        curon = FALSE;
                        break;

                case FMAIN :
                case FSIDE :
                        curon = TRUE;
                        break;

                default :
                        break;
                }
        }
        return;
```

```
	/* ARGSUSED */
}

/*
 * fixagain - revisiting routine for fixing rotations
 */
static
fixagain(db, index, ischief, islink, nsons)
int	db;
int	index, ischief, islink, nsons;
{
	if (index == aindex && nsons == 0) {
		if (fixmode == FSIDE)
			curon = FALSE;
		else if (fixmode == BSIDE)
			curon = TRUE;
	}
	return;
	/* ARGSUSED */
}

/*
 * checkrot - check if rotation matrices need updating
 */
checkrot(mfd, index, nx, ny, nz)
int	mfd, index;
float	nx, ny, nz;
{
	register		i, j;
	char	atname[AT_NAME_SIZE+1];

	for (i = 0; i < MAXINROT; i++) {
		if (rotation[i].r_type == INACTIVE)
			continue;
		if (rotation[i].r_model != mol)
			continue;
		for (j = 0; j < rotation[i].r_natom; j++) {
			if (smatch(resseq, rotation[i].r_resseq[j]))
				continue;
			(void) matom(mfd, index, atname);
			if (amatch(atname, rotation[i].r_atom[j]))
				continue;
			rotation[i].r_loc[j][0] = nx;
			rotation[i].r_loc[j][1] = ny;
			rotation[i].r_loc[j][2] = nz;
			rotation[i].r_type |= REMAKEMAT;
			break;
		}
	}
	return;
}

#if	GRAFENG == IRIS
/*
 * to_Angle:
 *	Return an angle number of type Angle from the given floating
 *	point number.  Angle's are integer values of 10ths of degrees.
 */
Angle
to_Angle(a)
ps_t	a;
{
	return (Angle) (a * 10.0 + 0.5);
}
#endif	IRIS
```

rotation.l

```
/* $Header: /usr/src/local/midas/src/editor/RCS/rotation.l,v 3.1 83/11/23 15:16:26 bin R
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 */

/*
 * The following code is used to test whether a rotation is legal or not.
 * The collected information is stored in the rotation structure.
 * r_natom = number of atoms selected (-1 on error)
 * If there are four atoms, they may be connected in one of three ways:
 * 1-2-3-4 or 2-1-3-4 or 4-1-2-3.  The pattern is kept track of in rotpat
 * and the data stored in the rotation structure is adjusted such that the
 * bond to be rotated is in the middle.
 */

#define  P1234   0
#define  P2134   1
#define  P4123   2

static   int     rotnum;
static   int     previndex[4];
static   int     waschief, waslink;
static   int     rotpat;

/*
 * setuprot - sets the global variables
 */
setuprot(n)
int      n;
{
         rotnum = n;
         rotation[rotnum].r_natom = 0;
         rotpat = P1234;
         editres = selres;
         prepres = NULL;
         realedit = editrot;
}

/*
 * editrot - does the real checking (see comment above as to what is a
 * legal rotation)
 */
editrot(selected, index, atname)
int      selected, index;
char     *atname;
{
         register        i, n;
         int     uncon;
         int     amchief, amlink;
         char    aname[AT_NAME_SIZE+1], rname[RES_SEQ_SIZE+1];
         double  sx, sy, sz;

         if (!selected)
                 return;
#ifdef   DEBUG
         if (debug & D_EDIT)
                 fprintf(stderr, "Rot - %s %d\n", atname,
                         rotation[rotnum].r_natom);
#endif
         if (rotation[rotnum].r_natom < 0)
                 return;
         n = rotation[rotnum].r_natom++;
         if (n >= 4)
                 return;
         amchief = mchief(mdb[mol].m_fd) == index;
         amlink = mlink(mdb[mol].m_fd) == index;
         if (n > 0) {
                 if (mol != rotation[rotnum].r_model) {
                         roterr();
                         return;
                 }
                 switch (mrconn(mdb[mol].m_fd, rotation[rotnum].r_resseq[n - 1],
                 resseq)) {

                 case -1 :
                 case 0  :
                 case 4  :
                 case 5  :
                         roterr();
                         return;

                 case 1  :
                         if (!waslink || !amchief) {
                                 roterr();
                                 return;
                         }
                         break;

                 case 2  :
                         if (!waschief || !amlink) {
                                 roterr();
```

```
                        return;
                    }
                    break;

                case 3 :
                    uncon = FALSE;
                    switch (n) {

                    case 1 :
                        i = maconn(mdb[mol].m_fd, index, previndex[0
                        if (i != 1 && i != 2)
                            uncon = TRUE;
                        break;

                    case 2 :
                        i = maconn(mdb[mol].m_fd, index, previndex[1
                        if (i == 1 || i == 2)
                            break;
                        i = maconn(mdb[mol].m_fd, index, previndex[0
                        if (i == 1 || i == 2) {
                            rotpat = P2134;
                            break;
                        }
                        uncon = TRUE;
                        break;

                    case 3 :
                        i = maconn(mdb[mol].m_fd, index, previndex[2
                        if (i == 1 || i == 2)
                            break;
                        i = maconn(mdb[mol].m_fd, index, previndex[0
                        if (i == 1 || i == 2) {
                            rotpat = P4123;
                            break;
                        }
                        uncon = TRUE;
                        break;
                    }
                    if (uncon) {
                        roterr();
                        return;
                    }
                    break;
            }
        } else
            rotation[rotnum].r_model = mol;
        strcpy(rotation[rotnum].r_resseq[n], resseq);
        strcpy(rotation[rotnum].r_atom[n], atname);
        rotation[rotnum].r_loc[n][0] = atomdata[index].x;
        rotation[rotnum].r_loc[n][1] = atomdata[index].y;
        rotation[rotnum].r_loc[n][2] = atomdata[index].z;
        previndex[n] = index;
        waschief = amchief;
        waslink = amlink;
        if (n == 3) {
            switch (rotpat) {

            case P1234 :
                break;

            case P2134 :
                strcpy(rname, rotation[rotnum].r_resseq[0]);
                strcpy(aname, rotation[rotnum].r_atom[0]);
                sx = rotation[rotnum].r_loc[0][0];
                sy = rotation[rotnum].r_loc[0][1];
                sz = rotation[rotnum].r_loc[0][2];
                rotshift(1, 0);
                strcpy(rotation[rotnum].r_resseq[1], rname);
                strcpy(rotation[rotnum].r_atom[1], aname);
                rotation[rotnum].r_loc[1][0] = sx;
                rotation[rotnum].r_loc[1][1] = sy;
                rotation[rotnum].r_loc[1][2] = sz;
                break;

            case P4123 :
                strcpy(rname, rotation[rotnum].r_resseq[0]);
                strcpy(aname, rotation[rotnum].r_atom[0]);
                sx = rotation[rotnum].r_loc[0][0];
                sy = rotation[rotnum].r_loc[0][1];
                sz = rotation[rotnum].r_loc[0][2];
                rotshift(3, 0);
                rotshift(2, 3);
                rotshift(1, 2);
                strcpy(rotation[rotnum].r_resseq[1], rname);
                strcpy(rotation[rotnum].r_atom[1], aname);
                rotation[rotnum].r_loc[1][0] = sx;
                rotation[rotnum].r_loc[1][1] = sy;
                rotation[rotnum].r_loc[1][2] = sz;
                break;
            }
        }
    }
}
/*
```

rotation.i

```
 * roterr - reports error in rotation specifications and marks the error
 */
roterr()
{
        rotation[rotnum].r_natom = -1;
        reply("Atoms are not connected.\n");
}

/*
 * rotshift - shifts rotation atom order (used by editrot)
 */
rotshift(from, to)
int     from, to;
{
        strcpy(rotation[rotnum].r_resseq[to], rotation[rotnum].r_resseq[from]);
        strcpy(rotation[rotnum].r_atom[to], rotation[rotnum].r_atom[from]);
        rotation[rotnum].r_loc[to][0] = rotation[rotnum].r_loc[from][0];
        rotation[rotnum].r_loc[to][1] = rotation[rotnum].r_loc[from][1];
        rotation[rotnum].r_loc[to][2] = rotation[rotnum].r_loc[from][2];
}
```

angle.i

```
/* $Header: /usr/src/local/midas/src/editor/RCS/angle.i,v 3.1 83/11/23 15:08:28 bin Rel
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 */

static    int       anglenum;

/*
 * setupang - check for angle specification validity
 */
setupang(n)
int       n;
{
          angles[n].a_type = INACTIVE;
          anglenum = n;
          angles[n].a_natom = 0;
          editres = selres;
          prepres = NULL;
          realedit = editang;
}

/*
 * editang - set up angle information and check for validity
 */
editang(selected, index, atname)
int       selected, index;
char      *atname;
{
          register            i;

          if (!selected || angles[anglenum].a_natom > 4)
                    return;
          i = angles[anglenum].a_natom++;
          switch (i) {

          case 2 :
                    angles[anglenum].a_type = ACTIVE;
                    /* Fall through */
          case 0 :
          case 1 :
          case 3 :
                    angles[anglenum].a_model[i] = mol;
                    strcpy(angles[anglenum].a_resseq[i], resseq);
                    strcpy(angles[anglenum].a_atom[i], atname);
                    break;

          case 4 :
                    angles[anglenum].a_type = INACTIVE;
                    break;
          }
          /* ARGSUSED */
}
```

struct.l

```
/* $Header: /usr/src/local/midas/src/editor/RCS/struct.l,v 3.1 83/11/23 15:18:08 bin Rel
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          5 Jul 1983
 * Release 2.1                          23 Aug 1983
 */

/*
 * setupadd - set up routines for addgrp command
 */
setupadd()
{
        editres = selres;
        prepres = NULL;
        realedit = editadd;
}

/*
 * editadd - call routine to insert selected atom into table
 */
editadd(selected, index, atname)
int     selected;
int     index;
char    *atname;
{
        if (selected)
                insertadd(mol, resseq, restype, &atomdata[index], atname);
}

/*
 * setupadd - set up routines for addaa, swapaa, swapnuc, etc. commands
 */
setupgetres()
{
        editres = selres;
        prepres = NULL;
        realedit = editgetres;
}

/*
 * editadd - call routine to insert selected residue into table
 */
editgetres(selected, index, atname)
int     selected;
int     index;
char    *atname;
{
        if (selected)
                insertgetres(mol, resseq, restype, &atomdata[index]);
        /* ARGSUSED */
}
```

/usr/src/local/midas/h/intr.h

```
/* $Header: intr.h,v 1.2 86/10/29 16:40:51 pett Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 2.0                          13 May 1983
 * Release 2.1                          26 Jul 1983
 * Release 2.2                          ?? Aug 1983
 */

#ifndef  vms

#include <mdbs.h>
#include <midas.h>

#if      GRAFENG == PS2
#include <ps.h>
typedef  int      fps_t;          /* This typedef is for backwards
                                  * compatibility for session files.
                                  * The type should really be ps_t
                                  * but the values in session files are
                                  * ints.  Ooops.  Conrad */
#endif   PS2
#if      GRAFENG == IRIS
#include <irisdefs.h>
typedef  ps_t     fps_t;
#endif   IRIS

#else    vms

#include "MIDAS$H:machdef.h"
#include "MIDAS$H:mdbs.h"
#include "MIDAS$H:midas.h"

#if      GRAFENG == MPS
#include "MIDAS$H:ps.h"
typedef  int      fps_t;          /* This typedef is for backwards
                                  * compatibility for session files.
                                  * The type should really be ps_t
                                  * but the values in session files are
                                  * ints.  Ooops.  Conrad */
#endif   MPS
#if      GRAFENG == IRIS
#include "MIDAS$H:irisdefs.h"
typedef  ps_t     fps_t;
#endif   IRIS
#endif   vms

/*
 * type for viewport coordinates.
 */

#if      GRAFENG == PS2 || GRAFENG == MPS
typedef int       vp_coord;
#endif
#if      GRAFENG == IRIS
typedef Scoord    vp_coord;
#endif

#if      GRAFENG == IRIS
typedef  double   knob_t;
#else    IRIS
typedef int       knob_t;
#endif   IRIS

#include <stdio.h>

/* #define        KDEBUG          /* display knob data during "set debug" */

#if      GRAFENG == PS2
#define  MATSIZE        18       /* MATCONR + 16 matrix + POPJ */
#endif   PS2
#if      GRAFENG == IRIS
#define  MATSIZE        16       /* 4x4 matrix */
#endif   IRIS
#if      GRAFENG == MPS
#define  MATCTRL        6        /* 6 control words */
#define  MATSIZE        (MATCTRL+18)    /* control + MATCONR + 16 matrix + POPJ */
#endif   MPS

#define  CURSCHR        '+'

#define  PI             3.1415926

#define  FRAMESIZE      5000
#define  GNOMONSIZE     50
#define  NMATRIX        (MAXINROT+MAXMOD+2)
#define  VNAMELEN       10       /* View name length */
#define  INACTIVE       0
#define  ACTIVE         01
#define  ISOBJ          01
#define  HASURF         02
#define  MODSEL         ((01 << MAXMOD) - 1)
#define  DEBUGSW        020000
```

/usr/src/local/midas/h/intr.h

```
#define  SDEBUGSW        040000
#define  RESETSW         0100000
#define  MAXSTACK        10          /* Input stack depth */
#define  MAXALIAS        20          /* Number of aliases */
# if     GRAFENG == MPS
#define  PVSIZE          1500        /* Pair view port size */
# endif
#define  PUBMODE         0644        /* Public file mode */
#define INBUFSIZE        256         /* Input buffer size */
#define  COMMENT         '#'

#ifndef  vms
#if GRAFENG != IRIS
#define  DOSTATS
#endif
#endif

#if      GRAFENG != IRIS
#define  INITCHSZ        3           /* Initial character size */
#define  INITHITHER      255         /* Initial hither plane intensity */
#define  INITYON         0           /* Initial yon plane intensity */
#define  INITSCALE       K16K        /* Initial scaling value (doubles window) */
#else    IRIS
#define  INITSCALE       ((ps_t) 2)
#define  INITHITHER      255
#define  INITYON         50
#endif   IRIS

#define  INITSPEED       10          /* Initial speed reduction */

#if      GRAFENG != IRIS
#define  WRATIO          1           /* Window width / displayed object width */
#define  DRATIO          3           /* Distance from screen / half-screen width */
#define  CUTOFF          -29000      /* Display extending beyond here is clipped */
#define  K_ANGLE         1           /* adjustment on angles read from knobs */
#define  K_DISTANCE      .25         /* adjustment on distance read from knobs */
#define  K_CLIP          1           /* adjustment on clipping distance */
#define  K_SCALE         2           /* adjustment on scale factors from knobs */
#else    IRIS
#define  WRATIO          1.0
#define  DRATIO          1.0
#define  K_ANGLE         2           /* adjustment on angles read from knobs */
#define  K_DISTANCE      .1667       /* adjustment on distance read from knobs */
#define  K_CLIP          .1667       /* adjustment on distance read from knobs */
#define  K_SCALE         .001667     /* adjustment on scale factors from knobs */
#endif   IRIS

#define  C_LEFT          0           /* Window and clipping plane indices */
#define  C_RIGHT         1
#define  C_BOTTOM        2
#define  C_TOP           3
#define  C_HITHER        4
#define  C_YON           5

#define  REASSIGN        01L         /* Settable flags */
#define  VERBOSE         02L
#define  STPAIR          04L
#define  ORTHO           010L
#define  DTEXT           020L
#define  INSTEREO        040L
#define  INDEP           0100L
#define  SHOWCOFG        0200L
#define  ONEBUF          0400L
#define  DLIGHTS         01000L
#define  RECORD          02000L
#define  DLABELS         04000L
#define  HALFBOND        010000L
#define  DDEBUG          020000L
#define  MMOVIE          040000L
#define  DCONTROL        0100000L    /* show control panel (IRIS) */
#define  XBOND           0200000L    /* show bond object (IRIS) */
#define  XLABEL          0400000L    /* show label object (IRIS) */
#define  XSURF           01000000L   /* show surface object (IRIS) */

#define  S_OPEN          1
#define  S_CLOSE         0
#define  S_EXPOSE        8           /* 8 frames for f5.6 and asa 160 */

/*
 * turn types for slow_down()
 */

#define  T_ROCK          0
#define  T_ANGLE         1
#define  T_SHIFT         2

#define  REPLY           repbuf[(nreply = (nreply % MAXREPLY + 1)) - 1]

#define  MODELSW(i)      (01 << (i))
#define  LINEWIDTH(i)    ((i - 3) * 2000)
#define  LINEHEIGHT(i)   (800 + 100 * (i))
#define  ABS(i)     (((i) < 0) ? -(i) : (i))

# if GRAFENG == IRIS
```

```
# define            COMMAND_Y       (textheight + 10)
# define            COMMAND_X       10
# endif  IRIS

long    lseek();
double  sin();
double  sqrt();
char    *strcpy(), *strcat(), *sprintf();

struct  model_def           {
        int         active;
        int         flags;
        int         nused;
        float       cofg[3];
        float       range;
        fps_t       com[3], tr[3];
        ps_t        matrix[4][4];
        char        molname[MOLNMSZ];
        char        altname[MOLNMSZ];
        char        surfname[MOLNMSZ];
};

struct  inrot_def           {
        int         r_stat;
        int         r_inita;
        angle_t     r_angle;
        char        r_label[60];
};

struct  dist_def            {
        int         d_stat;
#if     GRAFENG != IRIS
        unsigned            d_loc[2];
#else
        Matrix      d_mat[2];
        ps_t        d_x[2], d_y[2], d_z[2];
#endif
        ps_t        d_save[2][4];
        float       d_value;
        char        d_label[60];
};

struct  angle_def           {
        int         a_stat;
        int         a_natom;
#if     GRAFENG != IRIS
        unsigned            a_loc[4];
#else
        Matrix      a_mat[4];
        ps_t        a_x[4], a_y[4], a_z[4];
#endif
        ps_t        a_save[4][4];
        float       a_value;
        char        a_label[80];
};

struct  turn_def            {
        fps_t       t_speed;
        int         t_frames;
        int         t_wait;
};

struct  knob_def            {
        char        k_rot[3];
        char        k_tran[3];
        char        k_inrot[MAXINROT];
        char        k_clip[6];
        char        k_scale;
        char        k_section;
        char        k_thickness;
};

struct  move_def            {
        struct  turn_def        m_rot[3];
        struct  turn_def        m_rock[3];
        struct  turn_def        m_tran[3];
        struct  turn_def        m_clip[6];
        struct  turn_def        m_inrot[MAXINROT];
        struct  turn_def        m_scale;
        struct  turn_def        m_section;
        struct  turn_def        m_thickness;
};

struct  alias_def           {
        char        *a_name;                    /* Alias name */
        char        *a_line;                    /* Alias expansion */
};

#if     GRAFENG == MPS
struct  peobe   {
        struct  udl_preamble    preamble;
        struct  udl_entry       uentry[MAXPSOBS];
}       *peobe;
#endif
```

```
int     usermask;                       /* Original umask of user */
int     edit_fd[2];                     /* Pipe file descriptors */
int     edit_pid;                       /* Editor process id */
int     noeditor;                       /* Flag inhibiting spawn of editor */
fps_t   win[6], clip[6];                /* Window and clipping boundaries *
fps_t   wl, wr, wb, wt, wh, wy, weye;   /* Actual window parameters */
#if     GRAFENG == PS2 || GRAFENG == IRIS
char    *prompt;                        /* Current command prompt */
#endif
#if     GRAFENG == IRIS
vp_coord        vl, vr, vb, vt;         /* Viewport parameters */
Colorindex      vh, vy;                 /* Hither and yon intensities */
int     textheight, textwidth;          /* Width/height of regular (small) font */
int     normheight, normwidth;          /* Width/height of normal size font */
#else   IRIS
vp_coord        vl, vr, vb, vt, vh, vy; /* Viewport parameters */
#endif  IRIS
extern int      noknob;                 /* # knobs on system */
fps_t   scaleval;                       /* Scaling value */
fps_t   cx, cy, cz;                     /* Center of rotation */
#if     GRAFENG == IRIS
Colorindex      distcolor;              /* Color of lines for distances */
#else   IRIS
int     chsize;                         /* Label character size */
int     distcolor;                      /* Color of lines for distances */
#endif  IRIS
int     s_expose;                       /* Number of refreshes per frame */
char    setcofr;                        /* Flag indicating if cofr is set */
char    reduction;                      /* Knob value reduction factor */
char    nmol, nreply, nrot, ndist, nangle, nalias;      /* # of each type
char    repbuf[MAXREPLY][128];          /* Reply buffer */
char    combuf[INBUFSIZE];              /* Input command buffer */
char    lastcom[128];                   /* Last command typed in */
#if     GRAFENG != IRIS
char    usage[30];                      /* Usage message */
#endif
#ifdef  KDEBUG
char    knobuf[2][128];                 /* Debugging knob value buffer */
#endif
ps_t    matbuf[NMATRIX*MATSIZE];        /* PDP version of matrices */
#if     GRAFENG != IRIS
ps_t    frame[FRAMESIZE];               /* Rotation label (fixed protion) */
ps_t    gnomon[GNOMONSIZE];             /* Gnomon object buffer */
#endif  IRIS
ps_t    trans[3];                       /* Current amount of translation */
ps_t    cowin[3];                       /* Center of window */
ps_t    rightmat[4][4], leftmat[4][4];  /* Left and right stereo matrix */
FILE    *eo;                            /* Stdio editor output descriptor */
FILE    *recordfd;                      /* Record file descriptor */
float   rockpos[3];                     /* Extent of rocking */
float   com[3];                         /* Center of original window */
float   width;                          /* Width of original window */
unsigned        molrot[MAXMOD];         /* Global rotation matrix ...
                                         * location in matbuf */
unsigned        inrot[MAXINROT];        /* Internal rotation matrix ...
                                         * position in matbuf */
unsigned        genrot[2];              /* General working matrix ...
                                         * location in matbuf */
unsigned        switches;               /* Current switch value */
unsigned        selection;              /* Actual model selection */
long    flags;                          /* Various on/off flags */
struct  model_def       models[MAXMOD]; /* Molecule information */
struct  inrot_def       rotation[MAXINROT];     /* Int. rot. information */
struct  dist_def        distance[MAXDIST];      /* Dist. calc. information */
struct  angle_def       angles[MAXANGLE];       /* Angle calc. information */
struct  knob_def        knobs;          /* Knob assignment information */
struct  move_def        movecoms;       /* Keyboard movement command info */
struct  alias_def       atab[MAXALIAS]; /* Alias definition table */
#if     GRAFENG == MPS
int     arg0, arg1, arg2;               /* temporaries for argument passing */
#define PSMEMFULL       0x18a           /* mpsgsp errno for memory overflow */
int     psmemfull;                      /* non-zero when refresh mem overflows */
#endif

/*
 * This structure depends on which variables
 * must be saved for reconstructing the current view.
 */
struct  view_def        {
        ps_t    v_matrix[MAXMOD][4][4];
        fps_t   v_tr[MAXMOD][3], v_cofg[MAXMOD][3];
        int     v_angle[MAXINROT];
        fps_t   v_win[6], v_clip[6];
        fps_t   v_scaleval;
#if     GRAFENG != IRIS
        int     v_chsize;
        int     v_distcolor;
#else   IRIS
        Colorindex      v_distcolor;
#endif  IRIS
        float   v_rockpos[3];
        float   v_com[3];
        float   v_width;
```

```
        int     v_setcofr;
        fps_t   v_cx, v_cy, v_cz;
};

struct  vstck_def       {
        struct  vstck_def       *vs_next;
        struct  view_def        *vs_view;
};

struct  vheap_def       {
        struct  vheap_def       *vh_next;
        struct  view_def        *vh_view;
        char    vh_name[VNAMELEN];
};

struct  vstck_def       *viewstck;              /* View stack */
struct  vheap_def       *viewheap;              /* Named view heap */

int     slow_fac;                               /* slowdown factor */

knob_t  getknob(), readknob();

#if     GRAFENG == IRIS

Angle   rot_angle();

char    *knob_str();

# define        NOSWITCH        16

#endif  IRIS
```

midas.c

```
/* $Header: midas.c,v 3.43 86/11/05 13:10:51 pett Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          26 Jul 1983
 * Release 2.2                          28 Aug 1983
 */

#include "intr.h"

#if     GRAFENG != IRIS
char    id[]    =       "MIDAS INTERACTIVE MODULE";
#endif

#if     GRAFENG == PS2
#include <sys/ioctl.h>
#include <ps2.h>
#endif  PS2

#if     GRAFENG == IRIS
#include <device.h>

extern int          GI_r_coms;
extern int          GI_w_resp;
#endif  IRIS

#include <signal.h>

#ifdef  vms
#define index   strchr
#else
#if     GRAFENG != IRIS
#include <sys/local.h>
#endif  IRIS
#endif

#if     GRAFENG == MPS
int     extdmem = FALSE;        /* default is to use non-refresh partition */

mpserr(errbuf)
short *errbuf;
{
        static int recall = 0;

        if (errbuf[1] == PSMEMFULL) {
                psmemfull = 1;      /* refresh memory overflowed */
                return;
        }
        if (recall)
                goto bad;           /* recursive call */
        recall++;
        eprint(errbuf);
        fprintf(stderr, "\n MODULE: %s\n", id);
        fprintf(stderr, " Aborting...\n");
        mpstop();
bad:
        abort();
}
#endif  MPS

extern int          _cursor_pos;

/*
 *      MIDAS - Molecular Interactive Display And Simulation
 *
 *      Main program - starts up the editor, initialize variables
 *      and executes command until user quits.
 */
main(argc, argv)
int     argc;
char    **argv;
{
        register int        i;
#if     GRAFENG != IRIS
        register int        plusflag;
#endif
        int         count, matnam, chrstat, p[2];
        int         samepic;
        int         seenum;
        int         offset, pick_pos;
        char        session[80];
        char        *startup, rc[80];
        short       magic;
#if     GRAFENG == PS2
        struct      psobs   psb[MAXPSOBS];
#endif
#if     GRAFENG == MPS
        int         matloc;
#endif
#if     GRAFENG == IRIS
        register long       q;
```

```
        static short        input, x, y;
#endif
        extern int          _pekblen;
#if     GRAFENG != IRIS
        extern int          _ix, _iy;
#endif
        int         stop();
        char        *getenv(), *index();

        /* Check for arguments.  Only one argument allowed and must be
         * a saved session file. */
        sesnum = -1;
        count = 0;
        for (i = 1; i < argc; i++) {
                if (argv[i][0] != '-') {
                        sesnum = i;
                        count++;
                }
                else if (argv[i][1] == 'm') {
                        execl(MENUPROG, "midas.menu", 0);
                        perror(MENUPROG);
                }
#if     GRAFENG == MPS
                else if (argv[i][1] == 'x')
                        extdmem = TRUE;
#endif  MPS
        }
        switch (count) {
                case 0 :
                        break;
                case 1 :
                        strcpy(session, argv[sesnum]);
                        i = strlen(MSESSION);
                        count = strlen(session);
                        if (count > i) {
                                if (strncmp(&session[count-i], MSESSION, i))
                                        strcat(session, MSESSION);
                        } else
                                strcat(session, MSESSION);
                        i = open(session, 0);
                        if (i < 0) {
                                perror(session);
                                exit(1);
                        }
                        read(i, (char *) &magic, sizeof magic);
                        if (magic != SES_MAGIC) {
                                fprintf(stderr,
                                        "%s: Not a MIDAS session file.\n",
                                        session);
                                exit(1);
                        }
                        close(i);
                        zapchar(session, '.');
                        break;
                default :
                        fprintf(stderr,
                                "Usage: midas [-flags] [session_filename]\n");
                        exit(1);
        }

        /* Initialize the Picture System variables, knobs, etc. */
        setupps();
        vh = INITHITHER;
        vy = INITYON;
#if     GRAFENG == PS2
        i = open("/dev/ps.map", 0);
        if (i < 0) {
                printf("Waiting for Picture System.\n");
                while (open("/dev/ps.map", 0) < 0)
                        sleep(5);
                printf("Ready.\n");
        }
        close(i);
        psinit();
        BUFRPS(256);
        setps((psaddr_t) MAXPSMEM, MAXPSOBS, psb);
#endif  PS2
#if     GRAFENG == MPS
        /*
         * For the MPS we are very careful what we do here,
         * since the child process (MIDAS EDITOR) has not
         * yet been started and it will do an mpinit() as
         * well.  Basically we only do things that affect
         * variables in our address space and not the state
         * of the MPS hardware (other than very basic
         * initialization).
         */
        if (extdmem) {
                mpinit(&0, &1);
                /*
                 * The following is the only call to ucontx().
                 * This means that we are always in the MPSCTX
                 * context and hence never need to recall ucontx().
                 */
```

```
                ucontx(&MAXPSOBS, psobs, &MPSCTX);
        } else {
                mpinit(&NRMEMSZ);
                ucontx(&MAXPSOBS, psobs, &NORFCTX);     /* see comment
        }
        einit(mpserr);

        /*
         * The PS2 software maintains "lenp's" relative to
         * the beginning of PS memory, while the MPS software
         * maintains lenp's relative to the beginning of
         * each object.  We account for this difference below.
         */
        matloc = MATCTRL + 1;       /* loc of first data word in matbuf */
#endif  MPS
#if     GRAFENG == IRIS
        ginit();
        chunksize(252);
#ifndef NO_BUTTONBOX
        dbtext("");             /* the first time anyone uses the box after startup, */
        dbtext("");             /* the first two dbtext()s only bring it on line */
        dbtext("UC MIDAS");
        setdblights(0L);
#endif
        doublebuffer();
        gconfig();
        choose_colors();
        ir_color(BLACK);
        clear();
        iris_colors();
        swapbuffers();
        font(NORMFONT);
        normheight = getheight() + 2;
        normwidth = strwidth(" ");
        font(SMALLFONT);
        textheight = getheight() + 1;
        textwidth = strwidth(" ");
        curson();
        make_control();
#endif  IRIS

#if     GRAFENG == PS2 || GRAFENG == IRIS
        make_prompt("Command: ");
#endif

        /*
         * Set up the initial matrices as identity matrices.
         */
#if     GRAFENG == IRIS
        count = 0;
#endif  IRIS
        for (i = 0; i < MAXMOD; i++) {
                _imatx(models[i].matrix);
                matnam = ('m' << 8) | ('0' + i);
#if     GRAFENG == PS2
                maksps(matnam, &count);
                molrot[i] = count + 1;
                tran(0, 0, 0);
                stopps();
                matbuf[i*MATSIZE] = MATCONR;
                matbuf[i*MATSIZE + MATSIZE - 1] = POPJ;
                _imatx(&matbuf[molrot[i]]);
#endif  PS2
#if     GRAFENG == IRIS
                makeobj((Object) matnam);
                translate((Coord) 0, (Coord) 0, (Coord) 0);
                closeobj();
                molrot[i] = count++ * MATSIZE;
                _imatx(&matbuf[molrot[i]]);
#endif  IRIS
#if     GRAFENG == MPS
                ubegin(&matnam, &count);
                molrot[i] = matloc;
                ttran(&0, &0, &0);
                matloc += MATSIZE;
                uend();
#endif  MPS
        }
        for (i = 0; i < MAXINROT; i++) {
                matnam = ('r' << 8) | ('0' + i);
#if     GRAFENG == PS2
                maksps(matnam, &count);
                inrot[i] = count + 1;
                tran(0, 0, 0);
                stopps();
                matbuf[(MAXMOD+i)*MATSIZE] = MATCONR;
                matbuf[(MAXMOD+i)*MATSIZE + MATSIZE-1] = POPJ;
                _imatx(&matbuf[inrot[i]]);
#endif  PS2
#if     GRAFENG == IRIS
                makeobj((Object) matnam);
                translate((Coord) 0, (Coord) 0, (Coord) 0);
                closeobj();
                inrot[i] = count++ * MATSIZE;
```

```
                _imatx(&matbuf[inrot[i]]);
#endif  IRIS
#if     GRAFENG == MPS
                ubegin(&matnam, &count);
                inrot[i] = matloc;
                ttran(&0, &0, &0);
                matloc += MATSIZE;
                uend();
#endif  MPS
        }
        for (i = 0; i < 2; i++) {
                matnam = ('g' << 8) | ('0' + i);
#if     GRAFENG == PS2
                makeps(matnam, &count);
                genrot[i] = count + 1;
                tran(0, 0, 0);
                stopps();
                matbuf[(MAXMOD+MAXINROT+i)*MATSIZE] = MATCONR;
                matbuf[(MAXMOD+MAXINROT+i)*MATSIZE + MATSIZE-1] = PO
                _imatx(&matbuf[genrot[i]]);
#endif  PS2
#if     GRAFENG == IRIS
                makeobj((Object) matnam);
                translate((Coord) 0, (Coord) 0, (Coord) 0);
                closeobj();
                genrot[i] = count++ * MATSIZE;
                _imatx(&matbuf[genrot[i]]);
#endif  IRIS
#if     GRAFENG == MPS
                ubegin(&matnam, &count);
                genrot[i] = matloc;
                ttran(&0, &0, &0);
                matloc += MATSIZE;
                uend();
#endif  MPS
        }

#if     GRAFENG == PS2
        makeps('pc', &count);
        entryps('bd');
        tran(0, 0, 0);                          /* Make sure there's something ther
        stoppe();
        huesat(WHITE);

        /* Prepare the left and right matrices for stereo */
        getrot(leftmat, -546, YAXIS);
        getrot(rightmat, 546, YAXIS);
#endif  PS2
#if     GRAFENG == IRIS
        /* Prepare the left and right matrices for stereo */
        getrot(leftmat, -3.0, YAXIS);
        getrot(rightmat, 3.0, YAXIS);
#endif  IRIS
#if     GRAFENG == MPS
        /* now read the matrices back to initialize the incore copy */
        arg0 = psobs->preamble.u_min + 1;       /* loc of mats in PS memory */
        arg1 = sizeof(matbuf) / sizeof(ps_t);   /* amount to read */
        uread(&arg0, &arg1, matbuf);

        ubegin(&'pc', &count);
        ttran(&0, &0, &0);
        uend();
        ubegin(&'bd', &count);
        ttran(&0, &0, &0);
        uend();

        /* Prepare the left and right matrices for stereo */
        getrot(leftmat, &-546, &YAXIS);
        getrot(rightmat, &546, &YAXIS);
#endif  MPS

        /* Start up the editor which will handle the database end of things. */
        putps();                /* save UDL context for editor */
#if     GRAFENG == IRIS
        gflush();
#endif  IRIS
        if (noeditor) {
                /* Don't start editor process */
                edit_fd[0] = -1;
                edit_fd[1] = 1;
                goto noedit;
        }
        if (pipe(edit_fd) < 0 || pipe(p) < 0) {
                fprintf(stderr, "No pipes available\n");
                exit(-1);
        }
#ifndef vms
        if ((edit_pid = vfork()) == -1) {
                fprintf(stderr, "No more processes\n");
                exit(-1);
        }
        if (edit_pid == 0) {
#if     GRAFENG != IRIS
                extern  int _cdfid;
```

```
                extern  int _fsfid;
                extern  int _kbfid;
                extern  int _tabfid;
                extern  int _siafid;
                extern  int _psiofd;

                /* Close off unnecessary files */
                if (_cdfid != -1)
                        close(_cdfid);
                if (_fsfid != -1)
                        close(_fsfid);
                if (_kbfid != -1)
                        close(_kbfid);
                if (_tabfid != -1)
                        close(_tabfid);
                if (_siafid != -1)
                        close(_siafid);

                if (_psiofd != -1)
                        close(_psiofd);
#endif  IRIS

                /* Create the pipe */
                close(0);
                if (dup(edit_fd[0]) != 0)
                        fprintf(stderr, "Dup 0 failed\n");
                close(edit_fd[0]);
                close(edit_fd[1]);
#if     GRAFENG == IRIS
                if (dup2(p[1], IRIS_STDOUT) == -1)
#else   IRIS
                if (dup2(p[1], 1) == -1)
#endif  IRIS
                        fprintf(stderr, "Dup 1 failed\n");
                close(p[0]);
                close(p[1]);

                /* Fire up the editor */
                argv[0] = "EDITOR";
                if (seenum != -1) {
#if     GRAFENG != IRIS
                        argv[seenum] = "-?v";
#else   IRIS
                        argv[seenum] = "-v";
#endif  IRIS
                        execv(EDITOR, argv);
                } else if (argc == 1) {
#if     GRAFENG != IRIS
                        execl(EDITOR, "EDITOR", "-?v", 0);
#else   IRIS
                        execl(EDITOR, "EDITOR", "-v", 0);
#endif  IRIS
                } else {
                        char    buf[80];

                        strcpy(buf, argv[1]);
#if     GRAFENG != IRIS
                        strcat(buf, "?v");
#else   IRIS
                        strcat(buf, "v");
#endif  IRIS
                        argv[1] = buf;
                        execv(EDITOR, argv);
                }
                perror(EDITOR);
                exit(1);
        }
        close(edit_fd[0]);
        dup2(p[0], edit_fd[0]);
        close(p[0]);
        close(p[1]);
#else   vms
        /*
         * Setup standard pipe enviroment.
         * This is somewhat messy in VMS since
         * close and dup2 are not generic system
         * calls and hence cannot be done after
         * a vfork but before the execl, since
         * variables in the parent's address
         * space are then modified.
         */
        dup2(edit_fd[0], FROM_PARENT);
        close(edit_fd[0]);
        edit_fd[0] = FROM_PARENT;
        dup2(edit_fd[1], TO_CHILD);
        close(edit_fd[1]);
        edit_fd[1] = TO_CHILD;

        dup2(p[0], FROM_CHILD);
        close(p[0]);
        p[0] = FROM_CHILD;
        dup2(p[1], TO_PARENT);
        close(p[1]);
        p[1] = TO_PARENT;
```

midas.c

```
        if ((edit_pid = vfork()) == -1) {
                fprintf(stderr, "MIDAS: can't create subprocess\n");
                exit(1);
        }
        if (edit_pid == 0) {
                /* Fire up the editor */

                startup = EDITOR;
                if (access(startup, 05)) {
                        fprintf(stderr, "MIDAS: can't spawn editor\n");
                        perror(startup);
                        exit(1);
                }
                argv[0] = "EDITOR";
                if (sesnum != -1) {
                        argv[sesnum] = extdmem ? "-?vx":"-?v";
                        execv(startup, argv);
                } else if (argc == 1) {
                        execl(startup, argv[0], extdmem ? "-?vx":"-?v", 0);
                } else {
                        char    buf[80];

                        strcpy(buf, argv[1]);
                        strcat(buf, extdmem ? "?vx":"?v");
                        argv[1] = buf;
                        execv(startup, argv);
                }
                perror(startup);
                exit(1);
        }
        close(edit_fd[0]);
        edit_fd[0] = FROM_CHILD;
        close(p[1]);
#endif  vms

        if ((ec = fdopen(edit_fd[0], "r")) == NULL) {
                fprintf(stderr, "Out of file descriptors\n");
                stop();
                /* NOTREACHED */
        }
#ifdef  vms
        /* The first read on a pipe under vms always returns NULL
         * so we consume it here before anything gets messed up */
        fgets(rc, sizeof rc, ec);
#endif  vms
        if ((startup = fgets(rc, sizeof rc, ec)) == NULL || strcmp(rc, "SYNC\n")) {
                fprintf(stderr, "Out of sync with editor\n");
                if (startup == NULL)
                        fprintf(stderr, "(EOF on editor pipe)\n");
                fprintf(stderr, "Editor = \"%s\"\n", EDITOR);
                exit(1);
        }
noedit:
        if (signal(SIGINT, stop) != SIG_DFL)
                signal(SIGINT, SIG_IGN);
        signal(SIGTERM, stop);

#if     GRAFENG == MPS
        /* Finish MPS initialization */
        getps();                /* back to our context */
        sdbuff();
        lcolor(&WHITE);
#endif  MPS
#if     GRAFENG == IRIS
        /*
         * We have to send the color mappping down to the editor before
         * anything else can happen.
         */

        {
                extern int      c_range, c_factor;
                char            buf[80];

                sprintf(buf, "mapcolors %d %d", c_range, c_factor);
                sendcom(buf);
        }
#endif

        /* Create the display frame */
        makeframe();

#ifdef  FILLDISK
        fakefile(TMPDIR(fakefile), "w");
#endif  FILLDISK
#ifdef  PLAYBACK
        fakefile(TMPDIR(fakefile), "r");
#endif  PLAYBACK

        /* Set up the record files */
        usermask = umask(0);
        recordfd = fopen(RECORDFILE, "w+");
        if (recordfd == NULL) {
                perror(RECORDFILE);
```

```
                stop();
        }
#ifdef  SIGIO
        setlinebuf(recordfd);
#endif  SIGIO

        /* Initialize all variables */
        for (i = 0; i < 3; i++)
                knobs.k_rot[i] = knobs.k_tran[i] = noknob;
        for (i = 0; i < 6; i++)
                knobs.k_clip[i] = noknob;
        for (i = 0; i < MAXINROT; i++)
                knobs.k_inrot[i] = noknob;
        knobs.k_scale = noknob;
        knobs.k_section = noknob;
        knobs.k_thickness = noknob;
#ifdef  PLAYBACK
        getfake();
#endif  PLAYBACK
        switches = fswitch();
        getknob(0, 3);
#ifdef  FILLDISK
        putfake();
#endif
#if     GRAFENG != IRIS
        chsize = INITCHSZ;
#endif  IRIS
        reduction = INITSPEED;
        selection = switches;
        distcolor = YELLOW;
        scaleval = INITSCALE;

        /* Read the startup files */
        startup = getenv("MIDASRC");
        if (startup == NULL) {
                sprintf(rc, "source %s\n", MIDASRC);
                procom(rc);
        } else if (access(startup, 04) == 0) {
                sprintf(rc, "source %s\n", startup);
                procom(rc);
        }

        if (access(MENURC, 04) == 0) {
                strcpy(rc, "source ");
                strcat(rc, MENURC);
                strcat(rc, "\n");
                procom(rc);
                unlink(MENURC);
        }

        /* Read the user startup file(s) */
        startup = getenv("HOME");
        if (startup != NULL) {
                strcpy(rc, startup);
#ifndef vms
                strcat(rc, "/.midasrc");
#else
                strcat(rc, "midas.rc");
#endif
                if (access(rc, 04) == 0) {
                        sprintf(REPLY, "Reading '%s'\n", rc);
#ifndef vms
                        sprintf(rc, "source %s/.midasrc\n", startup);
#else
                        sprintf(rc, "source %smidas.rc\n", startup);
#endif
                        procom(rc);
                }
        }
#ifndef vms
        if (access(".midasrc", 04) == 0) {
                strcpy(REPLY, "Reading '.midasrc'\n");
                procom("source .midasrc\n");
        }
#else
        if (access("midas.rc", 04) == 0) {
                strcpy(REPLY, "Reading 'midas.rc'\n");
                procom("source midas.rc\n");
        }
#endif

        /* If a session was specified, read it in */
        procom("savepos");
        nreply = 0;
        if (seenum != -1) {
                strcpy(REPLY, "Restarting session.");
                drawem(TRUE);
                init(session);
        }
        if (flags & DLIGHTS)
                lights((int) selection);
#if     GRAFENG != IRIS
        rtp(1);
#endif  IRIS
```

midas.c

```
        calcent(selection);

        /* Loop until user quits */
        samepic = FALSE;
#if     GRAFENG != IRIS
        plusflag = 0;
#endif
        for (;;) {
#ifdef  PLAYBACK
                getfake();
#endif  PLAYBACK
                if (!samepic)
                        drawem(TRUE);
                else {
#if     GRAFENG != IRIS
                        rstfram();              /* No redraw unless ... */
#else   IRIS
                        ;
#endif  IRIS
#if     GRAFENG == PS2
                        if (flags & MMOVIE) {
                                shutter(S_OPEN);
                                expose(s_expose);
                                shutter(S_CLOSE);
                        }
#endif  PS2
                }

                chrstat = getchr(combuf, TRUE);
#if     GRAFENG == IRIS
                check_pick();
#endif
                if (chrstat > 0) {              /* Typed a complete line ! */
                        nreply = 0;
                        strcpy(REPLY, "Executing ...");
                        drawem(TRUE);

#if     GRAFENG != IRIS
                        rtp(0);
#endif  IRIS
                        nreply = 0;
                        fixtran(selection);
                        calcent(selection);
                        procom(combuf);
                        strcpy(lastcom, combuf);
                        if (flags & RECORD) {
                                fprintf(recordfd, "%s\n", combuf);
                                fflush(recordfd);
                        }
#if     GRAFENG != IRIS
                        plusflag = (index(combuf, CURSCHR) != NULL);
                        if (plusflag == 0)
                                autocur(0);
                        rtp(1);
#endif  IRIS
                }
#if     GRAFENG != IRIS
                else if (chrstat < 0) {
                        i = (index(combuf, CURSCHR) != NULL);
                        if (!plusflag && i)
                                autocur(1);
                        else if (plusflag && !i)
                                autocur(0);
                        plusflag = i;
                }
                if (plusflag) {
                        tablet();
                        if (ispchd() == PIDWU) {
                                rtp(0);
                                offset = strlen(combuf);
                                pick_pos = (int) (index(combuf,CURSCHR)-combuf);
                                dopick(combuf, _ix, _iy, sizeof combuf);
                                _pskblen = strlen(combuf);
                                if (pick_pos <= _cursor_pos)
                                        _cursor_pos += _pskblen - offset;
                                plusflag = (index(combuf, CURSCHR) != NULL);
                                if (plusflag == 0)
                                        autocur(0);
                                rtp(1);
                                chrstat = -1;
                        }
                }
#endif  IRIS
                samepic = (!movem()) && (chrstat == 0);
#ifdef  FILLDISK
                putfake();
#endif  FILLDISK
        }
}

#include <ctype.h>

dopick(s, x, y, s_size)
char    *s;
```

midas.c

```
int     x, y, s_size;
{
        register char       *cp, *s_pntr;
        ps_t                eye;
        char                buf[BUFSIZ];
        extern char         *index();
        int                 offset, old_len;
#if     GRAFENG == IRIS
# define        NUM_NAMES       6L

        register unsigned long      num;
        static unsigned short       namebuf[NUM_NAMES];
#endif

        if ((flags & (INSTEREO | STPAIR)) == STPAIR) {
                strcpy(REPLY, "Cannot pick in pair mode.\n");
                return;
        }

        /*
         * Load in the transformation since stereo puts the wrong one in
         */

        /* Position the eye of the window */
        eye = (flags & ORTHO) ? wh : weye;

#if     GRAFENG == PS2
        unit();
        vwport(vl, vr, vb, vt, vh, vy);
        window(wl, wr, wb, wt, wh, wy, eye);
        scale(K16K, K16K, -K16K, K16K);
        tran(-cowin[0], -cowin[1], -cowin[2]);
#endif  PS2
#if     GRAFENG == MPS
        tident();
        vbound(&vl, &vr, &vb, &vt);
        arg0=(vh>>2); arg1=(vy>>2);
        vinten(&arg0, &arg1);
        if (flags & ORTHO)
                twind(&wl, &wr, &wb, &wt, &wh, &wy);
        else
                twindp(&wl, &wr, &wb, &wt, &wh, &wy, &weye);
        tscale(&K16K, &K16K, &-K16K, &K16K);
        arg0 = -cowin[0]; arg1 = -cowin[1]; arg2 = -cowin[2];
        ttran(&arg0, &arg1, &arg2);
#endif  MPS
#if     GRAFENG == IRIS
        if (wl == wr)
                return;

        depthcue(FALSE);            /* picking won't work in depthcue, so be sure */
        pushmatrix();
        pushviewport();
        /*
         * the viewport command must be done before the pick; it doesn't
         * seem to matter that it is done again after in ir_showmat().
         */
        viewport(vl, vr, vb, vt);
        pick(namebuf, NUM_NAMES);
        ir_showmat((ps_t *) NULL, vl, vr, vb, vt, (flags & ORTHO) ? wh : weye);
        callobj(O_PNTS);
        nreply = endpick(namebuf);
        popviewport();
        popmatrix();
        if (nreply != 1)
                return;

        num = namebuf[2] | (((unsigned long) namebuf[3]) << 16);
        sprintf(repbuf[0], "#%d:%s@", namebuf[1], ra_num_to_name(num));
        strcat(repbuf[0], ra_num_to_name(namebuf[4]));
        nreply = 0;
#else   IRIS
        nreply = 0;
        sprintf(buf, "pick %d %d\n", x, y);
        sendcom(buf);
        if (repbuf[0][0] != '#')                    /* Did not return an atom specifier */
                return;
#endif
        old_len = strlen(s);
        offset = strlen(repbuf) - 1; /* less size of CURSCHR */
        if (old_len + offset >= s_size || offset < 0)
                return;
        cp = index(s, CURSCHR);
        for (s_pntr = s + old_len; s_pntr > cp; s_pntr--)
                *(s_pntr + offset) = *s_pntr;
        strncpy(cp, repbuf, offset + 1);
}

#if     GRAFENG == PS2 || GRAFENG == IRIS
/*
 * make_prompt:
 *      Set the current prompt to be the given one.
 */
make_prompt(str)
```

midas.c

```
char    *str;
{
        prompt = str;
#if     GRAFENG == IRIS
        make_command();
#endif
}
#endif

#ifdef  notdef
dumpmats()
{
        int     i;
        ps_t    *mp;

        mp = matbuf;
        for (i = 0; i < NMATRIX * MATSIZE; ) {
                fprintf(stderr, "%o\t", (*mp++) & 0xFFFF);
                if (++i % 10 == 0)
                        fprintf(stderr, "\n");
        }
        fprintf(stderr, "\n");
}
#endif  notdef
```

init.c

```
/* $Header: init.c,v 3.12 86/07/10 18:40:37 tef Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          19 Jul 1983
 */

#include "intr.h"

#if     GRAFENG == IRIS
#include        <device.h>
#endif

/*
 * Init - initialize MIDAS from a session file.
 */
init(file)
char    *file;
{
        register int        i;
        char    fname[80], *cp;
        char    buf[80];
        static  char    *msg    =       "Restarting session ...";
        int     len;
        FILE    *fd;
        struct  model_def       l_models[MAXMOD];
        struct  inrot_def       l_rotation[MAXINROT];
        struct  dist_def        l_distance[MAXDIST];
        struct  angle_def       l_angles[MAXANGLE];
        struct  vheap_def       *vheap;
        struct  vstck_def       *vstck;

        /* Put out the message */
        nreply = 0;
        strcpy(REPLY, msg);
        drawem(TRUE);

        /* Read in all the information from the session file */
        strcpy(fname, file);
        cp = &fname[strlen(fname)];
        strcpy(cp, MSESSION);
        fd = fopen(fname, "r");                 /* already know file is readable */
        fseek(fd, 2L, 0);                       /* Magic number */
        fread((char *) l_models, sizeof l_models, 1, fd);
        fread((char *) l_rotation, sizeof l_rotation, 1, fd);
        fread((char *) l_distance, sizeof l_distance, 1, fd);
        fread((char *) l_angles, sizeof l_angles, 1, fd);
        fread((char *) &knobs, sizeof knobs, 1, fd);
        fread((char *) &movecoms, sizeof movecoms, 1, fd);
        fread((char *) &reduction, sizeof reduction, 1, fd);
        fread((char *) &vh, sizeof vh, 1, fd);
        fread((char *) &vy, sizeof vy, 1, fd);
        fread((char *) &flags, sizeof flags, 1, fd);
        fread((char *) &selection, sizeof selection, 1, fd);

        fread((char *) &len, sizeof len, 1, fd);
        vheap = NULL;
        for (i = 0; i < len; i++) {
                if (vheap == NULL) {
                        vheap = (struct vheap_def *)
                                malloc(sizeof (struct vheap_def));
                        viewheap = vheap;
                } else {
                        vheap->vh_next = (struct vheap_def *)
                                malloc(sizeof (struct vheap_def));
                        vheap = vheap->vh_next;
                }
                vheap->vh_view = (struct view_def *)
                        malloc(sizeof (struct view_def));
                vheap->vh_next = NULL;
                fread((char *) vheap->vh_name, VNAMELEN, 1, fd);
                fread((char *) vheap->vh_view, sizeof (struct view_def), 1, fd);
        }

        fread((char *) &len, sizeof len, 1, fd);
        vstck = NULL;
        for (i = 0; i < len; i++) {
                if (vstck == NULL) {
                        vstck = (struct vstck_def *)
                                malloc(sizeof (struct vstck_def));
                        viewstck = vstck;
                } else {
                        vstck->vs_next = (struct vstck_def *)
                                malloc(sizeof (struct vstck_def));
                        vstck = vstck->vs_next;
                }
                vstck->vs_view = (struct view_def *)
                        malloc(sizeof (struct view_def));
                vstck->vs_next = NULL;
                fread((char *) vstck->vs_view, sizeof (struct view_def), 1, fd);
        }
```

init.c

```
        fclose(fd);

        /* Check the flags */
#if     GRAFENG != IRIS
        if (flags & INSTEREO)
                sia(1);
#endif
        if (flags & HALFBOND)
                sendcom("halfbond");
        if (flags & SHOWCOFG)
                makegnomon();
#if     GRAFENG == IRIS
        if (flags & ONEBUF) {
                singlebuffer();
                gconfig();
        }
        if (vh != INITHITHER || vy != INITYON)
                iris_colors();

        /*
         * If the flags aren't set according to the default, we fake
         * entries from the keypad to toggle them.  This modularizes
         * the toggling code, at the expense of some difficulty if the
         * pad toggles are rearranged.  If they are, these will have to
         * be changed.  (We don't need to qenter the upclick of the
         * buttons, since they are just ignored.)
         */

        if (!(flags & DCONTROL)) {
                flags |= DCONTROL;
                qenter(PADENTER, 1);
        }
        if (!(flags & DLABELS)) {
                flags |= DLABELS;
                qenter(PADCOMMA, 1);
        }
        if (!(flags & DTEXT)) {
                flags |= DTEXT;
                qenter(PADMINUS, 1);
        }
        if (flags & SHOWCOFG) {
                flags &= ~SHOWCOFG;
                qenter(PADPERIOD, 1);
        }
        if (flags & XBOND) {
                flags &= ~XBOND;
                qenter(PADPF1, 1);
        }
        if (flags & XLABEL) {
                flags &= ~XLABEL;
                qenter(PADPF2, 1);
        }
        if (flags & XSURF) {
                flags &= ~XSURF;
                qenter(PADPF3, 1);
        }
        if (flags & ORTHO) {
                flags &= ~ORTHO;
                qenter(PADPF4, 1);
        }
#endif  IRIS
#if     GRAFENG == PS2
        if (flags & ONEBUF)
                psbuf(1);
#endif
        if (!(flags & DLIGHTS))
                lights(0);

        /* Open up the models and add on the rotations and distances */

        nreply = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (l_models[i].active) {
                        if (l_models[i].flags & ISOBJ) {
                                sprintf(buf, "open object %d %s", i,
                                        l_models[i].molname);
                                sprintf(REPLY, "%s (%s)", msg, buf);
                                drawem(TRUE);
                                sendcom(buf);
                        } else {
#ifndef vms
                                sprintf(cp, ".%x", i);
#else
                                sprintf(cp, "_%x", i);
#endif
                                strcpy(l_models[i].altname, fname);
                                if (l_models[i].flags & HASURF)
                                        sprintf(buf, "open %d %s %s", i, fname,
                                                l_models[i].surfname);
                                else
                                        sprintf(buf, "open %d %s", i, fname);
                                sprintf(REPLY, "%s (%s)", msg, buf);
                                drawem(TRUE);
                                sendcom(buf);
```

```
                        }
                }
        }
        for (i = 0; i < MAXINROT; i++)
                if (l_rotation[i].r_stat & ACTIVE) {
                        sprintf(REPLY, "%s (%s)", msg, l_rotation[i].r_label);
                        drawem(TRUE);
                        sendcom(l_rotation[i].r_label);
                }
        for (i = 0; i < MAXDIST; i++)
                if (l_distance[i].d_stat & ACTIVE) {
                        sprintf(REPLY, "%s (%s)", msg, l_distance[i].d_label);
                        drawem(TRUE);
                        sendcom(l_distance[i].d_label);
                }
        for (i = 0; i < MAXANGLE; i++)
                if (l_angles[i].a_stat & ACTIVE) {
                        sprintf(REPLY, "%s (%s)", msg, l_angles[i].a_label);
                        drawem(TRUE);
                        sendcom(l_angles[i].a_label);
                }

        /* Reset to position and save it again (to set minstack properly) */
        procom("pop");
        procom("savepos");

        nreply = 0;
}
```

procom.c

```
/* $Header: procom.c,v 3.68 86/11/11 16:07:43 pett Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          30 Jun 1983
 * Release 2.2                          28 Aug 1983
 */

#include "intr.h"
#include <ctype.h>
#include <signal.h>

#ifdef  SIGIO
#define BSD
#endif

#if     GRAFENG == IRIS
#include <device.h>
#define HARDCOPY
#undef  HARDTRAN
#define ONE     1.0

#else

#ifndef ONE
#define ONE     K32K
#endif  ONE
#endif

#define MOVEMENT        0
#define SECTION         1
#define THICKNESS       2
#define SCALING         3
#define BUFSIZE         128

#define COMNOT          '~'
#define INTRPT          '\033'

int     doassign(),     docharsz(),     docopy(),       dochdir(),
        doclip(),       docofr(),       docpk(),        domapcolors(),
        dofix(),        dofixrev(),     dogetcrd(),     dohelp(),
        dointens(),     domatch(),      domove(),       dopush(),
        dopop(),        doreset(),      doroll(),       dorock(),
        dorecord(),     dorun(),        dosave(),       dosavepos(),
        doset(),        dosource(),     doscale(),      dosection(),
        doselect(),     dospeed(),      dostop(),       doescape(),
        dothickness(),  doturn(),       domatrix(),     dotran(),
        dounset(),      dowindow(),     dowait(),       dofreeze(),
        doalias(),      doalign(),      doslowdown(),   dorepeat(),
#ifndef vms
        dosleep(),
#ifdef  BSD
        dovmstat(), doframecnt(), domovie(),
#endif
#endif
        notimp();

char    *index(), *rindex(), *getinput();

struct  {
        char    *p_comname;
        int     (*p_func)();
        int     p_count;
} comlist[] = {
        "assign",       doassign,       0,
        "alias",        doalias,        0,
        "align",        doalign,        0,
        "charsz",       docharsz,       0,
        "copy",         docopy,         0,
        "cd",           dochdir,        0,
        "clip",         doclip,         0,
        "cofr",         docofr,         0,
        "cpk",          docpk,          0,
        "eval",         dorun,          0,
        "fix",          dofix,          0,
        "fixreverse",   dofixrev,       0,
        "freeze",       dofreeze,       0,
#ifdef  BSD
        "framecnt",     doframecnt,     0,
#else
        "framecnt",     notimp,         0,
#endif
        "getcrd",       dogetcrd,       0,
        "help",         dohelp,         0,
        "intensity",    dointens,       0,
        "match",        domatch,        0,
        "move",         domove,         0,
#ifdef  BSD
        "movie",        domovie,        0,
#else
        "movie",        notimp,         0,
```

```
#endif
        "mapcolors",    domapcolors,    0,
        "push",         dopush,         0,
        "pop",          dopop,          0,
        "reset",        doreset,        0,
        "roll",         doroll,         0,
        "rock",         dorock,         0,
        "record",       dorecord,       0,
        "repeat",       dorepeat,       0,
        "run",          dorun,          0,
        "save",         dosave,         0,
        "savepos",      dosavepos,      0,
        "set",          doset,          0,
        "source",       dosource,       0,
        "scale",        doscale,        0,
        "section",      dosection,      0,
        "select",       doselect,       0,
        "slowdown",     doslowdown,     0,
        "speed",        dospeed,        0,
#ifndef vms
        "sleep",        dosleep,        0,
#else
        "sleep",        notimp,         0,
#endif
        "stop",         dostop,         0,
        "system",       doescape,       0,
        "thickness",    dothickness,    0,
        "turn",         doturn,         0,
        "transform",    domatrix,       0,
        "translate",    dotran,         0,
#ifdef  BSD
        "vmstat",       dovmstat,       0,
#else
        "vmstat",       notimp,         0,
#endif
        "unset",        dounset,        0,
        "window",       dowindow,       0,
        "wait",         dowait,         0,
        0,              0,              0
};
static  int     intrpt;

/*
 * Procom - process command.
 */
procom(str)
char    *str;
{
        int     level;

        level = lex_string(str);
        if (level == -1)
                return;
        intrpt = FALSE;
        level = docommand(level);
        lex_done();
}

/*
 * Docommand - read commands downto 'level' and execute them.
 */
docommand(level)
int     level;
{
        register        i;
        char    buf[BUFSIZE], editcom[BUFSIZE];
        int     savedcom, retval, len, lastval;
        char    notcom;
        extern  int     stptr;

        retval = 0;

        while (stptr >= level) {
                editcom[0] = '\0';
                savedcom = FALSE;
                while (TRUE) {
                        cirinput(level);
                        len = getword(buf, sizeof buf, TRUE, level);
                        if (len <= 0)
                                break;

                        if (buf[0] == COMNOT) {
                                notcom = TRUE;
                                len = getword(buf, sizeof buf, TRUE, level);
                                if (len <= 0)
                                        break;
                        } else
                                notcom = FALSE;

                        /* Match for an interactive module command */
                        for (i = 0; comlist[i].p_comname != 0; i++)
                                if (strncmp(comlist[i].p_comname, buf, len)
                                == 0) {
                                        comlist[i].p_count++;
```

```
                        if (savedcom) {
                                /* Send saved edit comm
                                sendcom(editcom);
                                editcom[0] = '\0';
                                savedcom = FALSE;
                        }
#ifdef  DEBUG
                        if (flags & DDEBUG)
                                printf("Doing %s\n",
                                        comlist[i].p_com
#endif
                        /* Execute matched command */
                        lastval = (*comlist[i].p_func)(notcom,
                                level);
#ifdef  DEBUG
                        if (flags & DDEBUG)
                                printf("Done %s\n",
                                        comlist[i].p_com
#endif
                        retval += lastval;
                        break;
                }
                if (comlist[i].p_comname == 0) {
                        /*
                         * Current input token did not match one
                         * of the interactive module commands.
                         * If commands have already been saved,
                         * stick a semicolon between them and
                         * then save this one too.
                         */
                        if (savedcom)
                                strcat(editcom, " ; ");
                        if (notcom)
                                strcat(editcom, "~");
                        strcat(editcom, buf);
                        strcat(editcom, getinput(level));
                        savedcom = TRUE;
                }
        }
        if (savedcom) {
#ifdef  DEBUG
                if (flags & DDEBUG)
                        printf("Sending command:\n%s\n", editcom);
#endif
                sendcom(editcom);
        }
        notcom = FALSE;
    }
    return(retval);
}

/*
 * Doassign - assigns knobs to functions.
 */
doassign(notcom, level)
int     notcom, level;
{
        register int    i;
        register char   mode;
        register int    chan, rotnum;
        auto char       buf[BUFSIZE];
        auto char       answer[80];
        auto int        knum;

        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                prassign();
                return(0);
        }
        if (sscanf(buf, "%d", &knum) != 1 || knum < 0 || knum >= NOKNOB) {
                strcpy(REPLY, "Illegal knob number.");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                if (notcom) {
                        strcpy(buf, "nothing");
                        notcom = FALSE;
                }
                else {
                        strcpy(REPLY, "Incomplete specifications.");
                        return(-1);
                }
        }
        chan = knum - NOKNOB;
        mode = 1;
        if (!notcom && ((flags & REASSIGN) || strcmp(buf, "nothing") == 0)) {
                for (i = 0; i < 3; i++) {
                        if (knobs.k_tran[i] == chan || knobs.k_tran[i] == knum)
                                knobs.k_tran[i] = NOKNOB;
                        if (knobs.k_rot[i] == chan || knobs.k_rot[i] == knum)
                                knobs.k_rot[i] = NOKNOB;
                }
                for (i = 0; i < 6; i++)
                        if (knobs.k_clip[i] == chan || knobs.k_clip[i] == knum)
                                knobs.k_clip[i] = NOKNOB;
```

```
                for (i = 0; i < MAXINROT; i++)
                        if (knobs.k_inrot[i]==chan || knobs.k_inrot[i]==knum)
                                knobs.k_inrot[i] = NOKNOB;
                if (knobs.k_scale == chan || knobs.k_scale == knum)
                        knobs.k_scale = NOKNOB;
                if (knobs.k_section == chan || knobs.k_section == knum)
                        knobs.k_section = NOKNOB;
                if (knobs.k_thickness == chan || knobs.k_thickness == knum)
                        knobs.k_thickness = NOKNOB;
        }
        switch (buf[0]) {
                case 't' :                      /* Translation */
                case 'T' :
                        if (buf[1] == 'h' || buf[1] == 'H') {
                                if (notcom) {
                                        if (knobs.k_thickness != chan &&
                                           knobs.k_thickness != knum) {
                                                sprintf(REPLY, "Knob %d
                                                return(-1);
                                        }
                                        knobs.k_thickness = NOKNOB;
                                }
                                else
                                        knobs.k_thickness = chan;
                                if (flags & VERBOSE)
                                        sprintf(REPLY,
                                        "Thickness %s knob %2d.", notcom
                                        "deassigned from" : "assigned to",
                                        knum);
#if     GRAFENG == IRIS
                                set_assign(knum);
#endif
                                return(0);
                        }
                        mode = 0;
                        strcpy(answer, "Translation");
                        break;
                case 'r' :                      /* Rotation */
                case 'R' :
                        mode = 1;
                        strcpy(answer, "Rotation");
                        break;
                case 'c' :                      /* Clipping */
                case 'C' :
                        mode = 2;
                        strcpy(answer, "Clipping");
                        break;
                case 's' :                      /* Scaling */
                case 'S' :
                        if (buf[1] == 'e' || buf[1] == 'E') {
                                if (notcom) {
                                        if (knobs.k_section != chan &&
                                           knobs.k_section != knum) {
                                                sprintf(REPLY, "Knob %d not assigned to
                                                return(-1);
                                        }
                                        knobs.k_section = NOKNOB;
                                }
                                else
                                        knobs.k_section = chan;
                                if (flags & VERBOSE)
                                        sprintf(REPLY,
                                        "Section %s knob %2d.", notcom ?
                                        "deassigned from" : "assigned to",
                                        knum);
                        }
                        else {
                                if (notcom) {
                                        if (knobs.k_scale != chan &&
                                           knobs.k_scale != knum) {
                                                sprintf(REPLY, "Knob %d not assigned to
                                                return(-1);
                                        }
                                        knobs.k_scale = NOKNOB;
                                }
                                else
                                        knobs.k_scale = chan;
                                if (flags & VERBOSE)
                                        sprintf(REPLY,
                                        "Scale %s knob %2d.", notcom ?
                                        "deassigned from" : "assigned to",
                                        knum);
                        }
#if     GRAFENG == IRIS
                        set_assign(knum);
#endif
                        return(0);
                case 'n' :                      /* Nothing */
                case 'N' :
                        if (flags & VERBOSE)
                                sprintf(REPLY, "Knob %d deassigned.",
                                        knum);
#if     GRAFENG == IRIS
                        set_assign(knum);
```

```
#endif
                        return(0);
                default :
                        strcpy(REPLY, "No such type of movement.");
                        return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                strcpy(REPLY, "No direction specified.");
                return(-1);
        }
        do {
                if (buf[0] == '-') {
                        chan = knum - NOKNOB;
                        if (buf[1] == '\0') {
                                if (getword(buf, sizeof buf, TRUE, level) <= 0)
                                        break;
                        } else
                                buf[0] = buf[1];
                } else
                        chan = knum;
                switch(buf[0]) {
                        case 'x' :
                        case 'X' :
                                if (asaxis(notcom, mode, chan, 0,
                                   answer, " X") == -1)
                                        return(-1);
                                break;
                        case 'y' :
                        case 'Y' :
                                if (mode == 2) {
                                        if (asplane(notcom, mode, chan, C_
                                           answer, " back") == -1)
                                                return(-1);
                                } else {
                                        if (asaxis(notcom, mode, chan, 1,
                                           answer, " Y") == -1)
                                                return(-1);
                                }
                                break;
                        case 'z' :
                        case 'Z' :
                                if (asaxis(notcom, mode, chan, 2,
                                   answer, " Z") == -1)
                                        return(-1);
                                break;
                        case 'l' :
                        case 'L' :
                                if (asplane(notcom, mode, chan, C_LEFT,
                                   answer, " left") == -1)
                                        return(-1);
                                break;
                        case 'r' :
                        case 'R' :
                                if (asplane(notcom, mode, chan, C_RIGHT,
                                   answer, " right") == -1)
                                        return(-1);
                                break;
                        case 't' :
                        case 'T' :
                                if (asplane(notcom, mode, chan, C_TOP,
                                   answer, " top") == -1)
                                        return(-1);
                                break;
                        case 'b' :
                        case 'B' :
                                if (asplane(notcom, mode, chan, C_BOTTOM,
                                   answer, " bottom") == -1)
                                        return(-1);
                                break;
                        case 'h' :
                        case 'H' :
                                if (asplane(notcom, mode, chan, C_HITHER,
                                   answer, " front") == -1)
                                        return(-1);
                                break;
                        case '0' :
                        case '1' :
                        case '2' :
                        case '3' :
                        case '4' :
                        case '5' :
                        case '6' :
                        case '7' :
                        case '8' :
                        case '9' :
                                if (mode != 1) {
                                        strcpy(REPLY,
                                                "Only rotations permitted");
                                        return(-1);
                                }
                                rotnum = buf[0] - '0';
                                if (rotnum >= MAXINROT) {
                                        strcpy(REPLY, "No such rotation");
                                        return(-1);
```

```
                        }
                        if (notcom) {
                                if (knobs.k_inrot[rotnum] != chan &&
                                  knobs.k_inrot[rotnum] != knum) {
                                        sprintf(REPLY,
                                                "Knob %d not a
                                                knum, rotnum);
                                        return(-1);
                                }
                                knobs.k_inrot[rotnum] = NOKNOB;
                        }
                        else
                                knobs.k_inrot[rotnum] = chan;
                        strcat(answer, " ");
                        answer[strlen(answer) - 1] = buf[0];
                        break;
                }
        } while (getword(buf, sizeof buf, TRUE, level) > 0);
        if (flags & VERBOSE)
                sprintf(REPLY, "%s %s knob %2d.", answer,
                        notcom ? "deassigned from" : "assigned to", knum);
#if     GRAFENG == IRIS
        set_assign(knum);
#endif
        return(0);
}

/*
 * Prassign - print the assignment status of the knobs.
 */
prassign()
{
        register int            i, k;
        static char             buf[BUFSIZE];
        char                    *knob_str();

        nreply = 0;
        buf[0] = '\0';
        k = 0;
        for (i = 0; i < NOKNOB; i++) {
                sprintf(&buf[k], "%2d -%s", i, knob_str(i));
#if     GRAFENG == IRIS
                strcpy(REPLY, buf);
        }
#else
                k += strlen(&buf[k]);;
                if (k < 50) {
                        while (k % 25)
                                buf[k++] = ' ';
                } else {
                        buf[k] = '\0';
                        strcpy(REPLY, buf);
                        k = 0;
                }
        }
        if (k > 0) {
                buf[k] = '\0';
                strcpy(REPLY, buf);
        }
#endif  IRIS
}

/*
 * knob_str:
 *      Return a pointer to a printable version of what this knob
 *      does.
 */

#if     GRAFENG != IRIS
# define                S_ROTATE        " r "
# define                S_TRANSLATE     " tr "
# define                S_CLIP          " c "
# define                S_SCALE         " sc"
# define                S_SECTION       " se"
# define                S_THICKNESS     " th"
# define                S_UNUSED        " unused"
#else
# define                S_ROTATE        " rotate "
# define                S_TRANSLATE     " translate "
# define                S_CLIP          " clip "
# define                S_SCALE         " scale"
# define                S_SECTION       " section"
# define                S_THICKNESS     " thickness"
# define                S_UNUSED        " unused"
#endif

char *
knob_str(knob)
int     knob;
{
        register int            i;
        register char           *sp;
        register int            flag, used, chan;
        static char             buf[100];
```

```
        static char     axis[3] = { 'x', 'y', 'z' };
        static char     planes[6] = { 'l', 'r', 'b', 't', 'h', 'y' };

        used = FALSE;
        flag = TRUE;
        sp = buf;
        for (i = 0; i < 3; i++) {
                chan = (knobs.k_rot[i] < 0) ? knobs.k_rot[i] + NOKNOB :
                        knobs.k_rot[i];
                if (chan == knob) {
                        if (flag) {
                                strcpy(sp, S_ROTATE);
                                sp += sizeof S_ROTATE - 1;
                                flag = FALSE;
                                used = TRUE;
                        }
                        *sp++ = axis[i];
                        *sp = '\0';
                }
        }
        for (i = 0; i < MAXINROT; i++) {
                chan = (knobs.k_inrot[i] < 0) ?
                        knobs.k_inrot[i] + NOKNOB : knobs.k_inrot[i];
                if (chan == knob) {
                        if (flag) {
                                strcpy(sp, S_ROTATE);
                                sp += sizeof S_ROTATE - 1;
                                flag = FALSE;
                                used = TRUE;
                        }
                        *sp++ = i + '0';
                        *sp = '\0';
                }
        }
        flag = TRUE;
        for (i = 0; i < 3; i++) {
                chan = (knobs.k_tran[i] < 0) ?
                        knobs.k_tran[i] + NOKNOB : knobs.k_tran[i];
                if (chan == knob) {
                        if (flag) {
                                strcpy(sp, S_TRANSLATE);
                                sp += sizeof S_TRANSLATE - 1;
                                flag = FALSE;
                                used = TRUE;
                        }
                        *sp++ = axis[i];
                        *sp = '\0';
                }
        }
        flag = TRUE;
        for (i = 0; i < 6; i++) {
                chan = (knobs.k_clip[i] < 0) ?
                        knobs.k_clip[i] + NOKNOB : knobs.k_clip[i];
                if (chan == knob) {
                        if (flag) {
                                strcpy(sp, S_CLIP);
                                sp += sizeof S_CLIP - 1;
                                flag = FALSE;
                                used = TRUE;
                        }
                        *sp++ = planes[i];
                        *sp = '\0';
                }
        }
        chan = (knobs.k_scale < 0) ?
                knobs.k_scale + NOKNOB : knobs.k_scale;
        if (chan == knob) {
                strcpy(sp, S_SCALE);
                sp += sizeof S_SCALE - 1;
                used = TRUE;
        }
        chan = (knobs.k_section < 0) ?
                knobs.k_section + NOKNOB : knobs.k_section;
        if (chan == knob) {
                strcpy(sp, S_SECTION);
                sp += sizeof S_SECTION - 1;
                used = TRUE;
        }
        chan = (knobs.k_thickness < 0) ?
                knobs.k_thickness + NOKNOB : knobs.k_thickness;
        if (chan == knob) {
                strcpy(sp, S_THICKNESS);
                sp += sizeof S_THICKNESS - 1;
                used = TRUE;
        }
        if (!used) {
                strcpy(sp, S_UNUSED);
                sp += sizeof S_UNUSED - 1;
        }
        return buf;
}

/*
 * Asaxis - matches for the axis specifications.
```

```
 */
asaxis(notcom, mode, chan, axis, answer, s)
int     notcom;
int     mode, chan, axis;
char    *answer, *s;
{
        int     knum;

        if (notcom) {
                if (chan < 0)
                        knum = chan + NOKNOB;
                else
                        knum = chan - NOKNOB;
        }
        switch (mode) {
                case 0 :
                        if (notcom) {
                                if (knobs.k_tran[axis] != chan &&
                                  knobs.k_tran[axis] != knum) {
                                        sprintf(REPLY, "Knob %d not assig
                                                chan > 0 ? chan : knum, s
                                        return(-1);
                                }
                                else
                                        knobs.k_tran[axis] = NOKNOB;
                        }
                        else
                                knobs.k_tran[axis] = chan;
                        break;
                case 1 :
                        if (notcom) {
                                if (knobs.k_rot[axis] != chan &&
                                  knobs.k_rot[axis] != knum) {
                                        sprintf(REPLY, "Knob %d not assig
                                                chan > 0 ? chan : knum, s
                                        return(-1);
                                }
                                else
                                        knobs.k_tran[axis] = NOKNOB;
                        }
                        else
                                knobs.k_rot[axis] = chan;
                        break;
                case 2 :
                        strcpy(REPLY, "Can't clip an axis.");
                        return(-1);
        }
        strcat(answer, s);
        return(0);
}

/*
 * Asplane - matches for the plane specifications.
 */
asplane(notcom, mode, chan, plane, answer, s)
int     notcom;
int     mode, chan, plane;
char    *answer, *s;
{
        int     knum;

        if (mode != 2) {
                strcpy(REPLY, "Only clipping permitted.");
                return(-1);
        }
        if (notcom) {
                if (chan < 0)
                        knum = chan + NOKNOB;
                else
                        knum = chan - NOKNOB;
                if (knobs.k_clip[plane] != chan
                   && knobs.k_clip[plane] != knum) {
                        sprintf(REPLY, "Knob %d not assigned to clip%s.",
                                chan < 0 ? knum : chan, s);
                        return(-1);
                }
                else
                        knobs.k_clip[plane] = NOKNOB;
        }
        else
                knobs.k_clip[plane] = chan;
        strcat(answer, s);
        return(0);
}

/*
 * Dosource - gets commands from the given file.
 */
dosource(notcom, level)
int     notcom, level;
{
        char    buf[BUFSIZE];
        FILE    *fd;
        extern  char    _pskbchr;
```

procom.c

```
        extern  int     _ignornull;

        if (notcom) {
                strcpy(REPLY, "Don't source ???");
                return(-1);
        }
        _ignornull = FALSE;
        while (getword(buf, sizeof buf, TRUE, level) > 0) {
                if ((fd = fopen(buf, "r")) == NULL) {
                        if (flags & VERBOSE) {
                                strcat(buf, ": cannot access.");
                                strcpy(REPLY, buf);
                        }
                        continue;
                }
                while (fgets(buf, sizeof buf, fd) != NULL) {
                        if (intrpt)
                                break;
                        zapchar(buf, '\n');
                        if (flags & VERBOSE)
                                sprintf(REPLY, "Sourcing ... (%.40s)", buf);
                        drawem(TRUE);
                        nreply = 0;
                        if (getchr(combuf, TRUE) > 0) {
                                if (_pskbchr == INTRPT) {
                                        strcpy(REPLY, "Interrupt(S)!");
                                        intrpt = TRUE;
                                        break;
                                }
                                drawem(TRUE);
                                procom(combuf);
                        }
                        procom(buf);
                }
                fclose(fd);
        }
        _ignornull = TRUE;
        return(-1);                             /* Don't record this (was expanded)
}

/*
 * Dotum - rotate the system globally.
 */
dotum(notcom, level)
int     notcom, level;
{
        if (notcom)
                return(turn(0, level));
        else
                return(turn(1, level));
}

/*
 * Doroll - continuously rotate the system globally.
 */
doroll(notcom, level)
int     notcom, level;
{
        if (notcom)
                return(turn(0, level));
        else
                return(turn(-1, level));
}

/*
 * Dorock - rotate the system back and forth.
 */
dorock(notcom, level)
int     notcom, level;
{
        char    axis;
        char    buf[BUFSIZE];
        int     val;
        float   t;

        axis = 1;                       /* Default Y-axis */
        if ((val = getword(buf, sizeof buf, TRUE, level)) <= 0) {
                if (notcom)
                        movecoms.m_rock[axis].t_frames = 0;
                else {
                        rockpos[axis] = PI / 2;
                        movecoms.m_rock[axis].t_speed = MAPANGLE(3);
                        movecoms.m_rock[axis].t_frames = -1;
                        movecoms.m_rock[axis].t_wait = 0;
                }
                goto done;
        }
        switch (buf[0]) {
                case 'x' :
                case 'X' :
                        axis = 0;
                        val = getword(buf, sizeof buf, TRUE, level);
                        break;
                case 'y' :
```

```
                case 'Y' :
                        axis = 1;
                        val = getword(buf, sizeof buf, TRUE, level);
                        break;
                case 'z' :
                case 'Z' :
                        axis = 2;
                        val = getword(buf, sizeof buf, TRUE, level);
                        break;
        }
        if (val <= 0) {
                if (notcom)
                        movecoms.m_rock[axis].t_frames = 0;
                else {
                        rockpos[axis] = PI / 2;
                        movecoms.m_rock[axis].t_speed = MAPANGLE(3);
                        movecoms.m_rock[axis].t_frames = -1;
                        movecoms.m_rock[axis].t_wait = 0;
                }
                goto done;
        } else if (notcom) {
                strcpy(REPLY, "Too many arguments.");
                return(-1);
        }
        rockpos[axis] = PI / 2;
        if (sscanf(buf, "%f", &t) != 1)
                return(-1);
        while (t > 180.)
                t -= 360.;
        while (t < -180.)
                t += 360.;
        movecoms.m_rock[axis].t_speed = MAPANGLE(t);
        if (movecoms.m_rock[axis].t_speed == 0) {
                movecoms.m_rock[axis].t_frames = 0;
                goto done;
        }
        movecoms.m_rock[axis].t_frames = -1;
        movecoms.m_rock[axis].t_wait = 0;
        if (getword(buf, sizeof buf, TRUE, level) <= 0)
                goto done;
        movecoms.m_rock[axis].t_frames = atoi(buf);
        if (getword(buf, sizeof buf, TRUE, level) > 0)
                movecoms.m_rock[axis].t_wait = atoi(buf);
done:
        if (slow_fac != 0)
                slow_down(&movecoms.m_rock[axis], T_ROCK);
        return(0);
}

/*
 * domove - translate the system.
 */
domove(notcom, level)
int     notcom, level;
{
        if (notcom) {
                strcpy(REPLY, "Don't move ???");
                return(-1);
        } else
                return(shift(MOVEMENT, level));
}

/*
 * Scale - scale the system.  (Note: the scaling value is badly defined)
 */
doscale(notcom, level)
int     notcom, level;
{
        register int        i;

        if (notcom) {
                strcpy(REPLY, "Don't scale ???");
                return(-1);
        }
        else {
                i = shift(SCALING, level);
#if     GRAFENG == IRIS
                if (i == 0)
                        movecoms.m_scale.t_speed *= K_SCALE;
#endif
                return i;
        }
}

/*
 * Dosection - change the section.
 */
dosection(notcom, level)
int     notcom, level;
{
        if (notcom) {
                strcpy(REPLY, "Don't section ???");
                return(-1);
        } else
```

```
		return(shift(SECTION, level));
}

/*
 * Dothickness - change the thickness.
 */
dothickness(notcom, level)
int	notcom, level;
{
	if (notcom) {
		strcpy(REPLY, "Don't change thickness ???");
		return(-1);
	} else
		return(shift(THICKNESS, level));
}

/*
 * Doclip - move the clipping planes
 */
doclip(notcom, level)
int	notcom, level;
{
	char	axis;
	char	buf[BUFSIZE];
	int	val;
	float	t;

	if (notcom) {
		strcpy(REPLY, "Don't clip ???");
		return(-1);
	}
	axis = C_LEFT;			/* Default Left plane */
	if ((val = getword(buf, sizeof buf, TRUE, level)) <= 0) {
		if (flags & VERBOSE)
			strcpy(REPLY, "Missing specifications.");
		return(-1);
	}
	switch (buf[0]) {
		case 'l' :
		case 'L' :
			axis = C_LEFT;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'r' :
		case 'R' :
			axis = C_RIGHT;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'b' :
		case 'B' :
			axis = C_BOTTOM;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 't' :
		case 'T' :
			axis = C_TOP;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'h' :
		case 'H' :
			axis = C_HITHER;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'y' :
		case 'Y' :
			axis = C_YON;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
	}
	if (val <= 0) {
		movecoms.m_clip[axis].t_speed = MAPCRD(1.0);
		movecoms.m_clip[axis].t_frames = 1;
		movecoms.m_clip[axis].t_wait = 0;
		goto done;
	}
	if (sscanf(buf, "%f", &t) != 1) {
		sprintf(REPLY, "Unexpected argument '%s'.", buf);
		return(-1);
	}
	movecoms.m_clip[axis].t_speed = MAPCRD(t);
	if (movecoms.m_clip[axis].t_speed == 0) {
		movecoms.m_clip[axis].t_frames = 0;
		goto done;
	}
	movecoms.m_clip[axis].t_frames = 1;
	movecoms.m_clip[axis].t_wait = 0;
	if (getword(buf, sizeof buf, TRUE, level) <= 0)
		goto done;
	movecoms.m_clip[axis].t_frames = atoi(buf);
	if (getword(buf, sizeof buf, TRUE, level) > 0)
		movecoms.m_clip[axis].t_wait = atoi(buf);
done:
	if (slow_fac != 0)
		slow_down(&movecoms.m_clip[axis], T_SHIFT);
```

```
	return(0);
}

/*
 * Docharsz - change the character size.
 */
#if	GRAFENG == IRIS
/* ARGSUSED */
#endif
docharsz(notcom, level)
int	notcom, level;
{
#if	GRAFENG != IRIS
	char	buf[BUFSIZE];
	register	i;

	if (notcom) {
		strcpy(REPLY, "Don't change character size ???");
		return(-1);
	}
	if (getword(buf, sizeof buf, TRUE, level) <= 0)
		return(-1);
	i = atoi(buf);
	if (i < 1 || i > 7) {
		sprintf(REPLY, "Illegal character size %d", i);
		return(-1);
	} else {
		chsize = i;
		makeframe();
		return(0);
	}
#else	IRIS
	strcpy(REPLY, "Can't set character size on this device");
	return -1;
#endif	IRIS
}

/*
 * Dospeed - changes the degree of response of the knobs
 */
dospeed(notcom, level)
int	notcom, level;
{
	char	buf[BUFSIZE];
	int	len;
	register	i;

	if (notcom) {
		strcpy(REPLY, "Don't reduce knob response ???");
		return(-1);
	}
	if ((len = getword(buf, sizeof buf, TRUE, level)) <= 0) {
		sprintf(REPLY, "Speed at %d.", reduction);
		return(0);
	}
	i = atoi(buf);
	switch (buf[0]) {
		case '+' :
		case '-' :
			len = reduction + i;
			break;
		default :
			len = i;
			break;
	}
	if (len < 1 || len > 15) {
		if (flags & VERBOSE)
			sprintf(REPLY, "Illegal speed %d", len);
		return(-1);
	} else {
		reduction = len;
		sprintf(REPLY, "Speed set to %d.", reduction);
		return(0);
	}
}

/*
 * Dowindow - recalculates the window.
 */
dowindow(notcom, level)
int	notcom, level;
{
	if (notcom) {
		strcpy(REPLY, "Don't redo window ???");
		return(-1);
	} else {
		makewin();
		return(0);
	}
	/* ARGSUSED */
}

/*
 * Dointens - changes the hither and yon plane intensities.
```

```
 */
dointens(notcom, level)
int     notcom, level;
{
        char    buf[BUFSIZE];
        register        i, j;

        if (notcom) {
                strcpy(REPLY, "Don't change intensity ???");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                sprintf(REPLY, "Intensity set at %d %d.", vh, vy);
                return(0);
        }
        i = atoi(buf);
        if (i < 0 || i > 255) {
                sprintf(REPLY, "Illegal intensity %d.", i);
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                vh = i;
                sprintf(REPLY, "Intensity set to %d %d.", vh, vy);
                return(0);
        }
        j = atoi(buf);
        if (j < 0 || j > 255) {
                sprintf(REPLY, "Illegal intensity %d.", j);
                return(-1);
        }
        vh = i;
        vy = j;
#if     GRAFENG == IRIS
        iris_colors();
#endif
        sprintf(REPLY, "Intensity set to %d, %d", vh, vy);
        return(0);
}

/*
 * Domapcolors - change the color map (only works for the IRIS)
 */
domapcolors(notcom, level)
int     notcom, level;
{
#if     GRAFENG != IRIS
        strcpy(REPLY, "Can't change color mapping on this system");
        return -1;
#else
        register        i, j, tmp;
        auto char       buf[BUFSIZE];
        static int      numcolors = -1;         /* number of colors available */
        extern int      c_range, c_factor;

        if (notcom) {
                strcpy(REPLY, "Don't change color mapping ???");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                sprintf(REPLY, "Color map currently %d shades of %d colors.",
                        c_range, 64 / c_factor);
                return(0);
        }
        i = atoi(buf);
        if (i < 0) {
                sprintf(REPLY, "Illegal number of shades: %d.", i);
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0)
                j = 64 / c_factor;
        else {
                j = atoi(buf);
                if (j <= 0 || j > 64) {
                        sprintf(REPLY, "Illegal number of colors %d.", j);
                        return(-1);
                }
        }
        if (numcolors < 0)
                numcolors = 1 << getplanes();
        if (i * j > numcolors) {
                sprintf(REPLY,
                        "shades * colors must be <= number of bitplanes (%d)",
                        numcolors);
                return(-1);
        }
        tmp = 64 / j;
        if (tmp * j != 64) {
                strcpy(REPLY, "Number of colors must be factor of 64");
                return(-1);
        }
        sprintf(REPLY, "Color map set to %d shades of %d colors.", c_range,
                64 / c_factor);
        if (c_range != i || c_factor != tmp) {
                c_range = i;
```

```
                c_factor = tmp;

                /*
                 * redraw everything in the new colors
                 */

                iris_colors();
                sprintf(buf, "mapcolors %d %d ; redraw", c_range, c_factor);
                sendcom(buf);
                make_control();
                make_command();
                makeframe();
                cpk_init(TRUE);
        }
        drawem(TRUE);
        return(0);
#endif
}

/*
 * Dopush - push the current view on the stack.
 */
dopush(notcom, level)
int     notcom, level;
{
        struct  vstck_def *vs;

        if (notcom)
                return(dopop(FALSE, level));
        vs = (struct vstck_def *) malloc(sizeof (struct vstck_def));
        vs->vs_view = (struct view_def *) malloc(sizeof (struct view_def));
        putview(vs->vs_view);
        vs->vs_next = viewstck;
        viewstck = vs;
        if (flags & VERBOSE)
                sprintf(REPLY, "View pushed.");
        return(0);
}

/*
 * Dopop - pop the previous view off the stack.
 */
dopop(notcom, level)
int     notcom, level;
{
        struct  vstck_def       *vs;

        if (notcom)
                return(dopush(FALSE, level));
        if (viewstck == NULL) {
                strcpy(REPLY, "Nothing on the view stack.");
                return(-1);
        }
        getview(viewstck->vs_view);
        vs = viewstck->vs_next;
        free((char *) viewstck->vs_view);
        free((char *) viewstck);
        viewstck = vs;
        return(0);
}

/*
 * Doreset - moves all models to their original orientation.
 */
doreset(notcom, level)
int     notcom, level;
{
        struct  vheap_def       *vheap;
        char    buf[BUFSIZE];

        if (notcom) {
                strcpy(REPLY, "Don't reset ???");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0)
                strcpy(buf, "default");
        else if (strcmp(buf, "list") == 0)
                return(listpos());
        for (vheap = viewheap; vheap != NULL; vheap = vheap->vh_next)
                if (strncmp(buf, vheap->vh_name, VNAMELEN) == 0)
                        break;
        if (vheap == NULL) {
                sprintf(REPLY, "There is no view called '%s'.", buf);
                return(-1);
        }
        getview(vheap->vh_view);
        return(0);
}

/*
 * Dosavepos - save the current orientation as the original
 */
dosavepos(notcom, level)
int     notcom, level;
```

procom.c

```
{
        struct  vheap_def       *v, *vhprev;
        char    buf[BUFSIZE];
        int     nviews;
        char    *strncpy();

        if (notcom) {
                nviews = 0;
                while (getword(buf, sizeof buf, TRUE, level) > 0) {
                        vhprev = NULL;
                        for (v = viewheap; v != NULL; v = v->vh_next) {
                                if (strncmp(buf, v->vh_name, VNAMELEN) ==
                                        nviews++;
                                        free((char *) v->vh_view);
                                        if (vhprev == NULL)
                                                viewheap = v->vh_next;
                                        else
                                                vhprev->vh_next = v->vh_
                                        free((char *) v);
                                        break;
                                }
                                vhprev = v;
                        }
                }
                if (nviews == 0) {
                        strcpy(REPLY, "Usage: ~savepos view_name.");
                        return(-1);
                }
        } else {
                if (getword(buf, sizeof buf, TRUE, level) <= 0)
                        strcpy(buf, "default");
                else if (strcmp(buf, "list") == 0)
                        return(listpos());
                for (v = viewheap; v != NULL; v = v->vh_next)
                        if (strncmp(buf, v->vh_name, VNAMELEN) == 0)
                                break;
                if (v == NULL) {
                        v = (struct vheap_def *)
                                malloc(sizeof (struct vheap_def));
                        strncpy(v->vh_name, buf, VNAMELEN);
                        v->vh_view = (struct view_def *)
                                malloc(sizeof (struct view_def));
                        v->vh_next = viewheap;
                        viewheap = v;
                }
                putview(v->vh_view);
                sprintf(REPLY, "View saved as '%s'.", v->vh_name);
        }
        return(0);
}

/*
 * Listpos - list the saved position names
 */
listpos()
{
        int     nviews, len, needcomma;
        char    buf[BUFSIZE];
        struct  vheap_def       *v;

        strcpy(buf, "The saved views are: ");
        len = strlen(buf);
        nviews = 0;
        needcomma = FALSE;
        for (v = viewheap; v != NULL; v = v->vh_next) {
                if (strncmp(v->vh_name, "default", VNAMELEN) == 0)
                        continue;
                len += strlen(buf);
                if (needcomma)
                        len += 2;
                if (len > sizeof buf) {
                        strcpy(REPLY, buf);
                        len = strlen(buf);
                        buf[0] = '\0';
                        needcomma = FALSE;
                }
                if (needcomma)
                        strcat(buf, ", ");
                else
                        needcomma = TRUE;
                strcat(buf, v->vh_name);
                nviews++;
        }
        if (nviews > 0)
                sprintf(REPLY, "%s.", buf);
        else
                strcpy(REPLY, "There are no named views.");
        return(0);
}

/*
 * Putview - copy the current view info into the given buffer
 */
putview(view)
```

procom.c

```
struct	view_def		*view;
{
	register		i;

	for (i = 0; i < MAXMOD; i++) {
		blkcpy((char *) models[i].matrix, (char *) view->v_matrix[i],
			sizeof models[i].matrix);
		blkcpy((char *) models[i].tr, (char *) view->v_tr[i],
			sizeof models[i].tr);
		blkcpy((char *) models[i].com, (char *) view->v_cofg[i],
			sizeof models[i].com);
	}
	for (i = 0; i < MAXINROT; i++)
		view->v_angle[i] = rotation[i].r_angle;
	blkcpy((char *) win, (char *) view->v_win, sizeof win);
	blkcpy((char *) clip, (char *) view->v_clip, sizeof clip);
	view->v_scaleval = scaleval;
#if	GRAFENG != IRIS
	view->v_chsize = chsize;
#endif
	view->v_distcolor = distcolor;
	blkcpy((char *) rockpos, (char *) view->v_rockpos, sizeof rockpos);
	blkcpy((char *) com, (char *) view->v_com, sizeof com);
	view->v_width = width;
	view->v_setcofr = setcofr;
	view->v_cx = cx;
	view->v_cy = cy;
	view->v_cz = cz;
}

/*
 * Getview - retrieve the current view info from the given buffer
 */
getview(view)
struct	view_def		*view;
{
	register		i, j;
	ps_t		loc[4], new[4];
#ifdef	HARDTRAN
	int		ps2loc;
#endif

	for (i = 0; i < MAXMOD; i++)
		blkcpy((char *) view->v_matrix[i], (char *) models[i].matrix,
			sizeof models[i].matrix);
#ifdef	HARDTRAN
	unit();
	wbtmem(1);
	ps2loc = 0;
#endif
	for (i = 0; i < MAXMOD; i++) {
		if (!models[i].active)
			continue;
		for (j = 0; j < 3; j++)
			loc[j] = models[i].com[j] - view->v_cofg[i][j];
#ifdef	HARDTRAN
		bldcon(LOADMAT, models[i].matrix);
		moveto(loc[0], loc[1], loc[2]);
		rdtc(ps2loc, new);
		ps2loc += 4;
		for (j = 0; j < 3; j++)
			new[j] = new[j] * ((double) K32K) / new[3];
#else
		loc[3] = ONE;
		transform(models[i].matrix, loc, new, ONE);
#endif
		for (j = 0; j < 3; j++)
			models[i].tr[j] = view->v_tr[i][j] + new[j] - loc[j];
	}
#ifdef	HARDTRAN
	stopwb();
#endif
	for (i = 0; i < MAXINROT; i++) {
		rotation[i].r_angle = view->v_angle[i];
		if (rotation[i].r_stat == INACTIVE)
			continue;
#if	GRAFENG == PS2
		getrot(&matbuf[inrot[i]], rotation[i].r_angle, ZAXIS);
#endif
#if	GRAFENG == MPS
		getrot(&matbuf[inrot[i]], &rotation[i].r_angle, &ZAXIS);
#endif
#if	GRAFENG == IRIS
		pushmatrix();
		rotate(rotation[i].r_angle, 'z');
		loadmatrix(&matbuf[inrot[i]]);
		popmatrix();
#endif
	}
	blkcpy((char *) view->v_win, (char *) win, sizeof win);
	blkcpy((char *) view->v_clip, (char *) clip, sizeof clip);
	scaleval = view->v_scaleval;
#if	GRAFENG != IRIS
	chsize = view->v_chsize;
```

```
#endif
        distcolor = view->v_distcolor;
        blkcpy((char *) view->v_rockpos, (char *) rockpos, sizeof rockpos);
        blkcpy((char *) view->v_com, (char *) com, sizeof com);
        width = view->v_width;
        setcofr = view->v_setcofr;
        cx = view->v_cx;
        cy = view->v_cy;
        cz = view->v_cz;
        for (i = 0; i < 3; i++)
                cowin[i] = com[i];
        calcwin();
        makeframe();
        calcent(selection);
        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE, (ps_t *) NULL);
}

struct  {
        char    *f_name;
        unsigned        f_bit;
}       flaglist[]      =       {
        "reassign",     REASSIGN,
        "verbose",      VERBOSE,
        "pair",         STPAIR,
        "ortho",        ORTHO,
        "text",         DTEXT,
        "labels",       DLABELS,
#ifndef vms
#if     GRAFENG == IRIS
        "control",      DCONTROL,
#endif
#endif
        "stereo",       INSTEREO,
        "independent",  INDEP,
        "cofg",         SHOWCOFG,
        "single",       ONEBUF,
        "smear",        ONEBUF,
        "lights",       DLIGHTS,
        "record",       RECORD,
        "debug",        DDEBUG,
        "halfbond",     HALFBOND,
#ifndef vms
        "dobond",       XBOND,
        "dolabel",      XLABEL,
        "dosurface",    XSURF,
        "dovdw",        XSURF,
#endif
#if     GRAFENG == PS2
        "movie",        MMOVIE,         /* Set by domovie() */
#endif
        0,              0
};
#if     GRAFENG != IRIS
static  int     oldstat;
#endif

/*
 * Doset - sets a MIDAS variable.
 */
doset(notcom, level)
int     notcom, level;
{
        register int    i;
        register int    len, retval;
        register int    do_calc;
        char            buf[BUFSIZE];

        if (notcom)
                return(dounset(FALSE, level));
        retval = -1;
        do_calc = FALSE;
        while ((len = getword(buf, sizeof buf, TRUE, level)) > 0) {
                for (i = 0; flaglist[i].f_bit != 0; i++)
                        if (strncmp(flaglist[i].f_name, buf, len) == 0) {
                                retval = 0;
                                break;
                        }
                switch (flaglist[i].f_bit) {
                        case REASSIGN :
                        case VERBOSE :
                        case DLABELS :
                        case INDEP :
                        case RECORD :
                        case DDEBUG :
#ifndef vms
                        /* The C compiler on VMS cannot have cases
                         * that are not representable by signed 16-bit
                         * integers.  So we don't use them. */
                        case XBOND :
                        case XLABEL :
                        case XSURF :
#endif
                                break;
                        case STPAIR :
```

procom.c

```
#if     GRAFENG == PS2 || GRAFENG == IRIS
                        do_calc = TRUE;
#endif
                        break;
                case DTEXT :
#if     GRAFENG == IRIS
                        curson();
#endif
                        break;
#ifndef vms
                case DCONTROL :
#if     GRAFENG == IRIS
                        make_midas(DCONTROL);
#endif
                        break;
#endif
                case SHOWCOFG :
                        makegnomon();
                        break;
                case ONEBUF :
#if     GRAFENG == PS2
                        if (flaglist[i].f_name[1] == 'l')
                                psbuf(1);
                        else {
                                strcpy(REPLY, "\"smear\" not availa
                                retval = -1;
                        }
#endif
#if     GRAFENG == IRIS
                        if (flaglist[i].f_name[1] == 'm') {
                                singlebuffer();
                                gconfig();
                        }
                        else {
                                strcpy(REPLY, "\"single\" not availa
                                retval = -1;
                        }
#endif
#if     GRAFENG == MPS
                        sprintf(REPLY,
                                "\"%s\" not available on this device",
                                flaglist[i].f_name);
                        retval = -1;
#endif
                        break;
                case ORTHO :
#if     GRAFENG != IRIS
                        if (flags & INSTEREO) {
                                strcpy(REPLY,
                                        "Orthographic stereo not permitted.");
                                return(-1);
                        }
#endif
                        break;
                case INSTEREO :
#if     GRAFENG != IRIS
                        if (flags & INSTEREO) {
                                strcpy(REPLY, "Stereo mode already in effect.");
                                retval = -1;
                                break;
                        }
                        oldstat = (flags & ORTHO);
                        flags &= ~ORTHO;
                        sia(1);
#else
                        strcpy(REPLY, "Can't do sequential stereo on this device\n");
                        return -1;
#endif
                        break;
                case DLIGHTS :
                        lights((int) selection);
                        break;
                case HALFBOND :
                        sendcom("halfbond");
                        break;
                default :
                        strcpy(REPLY, "Unrecognized option.");
                        break;
                }
                flags |= flaglist[i].f_bit;
        }
        if (do_calc)
                calcwin();
        return(retval);
}

/*
 * Dounset - unsets the MIDAS variable.
 */
dounset(notcom, level)
int     notcom, level;
{
        char    buf[BUFSIZE];
        int     len, retval;
```

procom.c

```
        int     do_calc;
        register        i;

        if (notcom)
                return(doset(FALSE, level));
        retval = -1;
        do_calc = FALSE;
        while ((len = getword(buf, sizeof buf, TRUE, level)) > 0) {
                for (i = 0; flaglist[i].f_bit != 0; i++)
                        if (strncmp(flaglist[i].f_name, buf, len) == 0) {
                                retval = 0;
                                break;
                        }
                switch (flaglist[i].f_bit) {
                        case REASSIGN :
                        case VERBOSE :
                        case DLABELS :
                        case INDEP :
                        case SHOWCOFG :
                        case RECORD :
                        case DDEBUG :
#ifndef vms
                        case XBOND :
                        case XLABEL :
                        case XSURF :
#endif
                                break;
                        case STPAIR :
#if     GRAFENG == PS2 || GRAFENG == IRIS
                                do_calc = TRUE;
#endif
                                break;
                        case DTEXT :
#if     GRAFENG == IRIS
                                cursoff();
#endif
                                break;
#ifndef vms
                        case DCONTROL :
#if     GRAFENG == IRIS
                                make_midas(~DCONTROL);
#endif
                                break;
#endif
                        case ONEBUF :
#if     GRAFENG == PS2
                                psbuf(2);
#endif
#if     GRAFENG == IRIS
                                doublebuffer();
                                gconfig();
#endif
                                break;
                        case ORTHO :
#if     GRAFENG != IRIS
                                if (flags & INSTEREO)
                                        oldstat = 0;
#endif
                                break;
                        case INSTEREO :
#if     GRAFENG != IRIS
                                if (!(flags & INSTEREO)) {
                                        strcpy(REPLY, "Stereo mode not in effect.");
                                        retval = -1;
                                        break;
                                }
                                flags |= oldstat;
                                sia(0);
#endif
                                break;
                        case DLIGHTS :
                                lights(0);
                                break;
                        case HALFBOND :
                                sendcom("~halfbond");
                                break;
                        default :
                                strcpy(REPLY, "Unrecognized option.");
                                break;
                }
                flags &= ~flaglist[i].f_bit;
        }
        if (do_calc)
                calcwin();
        return(retval);
}

#if     GRAFENG == MPS
psbuf(n)
{
        strcpy(REPLY, "Sorry, can't change refresh buffer modes on the MPS");
        return(-1);
}
#endif
```

```
/*
 * Doselect - selects the specified models.
 */
doselect(notcom, level)
int     notcom, level;
{
        register int        i;
        register int        len;
        auto int            num;
        char                buf[BUFSIZE];

        while ((len = getword(buf, sizeof buf, TRUE, level)) > 0) {
                if (sscanf(buf, "%d", &num) != 1) {
                        sprintf(REPLY, "Illegal model number %s.", buf);
                        return(-1);
                }
                if (!models[num].active) {
                        sprintf(REPLY, "Model %d does not exist.", num);
                        return(-1);
                }
                if (notcom)
                        selection &= ~MODELSW(num);
                else
                        selection |= MODELSW(num);
        }
        if (flags & DLIGHTS)
                lights((int) selection);
        else
                lights(0);
        if (flags & VERBOSE) {
                buf[0]= '\0';
                len = 0;
                for (i = 0; i < MAXMOD; i++)
                        if (selection & MODELSW(i)) {
                                buf[len++] = '0' + i;
                                buf[len++] = ' ';
                                buf[len] = '\0';
                        }
                if (len > 0)
                        sprintf(REPLY, "Models %sselected.", buf);
                else
                        sprintf(REPLY, "No models selected.");
        }
        calcent(selection);
        return(0);
}
```

```
/*
 * Doescape - execute a shell command.
 */
doescape(notcom, level)
int     notcom, level;
{
        FILE    *pfd, *popen();
        char    buf[BUFSIZE], *shcom;

        if (notcom) {
                strcpy(REPLY, "Don't escape ???");
                return(-1);
        }
        shcom = getinput(level);
        usermask = umask(usermask);
        if (strlen(shcom) == 0)
                return(-1);
#ifndef vms
        sprintf(REPLY, "system ... (%.40s)", shcom);
        drawem(TRUE);
        if ((pfd = popen(shcom, "r")) == NULL) {
                sprintf(REPLY, "'%s': Unable to execute.", shcom);
                return(-1);
        }
        nreply = 0;
        while (fgets(buf, sizeof buf, pfd) != NULL)
                strcpy(REPLY, buf);
        pclose(pfd);
#else
        sprintf(REPLY, "check CRT terminal for any output");
        drawem(TRUE);
        vmscom(shcom);
#endif
        usermask = umask(usermask);
        return(0);
}

#ifdef  vms
#include <descrip.h>

vmscom(command)
char *command;
{
        struct dsc$descriptor_s string;

        string.dsc$w_length = strlen(command);
```

```
        string.dsc$a_pointer = command;
        string.dsc$b_class = DSC$K_CLASS_S;
        string.dsc$b_dtype = DSC$K_DTYPE_T;
        lib$spawn(&string);
}
#endif  vms

/*
 * Dorecord - records the commands up to now in the given file.
 */
dorecord(notcom, level)
int     notcom, level;
{
        char    buf1[BUFSIZE], buf2[BUFSIZE];
        FILE    *fd;

        if (notcom) {
                strcpy(REPLY, "Don't record ???");
                return(-1);
        }
        if (getword(buf1, sizeof buf1, TRUE, level) <= 0) {
                strcpy(REPLY, "Record file name missing.");
                return(-1);
        }
        if (getword(buf2, sizeof buf2, TRUE, level) > 0) {
                strcpy(REPLY, "record: too many arguments.");
                return(-1);
        }
        fd = fopen(buf1, "w");
        if (fd == NULL) {
                sprintf(REPLY, "%s: no write permission.", buf1);
                return(-1);
        }
        fseek(recordfd, 0L, 0);
        while (fgets(buf2, sizeof buf2, recordfd) != NULL)
                fputs(buf2, fd);
        fclose(fd);
        fseek(recordfd, 0L, 2);
        sprintf(REPLY, "Commands recorded in '%s'.", buf1);
        return(-1);                                     /* Don't record this */
}

/*
 * Dosave - saves the current session in the given file.  (MIDAS does not stop.)
 */
dosave(notcom, level)
int     notcom, level;
{
        char    buf1[BUFSIZE], buf2[BUFSIZE];
        char    dstat[MAXDIST], rstat[MAXINROT], astat[MAXANGLE];
        char    *cp, *base;
        int     fd;
        int     len;
        short   n;
        struct  vstck_def       *vs;
        struct  vheap_def       *vhp;
        register                i;
#ifndef vms
        char    *rindex();
#else
        char    *strrchr();
#endif

        if (notcom) {
                strcpy(REPLY, "Don't save ???");
                return(-1);
        }
        if (getword(buf1, sizeof buf1, TRUE, level) <= 0) {
                strcpy(REPLY, "Session file name missing.");
                return(-1);
        }
        if (getword(buf2, sizeof buf2, TRUE, level) > 0) {
                strcpy(REPLY, "save: too many arguments.");
                return(-1);
        }
#ifndef vms
        base = rindex(buf1, '/');
#else
        base = strrchr(buf1, ']');
#endif
        if (base == 0)
                base = buf1;
        else
                base++;
        if (strlen(base) > 8) {                 /* 8 = 14 - strlen(".0.ndx") */
                strcpy(REPLY, "Session file name must be less than 9 chars.");
                return(-1);
        }
        cp = &buf1[strlen(buf1)];
        strcpy(cp, MSESSION);
        fd = creat(buf1, PUBMODE);
        if (fd < 0) {
                sprintf(REPLY, "%s: no write permission.", buf1);
                return(-1);
```

```
	}
	*cp = '\0';

	for (i = 0; i < MAXMOD; i++)
		if (models[i].active) {
#ifndef vms
			sprintf(cp, ".%x", i);
#else
			sprintf(cp, "_%x", i);
#endif
			strcpy(models[i].altname, base);
		}

	procom("push");
	n = SES_MAGIC;
	write(fd, (char *) &n, sizeof n);
	write(fd, (char *) models, sizeof models);
	write(fd, (char *) rotation, sizeof rotation);
	write(fd, (char *) distance, sizeof distance);
	write(fd, (char *) angles, sizeof angles);
	write(fd, (char *) &knobs, sizeof knobs);
	write(fd, (char *) &movecoms, sizeof movecoms);
	write(fd, (char *) &reduction, sizeof reduction);
	write(fd, (char *) &vh, sizeof vh);
	write(fd, (char *) &vy, sizeof vy);
	write(fd, (char *) &flags, sizeof flags);
	write(fd, (char *) &selection, sizeof selection);
	len = 0;
	for (vhp = viewheap; vhp != NULL; vhp = vhp->vh_next)
		len++;
	write(fd, (char *) &len, sizeof len);
	for (vhp = viewheap; vhp != NULL; vhp = vhp->vh_next) {
		write(fd, (char *) vhp->vh_name, VNAMELEN);
		write(fd, (char *) vhp->vh_view, sizeof (struct view_def));
	}
	len = 0;
	for (vs = viewstck; vs != NULL; vs = vs->vs_next)
		len++;
	write(fd, (char *) &len, sizeof len);
	for (vs = viewstck; vs != NULL; vs = vs->vs_next)
		write(fd, (char *) vs->vs_view, sizeof (struct view_def));
	close(fd);

	for (i = 0; i < MAXINROT; i++)
		if (rotation[i].r_stat & ACTIVE) {
			rstat[i] = TRUE;
			sprintf(buf2, "~rot %d", i);
			sendcom(buf2);
		} else
			rstat[i] = FALSE;
	for (i = 0; i < MAXDIST; i++)
		if (distance[i].d_stat & ACTIVE) {
			dstat[i] = TRUE;
			sprintf(buf2, "~distance %d", i);
			sendcom(buf2);
		} else
			dstat[i] = FALSE;
	for (i = 0; i < MAXANGLE; i++)
		if (angles[i].a_stat & ACTIVE) {
			astat[i] = TRUE;
			sprintf(buf2, "~angle %d", i);
			sendcom(buf2);
		} else
			astat[i] = FALSE;
	for (i = 0; i < MAXMOD; i++)
		if (models[i].active) {
#ifndef vms
			sprintf(cp, ".%x", i);
#else
			sprintf(cp, "_%x", i);
#endif
			sprintf(buf2, "save %d %s", i, buf1);
			sendcom(buf2);
		}
	for (i = 0; i < MAXINROT; i++)
		if (rstat[i])
			sendcom(rotation[i].r_label);
	for (i = 0; i < MAXDIST; i++)
		if (dstat[i])
			sendcom(distance[i].d_label);
	for (i = 0; i < MAXANGLE; i++)
		if (astat[i])
			sendcom(angles[i].a_label);

	/* Clean up and return */
	procom("pop");
	*cp = '\0';
	nreply = 0;
	sprintf(REPLY, "Session saved in %s.", buf1);
	return 0;
}

/*
 * Docopy - copy the picture on the Versatec.
```

procom.c

```
 */
#if     GRAFENG != MPS
#include <setjmp.h>

static jmp_buf      printer_prob;
#endif

/* ARGSUSED */
docopy(notcom, level)
int     notcom, level;
{
#ifndef HARDCOPY
        strcpy(REPLY, "Sorry, no support for hardcopies on this system.");
        return(-1);
#else
#if     GRAFENG != MPS
        register int        title_given;
        register int        argc;
        register FILE       *copyf;
        register int        i;
        register long       o_flags;
        register int        (*fp)();
        register int        retval;
        auto char           printer[256];
        auto char           tbuf[4 * BUFSIZE];
        auto char           title[BUFSIZE];
        auto char           *argv[100];
        extern int          no_copy_pipe();
#if     GRAFENG == PS2
        extern  int     _ignomull;
        static  int     waitproc        =       0;
#endif
#endif  MPS

        if (notcom) {
                strcpy(REPLY, "Don't copy ???");
                return(-1);
        }

#if     GRAFENG == MPS
        cunit(&11);
        mpspol(&1, &1, &11.0, &0, &0);
#endif
#if     GRAFENG == PS2
        if (waitproc) {
                strcpy(REPLY, "Waiting for previous copy to finish.");
                drawem(TRUE);
                while (waitproc != wait((int *) 0))
                        continue;
        }
#endif

#if     GRAFENG == PS2 || GRAFENG == IRIS
        printer[0] = '\0';
        title[0] = '\0';
        title_given = FALSE;
        argv[0] = COPYPROG;
        argc = 1;
        while (getword(tbuf, sizeof tbuf, TRUE, level) > 0) {
                if (strcmp(tbuf, "date") == 0)
                        argv[argc++] = "date";
                else if (strcmp(tbuf, "box") == 0)
                        argv[argc++] = "box";
                else if (strcmp(tbuf, "color") == 0)
                        argv[argc++] = "color";
                else if (strcmp(tbuf, "bw") == 0)
                        argv[argc++] = "bw";
                else if (strcmp(tbuf, "printer") == 0) {
                        if (getword(tbuf, sizeof tbuf, TRUE, level) > 0) {
                                (void) strcpy(printer, tbuf);
                                argv[argc++] = "printer";
                                argv[argc++] = printer;
                        }
                        else
                                sprintf(REPLY, "must specify printer\n");
                }
                else if (strcmp(tbuf, "title") == 0) {
                        title_given = TRUE;
                        title[0] = '\"';
                        (void) strcpy(&title[1], getinput(level));
                        (void) strcat(title, "\"");
                        argv[argc++] = "title";
                        argv[argc++] = title;
                        break;
                }
                else {
                        sprintf(REPLY, "Unknown option '%s'.\n", tbuf);
                        return -1;
                }
        }

        o_flags = flags;
        flags |= DTEXT;
```

```
	combuf_clean(combuf);
	if (!title_given) {
		make_prompt("Title: ");
#if	GRAFENG == PS2
		_ignornull = FALSE;
#endif
		for (i = -1; i <= 0; i = getchr(combuf, FALSE)) {
#if	GRAFENG == IRIS
			register Device	d;
			auto short	input;

			/* need to toss any button strokes */
			while ((d = qtest()) != KEYBD && d != 0)
				(void) qread(&input);
#endif
			drawem(TRUE);
		}
#if	GRAFENG == PS2
		_ignornull = TRUE;
#endif
		(void) strcpy(title, combuf);
		combuf_clean(combuf);
		argv[argc++] = "title";
		argv[argc++] = title;
	}
	argv[argc] = NULL;

	/*
	 * Let the user know we're working on it.
	 */

	nreply = 0;
	make_prompt("Working on copy");
	drawem(TRUE);
	flags = o_flags;
	make_prompt("Command: ");

	tbuf[0] = '\0';
	for (i = 0; i < argc; i++) {
		if (i > 0)
			if (i + 1 >= argc)		/* quote the title */
				(void) strcat(tbuf, " \"");
			else
				(void) strcat(tbuf, " ");
		(void) strcat(tbuf, argv[i]);
		if (i + 1 >= argc)			/* quote the title */
			(void) strcat(tbuf, "\"");
	}
	if ((copyf = popen(tbuf, "w")) == NULL) {
		perror(tbuf);
		sprintf(REPLY, "Can't start up copy program");
		return 0;
	}

# if GRAFENG == PS2
#	define	COPY_FUNC	ps2_copy
# endif
# if GRAFENG == IRIS
#	define	COPY_FUNC	ir_copy
# endif

	fp = signal(SIGPIPE, no_copy_pipe);
	if (setjmp(printer_prob)) {
		sprintf(REPLY, "Printer problem (is \"%s\" known printer?)\n",
			printer);
		retval = -1;
		goto ret;
	}
	if (COPY_FUNC(copyf) < 0) {
		retval = -1;		/* the message has been set up by COPY_FUNC() */
		goto ret;
	}
	pclose(copyf);
#endif	GRAFENG == PS2 || GRAFENG == IRIS
	nreply = 0;
	strcpy(REPLY, "Copy should be coming out on the printer.");
	combuf_clean(combuf);
	retval = 0;
ret:
	(void) signal(SIGPIPE, fp);
	return retval;
	/* ARGSUSED */
#endif	HARDCOPY
}

no_copy_pipe()
{
	longjmp(printer_prob, 1);
}

/*
 * Dostop - terminate MIDAS.
 */
dostop(notcom, level)
```

procom.c

```
int	notcom, level;
{
#ifdef	vms
	int	i, j;
#endif

	if (notcom) {
		strcpy(REPLY, "Don't stop ???");
		return(-1);
	}
#ifdef	vms
	/*
	 * VMESS process signals don't work like
	 * UNIX signals.  We have to send a stop
	 * command to the editor if we want it to
	 * clean up properly.
	 */
	sendcom("stop");
	do {
		i = wait(&j);
	} while (i!=edit_pid && i!=-1);
#endif
	stop();
	/* NOTREACHED */
	/* ARGSUSED */
}

/*
 * Stop - actual code to stop midas (also here at interrupt)
 */
stop()
{
	register		i;

	signal(SIGINT, SIG_IGN);
	signal(SIGTERM, SIG_IGN);
	kill(edit_pid, SIGTERM);
	if (recordfd != NULL);
		fclose(recordfd);
	unlink(RECORDFILE);
	doneps();
#if	GRAFENG == IRIS
#ifndef	NOBUTTONBOX
	setdblights(0L);		/* clear off the lights */
	dbtext("");			/* .. and the text */
#endif
	greset();
	gexit();
#endif
#ifdef	DOSTATS
	cntinit(IRECFILE);
	for (i = 0; comlist[i].p_comname != 0; i++)
		cntincr(comlist[i].p_comname, comlist[i].p_count);
	cntfini(IRECFILE);
#endif
	exit(0);
}

/*
 * Dohelp - print help message.
 */
dohelp(notcom, level)
int	notcom, level;
{
#if	GRAFENG != IRIS
	register ps_t		i;
#else
	register Scoord		i;
#endif
	register int		notopic;
	register FILE		*fd;
	char	buf[BUFSIZE];
	char	helpfile[BUFSIZE];
#if	GRAFENG == IRIS
	register int			j;
	register Scoord			ypos;
	register Boolean		is_index = FALSE;
	static short			input;
	static Colorindex		clr, wtm;
	static Boolean			show_cur;
	static Scoord			ppos = 0;
#endif

	if (notcom) {
		strcpy(REPLY, "Don't help ???");
		return(-1);
	}
	if (getword(buf, sizeof buf, TRUE, level) <= 0)
		strcpy(buf, "index");
	sprintf(helpfile, HELPDIR, buf);
	if ((fd = fopen(helpfile, "r")) == NULL) {
		sprintf(helpfile, HELPDIR, "index");
		if ((fd = fopen(helpfile, "r")) == NULL) {
			strcpy(REPLY, "Index missing.  Notify management.");
```

```
                return(-1);
            }
            notopic = TRUE;
        } else
            notopic = FALSE;
#if     GRAFENG != IRIS
        if (flags & INSTEREO)
            sia(0);
#else
        is_index = (strcmp(buf, "index") == 0);
#endif
#if     GRAFENG == PS2
        unit();
        vwport(-2048, 2047, -2048, 2047, 255, 0);
        window(-K32K, K32K, -K32K, K32K);
        huesat(YELLOW, 6);
        charsz(4, 0);
        if (notopic) {
            moveto(-30000, -27000, 0);
            text("There is no topic ");
            text(buf);
        }
        do {
            moveto(-30000, -30000, 0);
            text("Press any key to continue");
            i = 30000;
            while (i > -28000 && (fgets(buf, sizeof buf, fd)) != NULL) {
                if (strncmp(buf, ".bp", 3) == 0)
                    break;
                moveto(-30000, i, 0);
                text(buf);
                i -= LINEHEIGHT(4);
            }
            moveto(0, 0, 0);            /* So amplifier won't work so hard */
            nufram();
            if (flags & MMOVIE) {
                shutter(S_OPEN);
                expose(s_expose);
                shutter(S_CLOSE);
            }
            while (getchr(combuf, FALSE) == 0) {
                rstfram();
                if (flags & MMOVIE) {
                    shutter(S_OPEN);
                    expose(s_expose);
                    shutter(S_CLOSE);
                }
            }
            combuf_clean(combuf);
        } while (!feof(fd));
#endif  PS2
#if     GRAFENG == MPS
        tident();
        vbound(&-2048, &2047, &-2048, &2047);
        vinten(&63, &63);
        twind(&-K32K, &K32K, &-K32K, &K32K, &-K32K, &K32K);
        if (notopic) {
            lcolor(&48, &6);
            csize(&4);
            damove(&-30000, &-27000, &0);
            ctext("There is no topic ");
            ctext(buf);
        }
        do {
            lcolor(&48, &6);
            csize(&4);
            damove(&-30000, &-30000, &0);
            ctext("Press any key to continue");
            i = 30000;
            while (i > -28000 && (fgets(buf, sizeof buf, fd)) != NULL) {
                if (strncmp(buf, ".bp", 3) == 0)
                    break;
                damove(&-30000, &i, &0);
                ctext(buf);
                i -= LINEHEIGHT(4);
            }
            damove(&0, &0, &0);         /* So amplifier won't work so hard */
            sdrep();
            while (getchr() == 0)
                rstfram();
            combuf_clean(combuf);
        } while (!feof(fd));
#endif  MPS
#if     GRAFENG == IRIS
                                        /* + 10 because there are 10 blanks in front of text */
#define TEXT_X              (XMAXSCREEN / 2 - normwidth * ((60 / 2) + 10))

        pushattributes();
        pushviewport();
        pushmatrix();
        getcursor(&input, &clr, &wtm, &show_cur);
        if (show_cur)
            cursoff();
        frontbuffer(TRUE);
```

procom.c

```
        viewport(0, XMAXSCREEN, 0, YMAXSCREEN);
        ortho2(0.0, (Coord) XMAXSCREEN, 0.0, (Coord) YMAXSCREEN);
        font(NORMFONT);
        blink(40, ir_colornum(MAGENTA), 0, 0, 255);
        do {
                ir_color(BLUE);
                clear();
                ir_color(MAGENTA);
                if (ppos == 0) {
                        ppos = strwidth("Press any key to continue");
                        ppos = (XMAXSCREEN - ppos) / 2;
                }
                cmov2i(ppos, COMMAND_Y);
                charstr("Press any key to continue");
                if (notopic) {
                        i = strwidth(buf) + strwidth("There is no topic \"\"");
                        cmov2i((XMAXSCREEN - i) / 2,
                            (YMAXSCREEN - normheight) / 2);
                        ir_color(WHITE);
                        charstr("There is no topic \"");
                        ir_color(CYAN);
                        charstr(buf);
                        ir_color(WHITE);
                        charstr("\"");
                        getstroke();
                        continue;
                }
                i = YMAXSCREEN - 2 - normheight;
                while (i > COMMAND_Y + normheight &&
                    fgets(buf, sizeof buf, fd) != NULL) {
                        if (strncmp(buf, ".bp", 3) == 0)
                                break;
                        cmov2i(TEXT_X, i);
                        print_str(buf);
                        i -= normheight;
                }
                if (is_index)
                        ir_help();
                curson();
                getstroke();            /* wait for a key to be typed */
                cursoff();
        } while (!notopic && !feof(fd));
        blink(0, ir_colornum(MAGENTA), 0, 0, 255);
        if (show_cur)
                curson();
        popmatrix();
        popviewport();
        popattributes();
        ir_color(BLACK);
        clear();
#endif  IRIS
#if     GRAFENG != IRIS
        if (flags & INSTEREO)
                sia(1);
#endif
        fclose(fd);
        return(0);
}

#if     GRAFENG == IRIS
/*
 * print_str:
 *      Print out a string, using WHITE for normal text, CYAN for underlined
 *      text.
 */
print_str(sp)
register char   *sp;
{
        register char           *osp;
        register Colorindex     clr;
        register int            x, newx;
        static char             buf[BUFSIZ];

        if (*sp == '_' && *(sp + 1) == '\b')
                clr = CYAN;
        else if (*(sp + 1) == '\b')
                clr = YELLOW;
        else
                clr = WHITE;
        x = 0;
        for (osp = buf; *sp; sp++) {
                if (*sp == '\t') {
                        for (newx = x + (8 - (x & 07)); x < newx; x++)
                                *osp++ = ' ';
                        continue;
                }
                x++;
                if (*(sp + 1) == '\b') {
                        if (*sp == '_' || sp[2] == '_') {
                                if (clr != CYAN) {
                                        *osp = '\0';
                                        ir_color(clr);
                                        charstr(buf);
                                        osp = buf;
```

```
                                clr = CYAN;
                        }
                        *osp++ = (*sp == '_' ? sp[2] : *sp);
                        sp += 2;
                }
                else {
                        if (clr != YELLOW) {
                                *osp = '\0';
                                lr_color(clr);
                                charstr(buf);
                                osp = buf;
                                clr = YELLOW;
                        }
                        while (sp[1] == '\b' && sp[2] == sp[0])
                                sp += 2;
                        *osp++ = *sp;
                }
        }
        else {
                if (clr != WHITE) {
                        *osp = '\0';
                        lr_color(clr);
                        charstr(buf);
                        osp = buf;
                        clr = WHITE;
                }
                *osp++ = *sp;
        }
    }
    if (osp > buf) {
        *osp = '\0';
        lr_color(clr);
        charstr(buf);
    }
}
#endif  IRIS

/*
 * Dowait - wait for all motion to stop.
 */
dowait(notcom, level)
int     notcom, level;
{
        register        i;
        unsigned        len, t;
        char            buf[BUFSIZE];
        extern  char    _pskbchr;
        extern  int     _ignomull;

        if (notcom) {
                strcpy(REPLY, "Don't wait ???");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                len = 0;
                for (i = 0; i < 3; i++) {
                        t = movecoms.m_rot[i].t_wait +
                            movecoms.m_rot[i].t_frames;
                        if (t > len)
                                len = t;
                }
                for (i = 0; i < 3; i++) {
                        t = movecoms.m_tran[i].t_wait +
                            movecoms.m_tran[i].t_frames;
                        if (t > len)
                                len = t;
                }
                for (i = 0; i < MAXINROT; i++) {
                        t = movecoms.m_inrot[i].t_wait +
                            movecoms.m_inrot[i].t_frames;
                        if (t > len)
                                len = t;
                }
                for (i = 0; i < 6; i++) {
                        t = movecoms.m_clip[i].t_wait +
                            movecoms.m_clip[i].t_frames;
                        if (t > len)
                                len = t;
                }
                t = movecoms.m_scale.t_wait + movecoms.m_scale.t_frames;
                if (t > len)
                        len = t;
                t = movecoms.m_section.t_wait + movecoms.m_section.t_frames;
                if (t > len)
                        len = t;
                t = movecoms.m_thickness.t_wait + movecoms.m_thickness.t_frames;
                if (t > len)
                        len = t;
        }
        else {
                if (sscanf(buf, "%d", &len) != 1) {
                        strcpy(REPLY, "Wait expects an integer argument.");
                        return -1;
                }
```

```
        }

        _ignornull = FALSE;
        for (i = 0; i < len; i++) {
                if (intrpt)
                        break;
                if (getchr(combuf, TRUE) > 0) {
                        nreply = 0;
                        if (_pskbchr == INTRPT) {
                                strcpy(REPLY, "Interrupt(W)!");
                                intrpt = TRUE;
                                break;
                        }
                        drawem(TRUE);
                        procom(combuf);
                }
                if (movem() == 0) {
#if     GRAFENG != IRIS
                        rstfram();
#else
                        ;
#endif
#if     GRAFENG == PS2
                        if (flags & MMOVIE) {
                                shutter(S_OPEN);
                                expose(s_expose);
                                shutter(S_CLOSE);
                        }
#endif
                }
                else
                        drawem(TRUE);
        }
        _ignornull = TRUE;
        fixtran(selection);
        calcent(selection);
        return(0);
        /* ARGSUSED */
}

/*
 * Dochdir - change directory.
 */
dochdir(notcom, level)
int     notcom, level;
{
        char    buf1[BUFSIZE], buf2[BUFSIZE];

        if (notcom) {
                strcpy(REPLY, "Don't change directory ???");
                return(-1);
        }
        if (getword(buf1, sizeof buf1, TRUE, level) <= 0) {
                strcpy(REPLY, "Don't know where to go.");
                return(-1);
        }
        if (getword(buf2, sizeof buf2, TRUE, level) > 0) {
                strcpy(REPLY, "cd: too many arguments.");
                return(-1);
        }
        if (chdir(buf1) < 0) {
                strcpy(REPLY, "Changing directory failed.");
                return(-1);
        }
        strcpy(buf2, "cd ");
        strcat(buf2, buf1);
        sendcom(buf2);
        return(0);
}

/*
 * The following routines pass several different data types
 * worth of data to the editor through a common buffer.
 */
extern  char    *cheatbuf;

/*
 * Dogetcrd - get the untransformed coordinate of an atom
 */
dogetcrd(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        int     *ip;
        float   x, y, z;
        float   *fp;

        if (notcom) {
                strcpy(REPLY, "Don't get coordinate ???");
                return(-1);
        }
        strcpy(editcom, "getcrd ");
        strcat(editcom, getinput(level));
        sendcom(editcom);
```

```
        ip = (int *) cheatbuf;
        if (*ip++ != 1) {                       /* Must have 1 molecule's atom */
                sprintf(REPLY, "Bad atom specification.");
                return(-1);
        }
        ip++;
        fp = (float *) ip;
        x = *fp++;
        y = *fp++;
        z = *fp++;
        sprintf(REPLY, "Coordinate is %.2f %.2f %.2f", x, y, z);
        uncheat();
        return(0);
}

/*
 * Docofr - set the center of rotation to the site of an atom
 */
docofr(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        int     *ip, molnum;
        ps_t    loc[4], cofg[4];
        float   *fp;

        if (notcom) {
                if (!setcofr) {
                        strcpy(REPLY, "The center of rotation wasn't set.");
                        return(-1);
                } else {
                        setcofr = FALSE;
                        calcent(selection);
                        strcpy(REPLY, "Center of rotation reset.");
                        return(0);
                }
        }
        strcpy(editcom, "getcrd ");
        strcat(editcom, getinput(level));
        sendcom(editcom);
        ip = (int *) cheatbuf;
        if (*ip++ != 1) {                       /* Must have 1 molecule's atom */
                sprintf(REPLY, "Bad atom specification.");
                return(-1);
        }
        molnum = *ip++;
        fp = (float *) ip;
        loc[0] = MAPCRD(*fp++) - models[molnum].com[0];
        loc[1] = MAPCRD(*fp++) - models[molnum].com[1];
        loc[2] = MAPCRD(*fp++) - models[molnum].com[2];
#if     GRAFENG != IRIS
        loc[3] = K32K;
#else
        loc[3] = 1;
#endif
#ifdef  HARDTRAN
        wbtmem(1);
        bldcon(LOADMAT, models[molnum].matrix);
        moveto(loc[0], loc[1], loc[2]);
        rdtc(0, cofg);
        for (i = 0; i < 3; i++)
                cofg[i] = cofg[i] * ((double) K32K) / cofg[3];
        stopwb();
#else
        transform(models[molnum].matrix, loc, cofg, ONE);
#endif
        cx = cofg[0] + models[molnum].com[0] + models[molnum].tr[0];
        cy = cofg[1] + models[molnum].com[1] + models[molnum].tr[1];
        cz = cofg[2] + models[molnum].com[2] + models[molnum].tr[2];
        setcofr = TRUE;

        sprintf(REPLY, "Center of rotation at (%d %d %d)", cx, cy, cz);
        uncheat();
        return(0);
}

/*
 * Dofix - fix the internal rotation.
 */
dofix(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        int     *ip;
        angle_t *ap;
        int     saveinit[MAXINROT];
        register        i;

        if (notcom) {
                strcpy(REPLY, "Don't fix rotation ???");
                return(-1);
        }
        ip = (int *) cheatbuf;
        *ip++ = 0;
```

```
        ap = (angle_t *) ip;
        for (i = 0; i < MAXINROT; i++) {
                *ap++ = rotation[i].r_angle;
                saveinit[i] = rotation[i].r_inita;
        }

        strcpy(editcom, "~ rot");
        strcat(editcom, getinput(level));
        sendcom(editcom);

        for (i = 0; i < MAXINROT; i++)
                if (rotation[i].r_stat == ACTIVE)
                        rotation[i].r_angle += saveinit[i] -
                                rotation[i].r_inita;
        uncheat();
        return(0);
}

/*
 * Dofixrev - fix and reverse the internal rotation.
 */
dofixrev(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        char    rotype[MAXINROT];
        int     *ip;
        angle_t *ap;
        register        i;

        if (notcom) {
                strcpy(REPLY, "Don't fix and reverse a rotation ???");
                return(-1);
        }
        ip = (int *) cheatbuf;
        *ip++ = 0;
        ap = (angle_t *) ip;
        for (i = 0; i < MAXINROT; i++) {
                *ap++ = rotation[i].r_angle;
                rotype[i] = rotation[i].r_label[0];
        }

        strcpy(editcom, "fixrev");
        strcat(editcom, getinput(level));
        sendcom(editcom);

        for (i = 0; i < MAXINROT; i++)
                if (rotation[i].r_stat == ACTIVE &&
                        rotype[i] != rotation[i].r_label[0])
                                rotation[i].r_angle = 0;
        uncheat();
        return(0);
}

/*
 * Domatch - superimpose two models.
 */
domatch(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        int     *ip;
        int     count;
        int     from, to;
        float   *fp;
        double  coord[2][4][3];
        register        i, j;

        if (notcom) {
                strcpy(REPLY, "Don't match ???");
                return(-1);
        }

        strcpy(editcom, "match ");
        strcat(editcom, getinput(level));
        sendcom(editcom);

        ip = (int *) cheatbuf;
        count = *ip++;
        if (count != 2 && count != -2) {                /* Must have 2 mol' atoms */
                strcpy(REPLY, "Bad atom specifications.");
                return(-1);
        }
        from = *ip++;
        fp = (float *) ip;
        for (i = 0; i < 4; i++)
                for (j = 0; j < 3; j++)
                        coord[0][i][j] = *fp++;
        ip = (int *) fp;
        to = *ip++;
        fp = (float *) ip;
        for (i = 0; i < 4; i++)
                for (j = 0; j < 3; j++)
                        coord[1][i][j] = *fp++;
```

```
        if (count > 0)
                i = superimpose(to, coord[1], from, coord[0]);
        else
                i = superimpose(from, coord[0], to, coord[1]);
        if (i == -1)
                strcpy(REPLY, "Too many iterations needed.");
        else
                sprintf(REPLY, "Superimposed %d on %d.",
                        count > 0 ? from : to, count > 0 ? to : from);
        uncheat();
        return(0);
}

/*
 * Docpk - make a space-filling model picture on the Versatec.
 */
/* ARGSUSED */
docpk(notcom, level)
int     notcom, level;
{
#ifdef  vms
        strcpy(REPLY, "Sorry, can't make space filling models on this system");
        return(-1);
#else
        int             plot, fuzzy;
        char            buf[BUFSIZE], *cpk_file;
        auto short      input;

        if (notcom) {
                strcpy(REPLY, "Don't make space filling model ???");
                return(-1);
        }
        fuzzy = FALSE;
        plot = FALSE;
        while (getword(buf, sizeof buf, TRUE, level) > 0) {
                if (strcmp(buf, "fuzzy") == 0)
                        fuzzy = TRUE;
                else if (strcmp(buf, "plot") == 0)
                        plot = TRUE;
                else
                        sprintf(REPLY, "Unknown option '%s'.\n", buf);
        }
#if     GRAFENG == IRIS
        if (plot) {
                strcpy(REPLY, "can't plot on the IRIS");
                return -1;
        }
        if (fuzzy)
                return ir_cpk();
#else
        if (fuzzy) {
                strcpy(REPLY, "fuzzy spheres not available on this device");
                return -1;
        }
#endif
        /*
         * let the user know something is going on
         */

        nreply = 0;
        strcpy(REPLY, "Working on cpk model");
        drawem(TRUE);

        cpk_file = TMPDIR(MCPK);
        (void) sprintf(buf, "save %s", cpk_file);
        procom(buf);
        nreply = 0;
        if (plot) {
#ifdef  NOTUSED
                (void) sprintf(buf,
                "midas.out -I %s 2> /dev/null | rsh cgl cpk - -vn '>' /dev/vp0",
                        cpk_file);
                system(buf);
                strcpy(REPLY, "Copy should be coming out on the Versatec.");
#else
                strcpy(REPLY,
                  "Sorry, cpk versatec output no longer available.");
#endif
        } else {
                (void) sprintf(buf,
                "midas.out -I %s 2> /dev/null | rsh socr cpk - -n '>' /dev/aed",
                        cpk_file);
                system(buf);
#if     GRAFENG != IRIS
                strcpy(REPLY, "Image should appear on AED 512.");
#else
                nreply = 0;
#endif
        }
        (void) sprintf(buf, "rm -f %s*", cpk_file);
        system(buf);
#if     GRAFENG == IRIS
#ifndef GL2_3BUGS
{
```

procom.c

```
#include        <gl2/grioctl.h>
        /*
         * On a 3.4 system you have to tell the system to give this
         * process input events, since the ginit() in cpk redirected
         * input events to "its" queue.  This may break on subsequent
         * systems.
         */
        grioctl(GR_GFINPUTCHANNEL, inchanget());
}
#endif
        /* wait for a keystroke */
        while (qread(&input) != KEYBD)
                continue;
        /* reset the color map, since cpk mucks around with it */
        iris_colors();
        drawem(TRUE);
#endif  IRIS
        return(0);
#endif  vms
}

/*
 * domatrix - reads a matrix and applies it to the selected models
 */
domatrix(notcom, level)
int     notcom, level;
{
        float   fmat[3][3];
        ps_t    mat[4][4];
        char    buf[BUFSIZE];
        register        i, j;

        if (notcom) {
                strcpy(REPLY, "Don't transform ???");
                return(-1);
        }
        for (i = 0; i < 9; i++) {
                if (getword(buf, sizeof buf, TRUE, level) <= 0)
                        break;
                if (sscanf(buf, "%f", &fmat[i % 3][i / 3]) != 1)
                        break;
        }
        if (i != 9) {
                strcpy(REPLY, "Incomplete matrix specifications.");
                return(-1);
        }
        for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++)
#if     GRAFENG != IRIS
                        mat[i][j] = fmat[i][j] * K16K;
#else
                        mat[i][j] = fmat[i][j];
#endif
                mat[3][i] = mat[i][3] = 0;
        }
#if     GRAFENG != IRIS
        mat[3][3] = K16K;
#else
        mat[3][3] = 1;
#endif
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || !(selection & MODELSW(i)))
                        continue;
#if     GRAFENG == PS2
                bidcon(LOADMAT, mat);
                bidcon(CONCATMAT, models[i].matrix);
                bidcon(STOREMAT, models[i].matrix);
#endif
#if     GRAFENG == IRIS
                pushmatrix();
                loadmatrix(mat);
                multmatrix(models[i].matrix);
                getmatrix(models[i].matrix);
                popmatrix();
#endif
#if     GRAFENG == MPS
                tset(mat);
                tcon(models[i].matrix);
                tget(models[i].matrix);
#endif
        }
        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE, (ps_t *) NULL);
        return(0);
}

/*
 * Dotran - do translation in molecular coordinate system
 */
dotran(notcom, level)
int     notcom, level;
{
        ps_t    loc[4], new[4];
        float   dist[3];
        char    buf[BUFSIZE];
```

```
        register        i;
#ifdef  HARDTRAN
        int     ps2loc;
#endif

        if (notcom) {
                strcpy(REPLY, "Don't translate ???");
                return(-1);
        }
        for (i = 0; i < 3; i++) {
                if (getword(buf, sizeof buf, TRUE, level) <= 0)
                        break;
                if (sscanf(buf, "%f", &dist[i]) != 1)
                        break;
        }
        if (i != 3)
                strcpy(REPLY, "Incomplete translation specifications.");
        loc[0] = MAPCRD(dist[0]);
        loc[1] = MAPCRD(dist[1]);
        loc[2] = MAPCRD(dist[2]);
#if     GRAFENG != IRIS
        loc[3] = K32K;
#else
        loc[3] = 1;
        pushmatrix();
#endif
#ifdef  HARDTRAN
        wbtmem(1);
        ps2loc = 0;
#endif
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || !(selection & MODELSW(i)))
                        continue;
#ifdef  HARDTRAN
                bldcon(LOADMAT, models[i].matrix);
                moveto(loc[0], loc[1], loc[2]);
                rdtc(ps2loc, new);
                ps2loc += 4;
                for (j = 0; j < 3; j++)
                        new[j] = new[j] * ((double) K32K) / new[3];
#else
                transform(models[i].matrix, loc, new, ONE);
#endif
                models[i].tr[0] += new[0];
                models[i].tr[1] += new[1];
                models[i].tr[2] += new[2];
        }
#ifdef  HARDTRAN
        stopwb();
#endif
#if     GRAFENG == IRIS
        popmatrix();
#endif
        calcent(selection);
        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE, (ps_t *) NULL);
        return(0);
}

/*
 * dorun - reads commands from a program and executes them
 */
dorun(notcom, level)
int     notcom, level;
{
        register char   *shcom;
        register FILE   *fd;
        auto char       buf[BUFSIZE];
        extern char     _pskbchr;
        extern int      _ignornull;
        extern FILE     *popen();

        if (notcom) {
                strcpy(REPLY, "Don't run ???");
                return(-1);
        }
        shcom = getinput(level);
        if (strlen(shcom) == 0)
                return(-1);
        if ((fd = popen(shcom, "r")) == NULL) {
                if (flags & VERBOSE)
                        sprintf(REPLY, "'%s': cannot execute.", shcom);
                return(-1);
        }
        _ignornull = FALSE;
        while (fgets(buf, sizeof buf, fd) != NULL) {
                if (intrpt)
                        break;
                zapchar(buf, '\n');
                if (flags & VERBOSE)
                        sprintf(REPLY, "Running ... (%.40s)", buf);
                drawscm(TRUE);
                nreply = 0;
                if (getchr(combuf, TRUE) > 0) {
                        if (_pskbchr == INTRPT) {
```

```
                                        strcpy(REPLY, "Interrupt(R)!");
                                        intrpt = TRUE;
                                        break;
                                }
                                drawem(TRUE);
                                procom(combuf);
                        }
                        procom(buf);
                }
                _ignornull = TRUE;
                pclose(fd);
                return(0);
}

/*
 * Notimp - prints a NOT IMPLEMENTED message.
 */
notimp(notcom, level)
int     notcom, level;
{
        strcpy(REPLY, "Command has not been implemented.");
        return(-1);
        /* ARGSUSED */
}

/*
 * Shift - move the models.
 */
shift(def, level)
int     def, level;
{
        char    axis;
        char    buf[BUFSIZE];
        int     val;
        float   t;
        struct  turn_def        *ptr;

        val = getword(buf, sizeof buf, TRUE, level);
        if (def == MOVEMENT) {                  /* Movement */
                axis = 0;                       /* Default X-axis */
                if (val <= 0) {
                        if (flags & VERBOSE)
                                strcpy(REPLY, "Missing specifications.");
                        return(-1);
                }
                switch (buf[0]) {
                        case 'x' :
                        case 'X' :
                                axis = 0;
                                val = getword(buf, sizeof buf, TRUE, level);
                                break;
                        case 'y' :
                        case 'Y' :
                                axis = 1;
                                val = getword(buf, sizeof buf, TRUE, level);
                                break;
                        case 'z' :
                        case 'Z' :
                                axis = 2;
                                val = getword(buf, sizeof buf, TRUE, level);
                                break;
                }
                ptr = &movecoms.m_tran[axis];
        } else if (def == SECTION)              /* Section */
                ptr = &movecoms.m_section;
        else if (def == THICKNESS)              /* Thickness */
                ptr = &movecoms.m_thickness;
        else
                ptr = &movecoms.m_scale;
        if (val <= 0) {
                ptr->t_speed = MAPCRD(1.0);
                ptr->t_frames = 1;
                ptr->t_wait = 0;
                goto done;
        }
        if (sscanf(buf, "%f", &t) != 1) {
                sprintf(REPLY, "Unexpected argument '%s'.", buf);
                return(-1);
        }
        ptr->t_speed = MAPCRD(t);
        if (ptr->t_speed == 0) {
                ptr->t_frames = 0;
                goto done;
        }
        ptr->t_frames = 1;
        ptr->t_wait = 0;
        if (getword(buf, sizeof buf, TRUE, level) <= 0)
                goto done;
        ptr->t_frames = atoi(buf);
        if (getword(buf, sizeof buf, TRUE, level) > 0)
                ptr->t_wait = atoi(buf);
done:
        if (slow_fac != 0)
                slow_down(ptr, T_SHIFT);
```

```
	return(0);
}

/*
 * Turn - turn the model.
 */
turn(def, level)
int	def, level;
{
	char	axis;
	char	buf[BUFSIZE];
	int	val;
	float	t;
	struct	turn_def		*array;

	array = movecoms.m_rot;
	axis = 1;				/* Default Y-axis */
	if ((val = getword(buf, sizeof buf, TRUE, level)) <= 0) {
		array[axis].t_speed = MAPANGLE(3);
		array[axis].t_frames = def;
		array[axis].t_wait = 0;
		goto done;
	}
	switch (buf[0]) {
		case 'x' :
		case 'X' :
			axis = 0;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'y' :
		case 'Y' :
			axis = 1;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case 'z' :
		case 'Z' :
			axis = 2;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
		case '0' :
		case '1' :
		case '2' :
		case '3' :
		case '4' :
		case '5' :
		case '6' :
		case '7' :
		case '8' :
		case '9' :
			axis = buf[0] - '0';
			if ((rotation[axis].r_stat & ACTIVE) == 0) {
				sprintf(REPLY, "Rotation %d is not active.",
					axis);
				return(-1);
			}
			array = movecoms.m_inrot;
			val = getword(buf, sizeof buf, TRUE, level);
			break;
	}
	if (val <= 0) {
		array[axis].t_speed = MAPANGLE(3);
		array[axis].t_frames = def;
		array[axis].t_wait = 0;
		goto done;
	}
	if (sscanf(buf, "%f", &t) != 1) {
		sprintf(REPLY, "Unexpected argument '%s'.", buf);
		return(-1);
	}
	if (t == 0) {
		array[axis].t_speed = 0;
		array[axis].t_frames = 0;
		array[axis].t_wait = 0;
		goto done;
	}
	while (t > 180.)
		t -= 360.;
	while (t < -180.)
		t += 360.;
	array[axis].t_speed = MAPANGLE(t);
	array[axis].t_frames = def;
	array[axis].t_wait = 0;
	if (getword(buf, sizeof buf, TRUE, level) <= 0)
		goto done;
	array[axis].t_frames = atoi(buf);
	if (getword(buf, sizeof buf, TRUE, level) > 0)
		array[axis].t_wait = atoi(buf);
done:
	if (slow_fac != 0)
		slow_down(&array[axis], T_ANGLE);
	return(0);
}

/*
```

procom.c

```
 * Dofreeze - stop all movements on the screen
 */
dofreeze(notcom, level)
int     notcom, level;
{
        register        i;

        if (notcom) {
                strcpy(REPLY, "Don't freeze ???");
                return(-1);
        }
        getknob(0, 3);
        for (i = 0; i < 3; i++) {
                movecoms.m_rot[i].t_wait = movecoms.m_rot[i].t_frames = 0;
                movecoms.m_rock[i].t_wait = movecoms.m_rock[i].t_frames = 0;
                movecoms.m_tran[i].t_wait = movecoms.m_tran[i].t_frames = 0;
        }
        for (i = 0; i < MAXINROT; i++)
                movecoms.m_inrot[i].t_wait = movecoms.m_inrot[i].t_frames = 0;
        for (i = 0; i < 6; i++)
                movecoms.m_clip[i].t_wait = movecoms.m_clip[i].t_frames = 0;
        movecoms.m_scale.t_wait = movecoms.m_scale.t_frames = 0;
        movecoms.m_section.t_wait = movecoms.m_section.t_frames = 0;
        movecoms.m_thickness.t_wait = movecoms.m_thickness.t_frames = 0;
#if     GRAFENG == IRIS
        freeze_knobs();
#endif
        return(0);
        /* ARGSUSED */
}

#define  MAXLINELEN     66              /* About right for the PS2 */
/*
 * Doalias - make entries in the alias table
 */
doalias(notcom, level)
int     notcom, level;
{
        register        i;
        int     len, total;
        char    buf[BUFSIZE], *def;
        char    editcom[BUFSIZE], answer[BUFSIZE];

        if (notcom) {                                   /* Remove an alias */
                strcpy(editcom, "~alias");
                answer[0] = '\0';
                total = 0;
                while (getword(buf, sizeof buf, FALSE, level) > 0) {
                        for (i = 0; i < nalias; i++)
                                if (strcmp(atab[i].a_name, buf) == 0)
                                        break;
                        if (i >= nalias) {
                                sprintf(REPLY, "No such alias: '%s'", buf);
                                continue;
                        }

                        strcat(editcom, " ");
                        strcat(editcom, atab[i].a_name);

                        if (total > 0)
                                strcat(answer, ", ");
                        else
                                strcat(answer, "");
                        strcat(answer, atab[i].a_name);
                        strcat(answer, "");

                        free(atab[i].a_name);
                        free(atab[i].a_line);
                        nalias--;
                        atab[i] = atab[nalias];
                        total++;
                }

                if (total > 0)
                        sendcom(editcom);
                sprintf(REPLY, "Alias%s %s removed",
                        (total > 1) ? "es" : "", answer);
        } else {                                        /* Add an alias */
                if (getword(buf, sizeof buf, FALSE, level) <= 0) {
                        sprintf(REPLY, "There are %d aliases :", nalias);
                        total = -2;
                        answer[0] = '\0';
                        for (i = 0; i < nalias; i++) {
                                len = strlen(atab[i].a_name) + 2;
                                if (total + len > MAXLINELEN) {
                                        strcpy(REPLY, answer);
                                        total = -2;
                                        answer[0] = '\0';
                                }
                                if (total > 0)
                                        strcat(answer, ", ");
                                strcat(answer, atab[i].a_name);
                                total += len;
                        }
```

```
                        if (total > 0)
                                strcpy(REPLY, answer);
                        return(0);
                }

                /* Is it possible to add it ? */
                for (i = 0; i < nalias; i++)
                        if (strcmp(atab[i].a_name, buf) == 0)
                                break;
                def = getinput(level);
                if (*def == '\0') {
                        if (i < nalias)
                                sprintf(REPLY, "'%s'   '%s'", atab[i].a_name,
                                        atab[i].a_line);
                        else
                                sprintf(REPLY, "No such alias: '%s'", buf);
                        return(0);
                }
                if (i >= nalias) {
                        nalias++;
                        if (nalias >= MAXALIAS) {
                                strcpy(REPLY, "Too many aliases.");
                                nalias--;
                                return(-1);
                        }
                }

                /* Enter alias into table */
                atab[i].a_name = malloc((unsigned) (strlen(buf) + 1));
                strcpy(atab[i].a_name, buf);
                atab[i].a_line = malloc((unsigned) (strlen(def) + 1));
                strcpy(atab[i].a_line, def);

                /* The editor should echo the alias definition
                sprintf(REPLY, "aliased '%s' to '%s'", atab[i].a_name,
                        atab[i].a_line); */
                sprintf(editcom, "alias %s %s", buf, def);
                sendcom(editcom);
        }
        return(0);
}

/*
 * Doalign - align two atoms on the z-axis
 */
doalign(notcom, level)
int     notcom, level;
{
        char    editcom[BUFSIZE];
        int     *ip;
        int     count;
        int     from, to;
        float   *fp;
        double  coord[2][3];
        ps_t    pscrd[2][4], newcrd[2][4];
        ps_t    mat[4][4], invmat[4][4];
        register        i, j;

        if (notcom) {
                strcpy(REPLY, "Don't align ???");
                return(-1);
        }

        strcpy(editcom, "align ");
        strcat(editcom, getinput(level));
        sendcom(editcom);

        ip = (int *) cheatbuf;
        count = *ip++;
        switch (count) {
                case 1 :
                case -1 :
                        from = to = *ip++;
                        fp = (float *) ip;
                        for (j = 0; j < 3; j++)
                                coord[0][j] = *fp++;
                        for (j = 0; j < 3; j++)
                                coord[1][j] = *fp++;
                        break;
                case 2 :
                case -2 :
                        from = *ip++;
                        fp = (float *) ip;
                        for (j = 0; j < 3; j++)
                                coord[0][j] = *fp++;
                        ip = (int *) fp;
                        to = *ip++;
                        fp = (float *) ip;
                        for (j = 0; j < 3; j++)
                                coord[1][j] = *fp++;
                        break;
                default :
                        sprintf(REPLY, "Bad atom specifications(%d).", count);
                        return(-1);
```

```
        }

        for (j = 0; j < 3; j++) {
                pscrd[0][j] = MAPCRD(coord[0][j]) - models[from].com[j];
                pscrd[1][j] = MAPCRD(coord[1][j]) - models[to].com[j];
        }
        pscrd[0][3] = pscrd[1][3] = ONE;
        transform(models[from].matrix, pscrd[0], newcrd[0], ONE);
        transform(models[to].matrix, pscrd[1], newcrd[1], ONE);

        i = 0;
        for (j = 0; j < 3; j++) {
                pscrd[0][j] = 0;
                if (count > 0)
                        pscrd[1][j] = (newcrd[1][j] + models[to].tr[j]) -
                                (newcrd[0][j] + models[from].tr[j]);
                else
                        pscrd[1][j] = (newcrd[0][j] + models[from].tr[j]) -
                                (newcrd[1][j] + models[to].tr[j]);
                if (pscrd[1][j] != 0)
                        i++;
        }
        if (i == 0) {
                strcpy(REPLY, "Cannot align superimposed atoms.");
                return(-1);
        }
#if     GRAFENG != IRIS
        lookat(mat, invmat, pscrd[0], pscrd[1]);
#else
        m_lookat(mat, invmat, pscrd[0], pscrd[1]);
#endif

        if (count > 0) {
                transform(mat, newcrd[0], pscrd[0], ONE);
                for (j = 0; j < 3; j++)
                        pscrd[0][j] += models[from].com[j] +
                                models[from].tr[j] - cowin[j];
        } else {
                transform(mat, newcrd[1], pscrd[0], ONE);
                for (j = 0; j < 3; j++)
                        pscrd[0][j] += models[to].com[j] +
                                models[to].tr[j] - cowin[j];
                from = to;
        }
#if     GRAFENG == IRIS
        pushmatrix();
#endif
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active)
                        continue;
#if     GRAFENG == PS2
                bldcon(LOADMAT, mat);
                bldcon(CONCATMAT, models[i].matrix);
                bldcon(STOREMAT, models[i].matrix);
#endif
#if     GRAFENG == MPS
                tset(mat);
                tcon(models[i].matrix);
                tget(models[i].matrix);
#endif
#if     GRAFENG == IRIS
                loadmatrix(mat);
                multmatrix(models[i].matrix);
                getmatrix(models[i].matrix);
#endif
                if (i != from) {
                        for (j = 0; j < 3; j++)
                                pscrd[1][j] = (models[i].com[j] +
                                        models[i].tr[j]) -
                                        (models[from].com[j] +
                                        models[from].tr[j]);
                        transform(mat, pscrd[1], newcrd[1], ONE);
                        for (j = 0; j < 3; j++)
                                models[i].tr[j] = newcrd[1][j] +
                                        models[from].com[j] +
                                        models[from].tr[j] -
                                        models[i].com[j];
                }
                for (j = 0; j < 3; j++)
                        models[i].tr[j] -= pscrd[0][j];
        }
#if     GRAFENG == IRIS
        popmatrix();
#endif
        uncheat();
        calcent(selection);
        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE, (ps_t *) NULL);
        return(0);
}


#ifndef vms
/*
 * dosleep - sleep the specified number of seconds
```

```
 */
dosleep(notcom, level)
int     notcom, level;
{
        char    buf[BUFSIZE];
        int     nsec;

        if (notcom) {
                strcpy(REPLY, "I'm awake.");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                strcpy(REPLY, "Usage: sleep number_of_seconds");
                return(-1);
        }
        if (sscanf(buf, "%d", &nsec) != 1 || nsec <= 0) {
                strcpy(REPLY, "Usage: sleep number_of_seconds");
                return(-1);
        }
        if (getword(buf, sizeof buf, TRUE, level) > 0) {
                strcpy(REPLY, "Usage: sleep number_of_seconds");
                return(-1);
        }
        sleep((unsigned int) nsec);
        return(0);
}
#else   vms
/*
 * VMS implements the sleep() function using the
 * SYS$HIBERNATE feature.  The midas rstfram()
 * routine also uses SYS$HIBERNATE and this usage
 * conflicts with the one above.  Hence the
 * midas "sleep" comand has not been implemented
 * under VMS.
 */
#endif  vms

/*
 * Doslowdown - change the slow-down factor
 */
doslowdown(notcom, level)
int     notcom, level;
{
        register int    i, j;
        auto char       buf[BUFSIZE];
        extern int      slow_fac;

        if (notcom)
                slow_fac = 0;
        else {
                if (getword(buf, sizeof buf, TRUE, level) <= 0) {
                        sprintf(REPLY, "Slowdown factor is currently %d.",
                                slow_fac);
                        return(0);
                }
                i = atoi(buf);
                if (i < 0) {
                        sprintf(REPLY, "Illegal slowdown factor %d.", i);
                        return(-1);
                }
                slow_fac = i;
        }
        sprintf(REPLY, "Slowdown factor set to %d", slow_fac);
        return(0);
}

#ifdef  BSD
#include <sys/time.h>
#include <sys/resource.h>

/*
 * dovmstat - starts/reports amount of time/resources used
 */
dovmstat(notcom, level)
int     notcom, level;
{
        static  struct  rusage  start, finish;
        int     utimes, utimeu, stimes, stimeu;
        int     nvcsw, nivcsw, nminflt, nmajflt;

        if (notcom) {                   /* report resource usage */
                getrusage(RUSAGE_SELF, &finish);
                utimes = finish.ru_utime.tv_sec - start.ru_utime.tv_sec;
                utimeu = finish.ru_utime.tv_usec - start.ru_utime.tv_usec;
                if (utimeu < 0) {
                        utimes--;
                        utimeu += 1000000L;
                }
                stimes = finish.ru_stime.tv_sec - start.ru_stime.tv_sec;
                stimeu = finish.ru_stime.tv_usec - start.ru_stime.tv_usec;
                if (stimeu < 0) {
                        stimes--;
                        stimeu += 1000000L;
                }
```

```
                nvcsw = finish.ru_nvcsw - start.ru_nvcsw;
                nivcsw = finish.ru_nivcsw - start.ru_nivcsw;
                nminflt = finish.ru_minflt - start.ru_minflt;
                nmajflt = finish.ru_majflt - start.ru_majflt;
                sprintf(REPLY, "%d.%06du %d.%06ds %dvc %divc %dlpf %dbpf",
                        utimes, utimeu, stimes, stimeu,
                        nvcsw, nivcsw, nminflt, nmajflt);
        }
        else                    /* record base counters */
                getrusage(RUSAGE_SELF, &start);
        return(0);
        /* ARGSUSED */
}

doframecnt(notcom, level)
int     notcom, level;
{
        char            buf[BUFSIZE];
        int             fps, hang;
        static time_t   start, finish;
        time_t          time();

        if (notcom)
                start = time((time_t *) NULL);
        else {
                finish = time((time_t *) NULL);
                hang = 0;
                fps = 24;
                while (getword(buf, sizeof buf, TRUE, level) > 0) {
                        if (strcmp(buf, "wait") == 0)
                                hang = 1;
                        else if (sscanf(buf, "%d", &fps) != 1) {
                                strcpy(REPLY, "Expecting integer or 'wait'");
                                return -1;
                        }
                }
                fprintf(stderr, "%d seconds @ %d fps = %d frames\n",
                        finish - start, fps, (finish - start) * fps);
                if (hang) {
                        fputs("Hit return to continue: ", stderr);
                        fgets(buf, sizeof buf, stdin);
                }
        }
        return 0;
}

domovie(notcom, level)
int     notcom, level;
{
        char    buf[BUFSIZE];
        int     rpf;                    /* Refreshes per frame */

        if (notcom) {
                flags &= ~MMOVIE;
                return 0;
        }
        if (getword(buf, sizeof buf, TRUE, level) > 0) {
                if (sscanf(buf, "%d", &rpf) != 1) {
                        strcpy(REPLY, "Expecting # refreshes per frame.");
                        return -1;
                }
                s_expose = rpf;
        }
        else
                s_expose = S_EXPOSE;
        flags |= MMOVIE;
        return 0;
}
#endif  BSD

dorepeat()
{
        strcpy(REPLY, "Use ^C or ^P instead of repeat.");
        return(-1);
}
```

matrix.c

```
/* $Header: matrix.c,v 3.19 86/05/09 15:52:23 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          15 May 1983
 * Release 2.2                          26 Jul 1983
 */

#include "intr.h"

#if     GRAFENG != IRIS
#define ROUND(x)        (int) ((x) < 0 ? (x) - 0.5 : (x) + 0.5)
#ifndef ONE
#define ONE             K32K
#endif  ONE
#else   IRIS
#define ROUND(x)        (x)
#define ONE             1.0
#endif  IRIS

#if     GRAFENG == IRIS
#undef  HARDTRAN
#endif  IRIS

/*
 * Makewin - recalculates the entire window.
 */
makewin()
{
        register        i, j;
        double  t;
        long    natom;

        /* Find the center of mass of the system */

        natom = 0;
        for (j = 0; j < 3; j++)
                com[j] = 0.;
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active)
                        continue;
                for (j = 0; j < 3; j++)
                        com[j] += (models[i].com[j] + models[i].tr[j])
                                * (double) models[i].nused;
                natom += models[i].nused;
        }
        if (natom > 0) {
                for (j = 0; j < 3; j++) {
                        com[j] /= natom;
                        cowin[j] = ROUND(com[j]);
                }

                /* Find the maximum distance from the center of the window and
                 * leave some safety margin so that the entire picture will
                 * fit comfortably on the screen */

                width = 0.;
                for (i = 0; i < MAXMOD; i++) {
                        if (!models[i].active)
                                continue;
                        for (j = 0; j < 3; j++) {
                                t = ABS(com[j] - models[i].com[j]
                                        - models[i].tr[j])
                                        + MAPCRD(models[i].range);
                                if (t > width)
                                        width = t;
                        }
#ifdef  DEBUG
                        if (flags & DDEBUG)
                                fprintf(stderr,
#if     GRAFENG == IRIS
                                        "Mod %d = (%.2f,%.2f,%.2f,%.2f)\n",
#else   IRIS
                                        "Mod %d = (%d,%d,%d,%.2f)\n",
#endif  IRIS
                                        i,
                                        models[i].com[0] + models[i].tr[0],
                                        models[i].com[1] + models[i].tr[1],
                                        models[i].com[2] + models[i].tr[2],
                                        MAPCRD(models[i].range));
#endif  DEBUG
                }
                width *= WRATIO;
#if     GRAFENG != IRIS
                for (j = 0; j < 3; j++) {
                        if (width + com[j] > K32K)
                                width = K32K - com[j];
                        if (com[j] - width < -K32K)
                                width = com[j] + K32K;
                }
#endif  IRIS
```

```
        } else {
                for (j = 0; j < 3; j++) {
                        com[j] = 0;
                        cowin[j] = 0;
                }
#if     GRAFENG != IRIS
                width = K32K;
#else   IRIS
                width = YMAXSCREEN;
#endif  IRIS
        }
#ifdef  DEBUG
        if (flags & DDEBUG) {
                fprintf(stderr, "Com = (%.2f,%.2f,%.2f)\n", com[0], com[1],
                                com[2]);
                fprintf(stderr, "Range = %.2f\n", width);
        }
#endif  DEBUG

        /* Clipping is reset to the window boundaries */

        clip[C_LEFT]    = win[C_LEFT]   = -width;
        clip[C_RIGHT]   = win[C_RIGHT]  = width;
        clip[C_BOTTOM]  = win[C_BOTTOM] = -width;
        clip[C_TOP]     = win[C_TOP]    = width;
#if     GRAFENG != IRIS
        clip[C_HITHER]  = win[C_HITHER] = -width;
        clip[C_YON]     = win[C_YON]    = width;
#else   IRIS
        clip[C_HITHER]  = win[C_HITHER] = width;
        clip[C_YON]     = win[C_YON]    = -width;
#endif  IRIS
#if     GRAFENG != IRIS
        if (win[C_HITHER] < CUTOFF)
                clip[C_HITHER] = win[C_HITHER] = CUTOFF;
#endif  IRIS

        calcwin();
}

/*
 * Calcent - calculates the center of mass of the SELECTED models.
 */
calcent(selected)
unsigned        selected;
{
        register        i, j;
        double  tot;
        double  fc[3];

        if (setcofr)
                return;

        for (j = 0; j < 3; j++)
                fc[j] = 0.;
        tot = 0.;
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || (selected & MODELSW(i)) == 0)
                        continue;
                for (j = 0; j < 3; j++) {
                        fc[j] += (models[i].com[j] + models[i].tr[j])
                                * (double) models[i].nused;
                }
                tot += models[i].nused;
#ifdef  DEBUG
                if (flags & DDEBUG)
                        fprintf(stderr,
#if     GRAFENG != IRIS
                                "Model %d %d %d+%d %d+%d %d+%d\n",
#else   IRIS
                                "Model %d %d %.2f+%.2f %.2f+%.2f %.2f+%.2f\n",
#endif  IRIS
                                i, models[i].nused,
                                models[i].com[0], models[i].tr[0],
                                models[i].com[1], models[i].tr[1],
                                models[i].com[2], models[i].tr[2]);
#endif  DEBUG
        }
        if (tot > 0) {
                fc[0] /= tot;
                cx = ROUND(fc[0]);
                fc[1] /= tot;
                cy = ROUND(fc[1]);
                fc[2] /= tot;
                cz = ROUND(fc[2]);
        }
#ifdef  DEBUG
        if (flags & DDEBUG)
#if     GRAFENG != IRIS
                fprintf(stderr, "Center of rotation: %d %d %d\n", cx, cy, cz);
#else   IRIS
                fprintf(stderr, "Center of rotation: %.2f %.2f %.2f\n", cx, cy,
                                cz);
#endif  IRIS
```

```
#endif	DEBUG
}

/*
 * Calcwin - recalculates the window without resetting the window
 * size or clipping planes.
 */

# if	GRAFENG == PS2
# define		HALF(x)		((ps_t) (((x) * 0.5) + 0.5))
# endif
# if	GRAFENG == IRIS
# define		HALF(x)		((x) * 0.5)
# endif

calcwin()
{
	register double	t, nv, nw, nwidth;

	if (width <= 0)
		return;

	nwidth = scaleval / (double) INITSCALE * width;
#if	GRAFENG != IRIS
	t = -nwidth - nwidth * DRATIO;
	if (t < -K32K)
		weye = -K32K;
	else
		weye = t;
#else	IRIS
	weye = nwidth + nwidth * DRATIO;
#endif	IRIS

	wh = clip[C_HITHER] / width * nwidth;
	wy = clip[C_YON] / width * nwidth;
#if	GRAFENG != IRIS
	if (wh < weye)
		wh = weye;
	if (wy > K32K)
		wy = K32K;
	nwidth = (wh - weye) / (double) DRATIO;
#else	IRIS
	if (wh > weye)
		wh = weye;
	nwidth = (weye - wh) / (double) DRATIO;
#endif	IRIS
	wt = wr = nwidth;
	wb = wl = -nwidth;
	nw = nwidth / width;
#if	GRAFENG == PS2 || GRAFENG == IRIS
	if (flags & STPAIR) {
		auto vp_coord	nvl, nvr, nvb, nvt;

		right_vp(&nvl, &nvr, &nvb, &nvt);
		t = (double) (nvt - nvb) / (double) (nvr - nvl);
		wb = wb * t;	/* DON'T use *= since it can cause unexpected
				   conversion errors when typeof wb != double */
		wt = -wb;
	}
#endif

	vt = vr = VIEWHILIM;
	vb = vl = VIEWLOLIM;
#if	GRAFENG != IRIS
	nv = K2K / width;
#else
	nv = VIEWHILIM / width;
#endif

	if (clip[C_LEFT] > win[C_LEFT]) {
		t = clip[C_LEFT] - win[C_LEFT];
		wl += t * nw;
		vl += t * nv;
	}

	if (clip[C_RIGHT] < win[C_RIGHT]) {
		t = win[C_RIGHT] - clip[C_RIGHT];
		wr -= t * nw;
		vr -= t * nv;
	}

	if (clip[C_BOTTOM] > win[C_BOTTOM]) {
		t = clip[C_BOTTOM] - win[C_BOTTOM];
		wb += t * nw;
		vb += t * nv;
	}

	if (clip[C_TOP] < win[C_TOP]) {
		t = win[C_TOP] - clip[C_TOP];
		wt -= t * nw;
		vt -= t * nv;
	}
```

matrix.c

```
        if (flags & SHOWCOFG)
                makegnomon();
#ifdef  DEBUG
        if (flags & DDEBUG) {
#if     GRAFENG != IRIS
                fprintf(stderr, "Window(%d,%d,%d,%d,%d,%d,%d)\n",
                        wl, wr, wb, wt, wh, wy, weye);
                fprintf(stderr, "Vwport(%d,%d,%d,%d,%d)\n",
                        vl, vr, vb, vt, vh, vy);
#else   IRIS
                fprintf(stderr,
                        "Window(%.2f,%.2f,%.2f,%.2f,%.2f,%.2f), weye = %.2f\n
                        wl, wr, wb, wt, wh, wy, weye);
                fprintf(stderr, "Vwport(%d,%d,%d,%d)\n", vl, vr, vb, vt);
#endif  IRIS
        }
#endif  DEBUG
}

/*
 * Fixtran - makes the current orientations the base orientation
 */
fixtran(selected)
unsigned        selected;
{
        register        i, j;
        ps_t    loc[4], new[4];
#ifdef  HARDTRAN
        int     ps2loc;
#endif  HARDTRAN

#if     GRAFENG == PS2
        bldcon(LOADMAT, &matbuf[genrot[0]]);
#endif  PS2
#if     GRAFENG == MPS
        tset(&matbuf[genrot[0]]);
#endif  MPS
#if     GRAFENG == IRIS
        pushmatrix();
        loadmatrix(&matbuf[genrot[0]]);
#endif  IRIS
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || (selected & MODELSW(i)) == 0)
                        continue;
#if     GRAFENG == PS2
                push();
                bldcon(CONCATMAT, models[i].matrix);
                bldcon(STOREMAT, models[i].matrix);
                pop();
                errcheck(models[i].matrix, 1);
#endif  PS2
#if     GRAFENG == MPS
                tpush();
                tcon(models[i].matrix);
                tget(models[i].matrix);
                tpop();
                errcheck(models[i].matrix, 1);
#endif  MPS
#if     GRAFENG == IRIS
                pushmatrix();
                multmatrix(models[i].matrix);
                getmatrix(models[i].matrix);
                popmatrix();
#endif  IRIS
        }
        if (!(flags & INDEP)) {
#ifdef  HARDTRAN
                wbtmem(1);
                ps2loc = 0;
                bldcon(LOADMAT, &matbuf[genrot[0]]);
#endif  HARDTRAN
                for (i = 0; i < MAXMOD; i++) {
                        if (!models[i].active || (selected & MODELSW(i)) == 0)
                                continue;
                        loc[0] = models[i].com[0] + models[i].tr[0] - cx;
                        loc[1] = models[i].com[1] + models[i].tr[1] - cy;
                        loc[2] = models[i].com[2] + models[i].tr[2] - cz;
                        loc[3] = ONE;
#ifdef  HARDTRAN
                        moveto(loc[0], loc[1], loc[2]);
                        rdtc(ps2loc, new);
                        ps2loc += 4;
                        for (j = 0; j < 3; j++)
                                new[j] = new[j] * ((double) K32K) / new[3];
#else   HARDTRAN
                        transform(&matbuf[genrot[0]], loc, new, ONE);
#endif  HARDTRAN
                        models[i].tr[0] = new[0] + (cx - models[i].com[0]);
                        models[i].tr[1] = new[1] + (cy - models[i].com[1]);
                        models[i].tr[2] = new[2] + (cz - models[i].com[2]);
                }
#ifdef  HARDTRAN
                stopwb();
#endif  HARDTRAN
```

```
        }
#if     GRAFENG == IRIS
        popmatrix();
#endif  IRIS
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || (selected & MODELSW(i)) == 0)
                        continue;
                for (j = 0; j < 3; j++)
                        models[i].tr[j] += trans[j];
        }
        if (setcofr) {
                cx += trans[0];
                cy += trans[1];
                cz += trans[2];
        }
        trans[0] = trans[1] = trans[2] = 0;
        _imatx(&matbuf[genrot[0]]);
}

#define NATOM   4

/*
 * Superimpose - superimposes mol0 on mol1.
 */
superimpose(mol0, coord0, mol1, coord1)
int     mol0;
double  coord0[NATOM][3];
int     mol1;
double  coord1[NATOM][3];
{
        double  newcoord1[NATOM][3];
# if    GRAFENG != IRIS
        double  err;
# endif
        double  errdist, delta;
        double  aa[3][3], rotmat[3][3];
        double  cofg[2][3];
        ps_t    c[2][3];
        ps_t    mat[4][4];
        ps_t    loc[4], new[4];
        char    iflag;
        int     ix, iy, iz;
        double  bb, cc, sig, gam, sg, xx;
        double  hypot(), fabs(), sqrt();
        int     k, icnt;
        register        i, j;

for (j = 0; j < 3; j++) {
        for (k = 0; k < 3; k++) {
                aa[j][k] = 0.;
                rotmat[j][k] = 0.;
        }
        rotmat[j][j] = 1.;
        cofg[0][j] = 0.;
        cofg[1][j] = 0.;
}
for (k = 0; k < NATOM; k++)
        for (i = 0; i < 3; i++) {
                cofg[0][i] += coord0[k][i];
                cofg[1][i] += coord1[k][i];
        }
for (i = 0; i < 3; i++) {
        cofg[0][i] /= NATOM;
        cofg[1][i] /= NATOM;
}
for (k = 0; k < NATOM; k++)
        for (i = 0; i < 3; i++) {
                xx = coord1[k][i] - cofg[1][i];
                for (j = 0; j < 3; j++)
                        aa[i][j] += xx * (coord0[k][j] - cofg[0][j]);
        }
icnt = 0;
iflag = FALSE;
ix = 0;
for (;;) {
        if (++icnt > 200)
                return(-1);
        iy = (ix == 2) ? 0  : ix + 1;
        iz = 3 - ix - iy;
        sig = aa[iz][iy] - aa[iy][iz];
        gam = aa[iy][iy] + aa[iz][iz];
        sg = sqrt(sig * sig + gam * gam);
        if (sg != 0.) {
                sg = 1. / sg;
                if (fabs(sig) >= 0.001 * fabs(gam)) {
                        for (i = 0; i < 3; i++) {
                                bb = gam * aa[iy][i] + sig * aa[iz][i];
                                cc = gam * aa[iz][i] - sig * aa[iy][i];
                                aa[iy][i] = bb * sg;
                                aa[iz][i] = cc * sg;
                                bb = gam * rotmat[iy][i] +
                                        sig * rotmat[iz][i];
                                cc = gam * rotmat[iz][i] -
                                        sig * rotmat[iy][i];
```

```
                                        rotmat[iy][i] = bb * sg;
                                        rotmat[iz][i] = cc * sg;
                                }
                                iflag = TRUE;
                        }
                }
                if (++ix > 2) {
                        if (iflag)
                                break;
                        iflag = FALSE;
                        ix = 0;
                }
        }

        errdist = 0;
        for (k = 0; k < NATOM; k++) {
                for (i = 0; i < 3; i++) {
                        newcoord1[k][i] = 0;
                        for (j = 0; j < 3; j++)
                                newcoord1[k][i] += rotmat[i][j] *
                                        (coord1[k][j] - cofg[1][j]);
                        newcoord1[k][i] += cofg[0][i];
                        delta = newcoord1[k][i] - coord0[k][i];
                        errdist += delta * delta;
                }
        }
        (void) sprintf(REPLY, "RMS error between atom sets is %.2f Angstoms.",
                sqrt(errdist) / NATOM);

#if     GRAFENG != IRIS
        mat[3][3] = K16K;
#else   IRIS
        mat[3][3] = 1;
#endif  IRIS
        for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++)
#if     GRAFENG != IRIS
                        mat[i][j] = rotmat[j][i] * K16K;
#else   IRIS
                        mat[i][j] = rotmat[j][i];
#endif  IRIS
                mat[3][i] = mat[i][3] = 0;
                c[0][i] = MAPCRD(cofg[0][i]);
                c[1][i] = MAPCRD(cofg[1][i]);
        }

#if     GRAFENG == PS2
        bldcon(LOADMAT, models[mol0].matrix);
        bldcon(CONCATMAT, mat);
        bldcon(STOREMAT, models[mol1].matrix);
        errcheck(models[mol1].matrix, 1);
#endif  PS2
#if     GRAFENG == MPS
        tset(models[mol0].matrix);
        tcon(mat);
        tget(models[mol1].matrix);
        errcheck(models[mol1].matrix, 1);
#endif  MPS
#if     GRAFENG == IRIS
        pushmatrix();
        loadmatrix(models[mol0].matrix);
        multmatrix(mat);
        getmatrix(models[mol1].matrix);
        popmatrix();
#endif  IRIS

#ifdef  HARDTRAN
        wbtmem(1);
        bldcon(LOADMAT, models[mol0].matrix);
#endif  HARDTRAN
        for (i = 0; i < 3; i++)
                loc[i] = c[0][i] - models[mol0].com[i];
        loc[3] = ONE;
#ifdef  HARDTRAN
        moveto(loc[0], loc[1], loc[2]);
        rdtc(0, new);
        for (i = 0; i < 3; i++)
                new[i] = new[i] * ((double) K32K) / new[3];
#else   HARDTRAN
        transform(models[mol0].matrix, loc, new, ONE);
#endif  HARDTRAN
        for (i = 0; i < 3; i++)
                c[0][i] = models[mol0].com[i] + new[i];

#ifdef  HARTRAN
        bldcon(LOADMAT, models[mol1].matrix);
#endif  HARDTRAN
        for (i = 0; i < 3; i++)
                loc[i] = c[1][i] - models[mol1].com[i];
        loc[3] = ONE;
#ifdef  HARDTRAN
        moveto(loc[0], loc[1], loc[2]);
        rdtc(8, new);
        for (i = 0; i < 3; i++)
```

```
                new[i] = new[i] ((double) K32K) / new[3];
#else   HARDTRAN
        transform(models[mol1].matrix, loc, new, ONE);
#endif  HARDTRAN
        for (i = 0; i < 3; i++)
                c[1][i] = models[mol1].com[i] + new[i];
#ifdef  HARDTRAN
        stopwb();
#endif  HARDTRAN

        for (i = 0; i < 3; i++)
                models[mol1].tr[i] = c[0][i] - c[1][i] + models[mol0].tr[i];
        calcent(selection);
        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE, (ps_t *) NULL);
        return(0);
}

#if     GRAFENG != IRIS
#ifdef  DEBUG
prmats(s)
char    *s;
{
        register        i, j, k;

        if (flags & DDEBUG) {
                fputs(s, stderr);
                fprintf(stderr, "genrot buffer\n");
                for (j = 0; j < 4; j++) {
                        for (k = 0; k < 4; k++)
                                fprintf(stderr, "\t%d",
                                        matbuf[genrot[0]+j*4+k]);
                        putc('\n', stderr);
                }
                for (i = 0; i < MAXMOD; i++) {
                        if (!models[i].active)
                                continue;
/*
                        fprintf(stderr, "%d buffer\n", i);
                        for (j = 0; j < 4; j++) {
                                for (k = 0; k < 4; k++)
                                        fprintf(stderr, "\t%d",
                                                matbuf[molrot[i]+j*4+k]);
                                putc('\n', stderr);
                        }
*/
                        fprintf(stderr, "%d matrix\n", i);
                        for (j = 0; j < 4; j++) {
                                for (k = 0; k < 4; k++)
                                        fprintf(stderr, "\t%d",
                                                models[i].matrix[j][k]);
                                putc('\n', stderr);
                        }
                }
        }
#endif  DEBUG
#endif  IRIS
```

move.c

```
/* $Header: move.c,v 3.19 86/06/04 17:26:34 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "intr.h"

/*
 * Movem - reads the input devices and executes whatever movement is necessary.
 */
movem()
{
        register int                    i, j;
        register unsigned int           newswitch;
        register double                 t, r;
        register int                    changed;
        register int                    anytran;
        register int                    anyrot;
#if     GRAFENG == IRIS
        register Object                 matnam;
#endif  IRIS
        static ps_t                     degtran[3];
        static ps_t                     degrot[3];
        static fps_t                    nclip[6];

        anyrot = anytran = FALSE;
        newswitch = fswitch();
#if     GRAFENG == IRIS
        changed = check_cntrl();
#else
        changed = FALSE;
        if (newswitch & RESETSW)
                getknob(0, 3);
        getknob(0, 0);                          /* Read in knob values */
#endif
        (void) readknob(noknob + 1);                    /* Reset values */

        /* Check the switches */

        if (newswitch != switches) {
                if ((newswitch & MODSEL) != (switches & MODSEL))
                        fixtran(selection);
                selection = newswitch;
                switches = newswitch;
                calcent(selection);
                if (flags & DLIGHTS)
                        lights((int) selection);
        }

        /* Check for global rotation */

        for (i = 0; i < 3; i++) {
                degrot[i] = readknob(knobs.k_rot[i]) * K_ANGLE;
                if (degrot[i] != 0)
                        anyrot = TRUE;
                else
                        degrot[i] = 0;

                if (movecoms.m_rot[i].t_frames != 0) {
                        if (movecoms.m_rot[i].t_wait > 0)
                                movecoms.m_rot[i].t_wait--;
                        else {
                                if (movecoms.m_rot[i].t_frames > 0)
                                        movecoms.m_rot[i].t_frames--;
                                degrot[i] += movecoms.m_rot[i].t_speed;
                                anyrot = TRUE;
                        }
                }

                if (movecoms.m_rock[i].t_frames != 0) {
                        if (movecoms.m_rock[i].t_wait > 0)
                                movecoms.m_rock[i].t_wait--;
                        else {
                                if (movecoms.m_rock[i].t_frames > 0)
                                        movecoms.m_rock[i].t_frames--;
                                if (slow_fac <= 1) {
                                        t = sin(rockpos[i]) *
                                           movecoms.m_rock[i].t_speed + 0.5;
                                        r = 0.1;
                                }
                                else {
                                        t = sin(rockpos[i]) *
                                           movecoms.m_rock[i].t_speed /
                                           (double) slow_fac + 0.5;
                                        r = 0.1 / (double) slow_fac;
                                }
                                degrot[i] += t;
                                rockpos[i] += r;
                                if (rockpos[i] >= 2 * PI)
                                        rockpos[i] = 0.;
```

```
                                anyrot = TRUE;
                        }
                }
        }

        /* Check for global translation */

        for (i = 0; i < 3; i++) {
                degtran[i] = readknob(knobs.k_tran[i]) * K_DISTANCE;
                if (degtran[i] != 0)
                        anytran = TRUE;

                if (movecoms.m_tran[i].t_frames != 0) {
                        if (movecoms.m_tran[i].t_wait > 0)
                                movecoms.m_tran[i].t_wait--;
                        else {
                                if (movecoms.m_tran[i].t_frames > 0)
                                        movecoms.m_tran[i].t_frames--;
                                degtran[i] += movecoms.m_tran[i].t_speed;
                                anytran = TRUE;
                        }
                }
        }

        /* Check for clipping */

        for (i = 0; i < 6; i++) {
                j = readknob(knobs.k_clip[i]) * K_DISTANCE;
                nclip[i] = clip[i];
                if (j != 0) {
# if    GRAFENG == IRIS
                        if (i == C_HITHER || i == C_YON)
                                nclip[i] -= j;
                        else
# endif
                                nclip[i] += j;
                        changed = TRUE;
                }

                if (movecoms.m_clip[i].t_frames != 0) {
                        if (movecoms.m_clip[i].t_wait > 0)
                                movecoms.m_clip[i].t_wait--;
                        else {
                                if (movecoms.m_clip[i].t_frames > 0)
                                        movecoms.m_clip[i].t_frames--;
# if    GRAFENG == IRIS
                                if (i == C_HITHER || i == C_YON)
                                        nclip[i] -= movecoms.m_clip[i].t_speed;
                                else
# endif
                                        nclip[i] += movecoms.m_clip[i].t_speed;
                                changed = TRUE;
                        }
                }
        }

        /* Check for sectioning */

        t = readknob(knobs.k_section) * K_DISTANCE;
        if (t != 0) {
# if    GRAFENG == IRIS
                nclip[C_HITHER] += t;
                nclip[C_YON] += t;
# else
                nclip[C_HITHER] -= t;
                nclip[C_YON] -= t;
# endif
                changed = TRUE;
        }
        if (movecoms.m_section.t_frames != 0) {
                if (movecoms.m_section.t_wait > 0)
                        movecoms.m_section.t_wait--;
                else {
                        if (movecoms.m_section.t_frames > 0)
                                movecoms.m_section.t_frames--;
# if    GRAFENG == IRIS
                        nclip[C_HITHER] -= movecoms.m_section.t_speed;
                        nclip[C_YON] -= movecoms.m_section.t_speed;
# else
                        nclip[C_HITHER] += movecoms.m_section.t_speed;
                        nclip[C_YON] += movecoms.m_section.t_speed;
# endif
                        changed = TRUE;
                }
        }

        /* Check for thickness */

        t = (readknob(knobs.k_thickness) * K_DISTANCE) / 2;
        if (t != 0) {
# if    GRAFENG == IRIS
                nclip[C_HITHER] -= t;
                nclip[C_YON] += t;
# else
```

```
                nclip[C_HITHER] += t;
                nclip[C_YON] -= t;
# endif
                changed = TRUE;
        }
        if (movecoms.m_thickness.t_frames != 0) {
                if (movecoms.m_thickness.t_wait > 0)
                        movecoms.m_thickness.t_wait--;
                else {
                        if (movecoms.m_thickness.t_frames > 0)
                                movecoms.m_thickness.t_frames--;
# if    GRAFENG == IRIS
                        nclip[C_HITHER] += movecoms.m_thickness.t_speed;
                        nclip[C_YON] -= movecoms.m_thickness.t_speed;
# else
                        nclip[C_HITHER] -= movecoms.m_thickness.t_speed;
                        nclip[C_YON] += movecoms.m_thickness.t_speed;
# endif
                        changed = TRUE;
                }
        }
        if (changed) {
                clipcheck(&nclip[C_LEFT], &nclip[C_RIGHT], &clip[C_LEFT],
                        &clip[C_RIGHT]);
                clipcheck(&nclip[C_BOTTOM], &nclip[C_TOP], &clip[C_BOTTOM]
                        &clip[C_TOP]);
#if     GRAFENG != IRIS
                clipcheck(&nclip[C_HITHER], &nclip[C_YON], &clip[C_HITHER],
                        &clip[C_YON]);
#else   IRIS
                clipcheck(&nclip[C_YON], &nclip[C_HITHER], &clip[C_YON],
                        &clip[C_HITHER]);
#endif  IRIS
        }

        /* Check for scaling */

        t = readknob(knobs.k_scale) * K_SCALE;
        if (movecoms.m_scale.t_frames != 0) {
                if (movecoms.m_scale.t_wait > 0)
                        movecoms.m_scale.t_wait--;
                else {
                        if (movecoms.m_scale.t_frames > 0)
                                movecoms.m_scale.t_frames--;
                        t += movecoms.m_scale.t_speed;
                }
        }
        if (t != 0) {
#if     GRAFENG == IRIS
                scaleval += t;
                changed = TRUE;
#else   IRIS
                t = scaleval + t;
                if (t > K32K)
                        t = K32K;
                if (t < K1K)
                        t = K1K;
                if (scaleval != t) {
                        scaleval = t;
                        changed = TRUE;
                }
#endif  IRIS
        }

        /* If the clipping, scaling, sectioning or thickness changed,
         * then we have to recalculate the window */

        if (changed)
                calcwin();

        /* Check for internal rotation */

        for (i = 0; i < MAXINROT; i++) {
                if (rotation[i].r_stat == INACTIVE)
                        continue;

                j = (readknob(knobs.k_inrot[i]) * K_ANGLE);
                if (j != 0) {
#ifndef SKF
                        rotation[i].r_angle = rotation[i].r_angle + j;
#else
                        /* faster bond rots for RGF */
                        rotation[i].r_angle += j<<3;
#endif
                        changed = TRUE;
                }

                if (movecoms.m_inrot[i].t_frames != 0) {
                        if (movecoms.m_inrot[i].t_wait > 0)
                                movecoms.m_inrot[i].t_wait--;
                        else {
                                if (movecoms.m_inrot[i].t_frames > 0)
                                        movecoms.m_inrot[i].t_frames--;
                                rotation[i].r_angle +=
```

```
                                                    movecoms.m_inrot[i].t_
                                changed = TRUE;
                        }
                }

#if     GRAFENG == PS2
                getrot(&matbuf[inrot[i]], rotation[i].r_angle, ZAXIS);
#endif  PS2
#if     GRAFENG == MPS
                getrot(&matbuf[inrot[i]], &rotation[i].r_angle, &ZAXIS);
#endif  MPS
#if     GRAFENG == IRIS
                matnam = ('r' << 8) | (i + '0');
                makeobj(matnam);
                        rotate(rotation[i].r_angle, 'z');
                closeobj();
#endif  IRIS
        }

        if (!anyrot && !anytran)
                return(changed);
        if (update(selection, anyrot, degrot, anytran, degtran))
                return(TRUE);
        else
                return(changed);
}

/*
 * Update - performs the matrix operations.
 */
update(selected, anyrot, degrot, anytran, degtran)
unsigned int    selected;
int             anyrot;
ps_t            degrot[3];
int             anytran;
ps_t            degtran[3];
{
        register        i;
        register        anychg;
        static ps_t     imat[16] = { -1 };

        if (imat[0] == -1)
                _imatx(imat);

        if (anytran) {
                for (i = 0; i < 3; i++)
                        if (degtran[i] != 0)
                                trans[i] += degtran[i];
        }

        if (anyrot) {
#if     GRAFENG == PS2
                unit();
                if (degrot[0] != 0)
                        rot(degrot[0], XAXIS);
                if (degrot[1] != 0)
                        rot(degrot[1], YAXIS);
                if (degrot[2] != 0)
                        rot(degrot[2], ZAXIS);
                bldcon(CONCATMAT, &matbuf[genrot[0]]);
                bldcon(STOREMAT, &matbuf[genrot[0]]);
                errcheck(&matbuf[genrot[0]], 0);
#endif  PS2
#if     GRAFENG == MPS
                tident();
                if (degrot[0] != 0)
                        trotx(&degrot[0]);
                if (degrot[1] != 0)
                        troty(&degrot[1]);
                if (degrot[2] != 0)
                        trotz(&degrot[2]);
                tcon(&matbuf[genrot[0]]);
                tget(&matbuf[genrot[0]]);
                errcheck(&matbuf[genrot[0]], 0);
#endif  MPS
#if     GRAFENG == IRIS
                pushmatrix();
                loadmatrix(imat);
                if (degrot[0] != 0)
                        rotate((Angle) degrot[0], 'x');
                if (degrot[1] != 0)
                        rotate((Angle) degrot[1], 'y');
                if (degrot[2] != 0)
                        rotate((Angle) degrot[2], 'z');
                multmatrix(&matbuf[genrot[0]]);
                getmatrix(&matbuf[genrot[0]]);
                popmatrix();
#endif  IRIS
        }

        anychg = 0;
        for (i = 0; i < MAXMOD; i++) {
                if (!models[i].active || (selected & MODELSW(i)) == 0)
                        continue;
```

```
                anychg++;
#if     GRAFENG == PS2
                unit();
                if (flags & INDEP) {
                        tran(trans[0] + models[i].tr[0] + models[i].com[0],
                                trans[1] + models[i].tr[1] + models[i].com[1],
                                trans[2] + models[i].tr[2] + models[i].com[2]);
                        bldcon(CONCATMAT, &matbuf[genrot[0]]);
                        bldcon(CONCATMAT, models[i].matrix);
                        tran(-models[i].com[0], -models[i].com[1],
                                -models[i].com[2]);
                } else {
                        tran(trans[0] + cx, trans[1] + cy, trans[2] + cz);
                        bldcon(CONCATMAT, &matbuf[genrot[0]]);
                        tran(-cx, -cy, -cz);
                        tran(models[i].com[0] + models[i].tr[0],
                                models[i].com[1] + models[i].tr[1],
                                models[i].com[2] + models[i].tr[2]);
                        bldcon(CONCATMAT, models[i].matrix);
                        tran(-models[i].com[0], -models[i].com[1],
                                -models[i].com[2]);
                }
                bldcon(STOREMAT, &matbuf[molrot[i]]);
#endif  PS2
#if     GRAFENG == MPS
                tident();
                if (flags & INDEP) {
                        arg0 = trans[0] + models[i].tr[0] + models[i].com[0];
                        arg1 = trans[1] + models[i].tr[1] + models[i].com[1];
                        arg2 = trans[2] + models[i].tr[2] + models[i].com[2];
                        ttran(&arg0, &arg1, &arg2);
                        tcon(&matbuf[genrot[0]]);
                        tcon(models[i].matrix);
                        arg0 = -models[i].com[0];
                        arg1 = -models[i].com[1];
                        arg2 = -models[i].com[2];
                        ttran(&arg0, &arg1, &arg2);
                } else {
                        arg0 = trans[0] + cx;
                        arg1 = trans[1] + cy;
                        arg2 = trans[2] + cz;
                        ttran(&arg0, &arg1, &arg2);
                        tcon(&matbuf[genrot[0]]);
                        arg0=(-cx); arg1=(-cy); arg2=(-cz);
                        ttran(&arg0, &arg1, &arg2);
                        arg0 = models[i].com[0] + models[i].tr[0];
                        arg1 = models[i].com[1] + models[i].tr[1];
                        arg2 = models[i].com[2] + models[i].tr[2];
                        ttran(&arg0, &arg1, &arg2);
                        tcon(models[i].matrix);
                        arg0 = -models[i].com[0];
                        arg1 = -models[i].com[1];
                        arg2 = -models[i].com[2];
                        ttran(&arg0, &arg1, &arg2);
                }
                tget(&matbuf[molrot[i]]);
#endif  MPS
#if     GRAFENG == IRIS
                pushmatrix();
                loadmatrix(imat);
                if (flags & INDEP) {
                        translate(trans[0] + models[i].tr[0] + models[i].com[0],
                                trans[1] + models[i].tr[1] + models[i].com[1],
                                trans[2] + models[i].tr[2] + models[i].com[2]);
                        multmatrix(&matbuf[genrot[0]]);
                        multmatrix(models[i].matrix);
                        translate(-models[i].com[0], -models[i].com[1],
                                -models[i].com[2]);
                }
                else {
                        translate(trans[0] + cx, trans[1] + cy, trans[2] + cz);
                        multmatrix(&matbuf[genrot[0]]);
                        translate(-cx, -cy, -cz);
                        translate(models[i].com[0] + models[i].tr[0],
                                models[i].com[1] + models[i].tr[1],
                                models[i].com[2] + models[i].tr[2]);
                        multmatrix(models[i].matrix);
                        translate(-models[i].com[0], -models[i].com[1],
                                -models[i].com[2]);
                }
                getmatrix(&matbuf[molrot[i]]);
                popmatrix();
                makeobj((Object) ('m' << 8 | (i + '0')));
                        multmatrix(&matbuf[molrot[i]]);
                closeobj();
#endif  IRIS
                }
                return(anychg);
}

prmat(str, matrix, outf)
char            *str;
register ps_t   matrix[4][4];
register FILE   *outf;
```

move.c

```
{
        register int        i, j;

        if (str != NULL)
                fprintf(outf, "%s:\n", str);
        for (i = 0; i < 4; i++) {
                for (j = 0; j < 4; j++)
                        fprintf(outf,
#if      GRAFENG == IRIS
                                "\t%.2f",
#else
                                "\t%d",
#endif
                                matrix[i][j]);
                putc('\n', outf);
        }
}

#if      GRAFENG == IRIS
/*
 * rot_angle:
 *      Convert the angles used by the picture system routines to those
 *      used by the IRIS 1400.  The ps routines use an unsigned integer
 *      0 <= n <= 2^16 - 1; the iris uses an integer which is 10 * angle.
 */

Angle
rot_angle(angle)
ps_t    angle;
{
        return (Angle) angle * 10.0 + .5;
}
#endif  IRIS
```

frame.c

```
/* $Header: frame.c,v 3.42 86/10/31 13:44:09 pett Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 * Release 2.1                          26 Jul 1983
 * Release 2.2                          28 Aug 1983
 */

#include <ctype.h>
#include "intr.h"
#if     GRAFENG == PS2
#include <sys/ioctl.h>
#include <ps2.h>
#endif  PS2
#define LABELSIZE       3

extern int              _cursor_pos;

/*
 * Makeframe - create the frame object (including the memory usage,
 * and rotation and distance labels).
 */
makeframe()
{
        register int            i, j;
#if     GRAFENG != IRIS
        int             cnt;
#endif  IRIS

        if (nrot == 0 && ndist == 0 && nangle == 0)
                return;

#if     GRAFENG == PS2
        makeob(frame, FRAMESIZE, &cnt);
        charsz(LABELSIZE, 0);
#endif  PS2
#if     GRAFENG == MPS
        makeob(frame, &FRAMESIZE, &cnt);
        csize(&LABELSIZE);
#endif  MPS
#if     GRAFENG == IRIS
        makeobj(O_FRAME);
        ir_color(YELLOW);
        font(SMALLFONT);
#endif  IRIS

        /* Display the rotation and distance labels in the user's selected
         * character size */
        for (i = 0, j = 0; i < MAXINROT; i++) {
                if (rotation[i].r_stat == INACTIVE)
                        continue;
#if     GRAFENG == PS2
                moveto(-30000, 30000 - j * LINEHEIGHT(LABELSIZE), 0);
                text(rotation[i].r_label);
#endif  PS2
#if     GRAFENG == MPS
                arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                damove(&-30000, &arg1, &0);
                ctext(rotation[i].r_label);
#endif  MPS
#if     GRAFENG == IRIS
                cmov2s(COMMAND_X, YMAXSCREEN - (j + 1) * textheight - 2);
                charstr(rotation[i].r_label);
                maketag((Tag) i);
                charstr("");
#endif  IRIS
                j++;
        }
        for (i = 0; i < MAXANGLE; i++) {
                if (angles[i].a_stat == INACTIVE)
                        continue;
#if     GRAFENG == PS2
                moveto(-30000, 30000 - j * LINEHEIGHT(LABELSIZE), 0);
                text(angles[i].a_label);
#endif  PS2
#if     GRAFENG == MPS
                arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                damove(&-30000, &arg1, &0);
                ctext(angles[i].a_label);
#endif  MPS
#if     GRAFENG == IRIS
                cmov2s(COMMAND_X, YMAXSCREEN - (j + 1) * textheight - 2);
                charstr(angles[i].a_label);
                maketag((Tag) MAXINROT + i);
                charstr("");
#endif  IRIS
                j++;
        }
        for (i = 0, j = 0; i < MAXDIST; i++) {
                if (distance[i].d_stat == INACTIVE)
                        continue;
```

```
#if     GRAFENG == PS2
                moveto(0, 30000 - j * LINEHEIGHT(LABELSIZE), 0);
                text(distance[i].d_label);
#endif  PS2
#if     GRAFENG == MPS
                arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                damove(&0, &arg1, &0);
                ctext(distance[i].d_label);
#endif  MPS
#if     GRAFENG == IRIS
                cmov2s(XMAXSCREEN / 2, YMAXSCREEN - (j + 1) * textheight -
                charstr(distance[i].d_label);
                maketag((Tag) MAXINROT + MAXANGLE + i);
                charstr("");
#endif  IRIS
                j++;
        }

#if     GRAFENG != IRIS
        stopob();
#else   IRIS
        closeobj();
#endif  IRIS
#ifdef  DEBUG
#if     GRAFENG != IRIS
        if (flags & DDEBUG)
                fprintf(stderr, "Frame size = %d words,\n", cnt);
#endif  IRIS
#endif  DEBUG
}

/*
 * Drawem sends the picture to the terminal (whether it gets drawn or not
 * depends on whether we are in write-back mode or not).
 */
drawem(psdisp)
int     psdisp;
{
        ps_t    eye;
#if     GRAFENG != IRIS
        extern  int     _memfid;
#else
        register int    i;
#endif

        /* First, send out the transformation matrices */
        if (psdisp) {
#if     GRAFENG == PS2
                lseek(_memfid, 0L, 0);
                write(_memfid, (char *) matbuf, sizeof (matbuf));
#endif
#if     GRAFENG == MPS
                arg0 = psobs->preamble.u_min + 1;
                arg1 = sizeof(matbuf) / sizeof(ps_t);
                uwrit(&arg0, &arg1, matbuf);
#endif
#if     GRAFENG == IRIS
                for (i = 0; i < MAXDIST; i++)
                        memset(distance[i].d_mat, 0, sizeof distance[i].d_mat);
                for (i = 0; i < MAXANGLE; i++)
                        memset(angles[i].a_mat, 0, sizeof angles[i].a_mat);
#endif

                if (ndist > 0 || nangle > 0)
                        calcdist();
        }

#if     GRAFENG == MPS
        psmemfull = 0;                  /* set asynchronously by mpserr() */
#endif  MPS
#if     GRAFENG != IRIS
        drawio(psdisp);                 /* Display replies etc. */
#endif  IRIS

        /* Position the eye of the window */
        eye = (flags & ORTHO) ? wh : weye;

        /* Draw the picture in the requisite manner */
#ifdef  SIA
        if (flags & INSTEREO && psdisp) {

                /* Sequential stereo */

                right();                        /* Right image */
                do_show(rightmat, vl, vr, vb, vt, eye);
                left();                         /* Left image */
                do_show(leftmat, vl, vr, vb, vt, eye);
        } else
#endif  SIA

        if (flags & (STPAIR | INSTEREO)) {

                /* Pair stereo */

                auto vp_coord       nvl, nvr, nvb, nvt;
```

```
                /*
                 * Right image
                 */

                right_vp(&nvl, &nvr, &nvb, &nvt);
                do_show(rightmat, nvl, nvr, nvb, nvt, eye);

                /*
                 * Left image
                 */

                left_vp(&nvl, &nvr);
                do_show(leftmat, nvl, nvr, nvb, nvt, eye);
        }
        else {

                /* Mono view */

                do_show((ps_t *) NULL, vl, vr, vb, vt, eye);
        }
#if     GRAFENG == IRIS
        if (psdisp && (flags & DCONTROL))
                callobj(O_CONTROL);
        drawio(psdisp);                         /* Display replies etc. */
#endif  IRIS

        if (psdisp) {                                   /* Drawing the picture ? */
#if     GRAFENG == IRIS
                swapbuffers();

                /*
                 * The clear is done afterwards as an optimization, so
                 * it can happen in the graphics engine while other
                 * computation is going on.  This also requires a
                 * clear() outside the main loop, and in the dohelp()
                 * function.
                 */

                lr_color(BLACK);
                clear();
#else   IRIS
#if     GRAFENG == PS2
                if (nufram())
#endif  PS2
#if     GRAFENG == MPS
                sdrop();
                if (psmemfull)
#endif  MPS
                {
                        nreply = 0;
                        strcpy(REPLY, "Too much stuff on the screen.");
                }
#if     GRAFENG == PS2
                if (flags & MMOVIE) {
                        shutter(S_OPEN);
                        expose(s_expose);
                        shutter(S_CLOSE);
                }
#endif  PS2
#endif  IRIS
        }
}

/*
 * Drawio - displays the replies and command lines
 */
drawio(showcom)
int     showcom;
{
        extern  int     _pskblen;
        register        i;
#if     GRAFENG != IRIS
        register        j;
#endif
        ps_t    psangle;
        float   angle;
#define VISLNLEN        67
        char    buf[(VISLNLEN < 80) ? 80 : VISLNLEN];

        /* Set up the window and viewport for text */

#if     GRAFENG == PS2
        unit();
        vwport(-2048, 2047, -2048, 2047, 255, 0);
        window(-K32K, K32K, -K32K, K32K);
        huesat(YELLOW, 6);
        charsz(4, 0);
        moveto(20000, -30000, 0);
        text("UCSF MIDAS");             /* Make sure program is identified on photos */
#endif  PS2
#if     GRAFENG == MPS
        tident();
        vbound(&-2048, &2047, &-2048, &2047);
```

```
        vinten(&63, &0);
        twind(&-K32K, &K32K, &-K32K, &K32K, &-K32K, &K32K);
        lspeed(&1);
        lcolor(&48, &6);                /* pale yellow */
        csize(&4);
        damove(&20000, &-30000, &-20000);
        ctext("UCSF MIDAS");            /* Make sure program is identified on photos */
#endif  MPS
#if     GRAFENG == IRIS
        callobj(O_MIDAS);
#endif  IRIS

        /* If the user does not want text AND he didn't type anything
         * at the keyboard, then we don't do anything */

        if (!(flags & (DTEXT | DLABELS | DCONTROL)) && _pskbien <= 0)
                return;

#ifdef  KDEBUG
        if (flags & DDEBUG) {
                readknob(-noknob - 1);
#if     GRAFENG == PS2
                charsz(3, 0);
                moveto(-30000, -31000);
                text(knobuf[0]);
                moveto(-30000, -32000);
                text(knobuf[1]);
#endif  PS2
#if     GRAFENG == MPS
                csize(&3);
                damove(&-30000, -31000);
                ctext(knobuf[0]);
                damove(&-30000, -32000);
                ctext(knobuf[1]);
#endif  MPS
        }
#endif  KDEBUG

        /* If the command line should be drawn, then do it */

        if ((flags & DTEXT || _pskbien > 0) && showcom) {
#if     GRAFENG == IRIS
                callobj(O_COMMAND);
                display_combuf(COMMAND_X, COMMAND_Y);
                make_reply();
                callobj(O_REPLY);
#endif  IRIS
#if     GRAFENG == PS2
                charsz(4, 0);
                moveto(-30000, -23500, 0);
                text(prompt);
                moveto(-22000, -23500, 0);
                display_combuf(-22000, -23500);
                if (nreply > 0) {
                        moveto(-30000, -25000, 0);
                        text("Reply: ");
                        for (i = 0; i < nreply; i++) {
                                moveto(-25000, -25000 - i * 1500, 0);
                                text(repbuf[i]);
                        }
                }
#endif  PS2
#if     GRAFENG == MPS
                csize(&4);
                damove(&-30000, &-23500, &-20000);
                ctext("Command: ");
                damove(&-22000, &-23500, &-20000);
                i = strlen(combuf);
                if (i > VISLNLEN)
                        i -= VISLNLEN;
                else
                        i = 0;
                ctext(&combuf[i]);
                if (nreply > 0) {
                        damove(&-30000, &-25000, &-20000);
                        ctext("Reply: ");
                        for (i = 0; i < nreply; i++) {
                                arg1 = -25000 - i * 1500;
                                damove(&-25000, &arg1, &-20000);
                                ctext(repbuf[i]);
                        }
                }
#endif  MPS

                /* Put up the labels and the memory usage message */

#if     GRAFENG == PS2
                charsz(2, 0);
                moveto(-30000, -30000);
                text(usage);
#endif  PS2
#if     GRAFENG == MPS
                csize(&2);
                damove(&-30000, &-30000);
```

```
                ctext(usage);
#endif  MPS
        }

        /* Put up the distance and rotation labels */
        if ((flags & DLABELS) && (nrot > 0 || ndist > 0 || nangle > 0)) {
#if     GRAFENG != IRIS
                drawob(frame);
#else   IRIS
                editobj(O_FRAME);
#endif  IRIS

                /* Put up the rotation angles */

#if     GRAFENG != IRIS
                j = 0;
#endif
                for (i = 0; i < MAXINROT && nrot > 0; i++) {
                        if (rotation[i].r_stat == INACTIVE)
                                continue;
                        psangle = rotation[i].r_angle + rotation[i].r_inita;
                        angle = UNMAPANGLE(psangle);
                        while (angle < 0)
                                angle += 360;
                        while (angle >= 360)
                                angle -= 360;
                        sprintf(buf, "%5.1f", angle);
#if     GRAFENG == PS2
                        moveto(-3500 + LINEWIDTH(LABELSIZE),
                                30000 - j * LINEHEIGHT(LABELSIZE), 0);
                        text(buf);
                        j++;
#endif  PS2
#if     GRAFENG == MPS
                        arg0 = -3500 + LINEWIDTH(LABELSIZE);
                        arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                        damove(&arg0, &arg1, &0);
                        ctext(buf);
                        j++;
#endif  MPS
#if     GRAFENG == IRIS
                        objreplace((Tag) i);
                        charstr(buf);
#endif  IRIS
                }

                /* Put up the angle calculations */
                for (i = 0; i < MAXANGLE && nangle > 0; i++) {
                        if (angles[i].a_stat) {
#if     GRAFENG == PS2
                                sprintf(buf, "%5.1f", angles[i].a_value);
                                moveto(-3500 + LINEWIDTH(LABELSIZE),
                                        30000 - j * LINEHEIGHT(LABELSIZE), 0);
                                text(buf);
                                j++;
#endif  PS2
#if     GRAFENG == MPS
                                arg0 = -3500 + LINEWIDTH(LABELSIZE);
                                arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                                damove(&arg0, &arg1, &0);
                                ctext(buf);
                                j++;
#endif  MPS
#if     GRAFENG == IRIS
                                objreplace((Tag) MAXINROT + i);
                                charstr(buf);
#endif  IRIS
                        }
                }

                /* Put up the distance calculations */

#if     GRAFENG != IRIS
                j = 0;
#endif
                for (i = 0; i < MAXDIST && ndist > 0; i++) {
                        if (distance[i].d_stat) {
                                sprintf(buf, "%6.2f", distance[i].d_value);
#if     GRAFENG == PS2
                                moveto(25000 + LINEWIDTH(LABELSIZE),
                                        30000 - j * LINEHEIGHT(LABELSIZE), 0);
                                text(buf);
                                j++;
#endif  PS2
#if     GRAFENG == MPS
                                arg0 = 25000 + LINEWIDTH(LABELSIZE);
                                arg1 = 30000 - j * LINEHEIGHT(LABELSIZE);
                                damove(&arg0, &arg1, &0);
                                ctext(buf);
                                j++;
#endif  MPS
#if     GRAFENG == IRIS
                                objreplace((Tag) MAXINROT + MAXANGLE + i);
```

```
                                        charstr(buf);
#endif  IRIS
                                }
                        }
#if     GRAFENG == IRIS
                        closeobj();
                        callobj(O_FRAME);
#endif  IRIS
        }
}

/*
 * Calcdist - calculates the distances by reading coordinates back from
 * PS memory.
 */
#if     GRAFENG == IRIS
# include       <do_emulate.h>

int     getdist(), getangle();

CFUNC_TAB       Cfunc_tab[] = {
                        DISTFUNC,       getdist,
                        ANGLEFUNC,      getangle,
                        NULL,           NULL,
                };
#endif

calcdist()
{
        double                  angle(), dihed();
        register int            i, j;
        register float          tx, ty, tz;
        static double           a[3], b[3], c[3], d[4];
#if     GRAFENG != IRIS
#if     GRAFENG == PS2
        static unsigned int     pslims[3];
        extern int              _psfid;

        unit();                 /* load identity matrix */
        wbtmem(1);              /* start the write-back */
        drawps('bd');           /* process the bond data */
        ioctl(_psfid, PSIOCGET, pslims);                /* get info on refresh buffe
        if (pslims[0] == pslims[1])             /* is it full? */
#endif  PS2
#if     GRAFENG == MPS
        extern int              wbmaoa;
        extern struct q0com     q0com;

        psmemfull = 0;
        tident();
        /*
         * Must run in "transform only mode" instead of
         * "transform/normalize" because there is a bug
         * in the MAP microcode and only 1 word of data
         * (not 4 as the manual states) is output when
         * FSM2 = 5-7.  BEWARE!  T. Ferrin  16jun83
         */
        fpmapx(&3);
        wbmaoa = q0com.gmaoa;   /* beginning loc for write-back data */
        udata(&'bd', psobs);
        fpstpx();
        if (psmemfull)
#endif  MPS
        {
                strcpy(REPLY, "Object too big to compute distances.");
                for (i = 0; i < MAXDIST; i++)
                        distance[i].d_value = -1.;
        }
        else {
                for (i = 0; i < MAXDIST; i++) {
                        /* is this an active distance request? */
                        if (!distance[i].d_stat || distance[i].d_value < 0)
                                continue;               /* no */
                        /* are both ends of the distance available? */
                        if (distance[i].d_loc[0] == 0 &&
                                distance[i].d_loc[1] == 0) {
                                        distance[i].d_value = -1.;
                                        continue;
                        }
                        /* ReaD Transformed Coordinates back from PS */
                        rdtc(distance[i].d_loc[0], distance[i].d_save[0]);
                        rdtc(distance[i].d_loc[1], distance[i].d_save[1]);
                        /* adjust for the homogenous coordinate */
                        normalize(distance[i].d_save[0]);
                        normalize(distance[i].d_save[1]);
                        /* compute distance */
                        tx = distance[i].d_save[0][0]
                                - distance[i].d_save[1][0];
                        ty = distance[i].d_save[0][1]
                                - distance[i].d_save[1][1];
                        tz = distance[i].d_save[0][2]
                                - distance[i].d_save[1][2];
                        distance[i].d_value =
                                UNMAPCRD(sqrt(tx * tx + ty * ty + tz * tz));
```

```
		}

		for (i = 0; i < MAXANGLE; i++) {
			/* is this an active distance request? */
			if (!angles[i].a_stat || angles[i].a_value < 0)
				continue;		/* no */
			/* are both ends of the distance available? */
			angles[i].a_value = 0;
			for (j = 0; j < angles[i].a_natom; j++)
				if (angles[i].a_loc[j] == 0) {
					angles[i].a_value = -1.;
					break;
				}
			if (angles[i].a_value < 0)
				continue;
			for (j = 0; j < angles[i].a_natom; j++) {
				/* ReaD Transformed Coordinates back from
				rdtc(angles[i].a_loc[j], angles[i].a_save[j]);
				/* adjust for the homogenous coordinate */
				normalize(angles[i].a_save[j]);
			}
			/* Put the stuff in the argument buffer */
			for (j = 0; j < 3; j++) {
				a[j] = angles[i].a_save[0][j];
				b[j] = angles[i].a_save[1][j];
				c[j] = angles[i].a_save[2][j];
				d[j] = angles[i].a_save[3][j];
			}
			if (angles[i].a_natom == 3)
				angles[i].a_value = angle(a, b, c);
			else
				angles[i].a_value = dihed(a, b, c, d);
			angles[i].a_value *= 180.0 / PI;
			while (angles[i].a_value < 0)
				angles[i].a_value += 360;
			while (angles[i].a_value >= 360)
				angles[i].a_value -= 360;
			if (angles[i].a_value < 0)
				angles[i].a_value += 360.;
		}
	}
#if	GRAFENG == PS2
	stopwb();		/* end writeback mode */
#endif	PS2

#else	IRIS
	register struct angle_def		*ap;
	register struct dist_def		*dp;
	static Matrix				imat = {
							1, 0, 0, 0,
							0, 1, 0, 0,
							0, 0, 1, 0,
							0, 0, 0, 1,
						};
	static Matrix				zmat = {
							0, 0, 0, 0,
							0, 0, 0, 0,
							0, 0, 0, 0,
							0, 0, 0, 0,
						};

	pushmatrix();
	loadmatrix(imat);
	callobj(O_BOND);
	popmatrix();
	for (dp = distance, i = 0; i < MAXDIST; i++, dp++) {
		/* is this an active distance request? */
		if (!dp->d_stat || dp->d_value < 0)
			continue;		/* no */
		/* do we have both ends of the distance? */
		if (memcmp(dp->d_mat[0], zmat, sizeof zmat) == 0 ||
		    memcmp(dp->d_mat[1], zmat, sizeof zmat) == 0) {
			dp->d_value = -1;
			continue;
		}
		d[3] = 1;
		for (j = 0; j < 2; j++) {
			d[0] = dp->d_x[j];
			d[1] = dp->d_y[j];
			d[2] = dp->d_z[j];
			transform(dp->d_mat[j], d, dp->d_save[j], 1.0);
		}

		/* compute distance */
		tx = dp->d_save[0][0] - dp->d_save[1][0];
		ty = dp->d_save[0][1] - dp->d_save[1][1];
		tz = dp->d_save[0][2] - dp->d_save[1][2];
		dp->d_value = UNMAPCRD(sqrt(tx * tx + ty * ty + tz * tz));
	}

	for (ap = angles, i = 0; i < MAXANGLE; i++, ap++) {
		/* is this an active distance request? */
		if (!ap->a_stat || ap->a_value < 0)
			continue;		/* no */
```

```
                /* do we have all ends of the angle? */
                if (memcmp(ap->a_mat[0], zmat, sizeof zmat) == 0 ||
                    memcmp(ap->a_mat[1], zmat, sizeof zmat) == 0 ||
                    memcmp(ap->a_mat[2], zmat, sizeof zmat) == 0 ||
                    (ap->a_natom == 4 &&
                     memcmp(ap->a_mat[3], zmat, sizeof zmat) == 0)) {
                        ap->a_value = -1;
                        continue;
                }
                /* Put the stuff in the argument buffer */
                d[3] = 1;
                for (j = 0; j < ap->a_natom; j++) {
                        d[0] = ap->a_x[j];
                        d[1] = ap->a_y[j];
                        d[2] = ap->a_z[j];
                        transform(ap->a_mat[j], d, ap->a_save[j], 1.0);
                }

                for (j = 0; j < 3; j++) {
                        a[j] = ap->a_save[0][j];
                        b[j] = ap->a_save[1][j];
                        c[j] = ap->a_save[2][j];
                        if (ap->a_natom == 4)
                                d[j] = ap->a_save[3][j];
                }
                if (ap->a_natom == 3)
                        ap->a_value = angle(a, b, c);
                else
                        ap->a_value = dihed(a, b, c, d);
                ap->a_value *= 180.0 / PI;
                while (ap->a_value >= 360)
                        ap->a_value -= 360;
                while (ap->a_value < 0)
                        ap->a_value += 360;
        }
#endif  IRIS
}

#if     GRAFENG == IRIS
/*
 * getdist:
 *      Get the matrix for the given distance calculation point; this
 *      is called indirectly via a callfunc() command.
 */
/* ARGSUSED */
getdist(n, d, a)
long    n, d, a;
{
        extern int              doing_copy;

        if (!doing_copy)
                getmatrix(distance[d].d_mat[a]);
}

/*
 * getangle:
 *      Get the matrix for the given angle calculation point; this
 *      is called indirectly via a callfunc() command.
 */
/* ARGSUSED */
getangle(n, d, a)
long    n, d, a;
{
        extern int              doing_copy;

        if (!doing_copy)
                getmatrix(angles[d].a_mat[a]);
}
#endif

/*
 * Linkdist - draws dotted lines connecting the two
 * atoms for each distance calculation.
 */

#if     GRAFENG == PS2
# define        move(x,y,z)     moveto(x,y,z)
# define        draw(x,y,z)     lineto(x,y,z)
# define        color(c)        huesat(c)
#endif
#if     GRAFENG == MPS
# define        move(x,y,z)     damove(&x,&y,&z)
# define        draw(x,y,z)     daline(x,&y,&z)
# define        color(c)        lcolor(&c)
#endif
#if     GRAFENG == IRIS
# define        color(c)        ir_srange(c)
#endif

linkdist()
{
        register struct angle_def               *ap;
        register struct dist_def                *dp;
```

```
        color(distcolor);
#if     GRAFENG == PS2
        txture(1);                      /* set PS for dashed lines */
#endif  PS2
#if     GRAFENG == MPS
        ldash(&2);
#endif  MPS
#if     GRAFENG == IRIS
        setlinestyle(L_DASH);
#endif  IRIS
        for (dp = distance; dp < &distance[MAXDIST]; dp++) {
                if (dp->d_stat != ACTIVE)
                        continue;
                if (dp->d_value < 0)
                        continue;
                move(dp->d_save[0][0], dp->d_save[0][1], dp->d_save[0][2]);
                draw(dp->d_save[1][0], dp->d_save[1][1], dp->d_save[1][2]);
        }
        for (ap = angles; ap < &angles[MAXANGLE]; ap++) {
                if (ap->a_stat != ACTIVE)
                        continue;
                if (ap->a_value < 0)
                        continue;
                move(ap->a_save[0][0], ap->a_save[0][1], ap->a_save[0][2]);
                draw(ap->a_save[1][0], ap->a_save[1][1], ap->a_save[1][2]);
                draw(ap->a_save[2][0], ap->a_save[2][1], ap->a_save[2][2]);
                if (ap->a_natom == 4)
                        draw(ap->a_save[3][0], ap->a_save[3][1],
                            ap->a_save[3][2]);
        }
#if     GRAFENG == PS2
        txture(0);                      /* return to solid lines */
#endif  PS2
#if     GRAFENG == MPS
        ldash(&0);
#endif  MPS
#if     GRAFENG == IRIS
        setlinestyle(0);
#endif  IRIS
}

#if     GRAFENG == PS2 || GRAFENG == MPS || GRAFENG == IRIS
#undef  move
#undef  draw
#undef  color
#endif
```

```
/*
 * Makegnomon - makes the 3-D cross that marks the center(s) of rotation.
 */
makegnomon()
{
        register ps_t           len;
#if     GRAFENG != IRIS
        register ps_t           offset;
#endif
        static int              cnt = 0;

        len = -scaleval / (float) INITSCALE * width / 25.0;
#if     GRAFENG == PS2
        offset = - len * 2;
        makeob(gnomon, GNOMONSIZE, &cnt);
        move(len, 0, 0);
        line(offset, 0, 0);
        move(len, 0, len);
        line(0, 0, offset);
        move(0, len, len);
        line(0, offset, 0);
        len /= 10;
        offset = len / 2;
        line(offset, len, 0);
        line(-2 * offset, 0, 0);
        line(offset, -len, 0);
        stopob();
#endif  PS2
#if     GRAFENG == IRIS
        if (cnt > 0)
                return;
        cnt++;

        makeobj(O_GNOMON);
        ir_srange(WHITE);
        move(-len, 0.0, 0.0);
        draw( len, 0.0, 0.0);
        move(0.0, -len, 0.0);
        draw(0.0,  len, 0.0);
        move(0.0, 0.0, -len);
        draw(0.0, 0.0,  len);
        closeobj();
#endif  IRIS
#if     GRAFENG == MPS
        len = - scaleval / (float) INITSCALE * width / 25.0;
        offset = - len * 2;
        makeob(gnomon, &GNOMONSIZE, &cnt);
```

```
        dbase(&0, &0, &0, &K32K);
        drmove(&len, &0, &0);
        drline(&offset, &0, &0);
        drmove(&len, &0, &len);
        drline(&0, &0, &offset);
        drmove(&0, &len, &len);
        drline(&0, &offset, &0);
        len /= 10;
        offset = len / 2;
        drline(&offset, &len, &0);
        arg0 = -2 * offset;
        drline(&arg0, &0, &0);
        arg1 = -len;
        drline(&offset, &arg1, &0);
        stopob();
#endif  MPS
}

/*
 * Drawaxis - draws the gnomon at the center(s) of rotation.
 */
drawaxis()
{
        register        i;

        if (nmol == 0)
                return;
#if     GRAFENG == PS2
        huesat(WHITE);
#endif  PS2
#if     GRAFENG == MPS
        lcolor(&WHITE);
#endif  MPS

        /* If the models are rotating independently, then
         * draw the gnomons for all the models, otherwise,
         * just draw one at the combined center of rotation
         */

        if (flags & INDEP) {
                for (i = 0; i < MAXMOD; i++) {
                        if (!models[i].active)
                                continue;
#if     GRAFENG == PS2
                        moveto(trans[0]+models[i].tr[0]+models[i].com[0],
                                trans[1]+models[i].tr[1]+models[i].com[1],
                                trans[2]+models[i].tr[2]+models[i].com[2]);
                        drawob(gnomon);
#endif  PS2
#if     GRAFENG == MPS
                        arg0 = trans[0]+models[i].tr[0]+models[i].com[0];
                        arg1 = trans[1]+models[i].tr[1]+models[i].com[1];
                        arg2 = trans[2]+models[i].tr[2]+models[i].com[2];
                        damove(&arg0, &arg1, &arg2);
                        drawob(gnomon);
#endif  MPS
#if     GRAFENG == IRIS
                        pushmatrix();
                        translate(trans[0] + models[i].tr[0] + models[i].com[0],
                                trans[1] + models[i].tr[1] + models[i].com[1],
                                trans[2] + models[i].tr[2] + models[i].com[2]);
                        callobj(O_GNOMON);
                        popmatrix();
#endif  IRIS
                }
        }
        else {
#if     GRAFENG == PS2
                moveto(trans[0]+cx, trans[1]+cy, trans[2]+cz);
                drawob(gnomon);
#endif  PS2
#if     GRAFENG == MPS
                arg0=(trans[0]+cx); arg1=(trans[1]+cy); arg2=(trans[2]+cz);
                damove(&arg0, &arg1, &arg2);
                drawob(gnomon);
#endif  MPS
#if     GRAFENG == IRIS
                pushmatrix();
                translate(trans[0] + cx, trans[1] + cy, trans[2] + cz);
                callobj(O_GNOMON);
                popmatrix();
#endif  IRIS
        }
}

do_show(mat, vl, vr, vb, vt, eye)
ps_t            *mat;
vp_coord        vl, vr, vb, vt;
ps_t            eye;
{
#if     GRAFENG == PS2
        unit();
        charsz(chsize, 0);
        vwport(vl, vr, vb, vt, vh, vy);
```

```
        window(wl, wr, wb, wt, wh, wy, eye);
        push();
        if (mat != NULL)
                bldcon(CONCATMAT, mat);
        scale(K16K, K16K, -K16K, K16K);
        tran(-cowin[0], -cowin[1], -cowin[2]);
        if (flags & SHOWCOFG)
                drawaxis();
        if (ndist > 0 || nangle > 0)
                linkdist();
#ifdef  DEBUG
        if (flags & DDEBUG)
                drawps('bd');
        else
#endif  DEBUG
        drawps('pc');
        pop();
#endif  PS2
#if     GRAFENG == MPS
        tpush();
        tident();
        csize(&chsize);
        vbound(&vl, &vr, &vb, &vt);
        arg0=(vh>>2); arg1=(vy>>2);
        vinten(&arg0, &arg1);
        twindp(&wl, &wr, &wb, &wt, &wh, &wy, &eye);
        if (mat != NULL)
                tcon(mat);
        tscale(&K16K, &K16K, &-K16K, &K16K);
        arg0=(-cowin[0]); arg1=(-cowin[1]); arg2=(-cowin[2]);
        ttran(&arg0, &arg1, &arg2);
        if (flags & SHOWCOFG)
                drawaxis();
        if (ndist > 0 || nangle > 0)
                linkdist();
        udata(&'pc', pcobe);
        udata(&'bd', pcobe);
        tpop();
#endif  MPS
#if     GRAFENG == IRIS
        pushmatrix();
        pushviewport();
        ir_showmat(mat, vl, vr, vb, vt, eye);
        depthcue(TRUE);
        if (ndist > 0 || nangle > 0)
                linkdist();
        if (!(flags & XBOND)) {
                callobj(O_BOND);
                callobj(O_SSBOND);
        }
        if (!(flags & XLABEL))
                callobj(O_LABEL);
        if (!(flags & XSURF))
                callobj(O_SURFACE);
        if (flags & SHOWCOFG)
                drawaxis();
        depthcue(FALSE);
        popviewport();
        popmatrix();
#endif  IRIS
}

#if     GRAFENG == IRIS
ir_showmat(mat, vl, vr, vb, vt, eye)
ps_t            *mat;
vp_coord        vl, vr, vb, vt;
ps_t            eye;
{
        if (wl - wr != 0) {
                viewport(vl, vr, vb, vt);
                if (flags & ORTHO) {
                        /*
                         * wy and wh are inverted because we have a
                         * right-hand coordinate system
                         */
                        ortho(wl, wr, wb, wt, wy, wh);
                        if (mat != NULL)
                                multmatrix(mat);
                }
                else {
                        window(wl, wr, wb, wt, eye - wh, eye - wy);
                        if (mat != NULL)
                                multmatrix(mat);
                        lookat(0.0, 0.0, eye, 0.0, 0.0, 0.0, (Angle) 0);
                }
                translate(-cowin[0], -cowin[1], -cowin[2]);
        }
}
#endif

#if     GRAFENG == MPS
# define        PFACTOR(double) PVSIZE / (double) K4K)
#endif
```

frame.c

```
/*
 * right_vp:
 *      Calculate the vieport coordinates for the right-hand image of a
 *      stereo pair.
 */
right_vp(nvl, nvr, nvb, nvt)
vp_coord        *nvl, *nvr, *nvb, *nvt;
{
#if     GRAFENG == MPS
        *nvl = (vl - VIEWLOLIM) * PFACTOR;
        *nvr = PVSIZE - (VIEWHILIM - vr) * PFACTOR;
        *nvb = (vb - VIEWLOLIM) * PFACTOR - PVSIZE / 2;
        *nvt = PVSIZE / 2 - (VIEWHILIM - vt) * PFACTOR;
#endif
#if     GRAFENG == IRIS || GRAFENG == PS2
        *nvl = VIEWLOLIM + (VIEWHILIM - VIEWLOLIM) / 2;
        *nvr = VIEWHILIM;
        *nvb = VIEWLOLIM;
        *nvt = VIEWHILIM;
#endif
}

/*
 * left_vp:
 *      Calculate the vieport coordinates for the left-hand image of a
 *      stereo pair.  It only calculates the left and right coordinates:
 *      It is assumed that right_vp() has been called to get the top
 *      and bottom coordinates.
 */
left_vp(nvl, nvr)
vp_coord        *nvl, *nvr;
{
#if     GRAFENG == MPS
        *nvl = (vl - VIEWLOLIM) * PFACTOR - PVSIZE;
        *nvr = (vr - VIEWHILIM) * PFACTOR;
#endif
#if     GRAFENG == IRIS || GRAFENG == PS2
        *nvr = *nvl;
        *nvl = VIEWLOLIM;
#endif
}


extern int      _cursor_pos;
static  int     line_base;

display_combuf(at_x, at_y)
int     at_x, at_y;
/*
 * display_combuf:
 * display the contents of the command buffer and cursor
 */
{
        register int    i, j;
        int             line_len, start, full_line, total_len, real_len;
        char            display_buf[VISLNLEN + 1];

        real_len = _cursor_pos - line_base;
        for (i = line_base; i < _cursor_pos; i++)
                if (iscntrl(combuf[i]))
                        real_len++;
        while (real_len < VISLNLEN / 4 && line_base > 0) {
                line_base--;
                real_len++;
                if (iscntrl(combuf[line_base]))
                        real_len++;
        }
        while (real_len >= VISLNLEN) {
                real_len--;
                if (iscntrl(combuf[line_base++]))
                        real_len--;
        }
        if (real_len > 3 * VISLNLEN / 4) {
                line_len = strlen(combuf);
                full_line = FALSE;
                total_len = real_len;
                for (i = _cursor_pos; i < line_len; i++) {
                        total_len++;
                        if (iscntrl(combuf[i]))
                                total_len++;
                        if (total_len > VISLNLEN) {
                                full_line = TRUE;
                                break;
                        }
                }
        }
        while (real_len > 3 * VISLNLEN / 4 && full_line) {
                real_len--;
                if (iscntrl(combuf[line_base++]))
                        real_len--;
        }
#if     GRAFENG == PS2
        moveto(at_x, at_y, 0);
#endif
```

frame.c

```
#if     GRAFENG == IRIS
        cmov2s(at_x, at_y);
        for (i = 0; i < strlen(prompt); i++)
                charstr(" ");
#endif
        display_buf[VISLNLEN] = '\0';
        for (i = line_base, j = 0; j < VISLNLEN; i++, j++) {
                if (iscntrl(combuf[i]) && combuf[i] != '\0') {
                        display_buf[j++] = '^';
                        if (j < VISLNLEN)
                                display_buf[j] = (combuf[i] < 64) ?
                                  combuf[i] + 64 : '?';
                } else {
                        display_buf[j] = combuf[i];
                        if (combuf[i] == '\0')
                                break;
                }
        }
#if     GRAFENG == PS2
        text(display_buf);
#endif
#if     GRAFENG == IRIS
        charstr(" ");
        charstr(display_buf);
#endif
#if     GRAFENG == IRIS
        cmov2s(at_x, at_y);
        for (i = 0; i < strlen(prompt); i++)
                charstr(" ");
        charstr(" ");
#endif
#if     GRAFENG == PS2
        moveto(at_x, at_y, 0);
#endif
        for (i = 0; i < real_len && i < VISLNLEN; i++)
                display_buf[i] = ' ';
        display_buf[i] = '\0';
#if     GRAFENG == PS2
        text(display_buf);
        text("\036");
#endif
#if     GRAFENG == IRIS
        charstr(display_buf);
        charstr("_");
#endif
}
```

editorio.c

```
/* $Header: editorio.c,v 3.21 86/09/15 14:29:29 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "intr.h"
#ifndef  vms
#include <sys/ioctl.h>
#if      GRAFENG == PS2
#include <ps2.h>
#endif   PS2
#endif   vms

#if      GRAFENG == IRIS
#undef   HARDTRAN        /* IRIS code doesn't know how to handle this */
#define  ONE     1.0
#else    IRIS
#ifndef  ONE
#define  ONE     K32K
#endif   ONE
#endif   IRIS

/*
 * Sendcom - send command string (+ misc garbage) to the editor
 */
sendcom(comline)
char     *comline;
{
         register        i;
         register        changed;
#if      GRAFENG == PS2
         static   int    ttyarg[3]       =       {
                  2,     4,      F_SPEEDS
         };
         int     memuse, newrate;
         extern  int     _psfid;
#endif   PS2

#if      GRAFENG == PS2
         _flwrbuf();
#endif   PS2
         putps();
#if      GRAFENG == IRIS
         gflush();
#endif   IRIS

#ifdef   DEBUG
         if (flags & DDEBUG)
                  fprintf(stderr, ">%s<\n", comline);
#endif   DEBUG
         i = strlen(comline);
         if (comline[i - 1] != '\n') {
                  comline[i++] = '\n';
                  changed = TRUE;
         } else
                  changed = FALSE;
         write(edit_fd[1], comline, i);
         if (changed)
                  comline[i - 1] = '\0';
                  /* Character changed must have been a null since strlen
                   * ended there */
         if (readeo(eo) == -1)
                  fputs("Communications error between intr and editor\n",
                           stderr);
#ifdef   vms
         if (strcmp(comline, "stop")==0)
                  return;   /* editor is dead, global section gone; just return */
#endif   vms
         getps();

#if      GRAFENG == PS2
#ifdef   LDEBUG
         if (flags & DDEBUG)
                  psmemcheck();
#endif   LDEBUG
         memuse = psmemuse('pc');
         /*
          * If the memory usage is low, then the update rate
          * may be too fast and consume a lot of CPU time.
          * Our solution is to lower to update rate from
          * 15hz to 12hz if memory usage drops below a
          * certain threshold.
          */
         if (memuse < 3000)
                  newrate = 5;
         else
                  newrate = 4;
         if (newrate != ttyarg[1]) {
                  ttyarg[1] = newrate;
                  ioctl(_psfid, PSIOCSET, ttyarg);
```

```
        }
        sprintf(usage, "Psmem usage : %d / %d", memuse, MAXPSMEM);
#endif  PS2
#if     GRAFENG == MPS
        i = psobs->preamble.u_max - psobs->preamble.u_min;
        sprintf(usage, "Psmem usage : %d / %d", psobs->preamble.u_base, i);
#endif  MPS

        makeframe();
        calcent(selection);
        if (ndist > 0 || nangle > 0)
                calcdist();
        getknob(0, 0);                          /* Read in knob values */
        readknob(noknob + 1);                   /* ... and throw it away */
}

/*
 * Readeo - read editor output
 */
readeo(fd)
FILE    *fd;
{
        char    inbuf[INBUFSIZE];
        char    comname[10], tbuf[2][MOLNMSZ];
        char    active[MAXMOD];
        ps_t    loc[4], new[4];
        float   ncofg[3];
        register        i, j;
        register        char    *p;
        int             n, num, k;
        float           angle;
        unsigned        loc1, loc2, loc3, loc4;
#ifdef  HARDTRAN
        int     ps2loc;
#endif  HARDTRAN
#ifdef  EMULATE
        static FILE     *ei = NULL;

        if (ei == NULL)
                ei = fdopen(edit_fd[1], "w");
#endif  EMULATE

        if (noeditor)
                return(-1);

#ifdef  EMULATE
        do_emulate(fd, ei);
#endif  EMULATE

        for (i = 0; i < MAXMOD; i++) {
                active[i] = models[i].active;
                models[i].flags = 0;
                models[i].active = FALSE;
        }
        if (fgets(inbuf, sizeof inbuf, fd) == NULL)
                return(0);
#ifdef  LDEBUG
        if (flags & DDEBUG)
                fputs(inbuf, stderr);
#endif  LDEBUG
        if (sscanf(inbuf, "%d molecules", &n) != 1) {
                resync(fd);
                return(-1);
        }
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "%d molecules\n", n);
#endif  DEBUG
#ifdef  HARDTRAN
        unit();
        wbtmem(1);
        ps2loc = 0;
#endif  HARDTRAN
        for (i = 0; i < n; i++) {
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
                sscanf(inbuf, "%d", &num);
                sscanf(inbuf, "%*d %d %f %f %f %f", &models[num].nused,
                        &ncofg[0], &ncofg[1], &ncofg[2], &models[num].range);
                models[num].active = TRUE;
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
```

```
                k = sscanf(inbuf, "%s %s %s", comname, tbuf[0], tbuf[1]);
                if (strcmp(comname, "object") == 0) {
                        models[num].flags |= ISOBJ;
                } else if (k == 3) {
                        models[num].flags |= HASURF;
                        strcpy(models[num].surfname, tbuf[1]);
                }
                if (lactive[num] || strcmp(tbuf[0], models[num].altname)) {
                        strcpy(models[num].altname, tbuf[0]);
                        strcpy(models[num].molname, tbuf[0]);
                        _imatx(models[num].matrix);
                        for (j = 0; j < 3; j++)
                                models[num].tr[j] = 0;
                        active[num] = FALSE;
                }
                if (active[num]) {
                        for (j = 0; j < 3; j++)
                                loc[j] = MAPCRD(ncofg[j] - models[num].cofg[j
                        loc[3] = ONE;
#ifdef  HARDTRAN
                        bldcon(LOADMAT, models[num].matrix);
                        moveto(loc[0], loc[1], loc[2]);
                        rdtc(ps2loc, new);
                        ps2loc += 4;
                        for (j = 0; j < 3; j++)
                                new[j] = new[j] * ((double) ONE) / new[3];
#else   HARDTRAN
                        transform(models[num].matrix, loc, new, ONE);
#endif  HARDTRAN
                        for (j = 0; j < 3; j++)
                                models[num].tr[j] += new[j] - loc[j];
                }
                for (j = 0; j < 3; j++) {
                        models[num].cofg[j] = ncofg[j];
                        models[num].com[j] = MAPCRD(ncofg[j]);
                }
        }
#ifdef  HARDTRAN
        stopwb();
#endif  HARDTRAN

        if (n != nmol) {
                if (nmol == 0) {
                        makewin();
                        (void) update(MODSEL, FALSE, (ps_t *) NULL, FALSE,
                                (ps_t *) NULL);
                        procom("savepos");
                }
                nmol = n;
        }

        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                resync(fd);
                return(-1);
        }
#ifdef  LDEBUG
        if (flags & DDEBUG)
                fputs(inbuf, stderr);
#endif  LDEBUG
        if (sscanf(inbuf, "%d rotations", &n) != 1) {
                resync(fd);
                return(-1);
        }
        nrot = n;
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "%d rotations\n", nrot);
#endif
        for (i = 0; i < MAXINROT; i++)
                rotation[i].r_stat = INACTIVE;
        for (i = 0; i < nrot; i++) {
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif
                if (sscanf(inbuf, "%d %f", &num, &angle) != 2) {
                        resync(fd);
                        return(-1);
                }
                rotation[num].r_inita = MAPANGLE(angle);
                for (p = inbuf, j = 0; j < 2; p++)
                        if (*p == ' ')
                                j++;
                strcpy(rotation[num].r_label, p);
                zapchar(rotation[num].r_label, '\n');
#if     GRAFENG == IRIS
                strcat(rotation[num].r_label, " ");
#endif
                rotation[num].r_stat = ACTIVE;
                readknob(knobs.k_inrot[num]);                   /* Throw away change */
```

```
        }
        for (i = 0; i < MAXINROT; i++)
                if (rotation[i].r_stat == INACTIVE) {
                        rotation[i].r_angle = 0;
#if     GRAFENG == PS2
                        getrot(&matbuf[inrot[i]], 0, ZAXIS);
#endif  PS2
#if     GRAFENG == IRIS
                        _imatrx(&matbuf[inrot[i]]);
#endif  IRIS
#if     GRAFENG == MPS
                        getrot(&matbuf[inrot[i]], &0, &ZAXIS);
#endif  MPS
                }

        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                resync(fd);
                return(-1);
        }
#ifdef  LDEBUG
        if (flags & DDEBUG)
                fputs(inbuf, stderr);
#endif  LDEBUG
        if (sscanf(inbuf, "%d distances", &n) != 1) {
                resync(fd);
                return(-1);
        }
        ndist = n;
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "%d distances\n", ndist);
#endif  DEBUG
        for (i = 0; i < MAXDIST; i++)
                distance[i].d_stat = INACTIVE;
        for (i = 0; i < ndist; i++) {
#if     GRAFENG != IRIS
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
                if (sscanf(inbuf, "%d %d %d", &num, &loc1, &loc2) != 3) {
                        resync(fd);
                        return(-1);
                }
                distance[num].d_loc[0] = loc1;
                distance[num].d_loc[1] = loc2;
#else   IRIS
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
                if (sscanf(inbuf, "%d", &num) != 1) {
                        resync(fd);
                        return -1;
                }

                for (j = 0; j < 2; j++) {
                        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                                resync(fd);
                                return(-1);
                        }
#ifdef  LDEBUG
                        if (flags & DDEBUG)
                                fputs(inbuf, stderr);
#endif  LDEBUG
                        if (sscanf(inbuf, "%f %f %f", &distance[num].d_x[j],
                                &distance[num].d_y[j], &distance[num].d_z[j])
                           != 3) {
                                resync(fd);
                                return -1;
                        }
                }
#endif  IRIS
                if (fgets(distance[num].d_label, sizeof distance[0].d_label, fd)
                   == NULL) {
                        resync(fd);
                        return -1;
                }
                zapchar(distance[num].d_label, '\n');
#if     GRAFENG == IRIS
                strcat(distance[num].d_label, " ");
#endif
                distance[num].d_stat = ACTIVE;
                distance[num].d_value = 0;
        }
```

```
        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                resync(fd);
                return(-1);
        }
#ifdef  LDEBUG
        if (flags & DDEBUG)
                fputs(inbuf, stderr);
#endif  LDEBUG
        if (sscanf(inbuf, "%d angles", &n) != 1) {
                resync(fd);
                return(-1);
        }
        nangle = n;
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "%d angles\n", nangle);
#endif  DEBUG
        for (i = 0; i < MAXANGLE; i++)
                angles[i].a_stat = INACTIVE;
        for (i = 0; i < nangle; i++) {
#if     GRAFENG != IRIS
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
                if (sscanf(inbuf, "%d %d %d %d %d", &num, &loc1, &loc2,
                        &loc3, &loc4) != 5) {
                                resync(fd);
                                return(-1);
                }
                angles[num].a_loc[0] = loc1;
                angles[num].a_loc[1] = loc2;
                angles[num].a_loc[2] = loc3;
                angles[num].a_loc[3] = loc4;
                angles[num].a_natom = (loc4 == 0) ? 3 : 4;
#else   IRIS
                if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif  LDEBUG
                if (sscanf(inbuf, "%d", &num) != 1) {
                        resync(fd);
                        return -1;
                }

                for (j = 0; j < 4; j++) {
                        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                                resync(fd);
                                return(-1);
                        }
#ifdef  LDEBUG
                        if (flags & DDEBUG)
                                fputs(inbuf, stderr);
#endif  LDEBUG
                        if (sscanf(inbuf, "%f %f %f", &angles[num].a_x[j],
                                &angles[num].a_y[j], &angles[num].a_z[j])
                          != 3) {
                                        resync(fd);
                                        return -1;
                        }
                }
                if (angles[num].a_x[3] == 0.0 && angles[num].a_y[3] == 0.0 &&
                  angles[num].a_z[3] == 0.0)
                        angles[num].a_natom = 3;
                else
                        angles[num].a_natom = 4;
#endif  IRIS
                if (fgets(angles[num].a_label, sizeof angles[0].a_label, fd)
                  == NULL) {
                        resync(fd);
                        return -1;
                }
                zapchar(angles[num].a_label, '\n');
#if     GRAFENG == IRIS
                strcat(angles[num].a_label, " ");
#endif
                angles[num].a_stat = ACTIVE;
                angles[num].a_value = 0;
        }

        if (fgets(inbuf, sizeof inbuf, fd) == NULL) {
                resync(fd);
                return(-1);
        }
#ifdef  LDEBUG
        if (flags & DDEBUG)
```

```
                fputs(inbuf, stderr);
#endif  LDEBUG
        if (sscanf(inbuf, "%d replies", &n) != 1) {
                resync(fd);
                return(-1);
        }
        nreply = n;
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "%d replies\n", nreply);
#endif  DEBUG
        for (i = 0; i < nreply; i++) {
                if (fgets(repbuf[i], sizeof inbuf, fd) == NULL) {
                        resync(fd);
                        return(-1);
                }
#ifdef  LDEBUG
                if (flags & DDEBUG)
                        fputs(inbuf, stderr);
#endif
                zapchar(repbuf[i], '\n');
        }
#ifdef  DEBUG
        if (flags & DDEBUG)
                fputs("Done\n", stderr);
#endif
        resync(fd);
        return(0);
}

resync(fd)
FILE    *fd;
{
        char    buf[BUFSIZ];

        while (fgets(buf, sizeof buf, fd) != NULL)
                if (strncmp(buf, "SYNC", 4) == 0)
                        return;
        fprintf(stderr, "Communication synchronization error\n");
        fprintf(stderr, "Expected \"SYNC\", got \"%.4s\"\n", buf);
        return;
}
```

misc.c

```
/* $Header: misc.c,v 3.17 86/06/04 17:26:32 arnold Exp $ */
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include "intr.h"
#ifdef  vms
#define rindex  strrchr
#endif
int noknob = NOKNOB;        /* everything that depends on NOKNOB is in here */

/*
 * Zapchar - changes the character c in str into a null.
 */
zapchar(str, c)
char    *str, c;
{
        char    *p;
        char    *rindex();

        if (p = rindex(str, c))
                *p = '\0';
}

/*
 * Readknob - reads the knobs.  If a knob is read twice, the values
 * returned are the same both times (getknob returns 0 the second time).
 */

#if     GRAFENG != IRIS
#define THRESHOLD 64
#else
#define THRESHOLD 0
#endif

knob_t
readknob(knob)
int     knob;
{
        register int i;
        register knob_t     val;
        static double       knbscale;
        static knob_t       knobval[NOKNOB];
        static char         gotknob[NOKNOB];

        if (knob == NOKNOB)
                return 0;
        if (knob < -NOKNOB) {
#ifdef  KDEBUG
                /* These formats vary according to the number of knobs
                 * available on the system.  You have to adjust them
                 * if NOKNOB != 14. */
                if (flags & DDEBUG) {
                        sprintf(knobuf[0],
#if     GRAFENG == IRIS
                                "%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
#else
                                "%6d %6d %6d %6d %6d %6d %6d %6d",
#endif
                                knobval[0], knobval[1], knobval[2], knobval[3],
                                knobval[4], knobval[5], knobval[6], knobval[7]);
                        sprintf(knobuf[1],
#if     GRAFENG == IRIS
                                "%6.2f %6.2f %6.2f %6.2f %6.2f %6.2f",
#else
                                "%6d %6d %6d %6d %6d %6d",
#endif
                                knobval[8], knobval[9], knobval[10],
                                knobval[11], knobval[12], knobval[13]);
                }
#endif  KDEBUG
                return 0;
        }
        if (knob > NOKNOB) {
                for (i = 0; i < NOKNOB; i++)
                        gotknob[i] = FALSE;
#if     GRAFENG != IRIS
                knbscale = (double) (1 << reduction) / (double) K32K;
#else
                knbscale = reduction;
#endif
                return 0;
        }
        i = (knob < 0) ? (knob + NOKNOB) : knob;
        if (!gotknob[i]) {
                /* Assumption: all absolute-value knobs have numbers
                 * lower than all rate-value knobs. */
                val = getknob(i, (i < RATENOB) ? 2 : 1);
                knobval[i] = (abs(val) < THRESHOLD) ? 0 : val * knbscale;
                gotknob[i] = TRUE;
```

```
        }
        return ((knob < 0) ? -knobval[i] : knobval[i]);
}

/*
 * Clipcheck - checks that the clipping planes don't get too close.
 */
clipcheck(newlow, newhi, oldlow, oldhi)
register fps_t      *newlow, *newhi, *oldlow, *oldhi;
{
        double  dist;
        double  lowmove, himove, total;

        if (*newlow < *newhi) {
                *oldlow = *newlow;
                *oldhi = *newhi;
                return;
        }
        dist = *oldhi - *oldlow - width / 50.0;
        if (dist <= 0)
                return;
        lowmove = *newlow - *oldlow;
        himove = *oldhi - *newhi;
        if (lowmove < 0 || himove < 0) {
                fprintf(stderr, "lowmove = %.2f, himove = %.2f\n", lowmove, himov
                return;
        }
        total = himove + lowmove;
        *oldlow = *oldlow + lowmove / total * dist;
        *oldhi = *oldhi - himove / total * dist;
}

#if     GRAFENG != IRIS
/*
 * Normalize - normalizes the given vector to K32K.
 */
normalize(x)
ps_t    x[4];
{
        float   fac;

        fac = 32768. / x[3];
        x[0] = x[0] * fac;
        x[1] = x[1] * fac;
        x[2] = x[2] * fac;
}
#endif
```

```
int     slow_fac = 0;

slow_down(tp, type)
register struct turn_def            *tp;
int                                 type;
{
        register double     total;
        register int        d;

#if     GRAFENG == IRIS || GRAFENG == PS2
        tp->t_wait *= slow_fac;

        if (tp->t_frames == -1) {
                tp->t_speed /= slow_fac;
                return;
        }

#if     GRAFENG == IRIS
        /*
         * IRIS coordinates are floating point, so we don't have to
         * find "best match".
         */
        if (type == T_SHIFT) {
                tp->t_speed /= slow_fac;
                tp->t_frames *= slow_fac;
                return;
        }
#endif

        total = tp->t_speed * tp->t_frames;
        if (total < 0)
                d = total / (tp->t_frames * slow_fac) - 0.5;
        else
                d = total / (tp->t_frames * slow_fac) + 0.5;
        tp->t_frames = total / d + 0.5;
        if (type != T_ROCK)
                tp->t_speed = d;
#else
/* ARGSUSED */
#endif
}
```

lex.c

```
/* $Header: /usr/src/local/midas/src/intr/RCS/lex.c,v 3.2 84/07/18 13:07:26 arnold Exp
/*
 * Copyright (c) 1983 by the Regents of the University of California.
 * All rights reserved.
 *
 * Release 1.0                          9 Apr 1982
 * Release 2.0                          13 May 1983
 */

#include <ctype.h>
#include "intr.h"

#define  isterm(c)          ((c) == '\n' || (c) == '\0' || (c) == ';')
#define  isletter(c)        ((c) != ';' && (c) != '~' && (c) != '\t' &&\
                            (c) != ' ' && (c) != '\n' && (c) != '\0')

struct   cstack_def         {
         int        c_index;                        /* Buffer location */
         union      {
                    char    *c_line;                /* Input buffer (alias) */
                    FILE    *c_strm;                /* ... or input stream */
         }          c_input;
         int        c_saveeoc;                      /* Saved value of eocom *
}        cstck[MAXSTACK];
int      stptr      =       -1;
static   int        eocom;

#define  CURLINE cstck[stptr].c_input.c_line
#define  CURSTRM cstck[stptr].c_input.c_strm
#define  CURNDX  cstck[stptr].c_index
#define  CUREOC  cstck[stptr].c_saveeoc
#define  NEXTC   (CURNDX < 0) ? getc(CURSTRM) : CURLINE [ CURNDX++ ]

/*
 * lex_string - puts the string on the lexical input stack.
 */
lex_string(str)
char     *str;
{
         if (*str == COMMENT)
                  return(-1);
         if (stptr + 1 >= MAXSTACK) {
                  strcpy(REPLY, "Input nesting too deep.");
                  return(-1);
                  /* Should never happen */
         }
         stptr++;
         CUREOC = eocom;
         CURNDX = 0;
         CURLINE = str;
         eocom = TRUE;
         return(stptr);
}

/*
 * lex_file - puts the file on the lexical input stack.
 */
lex_file(file)
char     *file;
{
         if (stptr + 1 >= MAXSTACK) {
                  strcpy(REPLY, "Input nesting too deep.");
                  return(-1);
                  /* Should never happen */
         }
         stptr++;
         CUREOC = eocom;
         CURSTRM = fopen(file, "r");
         if (CURSTRM == NULL) {
                  stptr--;
                  sprintf(REPLY, "%s: cannot access.\n", file);
                  return(-1);
         } else {
                  CURNDX = -1;
                  eocom = TRUE;
                  return(stptr);
         }
}

/*
 * lex_done - resets the value of eocom to the current level of input.
 */
lex_done()
{
         eocom = CUREOC;
}

/*
 * getinput - returns the remainder of the command.
 */
char     *
getinput(level)
int      level;
{
```

lex.c

```
        static  char    inputbuf[256];
        register        i, c;
        char    getch();

        if (eocom) {
                inputbuf[0] = '\0';
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "Getinput: >%s<\n", inputbuf);
#endif
                return(inputbuf);
        }
        for (i = 0; i < 255; i++) {
                c = getch(level);
                if (c == ';' || c == '\n' || c == '\0')
                        break;
                inputbuf[i] = c;
        }
        inputbuf[i] = '\0';
        eocom = TRUE;
#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "Getinput: >%s<\n", inputbuf);
#endif
        return(inputbuf);
}

/*
 * clrinput - clears any junk from last command from the lexical stack./
 */
clrinput(level)
int     level;
{
        register        char    c;
        char    getch();

        if (!eocom)
                do
                        c = getch(level);
                while (!isterm(c));
        else
                c = '\0';
        while (stptr >= level && isterm(c))
                c = getch(level);
        if (!isterm(c))
                backspace(c, level);
        eocom = FALSE;
}

/*
 * getword - reads in the next token.
 */
getword(buffer, size, mode, level)
char    *buffer;
int     size;
int     mode, level;
{
        char    lastc, getch();
        register        i, len;

        /* If we are at the end of a command, quit */
        if (eocom) {
                buffer[0] = '\0';
                return(0);
        }

        /* Skip white space */
        do
                lastc = getch(level);
        while (lastc == ' ' || lastc == '\t');

        /* Store the character and determine the token lexical code */
        buffer[0] = lastc;
        buffer[1] = '\0';
        if (lastc == ';' || lastc == '\n' || lastc == '\0') {
                eocom = TRUE;
                len = 0;
        } else if (isletter(lastc)) {
                /* Read in the entire token and put back the
                 * unused character */
                for (i = 1; i < size - 1; i++) {
                        lastc = getch(level);
                        if (!isletter(lastc))
                                break;
                        buffer[i] = lastc;
                }
                buffer[i] = '\0';
                backspace(lastc, level);

                /* Try to expand the name */
                if (mode) {
                        len = expand(buffer, size, level);
                        if (len == 0)
                                len = i;
```

```
                } else
                        len = i;
        } else
                len = 1;

#ifdef  DEBUG
        if (flags & DDEBUG)
                fprintf(stderr, "Getword: (%d) >%s<\n", len, buffer);
#endif
        return(len);
}

/*
 * getch - gets the next character from the input
 */
char
getch(level)
int     level;
{
        char    c;

        if (stptr < level)                              /* Hit bottom */
                return('\0');
        while (stptr >= level && (c = NEXTC) == '\0')           /* Hit end of lev
                stptr--;
        if ((int) c == -1) {                            /* Read failed */
                fclose(CURSTRM);
                stptr--;
                return('\0');
        }
        return(c);
}

/*
 * backspace - puts 'c' back onto the input stream
 */
backspace(c, level)
char    c;
int     level;
{
        if (stptr < level)
                return;
        if (CURNDX > 0)
                CURNDX--;
        else if (CURNDX < 0)
                ungetc(c, CURSTRM);
}
```

```
/*
 * expand - expands the current token if it is an alias
 */
expand(buffer, size, level)
char    *buffer;
int     size;
int     level;
{
        register        i;

        for (i = 0; i < nalias; i++)
                if (!strcmp(buffer, atab[i].a_name))
                        break;
        if (i >= nalias)
                return(0);
        if (stptr + 1 >= MAXSTACK) {
                strcpy(REPLY, "Alias nesting too deep.");
                return(0);
        }
        stptr++;
        CURNDX = 0;
        CURLINE = atab[i].a_line;
        return(getword(buffer, size, TRUE, level));
}
```