VMD User's Guide – Draft version

A. Dalke, J. Gullingsrud, W. Humphrey, S. Izrailev, J. Stone, J. Ulrich

Version 1.3

April 6, 1999

Theoretical Biophysics Group[1]
University of Illinois and Beckman Institute
405 N. Mathews
Urbana, IL 61801
VMD WWW home page: `http://www.ks.uiuc.edu/Research/vmd/`

Description

The VMD User's Guide – Draft version describes how to run and use the molecular visualization and analysis program VMD. This guide documents the usage of both the graphical user interface and the text console interface for displaying and grapically manipulating molecules, and describes how to customize the appearance and behavior of VMD for each user.

[1]http://www.ks.uiuc.edu/

Contents

List of Figures

List of Tables

Chapter 1

Introduction

VMD is a molecular graphics program designed for the interactive visualization and analysis of biopolymers such as proteins, nucleic acids, lipids, and membranes. Currently VMD runs on SGI workstations with IRIX 5.3 or higher, Hewlett-Packard workstations with HP-UX 10.20, Sun workstations with Solaris 2.6 or higher, IBM RS/6000 workstations with AIX 4.3, and PC's with RedHat Linux.

Online information about VMD is available from `http://www.ks.uiuc.edu/Research/vmd/`.

VMD has several features:

- **General molecular visualization**
  At its heart, this program is a general application for displaying molecules containing any number of atoms. It is similar in basic capabilities to commercial programs such as Quanta and non-commercial programs such as RasMol, XMol, and Ribbons. It can read PDB files or use Babel (if available) to convert other formats automatically. Once loaded, user-defined subsets of the molecule can be displayed in various ways including licorice, ribbons, van der Waal spheres, and molecular surfaces. The display can be saved directly to a postscript file or in a format suitable for use by ray shading programs such as Raster3D, POV, and Rayshade.

- **Visualization of dynamic molecular data**
  VMD can read molecular trajectories from DCD and Amber files, or it can acquire timesteps from a running molecular dynamics program. The data can be used to animate the molecule or to plot the change in molecular properties such as interatomic distances, angles, or dihedrals over time.

- **Display and control of molecular dynamics simulations**
  VMD can be used as a graphical front end to a molecular dynamics (MD) program running on a remote supercomputer or high-performance workstation. VMD uses the MDComm software to set up, initiate, and interactively display and control the MD simulation as the simulation is running. The user can disconnect from the simulation and let it continue, reattach to a running simulation, or halt the MD program. Any number of simulations may be simultaneously displayed and controlled by a single VMD session.

- **Support for several input and display (output) devices**
  A number of different visual display and control systems are supported in addition to the usual monitor, keyboard, and mouse. The UNC tracker library is used to get position and orientation information from a wide variety of spatial input devices, including a Polhemus

Fastrak. An interface to the CAVE library has been developed for use in many different types of stereo projection facilities.

- **Tcl scripting language**
  VMD uses the freely available Tcl scripting language for processing text commands. This is a very common and popular language which contains variables, loops, subroutines, and much more. The TclX and Tcl-DP extension packages have been included to improve the usefulness of the language.

- **Molecular analysis commands**
  Many new Tcl commands have been added for doing molecular analysis. These include methods to extract information about a set of atoms and molecules, vector and matrix routines for coordinate manipulation, and functions for computing values like the center of mass and radius of gyration.

VMD is the visualization component of MDScope, a computational environment for structural biology and interactive molecular dynamics.

1.1 For more information on VMD and MDScope

MDScope (a Molecular Dynamics computational environment) is a complete software environment for interactive simulation and display of biopolymers. It consists of three components:

- VMD
  A general molecular visualization program capable of interactive display and concurrent control of a molecular dynamics simulation running on a remote computer. VMD uses an object-oriented design, and is written in C++. This document describes the VMD component of MDScope.

- NAMD
  A molecular dynamics program designed specifically to run in parallel on a number of parallel computing platforms. NAMD employs the CHARMm energy function and accepts molecular structure and parameter files compatible with the program X-PLOR. NAMD fully supports connection to VMD, as well as a variety of molecular dynamics simulation options such as, and inclusion of complete long-range electrostatic forces through the use of the Fast Multipole Algorithm.

- MDComm
  A set of tools which allows fast, reliable communication and control between a molecular visualization program (such as VMD) and a molecular dynamics simulation program (such as NAMD). MDComm defines a protocol for transfer of static and dynamic molecular data between applications, provides a library for use by applications to send and receive this data, and includes daemons which coordinate and buffer the communication. Both VMD and NAMD use the MDComm software, which links these two applications into a single environment for interactive simulation of biomolecular assemblies.

For more information on MDScope, or on any of the individual components VMD, NAMD, or MDComm, see the Theoretical Biophysics Group WWW home page[1]. Full source code and

[1] http://www.ks.uiuc.edu/

documentation for all components of MDScope may be obtained from the Theoretical Biophysics Group anonymous FTP server[2], in the directory `pub/mdscope`.

1.2 Contacting the authors

The current authors of VMD are A. Dalke, J. Gullingsrud, W. Humphrey, S. Izrailev, J. Stone, J. Ulrich. We are very interested in and grateful for any user comments and reports of program bugs or inaccuracies. If you have any suggestions, bug reports, or general comments about VMD, please send them to us at `vmd@ks.uiuc.edu`.

1.3 Credits and Program Reference

The authors request that any published work or images created using VMD include the following reference:

Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics" *J. Molec. Graphics* **1996**, *14.1*, 33-38.

VMD has been developed by the Theoretical Biophysics group at the University of Illinois and the Beckman Institute. The main authors of VMD are A. Dalke, J. Gullingsrud, W. Humphrey, S. Izrailev, J. Stone, J. Ulrich. This work is supported by grants from the National Institutes of Health (grant number PHS 5 P41 RR05969-04), the National Science Foundation (grant number BIR-9423827 EQ), and the Roy J. Carver Charitable Trust.

1.4 Copyright and Disclaimer

The terms for using, copying, modifying, and distributing VMD are specified by the VMD License. The license agreement is distributed with VMD in the file LICENSE. If for any reason you do not have this file in your distribution, it can be downloaded from `ftp://ftp.ks.uiuc.edu/pub/mdscope/vmd/LICENSE` .

Please take a few minutes to fill out a registration form at `http://www.ks.uiuc.edu/Research/vmd/VMDregistration.html` if you are going to use VMD.

Some of the code and executables used by VMD have different restrictions. They are:

1) STRIDE, the program used for secondary structure calculation, is free to both academic and commercial sites provided that STRIDE will not be a part of a package sold for money. The use of STRIDE in commercial packages is not allowed without a prior written commercial license agreement. See http://www.embl-heidelberg.de/argos/stride/stride_info.html

2) The source code for SURF is copyrighted by the original author, Amitabh Varshney, and the University of North Carolina at Chapel Hill. Permission to use, copy, modify, and distribute this software and its documentation for educational, research, and non-profit purposes is hereby

[2] ftp://ftp.ks.uiuc.edu/

granted, provided this notice, all the source files, and the name(s) of the original author(s) appear in all such copies.

BECAUSE THE CODE IS PROVIDED FREE OF CHARGE, IT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED.

This software was developed and is made available for public use with the support of the National Institutes of Health, National Center for Research Resources under grant RR02170.

See ftp://ftp.cs.unc.edu/pub/projects/GRIP/SURF/surf.tar.Z .

3) The perl script url_get, was written by Jack Lund at the University of Texas as Austin. There appear to be no restrictions on its use.

4) Some parts of the GNU C++ class library are used in VMD, including the String and Regex classes. As per the GNU General Public License v. 2, the modified code is available as part of the general VMD source distribution and may be freely used and altered.

1.5 Registering VMD

VMD is made available free of charge for non-commercial use to all interested users (but please see the Disclaimer below). We would like to request that you register with us that you are using VMD. This is so that we can maintain some idea of the number of users of the program and so that we know who to contact about program updates, bug fixes, etc. To register, please fill out the form at

`http://www.ks.uiuc.edu/Research/vmd/VMDregistration.html`.

We appreciate your help in this service.

1.6 Acknowledgments

The authors would particularly like to thank the members of the Theoretical Biophysics group, past and present, who have helped tremendously in making suggestions, pushing for new features, and trying out quite often very bug-ridden code. Tom Bishop, the original (and for some time only) VMD user, has been a driving force in suggesting (well, demanding) new features and bug fixes. As well, thanks go to Alexander Balaeff, Ivo Hofacker, Xiche Hu, Barry Israeliwitz, Dorina Kosztin, Ilya Logunov, Jim Phillips, Ari Shinozaki, Svilen Tzonev, Willy Wriggers, Dong Xu, Feng Zhou, and our agent in the field, Daniel Barsky. Thanks also to all of you who have tried out the program.

Many external libraries and packages are used in VMD, and the program would not be possible without them. The authors wish to thank Jon Leech for the code to compute the uniform point distributions; T. C. Zhao and Mark Overmars, authors of the excellent XFORMS library; John Ousterhout and the other authors of the Tcl, Tk, TclX, TkX and Tcl-DP packages; the authors of the Tracker library from the University of North Carolina; Amitabh Varshney, author of SURF, also from UNC; Dmitrij Frishman at EMBL for developing STRIDE; Jack Lund for the url_get perl script; and Ethan Merrit from the University of Washington for developing the algorithm for drawing ribbons.

We also received invaluable assistance from people who got the source code and sent in patches and explicit bug reports. The VMD developers would like to thank Axel Berg, Andrew Dalke, Rick Kufrin, Joe Landman, Clare Macrae, Lukasz Salwinski, Stephen Searle, Charles Schwieters, Michael Tiemann, Raymond de Vries, and Simon Warfield for their bug fixes and correspondence.

Chapter 2

Tutorials

2.1 Rapid Introduction to VMD

For those of you who don't like reading the manual, here is a quick introduction to VMD. To start it type `vmd` on the command line of your shell. (If that doesn't work, make sure it is in your path; you may have to check with your system administrator about how to fix it). VMD should start up in a window titled vmd console, a display window entitled OpenGL Display, and a button bar entitled main. Text commands are typed in the console window, graphics are displayed and manipulated in the display window, and many commands are available from the menu interface, accessible through the button bar. (The menus are based on the Forms library so all future references in the documentation will refer to them as *forms* and not *menus*.) All the forms can be turned off by pressing the button in the center top; for instance, press the button labeled VMD in the main form to turn it off.

There are two ways to perform almost all functions in VMD, either use the forms or the text console. For many of the commands, you can also use the popup menu available in the display window by pressing the right mouse button. Some of the more sophisticated commands, such as Tcl scripting, are only available in the text interface. Now that the button bar is turned off, you probably will want it back. The easiest way to access any form is by placing the cursor in the display window and holding down the right button on the mouse to obtain a popup menu. Select the option 'show form' on the popup menu, and then select the form 'main'. The button bar should reappear. The same effect may be accomplished by typing the command `menu main on` in the text console window.

2.2 Viewing a molecule: Myoglobin

You have probably obtained VMD in order to visualize molecules, so we'll load up one of the provided molecular structures to demonstrate the capabilities of VMD. On the button bar, click on the Mol button. This brings up the Molecules form [§3.3.2]. (As before, this form may be turned off by clicking the button in the center top of the form, labeled Molecules). Select Load From Files on the Molecules form to bring up the Files form [§3.3.3].

We will load a PDB (Protein Data Bank) file containing the coordinates of the atoms in myoglobin (compliments of Joel Berendzen from the Biophysics Group at Los Alamos National Laboratory). In the browser on the left, select the line that says pdb only then click on the button labeled Select pdb. Use the file browser that appears to go to the subdirectory `proteins/` of the VMD distribution (you may have to ask where this is located; try /usr/local/lib/vmd or /usr/local/vmd). Once there, select the file mbco.pdb. (Be careful when changing directories as the Forms library does not understand double clicks, and clicking too rapidly may cause the form to get very upset and core dump. This problem will be fixed in future versions of VMD.) Once the PDB file is selected, click on the Load Molecule button in the center of the Files form. You have loaded a myoglobin structure. Figure 2.1 shows an example of VMD displaying this protein.

You can use the mouse to manipulate the structure in the display window. There are three basic mouse modes [§3.1.1]: rotation[§3.1.1], translation [§3.1.1], scaling[§3.1.1]. The mode can be changed from the display window popup menu by pressing the right mouse button and picking the option under 'Mouse Mode', or by pressing r, t, or s on the keyboard. Experiment with these modes, and note how the cursor changes to indicate the current mode. In rotation mode, the left mouse button controls rotation about axes parallel to the screen, and the middle button controls rotation about the axis perpendicular to the screen. In translation mode, the left mouse button controls translation parallel to the screen, while the middle button controls translation in and out of the screen. Finally, in scaling mode, both the left and middle buttons control global scaling when the mouse is moved left or right, but the middle button causes larger changes. In all modes, the right mouse button controls the display window popup menu.

By default the myoglobin bonds are represented as lines and non-bonded atoms as points, with the color in both cases representing the atom type. This representation is easy for the computer to draw but is not always informative, especially when there are a large number of atoms. VMD allows you to display many of the common molecular representations. To access these, open the Graphics form [§3.3.4] using the button bar.

Suppose you would like to view the myoglobin structure with its protein backbone represented as a tube, the heme represented as licorice, the SO_4 ion and CO molecule represented as van der Waals spheres, and histidines 64 and 93 represented as CPK models. First, type backbone in the atom selection text entry area and press 'enter' to select the myoglobin backbone. All of the protein except for the backbone will disappear. Choose drawing method 'Tube' from the drawing method chooser to render the backbone as a tube, and chose coloring method 'Backbone' from the coloring method chooser to color the tube with the predefined backbone color. Click on the Create New button. This causes a new line to appear on the browser identical to the first line. The new line can be changed without affecting the first one, so clear the atom selection text area and then enter resname HEM to select the heme. At this point the heme isn't visible because it cannot be drawn as a tube, so choose the 'Licorice' drawing method to make it appear. Click on Create New again to make a new view, and enter resname SO4 CO to select the SO_4 ion and the CO molecule, and choose the drawing method 'VDW' to render them as Van der Waal spheres. Once again, press the Create New button and enter resid 93 64 to select the two histidines, and render them as 'CPK'. If you followed all that, then congratulations, you have made a beautiful image of myoglobin! Many more ways to represent atoms are possible. Please experiment with the options available in the Graphics form.

2.3 Rendering an Image

This tutorial assumes Raster3D is installed on your system and the executable render is in your path.

Find an interesting view of the molecule from the previous tutorial. Suppose you want to publish this view in a journal and want a high quality image, or you want to make a large poster. Taking the image from a screen capture results in a rather grainy image as the size of the pixels becomes apparent, so you want something with more resolution. There are several programs available which can render a high-quality raster image, based on an input script. VMD has the option to create input scripts for many of these image processing programs, which may then be processed to create a higher quality image of the scene displayed by VMD at the time the script was created. See the section on rendering [§ 8] for a further description of how this works.

Open the Render form [§ 3.3.10] and click on the 'Raster3D' Output Format. Both of the text boxes will fill with default values and do not need to be changed. Press the Go button. If everything works correctly, a message starting with:

```
Info 1) Rendering current scene to 'plot.r3d' ...
Info 1) Raster3D file generation finished
Info 1) Executing post-render cmd ' render < %s -sgi %s.rgb; ipaste %s.rgb' ...
 Raster3D rendering program V2.0 Apr 1994
```

will appear in the text console. After a few moments, a window should open with an image similar to the one in the VMD display. This image is in the RGB graphics format and can be read by many programs (such as `xv` and `ipaste`).

2.4 A Quick Animation

Another strength of VMD lies in its ability to playback trajectories resulting from molecular dynamics simulations. A sample trajectory, `alanin.DCD` is provided in the `proteins` directory included with VMD. To load it, click on the Mol button of the button bar to bring up the Molecules form [§3.3.2]. Next click on the Load From Files button and select the `psf and dcd` option under the Molecule File Types listing. Select `alanin.psf` for the psf file and `alanin.DCD` for the DCD by clicking on the Select psf and Select dcd buttons respectively. After this, you need only click the Load Molecule button in order to begin reading the trajectory. In the display window you should see a simulation of an alanin residue in vacuo. It isn't particularly informative, but you can easily see that the structure is quite unstable in an isolated environment. After the DCD file has loaded, it will by default stop. To see it again or to fine- tune playback, open the Animate form [§3.3.5] by clicking on the appropriate button in the button bar. Press the button that looks like >> to play the animation. Use the slider at the bottom of the form to change the speed of playback. By rotating the molecule around, etc. you should get an idea about how the system destabilizes over the course of the simulation.

2.5 An Introduction to Atom Selection

In this section it is assumed that you have the myoglobin structure `mbco.pdb` loaded and the views discussed in 2.2 created. If this is not true, go back to 2.2 and repeat the process described there.

VMD has a powerful atom selection method which is very helpful when generating attractive, informative, and complex graphics. In the previous section you used a few of these atom selection tools. This tutorial assumes that you have already loaded the myoglobin molecule, but it isn't necessary to recreate all the graphical representations.

To change which atoms are used to display each representation of the molecule shown in the display window, open the Graphics form [§ 3.3.4] and select the representation you want to change. You can then either edit the different fields (selection, coloring method, or drawing method) or use the Delete button to delete the view entirely. Try changing or deleting some of the views. When finished, delete all representations for the myoglobin structure. To get the basic line drawing view back, clear the atom selection text entry area (press `Esc`), enter `all` and press the Create New button.

Atoms may be selected on the basis of a property, i.e. `protein` or `not protein`, `water`, or `nucleic backbone`. They may also be selected by atom name, such as `atom C`, by residue name, such as `resname HEM`, or by many other identifiers. Multiple atoms may be specified with one keyword. For example, the selection `name C CA N O` will select the backbone atoms. (A similar effect may be obtained with the command `protein backbone`.) VMD can handle regular expressions, so that `name "C.*"` will select all atoms with names starting with C. VMD also understands the boolean operators `and`, `or`, and `not`, so the selection `resname HEM and not name "N.*"` selects all non-nitrogen atoms in the heme group of myoglobin.

Several more abstract selection criteria are available. For instance, the selection `x > 5` finds all atoms with an x coordinate greater than 5, while `mass >12 and mass < 14` selects all atoms with mass greater than 12 and less than 14 atomic mass units. Many math functions are also provided, so the selection `sqrt( sqr(x) + sqr(y) + sqr(z) ) < 10` will select atoms in a spherical region of radius 10 Å centered about the origin of the coordinate space. You can pick atoms nearby a selection with the phrase "within <distance> of <selection>" and all residues with the same property as a given selection as "same <property> as <selection>".

See section § 7 for a full description of the selection command.

2.6 Comparing Two Structures

Let's start from scratch by deleting everything: open the Mol form [§ 3.3.2], select every line in the browser (there should be only one), and press the Delete button.

Start by loading the `mbco.pdb` structure with the Files form. Turn on just the heme, CO, and histidines by using the selection commands `resname HEM CO or resid 64 93`. The dot (probably green) in the middle is the iron and you can verify that by clicking on it with the mouse. The label `HEM154:FE` should appear both on the display and in the text console.

To get the distance between the iron and the oxygen of the CO, click with the middle mouse button first on the iron and then on the oxygen. The first click turned the FE label off and the second turned the O label on and drew a line between the two atoms with the distance drawn in the middle and a bit to the right. The distance between the two atoms is 2.94 Å, as compared to 2.93 Å in the paper; not bad. However, picking the distance between the FE and the C of the CO reveals a distance of 1.91 Å as compared to 1.85 Å in the paper. The difference is that the structures in the VMD distribution are actually preliminary structures obtained before the final coordinates were determined.

In order to experiment with more complex picking modes, consider the angle made by the O of the CO with the FE of the heme and the NE2 of residue 93 (you can click on the atoms to find which ones are which). Using the right mouse button in the Display window, select 'Mouse Mode', then 'Pick Item', then 'Angles.' This should cause the cursor to become a red crosshair. Click on each of the three atoms. After the third pick, a shallow angle will appear indicating an 8.71 degree angle between the three atoms.

Now load the intermediate `star.pdb` file which can also be found in the `proteins` directory of your distribution. Again use the Files form to do this. Both of the molecules will be loaded side by side. Go to the Graphics form and change the selection so it the same as the first, i.e. `resname HEM`

CO or resid 64 93. The two molecules are almost atop each other, making it hard to distinguish the two, so change the colors to simplify things.

First, in the Graphics form, change the Coloring method to 'Molecule'. Use the Selected Molecule chooser to change the mbco.pdb Coloring method to 'Molecule' as well. Open the Color form [§3.3.9] and scroll the Category browser down until the line 'Molecule' is visible. Click on it then click on the line which says mbco.pdb. (There may be two mbco lines if the file had been loaded before in this session.) Scroll the Colors browser up to click on 'blue'. This should change one of the molecules in the display to blue.

Next, click on the last line in the Names chooser, which says star.pdb. This time, choose 'red' from the Colors chooser. The display should be much easier to understand. The myoglobin with the bound CO is in blue and the intermediate state is in red. At this point it is easy to measure the change in position between the two different states by using the middle mouse button to pick the same atom in the two conformations.

Once that is done, it is easy to point out one interesting aspect of the way VMD handles the graphics. Go to the Mol form, select one of the two molecules, and press Toggle Fixed. Enter translation mode and move the other molecule around. Notice that the number which lists the distance between the two atoms never changes. That's because the mouse only affects the way the coordinates are translated to the screen image. It does not affect the real coordinates at all. It is possible to change the coordinates in a molecule using the text command interface, or by using the atom move pick modes [§3.1.1]).

By the way, unfix the molecules and do a 'Reset View' from the right pull-down menu to reset everything. Load up the third structure, deoxy.pdb and give it the same selection as the other two molecules. However, color this one green. Pull out Nature v. 371, Oct. 27, 1994 and turn to page 740. With a bit of manipulation you should be able to recreate the image that appears there.

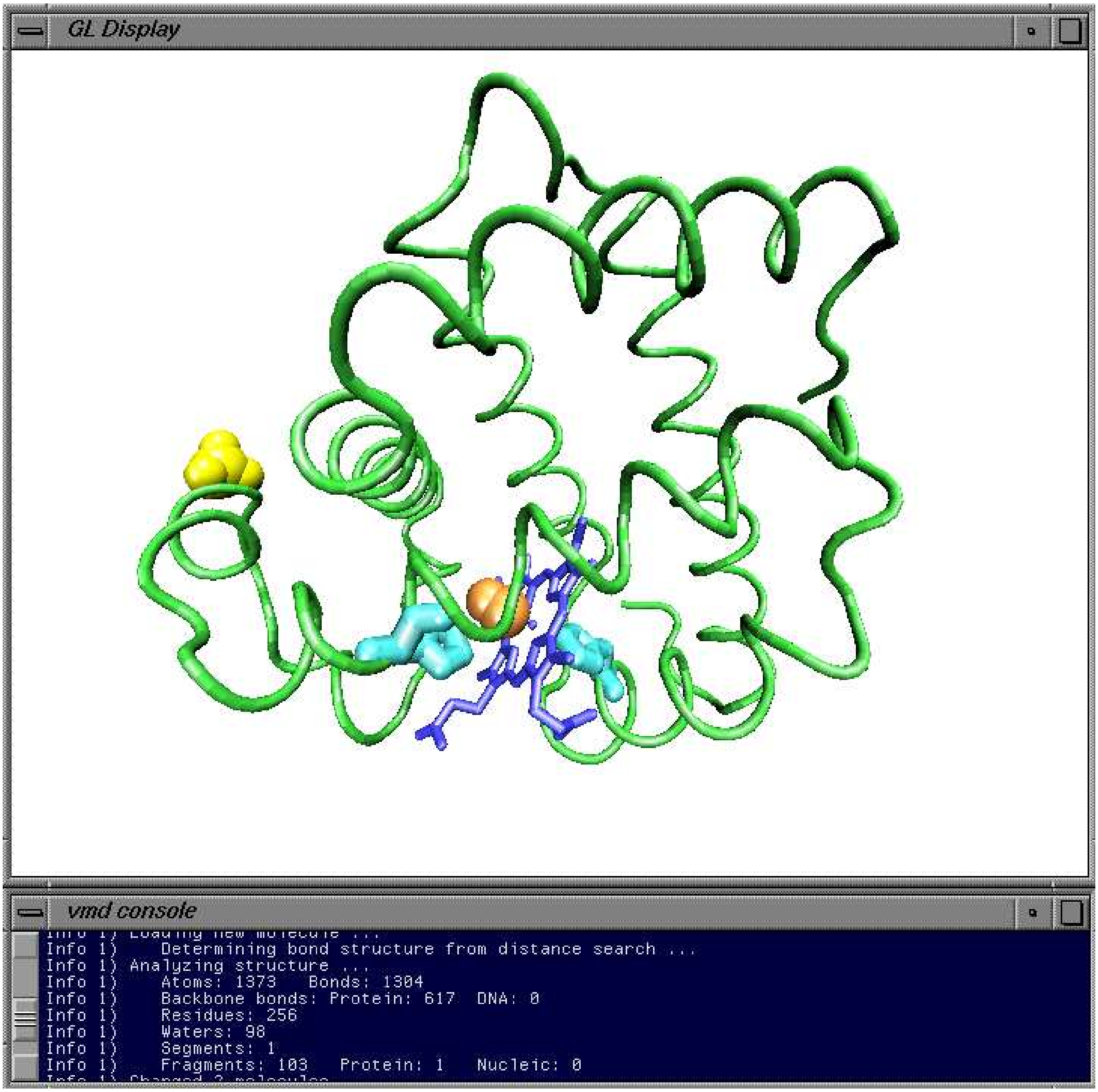


Figure 2.1: Sample VMD session displaying myoglobin.

Chapter 3

User Interface Components

There are several methods available in VMD for the user to control and interact with the molecular display. The primary methods are by using the mouse, either in the VMD graphical display window (where the molecule is displayed) or in the different graphical user interface (GUI) *forms* provided by the program. There is also a third user interface component, the text console interface, in which the user can enter and execute simple commands or more sophisticated VMD scripts. Also under development, but not documented here due to their experimental nature, are user interface components based on three-dimensional input devices and on speech- and gesture-recognition systems. This chapter describes how to use the mouse-based user interface components; it describes the basic operation of the graphical user interface components, and gives a quick overview of each available form. In VMD, the GUI is composed of several distinct windows, called forms, which provide a set of buttons, sliders, choosers, etc. to control some specific element of the program.

The other primary user interface component, the text user interface, is described fully in chapter § 11.

3.1 Using the Mouse in the Graphics Window

The graphics display window is labeled OpenGL Display and contains a display of the various molecules and other graphical objects, like the axes, which make up the scene. When the mouse cursor is located inside the graphics display window, it may be used to perform the following actions:

- Rotate, translate, or scale the displayed molecules
- 'Pick' atoms or other objects. That is, select them in order to move them around, label them etc.
- Execute commands from the *popup menu*, available by pressing the *right* mouse button
- Move the lights
- Translate and rotate a set of atoms
- Apply a force (acceleration) to a set of atoms

There are also *hot keys*, or keyboard accelerators, available when the mouse is in the graphics display window. These hot keys are bound to some VMD text commands, which are executed when a hot key is pressed. VMD has several built-in hot key commands (see Tables 3.1, 3.2, 3.3 and 3.4), and the user can add new ones as well (overriding the built-in ones if desired). A description of how to set up personal hot key commands is given in section § 15.1.

3.1.1 The Popup Menu

Pressing the right mouse button while the mouse cursor is in the graphics display window activates the VMD *popup menu*. This menu contains a number of commands, and also submenus with commands, which may be executed by selecting the option while keeping the third mouse button pressed and then releasing the button. The commands accessed via the popup menu simply execute a text command which is bound to the selected popup option, just as if that command had been entered at the VMD command prompt. There are options available in VMD to add your own submenus and commands to the main VMD popup menu, to customize this GUI control to your personal taste. See section § 15.1 for a complete description of how to do this, and how to make these popup menu customizations available every time you start up VMD.

In addition to this general popup menu, there is also an atom/molecule specific popup menu which is available when one holds down both the shift key and the rightmost mouse button over any atom in a molecule. This submenu contains options to change how the molecule is rendered, to display information about the molecule, and to perform actions such as fix the molecule in place or hide it from view. The following section describe the menu options for both the general and atom-specific popup menus.

The following sections describe the submenus and options available in the general VMD popup menu.

Mouse Mode Submenu

The mouse, while positioned within the graphics display window, is used to perform a variety of functions. There are several different *modes* which the mouse may be in at any time; the current mouse mode determines what the effect is when the user presses the left or middle button. Each mouse mode, except the lights mode (see below), causes the cursor to acquire a characteristic shape. The mouse mode is changed via the popup menu; pressing the right mouse button brings up this menu, which contains several submenus with various commands. The first two submenus, 'Mouse Mode' and 'Picking Mode', are used to set the mouse mode. Note: the actions performed from the 'Mouse Mode' submenu *do not* change the atom coordinates, while the actions performed from the 'Picking Mode' do. The available modes are as follows:

- **Mouse Mode → Rotation Mode**
  When the mouse is in this mode, holding the left mouse button down and moving the mouse rotates the molecules about axes parallel to the screen, in a 'virtual trackball' behavior. To get a rotation around the axes coming out of the screen (the 'z' axis), hold the middle button down and move the mouse left (clockwise) or right (counter-clockwise).

  You can keep the molecules rotating without continuously moving the mouse. Start the molecule moving with the mouse, as above, then release the mouse button before you stop moving the mouse. With some practice it becomes easy to impart a slight spin on the molecule, or whirl it about madly. To stop the rotation, either press and hold the left mouse button down until the molecule stops moving, or use the pop-up menu and select 'Stop Rotation.'

The hot key to enter rotation mode is `r`. Also, pressing `r` or any of the other mouse mode hot keys causes the rotation to stop.

- **Mouse Mode → Translation Mode**
  When the mouse is in translation mode, holding the left button down allows you to move the molecules parallel to the screen plane, so, for example, moving the mouse left moves the system left. To move the molecule towards or away from you, hold the middle button down and move the mouse right or left, respectively.

  The translation hot key is `t`; pressing `t` while the mouse is in the graphics display window will change the mouse to translation mode.

- **Mouse Mode → Scaling Mode**
  The scaling mode has the simplest mouse controls as pressing either the left or middle button down and moving to the right enlarges the molecules, and moving the mouse left shrinks them. The difference is that the middle button scales faster than the left button.

  The scaling hot key is `s`.

- **Mouse Mode → Move Lights Mode**
  VMD provides up to four separate light sources to illuminate the graphics objects in the display window. These light sources act independently, and use the native lighting hardware acceleration available on the graphics workstation (if any). Thus, they do not cast shadows, and their effect is not calculated with any form of raytracing or similar algorithm. They do, however, add a nice realistic effect to the objects and help give the appearance of material characteristics to spheres, cylinders, and so forth. You can use the mouse to rotate each of the light sources in space to a new position; you can even set a specific light to rotate continuously about the origin.

  The light sources are located at infinity, so only their orientation is important. To enter the move light mode in order to position a specific light, select the 'Move Light' submenu from the 'Mouse Mode' menu, and select a light. Not all lights are turned on; those that are on are indicated with a check mark. Once a light is chosen, the mouse is set to rotate that light source; a line should appear from the origin towards the chosen light. The controls for this mode are the same as the rotation mode controls, so holding the left button down allows you to rotate the light around an axis parallel to the screen, while holding the middle button allows for rotations perpendicular to the screen. To stop moving the light source, change the mouse to another mode via the popup menu or by pressing a hot key.

  If the light is not currently on, moving that light source will not affect any change in the displayed image. To turn a light on or off, use the Lights browser in the upper right of the Display form [§3.3.8].

Pick Item Mode Submenu

The mouse can also be used to select things from the screen. As with many molecular graphics programs, an atom can be picked by moving the cursor over it and clicking the left mouse button. When an atom is picked for the first time, a text label appears which shows the atom residue name and number, and the atom name. Clicking on the atom again turns the label off.

Picking atoms with the mouse is used to turn on or off various types of labels, to query for information about an object, or to move items around on the screen. You can label an atom (and display the atom name), or you can label geometric values such as the distance between two atoms (a *bond* label), an angle between three atoms (an *angle* label), or the dihedral angle formed by four atoms (a *dihedral* label). This is done by setting the mouse into the proper picking mode and then selecting the relevant atoms with the mouse.

You first select the proper picking mode by using the popup menu, and choosing the mode required to perform the desired action. The available actions when in pick item mode are:

- **Pick Item Mode → Query**
  Clicking with the left or middle mouse button on an item will print out the name of the item (e.g. the atom name) to the text console window. The hot key to set this mode is '0'.

- **Pick Item Mode → Center**
  This mode is used to change the point about which a molecule rotates when the molecule is rotated. To cause a molecule to rotate about a specific atom, select this mode and then click on that atom. The rotation point may be restored to its default position (the center of volume of the molecule) by executing the 'Reset View' option from the popup menu (see below). The hot key to set the mouse to Pick Item Mode → Centering mode is 'c'.

- **Pick Item Mode → Atom**
  Clicking on an atom will toggle on/off a label for the atom. The hot key to set this mode is '1'.

- **Pick Item Mode → Bond**
  Clicking on two atoms in a row will toggle on/off a bond distance label between the two atoms (a dotted line with the distance printed at the midpoint). The hot key to set this mode is '2'.

- **Pick Item Mode → Angle**
  Clicking on three atoms in a row will toggle on/off a label showing the angle formed by the three atoms. The hot key to set this mode is '3'.

- **Pick Item Mode → Dihedral**
  Clicking on four atoms in a row toggles on/off a label showing the dihedral angle formed by the four atoms. The hot key to set this mode is '4'.

- **Pick Item Mode → MoveAtom**
  In this mode, the position of an atom can be changed by clicking on the desired atom, and dragging with the mouse while the button is still pressed. This will change the atom coordinates. The hot key to set this mode is '5'.

- **Pick Item Mode → MoveResidue**
  This mode may be used to move all the atoms in a selected residue at the same time. Select an atom in a residue, and move it to a new position while keeping the mouse button pressed. All the atoms in the same residue as the selected one will be moved the same amount. This will change the atome coordinates. The hot key to set this mode is '6'.

- **Pick Item Mode → MoveFragment**
  A *fragment* is a set of atoms all connected by a series of covalent bonds. This mode acts just like MoveResidue, except that the atoms which are moved are all in the selected fragment rather than in the selected residue. This will change the atom coordinates. The hot key to set this mode is '7'.

- **Pick Item Mode → MoveMolecule**
  This mode may be used to move all the atoms in a selected molecule at the same time. Select an atom in a molecule, and move it to a new position while keeping the mouse button pressed. All the atoms in the same molecule as the selected one will be moved the same amount. This will change the atom coordinates. The hot key to set this mode is '8'.

When a label is added to a molecule (say, for a bond, or just to show the name of an atom), a new entry will be added to the list of current labels which is available via the Labels form [§ 3.3.7], or from the Labels popup menu commands. These controls also contain options to turn the labels on or off, or to delete them entirely from memory.

Since atom and bond picking is a very common action, these two types of labels can be added without having to set the mouse to a picking mode. To just add an atom label, click on and release the *left* mouse button while pointing to an atom, without moving the mouse during the click and release. To add a bond label, click on and release the *middle* mouse button on first one, and then the other, of the two atoms involved. Again, make sure not to move the mouse while the button is pressed down.

Display Modes Submenu

From the popup menu you can control certain settings for the graphics display window. This submenu is used to control what stereo mode is available, and how the 3D graphics are projected onto the 2D screen. The commands available in this menu are:

- **Display Modes → Perspective**
  In *perspective* mode, items closer to the viewer appear larger than items further away from the viewer. This is the default projection in VMD. Selecting this option sets the display for perspective projection.

- **Display Modes → Orthographic**
  In *orthographic* mode, graphical objects are merely projected straight onto the screen, with no consideration of their distance from the viewer. Selecting this option sets the display to use orthographic projection.

- **Display Modes → Stereo Off**
  This command sets the display window back to a non-stereo mode, if it was in another mode displaying stereo images.

- **Display Modes → CrystalEyes**
  Selecting this mode tells VMD to display the graphics in stereo, using Stereographics Corp. Crystal Eyes stereo equipment (if available). In this mode, the monitor is set to a special scanning state to display overlapping left and right eye images, and the user must wear special glasses to see the stereo image. This is described in depth in section §9.

- **Display Modes → CrossEyes**

- **Display Modes → SideBySide**
  In these stereo modes, the graphics display window is split into two side-by-side halves, and separate right and left eye images are shown in the two halves. In SideBySide mode, the left

eye image is shown on the left side, and the right eye image is shown on the right side. In CrossEyes mode, the images are reversed.

- **Display Modes → Left**

- **Display Modes → Right**
  These stereo display modes actually only show a single image; the image shown, however, is that which would be shown for the left or right eye, respectively, if any of the other stereo display modes were selected.

Fit Submenu

When one has two similar structures, one often wants to compare them. What's the difference between two X-ray structures? How much did the structure change during a simulation? To answer these questions, you must first figure out how to compare two structures, which usually means that you must find the root mean square deviation (RMSD).

Formally, given N atom positions from structure x and the corresponding N atoms from structure y with a weighting factor $w(i)$, the RMSD is defined as:

$$RMSD\,(N; x, y) = \left[\frac{\sum_{i=1}^{N} w_i \|x_i - y_i\|^2}{N \sum_{i=1}^{N} w_i}\right]^{\frac{1}{2}}$$

Using this equation by itself probably won't give you the answer you are looking for. Imagine two identical structures offset by some distance. The RMSD should be 0, but the offset prevents that from happening. What you really want is the minimum RMSD between two given structures; the best fit. There are many ways to do this, but for VMD we have implemented the method of Kabsch (Acta Cryst. (1978) A34, 827-828 or see file Measure.C in the VMD source code). This algorithm computes the transformation, needed to move one structure onto another in order to minimize the RMSD.

With the mathematical prerequisites behind us, we still need to be able to specify how to choose the atoms to compare. If you want to compare all the atoms in both structures, and they both have the same number of atoms, then the problem is easy – N is everything. This occurs most often in MD simulations when the only thing different between two structures are the coordinates.

But what about homologous sequences? In this case, the number of atoms differ because while the number of residues is the same, the sidechains have different numbers of atoms. The usual solution is to determine the RMSD based solely on the backbone atoms or, in some X-ray structures where only the C_α atoms have been determined, based on the C_α atoms. In addition, VMD allows two other methods for fitting. Fitting by heavy atoms omits the hydrogens, since their positions are often not well determined. Fitting by "picked atoms" performs only a translation to bring one atom directly on top of another molecule.

Hopefully the previous discussion revealed the importance of the options available in the fit submenu. Before examining each of them in turn, you should be aware of some VMD definitions. In VMD, a "molecule" refers to all the atoms from a structure file. The file may contain multiple molecules under standard chemical usage, but VMD still thinks of them as one molecule. Instead, those individual parts are called "fragments," for lack of a better term. With this said, the fit options in the submenu include:

- **Fit → Do RMSD Fit**
  The fitting will change the stored coordinates of one of the molucules.
- **Fit → Print RMSD**
  VMD wll print out the RMSD between the molecules after fitting them together, without changing their stored coordinates.
- **Fit → Two Molecules**

- **Fit → Two Fragments**
  These two submenus contain the same options, which select which atoms of the two molecules should be used to do the least squares fit. For the first submenu, the atoms used will be taken from all the atoms in two different molecules; for the second submenu, the atoms used will only be taken from two separate fragments, where a fragment is a connected chain of residues. The available atom selections for fitting the molecules are:
  - All atoms;
  - Heavy atoms (i.e., all except hydrogen);
  - Backbone atoms;
  - CA (that is, C_α) atoms;
  - Picked atoms (i.e., atoms currently labeled on the screen).

  Presently the VMD feedback for RMSD picks is a little terse. To make things clear, here is the process: First select one of the options (i.e., All atoms; Heavy atoms; etc.) from the fit submenu. VMD will respond with a confirmation of your choice and then it will ask you to click on one atom of each of the two selections you would like to compare. The selection you click on first will be moved to the selection you click on second. The atoms that you click on are representative of the entire molecule, fragment, set of heavy atoms, etc. associated with those atoms. Thus, if you have chosen to RMSD fit two molecules, then clicking on one atom for each of the two molecules *does* in fact specify an RMSD calculation involving *all* atoms of the two molecules.

  As a simple example of computing RMSD, load the same molecule twice. Press the '8' key (this puts VMD into the "MoveMolecule" pick mode and lets you use the mouse to change the coordinates of a molecule). Click and drag one of the molecules away from the other so there is a space between the two.

  By default the fit routines are configured to do the best fit but for this example you will only compute the RMSD, so pick "Fit → Print RMSD." Now go to the pop-up menu and pick "Fit → Two Molecules → All Atoms." Look at the VMD console window and you'll see a new line was printed to show the current state. The mouse should look like a crosshair. Pick one of the atoms in the first molecule and then one of the atoms in the second molecule. The value of the RMSD will be printed to the screen, for example:

  ```
  RMSD between the two molecules is: 1.123410
  ```

  In case of two identical molecules the RMSD calculation should, of course, give zero as a result. To stop computing RMSDs, simply change the mouse mode to what you want to do next, for example, press `r` to go to the rotate mode.

A discussion of advanced RMSD features available through the scripting interface can be found in section §16.7.

VMD's atom alignment should be able to handle most common tasks, but there are some it cannot do. If this occurs, you might want to look at Andrew Martin's ProFit at `http://www.biochem.ucl.ac.uk/~martin/text/ProFit.readme`

Labels Submenu

The labels submenu is used to control the appearance of text labels which may be placed on atoms, and of bond, angle, and dihedral angle labels which may be created between atoms. Once a label is placed on an atom or set of atoms, through the use of the mouse (see section § 3.1.1), it may be turned on or off, or deleted entirely, through use of this submenu and also through use of the Labels form [§ 3.3.7] form as explained later in this manual. The labels submenu has the following options:

- **Labels → Show**
  If a label has been previously added to, e.g., an atom, but has been hidden from view, this submenu contains options to turn the label back on. It contains submenus for the following categories of labels:
  - **Labels → Show → Atoms**
  - **Labels → Show → Bonds**
  - **Labels → Show → Angles**
  - **Labels → Show → Dihedrals**

  Each submenu contains a list of the labels which have been added to that menu; selecting the label from the submenu turns on that label.
- **Labels → Hide**
  This acts identically to the **Labels → Show** menu, except that it is used to *hide* labels from view instead of to display them. It contains the same submenus of label categories.
- **Labels → Delete**
  This acts identically to the **Labels → Show** menu, except that is is used to *delete* labels entirely. It contains the same submenus of label categories.
- **Labels → Delete all**
  This command will delete all the labels currently stored in each label category.

Animate Scene Submenu

A molecule in VMD can contain any number of coordinate sets; the resulting molecular trajectory may be played back in order to animate the molecule. This submenu contains commands to control the animation, much like the controls for a VCR or tape player. The animation can also be controlled via the Animate form [§ 3.3.5]. The commands available in this submenu are:

- **Animate Scene → play forward**
  Causes the animation to start playing forward through the frames which have been read into memory.

- **Animate Scene → play reverse**
  Plays the animation in reverse.

- **Animate Scene → step 1 forward**
  Advances the animation one frame forward, and then pauses it. If the end is reached, the animation goes back to the initial frame.

- **Animate Scene → step 1 reverse**
  Advances the animation one frame backwards, and then pauses it. If the start of the animation is reached, the last frame is displayed.

- **Animate Scene → pause**
  Pauses the display at the current frame.

- **Animate Scene → goto start**
  Rewinds the animation to the initial frame, and pauses the animation.

- **Animate Scene → goto end**
  Fast-Forwards the animation to the final frame, and pauses the animation.

- **Animate Scene → Style**
  This submenu controls how the animation behaves when it reaches the beginning or end of the trajectory. The following styles may be selected:

  - **Animate Scene → Style → Once**
    When the end of the trajectory is reached (or the beginning, if the molecule is being animated in reverse), the animation will be paused at the last (first) frame.
  - **Animate Scene → Style → Loop**
    Causes the animation to loop around back to the start of a trajectory when the end is reached, or vice versa if playing in reverse.
  - **Animate Scene → Style → Rock**
    Causes the direction in which the animation is playing to reverse when the end or start of the trajectory is reached.

Spin/Rock Scene Submenu

These commands allow you to spin or rock the displayed molecules. A *spin* is a constant rotation in a given direction, and a *rock* is rotation that alternates between one direction and its opposite. Thus, 'Spin X' rotates the system in a constant speed around the X axis and 'Rock Z' rocks it around the Z axis. The speed of rotation and total rock angle are currently fixed. Available commands are:

- **Spin/Rock Scene → Stop Rotation**
  Halts the rotational motion of the displayed molecules, i.e. sets their spin and rock values to be zero.

- **Spin/Rock Scene → Spin (X, Y or Z)**
  Imparts a constant angular velocity (spin) to the molecules around the selected axis.

- **Spin/Rock Scene → Rock (X, Y or Z)**
  Imparts a constant angular velocity (spin) to the molecules around the selected axis. After the molecule rotates through a 180° arc, the rotation direction is reversed.

Stop Rotation

Halts the rotational motion of the displayed molecules, i.e. sets their spin and rock values to be zero.

Reset View

VMD does not actually alter a molecule when it is translated or scaled. Instead, it changes the transformation matrix used to convert the atomic coordinates to screen coordinates. The initial definition of this matrix is set up so that the top molecule [§ 3.3.2] takes up most of the screen. However, after moving things about, you may want to reset the view to bring the molecule back to the initial position. Selecting 'Reset View' allows you to do just that.

Show Form Submenu

The GUI controls for VMD, also known as *forms*, consist of separate windows containing controls for the program. The forms may be individually displayed on the screen or hidden from view. This menu provides one way to display a form. To do so, select the name of the form to display from the 'Show Form' submenu. If the box to the left is checked, it is already turned on and selecting it again will not affect anything. To bring the form to the front of screen, you will have to turn the form off then back on.

Hide Form Submenu

This menu provides another way to close a form. Select the form name to turn it off.

Help

Starts an HTML viewer (like Mosaic or Netscape) and displays on-line VMD help documents. The viewer is designated by the environment variable `VMDHTMLVIEWER` [§ 17.2]. Starting help multiple times will start multiple viewers. The default web browser is Netscape. The following documents can be accessed from this submenu:

- Quick Help - A listing of common procedures in VMD
- User's Guide - An online version of what you are presently reading
- VMD Home Page - The WWW site from which you can access many VMD docs
- VMD FAQ - A listing of fequently asked VMD questions (plus answers)
- Mailing List - Allows you to access an archive of past VMD discussions
- Tcl - If you need hints on programming the scripting interface
- Software - Links to common programs used by VMD (i.e. renderers)

Note, you may need to have Netscape or Mosaic up and running before successfully selecting these options. That is, if you obtain a message such as ``netscape: not running on display :0.0'', then you will have to start up the program yourself, after which VMD will be able to direct you to the correct page.

Quit Submenu

Opens a submenu asking if you really want to quit.

Play File

Forces VMD to execute the Tcl commands found in a file of your choosing.

Save Config

With this option, VMD saves information about the representations and viewing angle characterizing your display of the scene. It creates a log file of Tcl commands which can be read back in at a later time to reproduce the images seen in the output window. See also section § 16.3 for text interface that can be used to save/restore VMD state information.

3.1.2 Atom/Molecule Specific Popup Menu

In addition to the general popup menu which is always available with a right mouse button click, there is also an atom-specific menu which appears when you hold down both the `Shift` key and the right mouse button while positioned over an atom in the display. The options available in this menu allow one to quickly customize selections and representations.

Center

Designates the position of the atom being clicked on to be the new center of transformations such as rotation, scaling, and translation.

Pick

Performs a selection of the atom being clicked on. This will toggle labels on and off

Info

The Info menu gives you a synopsis of properties for the atom that you are holding the mouse over. In particular, information about the following fields is identified:

- Name - the name of the atom as it appeared in the coordinate file
- Type - the type of the atom, as determined by an internal VMD match-up of the given name to a likely atom type associated with that name
- Index - the internal VMD index used to identify the atom; this is useful for specifying selection syntax to generate different representation styles for particular atoms. For PDB files Index corresponds to the atom number listed in the file minus 1 (so that the index starts with 0).
- Resname - the type of the amino or nucleic acid to which this atom belongs

- ResId - the internal VMD ID number of the entire residue to which the particular atom belongs. E.g., ResId for an atom of a protein is the same as the residue number of that atom as listed in its PDB file.
- Chain - if the coordinate file contained data in the "Chain" field for this atom, then that data is given here.
- Segname - the name of the segment to which this atom belongs
- X, Y, Z - the position of the atom in 3D space

In addition, there is an option "Print" in the Info submenu which simply prints the information into the console window, in case you would like to cut and paste the data somewhere else.

Mol Status

As discussed in section §3.3.2, there are a number of attributes which characterize the display of a given molecule on the screen. A few of the most important properties which can be toggled on and off with the MolStatus menu include:

- Top - makes the molecule to which the picked atom belongs to the top one (a check mark indicates its status)
- Make Active/Make Inactive - several VMD commands operate on multiple *active* molecules. Toggling this parameter will allow you to control whether or not these actions are done to your molecule of interest.
- Fix/Free - *Fixed* molecules do not undergo rotation, translation, or scaling. *Free* molecules do.
- Hide - Refrain from displaying the selected molecule

Atom Selection

This submenu allows you to selectively display portions of a molecule based on relationships to the picked atom. Presently this is the only way to define atom selections in a graphical fashion. All other selections must be described via a text-based entry form. The following criteria may be used. In each case, VMD will render only those atoms which have the following property in common with the selected atom

- Same Residue
- Same Chain
- Same Segname
- Same Fragment

In addition, the currently selected atom can be used as a representative of the entire molecule by choosing to render only the following atoms of the molecule.

- backbone - VMD should display backbone atoms only

- protein - only atoms comprising a protein should be displayed
- nucleic - only display atoms comprising nucleic acids
- water - only display water atoms in the molecule
- not waters - only display non-water atoms in the molecule
- heavy atoms- only display the heavy atoms of the molecule

Note that this submenu only affects the first representation (view) listed in the Graphics form [§3.3.4] for the given molecule.

Rendering Method

With this submenu, you can again use the currently picked atom as a representative of the entire selection it belongs to. You can switch the representation style to any of those listed in section §5. This will affect only the first representation containing the picked atom in the list of representations accessible through Graphics form.

Coloring Method

Much in the spirit of the Rendering Method submenu, with this option you can change the way in which the rendered selection associated with the picked atom is colored. Any of the options described in the section on coloring methods [§ 6] are valid.

3.1.3 Delete Representation

As its name suggests, with this option of the atom specific popup menu, you may delete the current representation of which the atom pick is a part of. The lowest-numbered representation containing the picked atom will be deleted.

3.1.4 Hot Keys

When the mouse is in the graphics window, several more commands are accessible via programmable hot keys. They allow you to do things like change mouse modes or advance the animation by a frame by simply pressing a key. The default key bindings are listed in the following tables. The commands listed are the text commands which are executed when the hot key is pressed; these text commands are explained in section §11.3. See section §15.1 for information on how to customize the behavior of these hot keys and how to add new hot key commands.

Hot Key	Command	Purpose
r, R	mouse mode 0 0	enter rotate mode; stop rotation
t, T	mouse mode 1 0	enter translate mode
s, S	mouse mode 2 0	enter scaling mode
0	mouse mode 4 0	query item
c	mouse mode 4 1	pick center
1	mouse mode 4 2	pick atom
2	mouse mode 4 3	pick bond (2 atoms)
3	mouse mode 4 4	pick angle (3 atoms)
4	mouse mode 4 5	pick dihedral (4 atoms)
5	mouse mode 4 6	move atom
6	mouse mode 4 7	move residue
7	mouse mode 4 8	move fragment
8	mouse mode 4 9	move molecule
%	mouse mode 4 10	force on atom
^	mouse mode 4 11	force on residue
&	mouse mode 4 12	force on fragment

Table 3.1: Mouse control hot keys.

3.2 Using the Forms library

The various menus used in VMD are based on the XForms Library developed by Mark Overmars and T. C. Zhao. In this library, each graphical user interface component window is referred to as a *form*. A form contains a collection of graphical controls, such as buttons, sliders, browsers, etc, described in the following sections.

3.2.1 Buttons

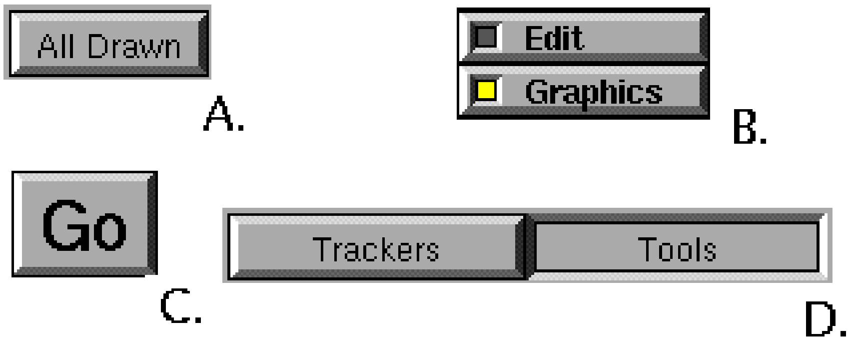


Figure 3.1: Simple buttons

Figure 3.1 shows some of the buttons used by VMD. For example, the button in 3.1 A draws all the loaded files and the one in 3.1 C writes the current screen description for use in a ray-tracing

Hot Key	Command	Purpose
x	`rock x by 1 -1`	spin about x axis
X	`rock x by 1 70`	rock about x axis
y	`rock y by 1 -1`	spin about y axis
Y	`rock y by 1 70`	rock about y axis
z	`rock z by 1 -1`	spin about z axis
Z	`rock z by 1 70`	rock about z axis
j, Cntl-n	`rotate x by 2`	rotate 2° about x
k, Cntl-p	`rotate x by -2`	rotate -2° about x
l, Cntl-f	`rotate y by 2`	rotate 2° about y
h, Cntl-b	`rotate y by -2`	rotate -2° about y
g	`rotate z by 2`	rotate 2° about z
G	`rotate z by -2`	rotate -2° about z
Cntl-a	`scale by 1.1`	enlarge 10 percent
Cntl-z	`scale by 0.9`	shrink 10 percent

Table 3.2: Rotation & scaling hot keys.

program. Some buttons, as in 3.1 B, indicate if something is turned on or off by turning an indicator light on or off. They can be turned on or off by clicking on the button. Others, shown in 3.1 D, indicate that they are pressed by appearing recessed.

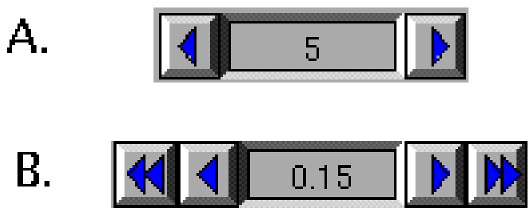


Figure 3.2: A common button motif

One common button usage motif in VMD is shown in figure 3.2. These indicate that the value in the middle can be increased by pressing the button(s) on the right and decreased by pressing the button(s) on the left. If there are two sets of buttons, as in 3.2 B, then the outer set of buttons change the number in larger increments than the inner set.

3.2.2 Sliders

Sliders are used somewhat rarely in VMD, typically to control some smoothly varying quantity that should always be set within some definite limits. An example from each is shown in figure 3.3. To use a slider, click and drag on the interior rectangle.

Hot Key	Command	Purpose
`Alt-M`	`menu main off;menu main on`	Show main menu
`Alt-m`	`menu mol off;menu mol on`	Show mol menu
`Alt-f`	`menu files off;menu files on`	Show files menu
`Alt-a`	`menu animate off;menu animate on`	Show animate menu
`Alt-e`	`menu edit off;menu edit on`	Show edit menu
`Alt-g`	`menu graphics off;menu graphics on`	Show graphics menu
`Alt-l`	`menu labels off;menu labels on`	Show labels menu
`Alt-r`	`menu render off;menu render on`	Show render menu
`Alt-d`	`menu display off;menu display on`	Show display menu
`Alt-c`	`menu color off;menu color on`	Show color menu
`Alt-s`	`menu sim off;menu sim on`	Show sim menu
`Alt-t`	`menu tracker off;menu tracker on`	Show tracker menu
`Cntl-r`	`display resetview`	Reset display
`Alt-q`	`quit confirm`	Quit VMD with confirmation
`Alt-Q`	`quit`	Quit VMD
`Alt-h`	`hyperref invert`	Invert hyper text mode (NOT help)

Table 3.3: Menu control hot keys.

Hot Key	Command	Purpose
`+,f,F`	`animate next`	move to next frame
`-,b,B`	`animate prev`	move to previous frame
`.,>`	`animate forward`	play animation forward
`,`	`animate reverse`	play animation reverse
`<`	`animate reverse`	play animation reverse
`/, ?`	`animate pause`	stop animation

Table 3.4: Animation hot keys.

3.2.3 Choosers

Choosers look like the items in figure 3.4. They are used to select one of a list of possible choices. There are two ways to change the selection. Clicking with the left mouse button brings up the list of choices. Move the mouse to the correct selection and release the button to pick that item, or release the button outside of the menu to keep the original selection. Clicking with the right mouse button advances the selection through the list by one. If the currently selected item is the last element of the list, the selection loops around to the first item.

3.2.4 Text Entry Areas

Figure 3.5 shows an example text entry. These are used for things such as atom selection or for entering filenames. To select a text entry, use the mouse and click on the appropriate field, or press the tab key to move between different text entry areas. The currently active area should have a brighter background. Some emacs-style control keys are allowed, so `^A` (control-A) goes to the beginning of the line, `^E` goes to the end, `^B` moves the cursor back by one without deletion, and `^F` moves it forward one. `^H` or backspace deletes the previous character, and `^K` deletes everything

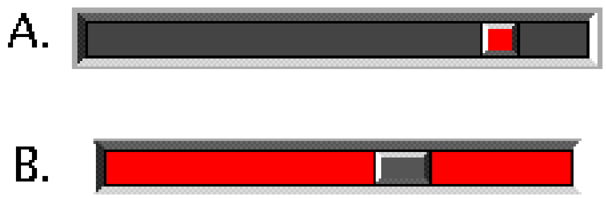


Figure 3.3: Sliders

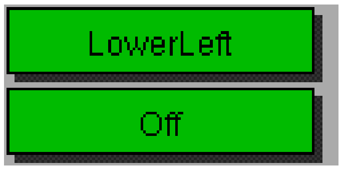


Figure 3.4: Choosers

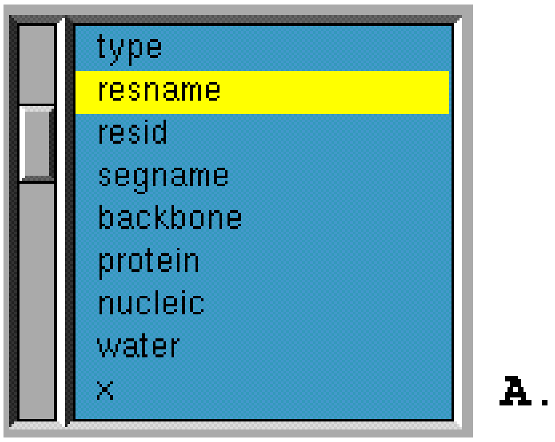
from the cursor to the end of the line. The mouse can be used to select text inside the entry field. If text is selected, pressing any normal key replaces the text, and pressing backspace deletes it. Cut-and-paste operations among XForms text entry regions and other X-Windows applications are allowed.

3.2.5 Browsers

Another way to view a list of items is to use a browser, as shown in figure 3.6. A browser displays several items on the screen (unlike the chooser which only shows one) and has a slider which is used to scroll though all the available options. Selected items are highlighted. Some browsers only allow one item to be selected, while others allow multiple selections and do not automatically deselect a line when a new item is picked. To turn off a selected item when multiple selections are allowed, click on the item, and the highlight will disappear.

One common browser motif uses coupled browsers to describe two-level hierarchies. Selecting a line in the first browser determines the possible selections in the second. An example is shown in figure 3.6 B.

Figure 3.5: A Text Entry Area



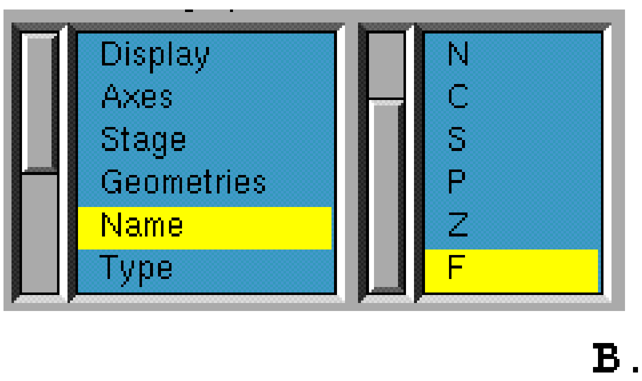


Figure 3.6: Several Browsers

3.2.6 File Browser

This form, as shown in figure 3.7, is used for input or output file selection. The current directory is shown on the line labeled Directory, and is changed by clicking on the current directory name and entering the new name into the text entry field which appears. On the left side of the form is a browser which alphabetically lists all of the available directories and the files which match the Pattern given on the second line. Note that the file selection pattern will use glob-style pattern matching, not regular expression matching.

The browser can be used to traverse the file structure. Clicking once on a directory name changes the current directory to the new one. If you click once on a file name, the name appears in the file name text entry field and can be modified as needed. To select the file name given in the field, press the enter key or click on the Ready button, or else double-clock the file name entry. To cancel this file selection, click on the Cancel button. To refresh the list of files in the directory press Rescan.

3.3 Description of each VMD form

VMD uses several different GUI forms, each designed to control a specific aspect of the molecular display (e.g., to control the appearance of the graphics display window, or to change the colors of displayed objects). Each form has a unique name, includes a button with the name of the form near the top of the window. Pressing this button will hide the form from view. The following sections give a brief description of the forms available in VMD; the remaining chapters in this manual describe the actions which these forms make available in greater detail.

3.3.1 Main Form

The Main form, also called the button bar, can turn most of the other forms on or off. It can also be used to start an HTML viewer to see the VMD quick help file, and to exit the program. The buttons with lights control which forms are turned on or off; if the light is on, the form is being displayed. Sometimes a form is turned on but is hidden behind other forms or windows. A quick

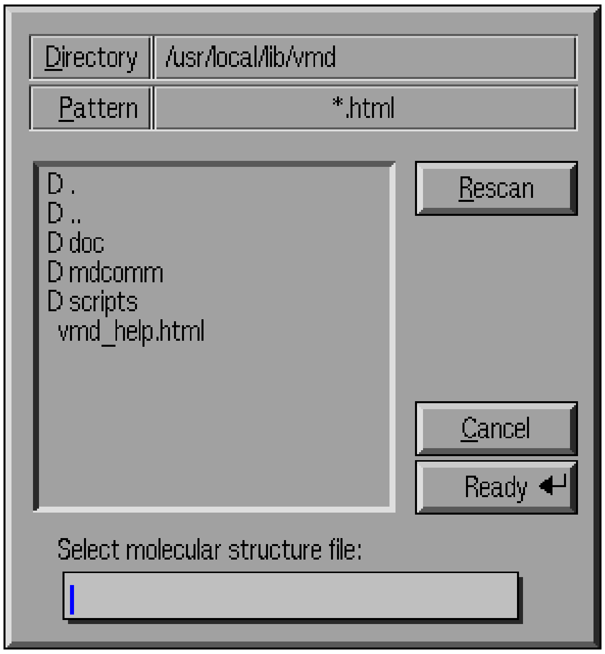


Figure 3.7: The File Browser

way to bring the form to the top is to turn it off and then on again, as implemented for you using the Menu shortcut keys described in Table 3.3.

The button bar can be closed by clicking on the Main button and, as with all other forms, by clicking on the button in the center top. There are several ways to re-open the form. The simplest is to go to the window labeled OpenGL Display and use the pop-up menu that appears when you press the right mouse button (see section §3.1 to select 'Show Form' followed by 'main'. It can also be turned on with the text command `menu main on`.

The Help button starts up an HTML viewer (such as Mosaic or Netscape) to view the Quick Help file (see General Help, §3.1.1).

Press the Quit button to exit VMD. This will bring up another form which verifies that you do indeed wish to exit. Press Yes to quit, or No to return to VMD.

3.3.2 Molecules (Mol) Form

The Mol, or Molecules, form shows the global status of the loaded molecules. Any number of molecules may be displayed by VMD simultaneously. Each molecule can separately be hidden from view, fixed in place (e.g., prevented from being affected by mouse rotation commands). This form contains controls to change the status of the molecules individually or in groups.

Loading a New Molecule

The Load From Files button will activate the Files form [§3.3.3], which is used to read a new molecular structure in from a file or set of files. Similarly, the Setup Remote Connection will display the Remote form [§3.3.12] which is used to initialize or connect to a simulation running on a remote computer. Chapter § 4 describes fully how to load a molecule and what file formats are supported.

Figure 3.8: The Main form

The Molecule List browser

The browser at the center displays information about each molecule. The Name is the file name which contained the topology information, followed by a unique integer ID which is assigned to each molecule by VMD when it is loaded. Atoms shows the number of atoms in the molecule, Frames gives the number of timesteps associated with the file, and Source is either File or Remote, reflecting whether the information was acquired from a file or a remote simulation.

Next to each molecule is a set of status flags, which indicate the current Status of each molecule. Each molecule has the following characteristics, which can be *on* or *off*:

- **Top (T)**
  Top indicates the default molecule used in the text commands when nothing is specified for the `mol` text command. It is also used in some forms (like Graphics and Animate) to determine certain values. There can be only one top molecule at a time.

- **Active (A)**
  Several commands and actions in VMD operate on many molecules. These commands, unless

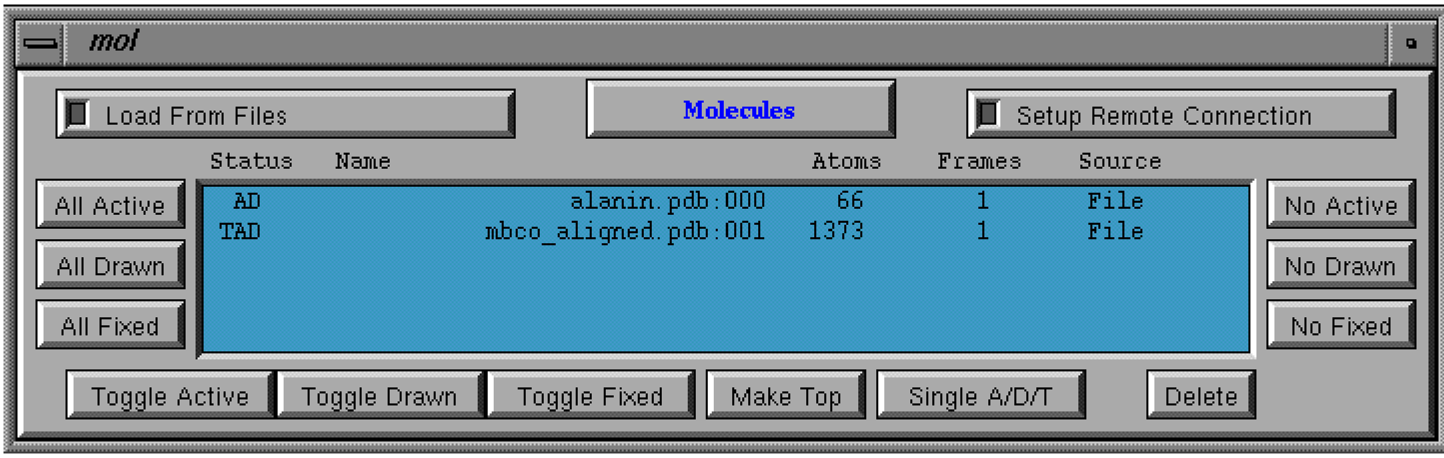


Figure 3.9: The Molecules form

specifically specified otherwise, will do their action for all the *active* molecules.

- **Drawn (D)**
  If a molecule is *Drawn* then it is being displayed in the graphics display window. This is useful for temporarily hiding a molecule from view without deleting it.

- **Fixed (F)**
  Fixed molecules do not undergo rotation, translation, or scaling. Note that while it may seem that one molecule has been moved relative to another, the difference is only apparent. The internal coordinates do not change when a standard rotation is applied by using, for example, the mouse. It is possible, however, to change the coordinates of atoms in a molecule, using the text command interface, and by using the atom move picking modes.

Changing the Molecule's Status

The status of a given molecule can be changed by selecting the molecule in the browser and pressing the toggle that controls the appropriate flag. The active, drawn, and fixed status values can be changed for several molecules at the same time by selecting several molecules before pressing the toggle. However, only one molecule can be top at any one time, so Make Top can only be applied to one selection. Pressing one of the All ... or No ... buttons either sets or unsets corresponding flag on all the loaded molecules.

The Single A/D/T button makes the selected molecule active, displayed, and top. It also resets the scene so the given molecule roughly fills the screen. It is a quick way to switch from viewing one molecule to another when it is desirable to only show one molecule within the graphics display window at a time.

Deleting a Molecule

The Delete button deletes all selected molecules. There is no prompt verifying the deletion, so take some care. If a deleted molecule was the top molecule, a new top molecule will be set from the remaining structures.

3.3.3 Files Form

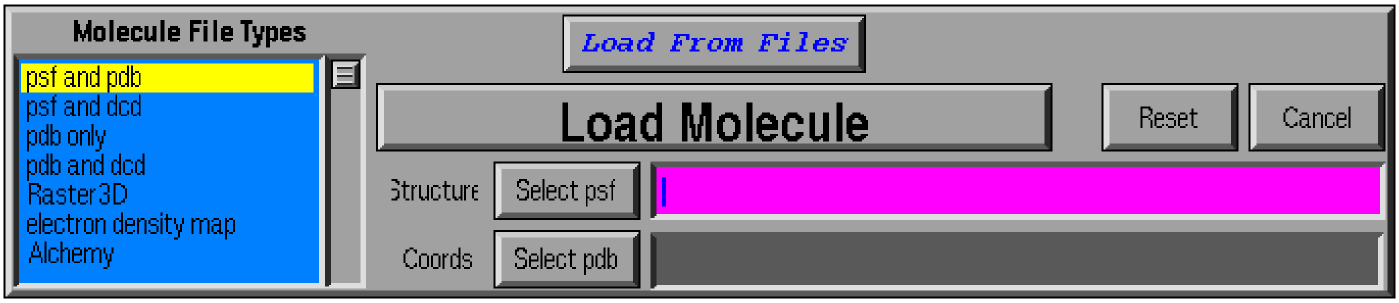


Figure 3.10: The Files form

The Files form is used to load a file from disk. It is not accessible via the button bar on the Main form. Instead, use the Load From Files button in the Molecules form or select the 'files' submenu in the 'Show Form' option available from the popup menu in the graphics display.

To use this form, first select the appropriate file type from the browser on the left (the available file types are described in section §4). One or two text input lines (depending on the molecular structure and coordinate file formats selected) will be displayed in the center of the screen. These need to be filled with the appropriate file names, which can be done in two ways. The easiest is to press the button to the left of the text area to bring up the file browser. The other way is to type the name directly into the text area. To clear the file selection, press Reset; to cancel the operation press Cancel.

Once you've selected the correct file (or files), press the large button that says Load Molecule. The button will disappear and be replaced with a message that says Loading . . . Please Wait. After the files are completely processed, the structural information will be displayed in the text window, and the Files form will be closed automatically.

3.3.4 Graphics Form

The Graphics form is the most complicated form in VMD, which could be expected from a visualization program. The details of most of the subjects discussed below are covered in the special topics on selections [§ 7], drawing methods [§ 5], and coloring methods [§ 6]. In short, a molecule can have many different representation, also referred to as *views*. Each view consists of three parts: a selection, a drawing method (also called representation style, and a coloring method. The selection determines which part of the molecule is drawn, the drawing method defines which rendering representation is used, and the coloring method gives the the color of each part of the representation.

There are actually two parts to the graphics form. The Image Controls button brings up the controls used to alter the drawing and coloring methods. The Atom Name Lists button provides access to browsers which display the lists of atom names, residue names, and so forth for the selected molecule.

This form is used to control the appearance of one molecule at a time. The molecule it affects is selected in the 'Selected Molecule' chooser at the top of the form. The browser below this chooser lists the views available for the molecule. Each line of the browser shows information about the drawing method, the coloring method, and the selection which completely specify the view. To change the attributes of a given view, click on the view that should be changed. The atom selection

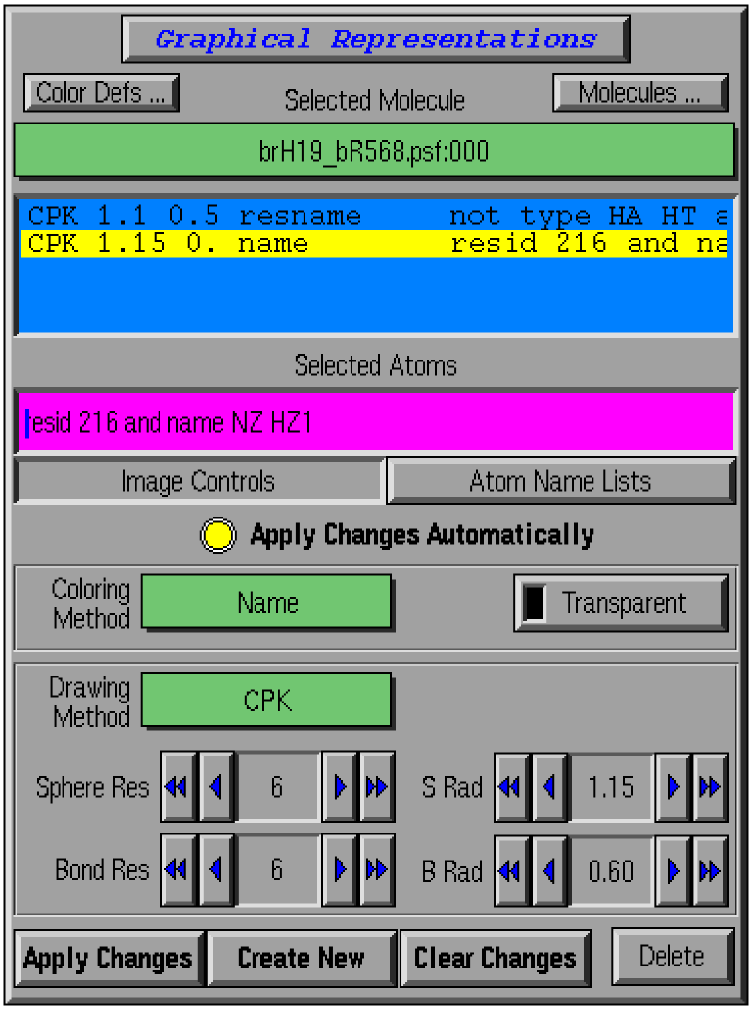


Figure 3.11: The Graphics form (in Image Controls mode)

for that view will appear in the Atom Selection text area and, if the Image Controls button is pressed, the drawing and coloring method choosers will also change to reflect the current view.

To add a new view of the molecule, enter the selection into the Atom Selection text area (or keep what is there) and press Create New. This adds the view to the currently selected molecule.

To delete a view, select the view in the browser and press the Delete button. Bear in mind that this does not delete the molecule, it only deletes a view of the molecule. (To delete the molecule, use the Mol form [§ 3.3.2].)

Image Controls

The drawing method indicates how the view's selected atoms are displayed in the graphics display window, and the coloring method indicates how to color the displayed atoms. The selected view's drawing [§ 5] and coloring [§ 6] methods are changed via the corresponding chooser. Some of the methods have additional controls which will appear when that particular method (either drawing or coloring) is chosen. The controls for the drawing method are:

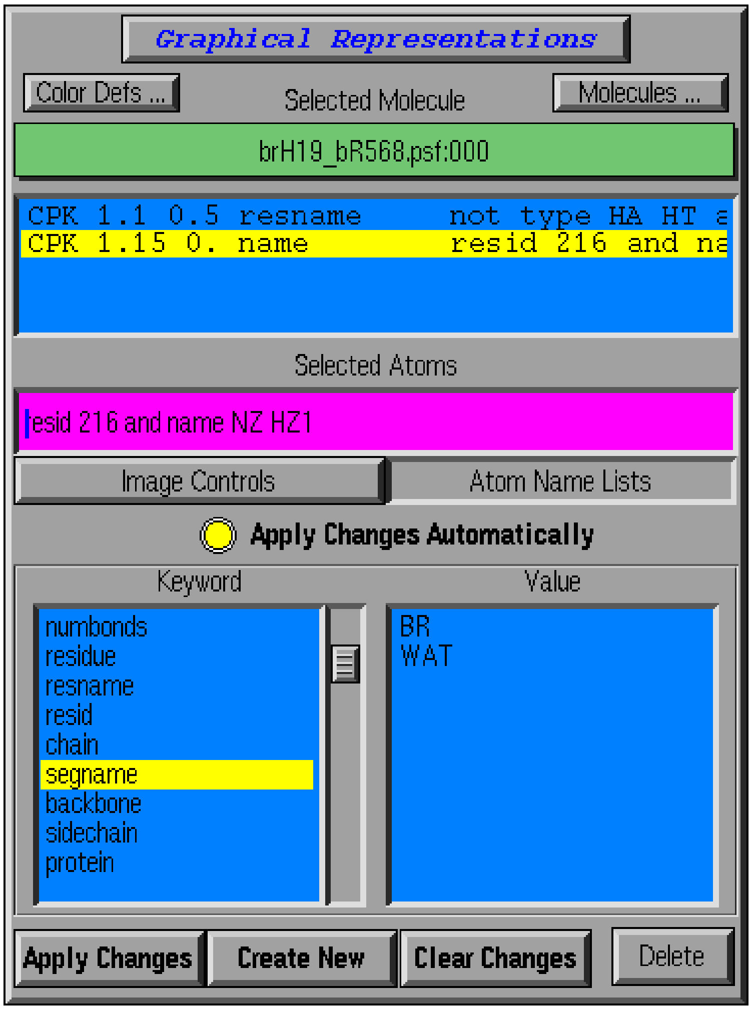


Figure 3.12: The Graphics form (in Atom Name Lists mode)

- Line Width – only works on some versions of IRIX (see section §13.1).
- Sphere Res – detail level for rendered spheres
- Sphere Rad – sphere radius scaling factor
- Cylinder Res – cylinder resolution (number of sides in the polygon approximation)
- Cylinder Rad – cylinder radius scaling factor

For the coloring method, the `transparent` button allows you to make the drawing representation semi-transparent (see §6.3). Also, when 'ColorID' is chosen for the coloring method, a text entry box is shown allowing you to specify the index of a color to use for the selection, which may be a number from 0 to 15.

Atom Name Lists

When the Atom Name Lists button is pressed, two browsers appear in place of the drawing and coloring method controls. These are used to list the available keywords and values for use in choosing atoms for the selected views. The left browser contains a list of the keywords and functions understood by the selection command [§7]. If a keyword is selected which can take on a value (for instance, `name` and `index`), then the possible names will be displayed in the rightmost browser. The functions can be identified by the `(` to the right of the name. After selecting a keyword, the right browser will display all the names associated with the keyword. For example, selecting *resname* in the left browser will show all the three-letter residue names known for the selected molecule.

Clicking on a field in the value browser will add it to the selection text field. *Double* clicking a keyword field adds the keyword to the text field. A double click is used so that the single click is available for simple viewing of the possible keyword values.

Changing Views

In addition to adding or removing views from a molecule, this form is also used to change how an existing view is displayed. After selecting a view for a specific molecule, the form's controls are updated to show the current settings for that view. Changing the settings will automatically change the respective view, and the new format will be shown in the graphics display window. The display will be updated after every change, however, which is sometimes not always desirable (for example, if a number of different aspects need to be changed at once). The *Apply Changes Automatically* button may be toggled off to change this behavior – if it is turned off, selected changes will only be applied when the *Apply Changes* button is pressed. Selecting a different view, or pressing the *Clear Changes* button, before applying the changes will reset the form controls to the current settings of the selected view.

To clear the selection text for a given view *double* click on the *Clear Changes* button. The first (and every odd) click reverts the text to its previous setting, the second (and every even) click clears it completely.

A few details regarding the mechanism for 'selections' is in order. For beginners, it is best to use the `Atom Name Lists` and the `Image Controls` together in order to generate and organize views. For instance, load a molecule and choose the `Atom Name Lists` option. If you *double* click on the `name` keyword, you will notice that two things happen. First, a listing of values appears to the right of the keyword. Second, the word "name" appears in the text selection window. Now if you go ahead and *single* click on some of the available values, you will notice that they are also printed in the text selection box. In this way you can use the mouse to generate a description of your selection and then hit `Apply Changes` in order to have that description register with VMD. The moral is, if you want to create a complex description quickly with the mouse:

- Use the `Atom Name Lists` to see what your available options are.
- *Double* click on the keyword in order to enter it in the selection text box.
- *Single* click on each of the desired values for the keyword that you want to display.
- Hit the `Apply Changes` button.
- Go back to `Image Controls` and customize your style of presentation

As a final note, keep in mind that both the Color form [§3.3.4] and the Molecule form [§3.3.2] can be accessed from the Graphics form by clicking on the buttons located in the upper corners of the form.

3.3.5 Animate Form

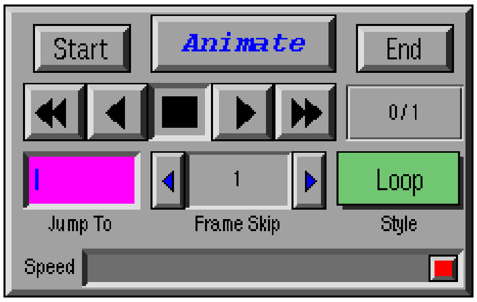


Figure 3.13: The Animate form

Each molecule displayed by VMD can contain multiple sets of atomic coordinates, which may be played back to animate the molecule and show its motion. The source of the coordinates can be, for example, from a molecular dynamics simulation, or simply multiple versions of the same molecular structure. The Animate form is one way to control the playback of these trajectories.

The Animate form handles the trajectory playback of the active molecules [§3.3.2]. The second line of controls contains five buttons which act like the buttons on a tape player or VCR. The center button (with the square) stops the animation. The button to the right of the stop button advances the animation one step forward in the trajectory, and the next button to the right continually steps the animation forward. Similarly, the buttons to the left of the stop button steps the animation backwards either once or continuously. (Just remember that the button that looks like fast-forward is really the play button, and that the one that looks like the play button is really the single step forward button.)

The molecular status shown in this form reflects the state of the *top* molecule. Commands entered via this form, however, affect all *active*, not just the top molecule. This makes it possible to position several different molecules at the same frame, and to start them in motion at the same time with one command. But, since it is not immediately obvious from the information shown in this form what molecules will be affected, some care must be taken to make sure you have the proper molecules active that you wish to animate.

Animation Speed

The rate of playback can be controlled in two ways. The Frame Skip controls change the step size. By default, the frame skip is 1, so each step of the playback increases (or decreases) the animation frame number by one. If the frame skip is 5 then the animation proceeds five times faster because only 1/5 of the frames are shown.

The Speed slider at the bottom of the form also affects the playback speed. Internally, this controls how many screen updates are needed between each step. By default, the slider is at the far right indicating that one step is performed for each screen redraw. Moving the slider to the left increases the minimum time required between updates.

Jumping to Specific Frames

The Start and End buttons are used to simplify the comparison between the initial and final structures - Start resets the current animation to the first frame, and End jumps to the last frame. If you need to jump to a specific frame, enter the frame number in the Jump To text area and press enter. One thing to bear in mind is that the frame number starts at 0, so to jump to the 5th frame, you must actually enter 4 here.

All of the controls affect the active molecules, but two parts of the animate menu use information from the top molecule [§3.3.2]. There is a display just below the End button which shows something like 0/1 or 23/59. The first number specifies the current frame number, starting at 0, and the second gives the total number of frames.

Looping Styles

When the animation is playing forward and reaches the end of the data available for the top molecule, one of three possible actions takes place, as specified in the Style chooser. The default is 'Loop', which will reset the active molecules to the first frame and continue playing forward. 'Once' will stop the animation when it reaches the last frame, and 'Rock' reverses the direction of animation. The actions are symmetrical when the animation is playing in reverse.

3.3.6 Edit Animation Form

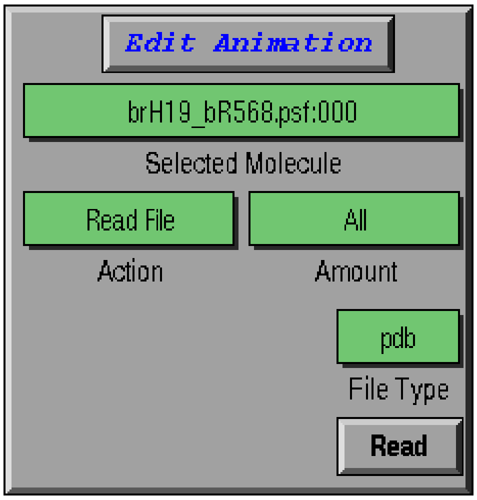


Figure 3.14: The Edit Animation form

This form is used to add, save, or delete coordinates sets (also referred to here as *frames*) from a molecule. Usually these coordinates come from successive frames of a trajectory but, as shown in the tutorial **A Quick Animation** [§3.3.6], can also be distinct conformations of the same structure.

When this form is used, it will affect only the molecule selected in the Selected Molecule chooser. There are three types of actions which can take place; read frames from or write frames to a file (in the CRD, DCD or PDB format) or delete frames from the current animation.

Reading Frames

VMD can read in new coordinate sets from either PDB files (to add a single new frame to the animation), from ascii CRD files (which may contain several frames), or from binary DCD files (which may also contain several frames). The new coordinate sets are appended to the end of the stored animation list for the selected molecule. At present, there is no way to use Babel to append non-PDB and non-CRD/DCD files. The format is determined by selecting the appropriate name in the File Type chooser. If you want to read in all the information from the file, make sure that the Amount chooser says 'All', then press the Read button in the bottom right corner. This will bring up the file browser so you can select the file. (Unlike the Files form, the file is loaded immediately after the filename is entered in the file browser.)

Sometimes you may not want to read in a whole CRD/DCD file. For example, you may only want the last frame, or every tenth frame. You can do this by choosing the 'Selected' option in the Amount chooser[§ 3.3.6]. This brings up the frame skip selection controls. Once you've chosen the appropriate values, press the Read button to bring up the file browser and finish as mentioned in the previous paragraph. The Amount options are ignored when reading in PDB files.

Writing Frames

Using the 'Write File' action, you can write the loaded frames to file in either the PDB or the CRD ASCII formats, or you may write to a binary file following the DCD format. This may be used to write out a new trajectory in a single file after assembling many frames from different sources (such as PDB CRD, or DCD files, or even from a remote simulation). You can also use this, in combination with Read File, as a way to make PDB files from a DCD/CRD trajectory.

You can either save the entire stored trajectory, or a slice of the data by using the Amount chooser [§ 3.3.6]. Then select the appropriate output file type in the File Type chooser, and press the Write button in the bottom right corner. This brings up the file browser, which you can use to enter the new filename. Once you press the Write button in the browser, the file will be written without further confirmation. See section on `atomselect` command [§ 14] for information on how to write atom coordinates for an atom selection in a PDB file.

Deleting Frames

This provides a way to delete frames from memory. First, choose the frames you wish to delete with the Amount chooser and (possibly) the frame skip controls, then press the Delete button. There is no confirmation of deletions.

One problem with this mechanism is there is no way to delete every frame except for those given by the frame skip. You can get around this by writing the skip selection to a CRD/DCD file, deleting all the frames from memory, then reading the skip back in from disk.

Amount Chooser

The meaning of this option varies depending on the action. If 'All' is selected, then all the frames will be read from the file, or all the frames will be written to the file, or all the frames will be deleted from memory.

The other option is 'Selected.' This will bring up three controls, labeled Begin, End, and Skip. These make it possible to use a subset of the frames, starting at frame Begin and selecting every

Skip frames until the End is reached. For instance, to select every fifth frame between frames 14 and 98, set:

- Begin to 14
- End to 98
- Skip to 5

(Remember that frame numbers in VMD start at 0, so frame 0 is the first frame.) The value '-1' is a special number; setting Begin to -1 is the same as starting at the first frame, End = -1 is the same as ending at the last frame, and Skip = -1 is the same as taking one step.

When the Action is 'Read File', the selection is applied to the frames from the file to be read. When it is 'Write File', the selection determines the frames to be written, and when 'Delete Frames', the selection determines the frames to be deleted from memory.

3.3.7 Labels Form

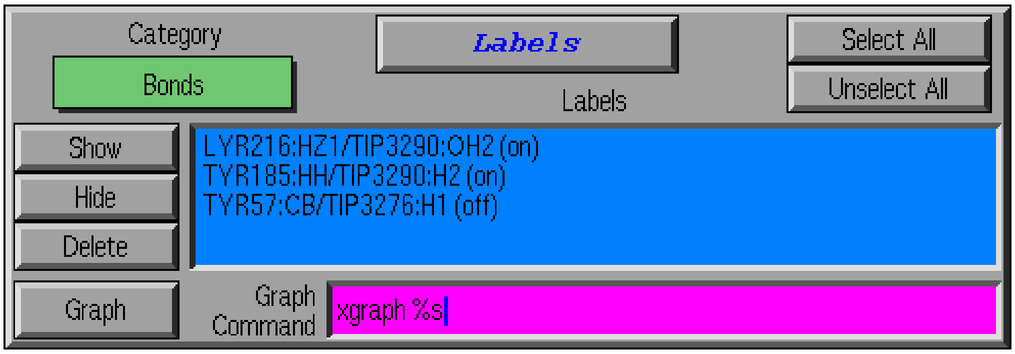


Figure 3.15: The Labels form

This form is used to manipulate the labels which may be placed on atoms, and the geometry monitors which may be placed between atoms. Labels are selected with the mouse, as discussed in §3.1.1. Once selected, the Labels form can be used to turn different labels on or off or to delete them entirely. Also, labels displaying geometrical data such as bond lengths may be graphically displayed using this form.

Label categories

The Category chooser (in the upper left) is used to select which category of labels to manipulate. The different label categories include:

- Atoms, which are shown as a text string next to the atom listing the name and residue of the atom;
- Bonds, which are shown as dotted lines between the atoms with the bond length displayed at the bond midpoint;

- Angles, which are shown as dotted lines between the three atoms with the angle displayed at the center of the defined triangle;
- Dihedrals, which are shown as dotted lines between the four atoms with the dihedral angle (the angle between the planes formed by the first three atoms and the last three atoms) shown at the midpoint of the torsional bond.
- dihedrals

All the labels for the selected category which have been previously added are displayed in the browser in the center of the form. The line itself contains from 1 to 4 atom names, depending on the category; the atom names have the form `<residue name><residue id>:<atom name>` followed by either `(on)` or `(off)`. The last word indicates if the label is turned on or off.

Modifying or deleting a label

A label can be turned on or off without deleting it, by selecting the label in the central browser and pressing the Hide button. To turn it back on, select it again then press the Show button. Press the Delete button to delete it. This browser allows multiple selections, which, for example, allows you to delete several labels at once. To select everything in the current category, press Select All; to unselect them, press Unselect All. If nothing is selected, the action is applied to everything. Thus, one way to turn everything off is to press Unselect All then press Hide. (It may seem counterintuitive, but it was done this way so all the labels could be deleted by just pressing Delete.)

Plotting a label's value

If the label has a numeric value (such as a bond length geometry monitor), it is easy to graph the change of the value over time (for multiple frames in an animation). The Graph button creates a temporary file for use by a graphing program, then optionally starts such a program to display the data. Each line of the file contains the frame number (starting at zero and expressed as a floating point number) followed by the value of the label for that frame.

Once the file is created, the text in Graph Command is executed to plot the data. By default, the text is `xmgr %s`, where the `%s` is automatically replaced with the appropriate temporary file name. When the graphing program finishes, the temporary file is deleted.

The default setup causes VMD to freeze until the graphing program finishes. It is possible to get around this by including an `&` at the end of the graph command, as in:

```
xmgr %s &
```

This may sometimes cause a problem because VMD might delete the file before the program finishes reading it. If this is a problem, try:

```
csh -c "xmgr %s &; sleep 4"
```

to cause VMD to wait a few seconds before deleting the file. You may have to increase the delay depending on the file size and type of program used. You may also try variations on the theme; for instance

```
csh -c "xterm -e vi %s ; sleep 4"
```

will bring up a vi window with the data file.

3.3.8 Display Form

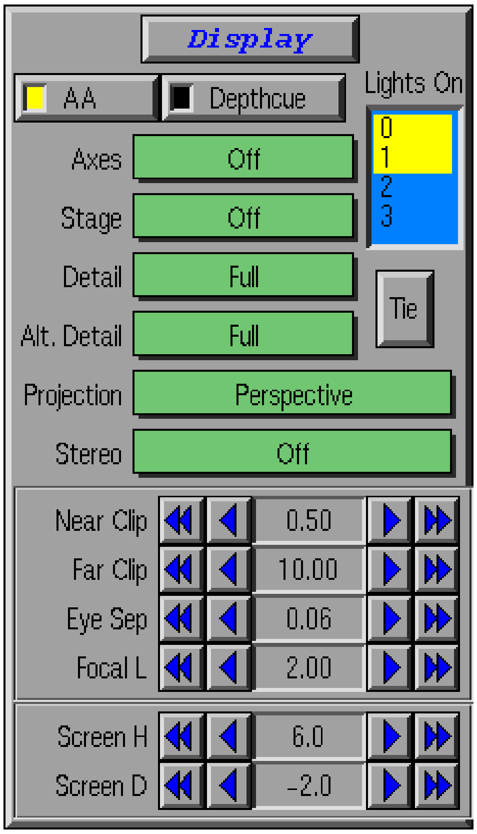


Figure 3.16: The Display form

The Display form controls many of the characteristics of the graphics display window. The characteristics which may be modified include:

- **AA** – Turns antialiasing on or off. Antialiasing helps smooth out the sometimes jagged appearance of lines resulting from the discrete pixels used to represent a line on the screen. It is highly recommended to leave this on, but on some versions of IRIX (i.e., 6.3) antialiasing is known to be buggy.

- **Depthcue** – Turns depth cueing on or off. Depth cueing causes the more distant objects to blend into the background color. This can help give a 3D effect when viewing in a non-stereo mode. **Warning**: There is a known bug that causes the antialias and depth cue buttons to stop working. The existence of the bug is known, but the solution is not (yet). Until this is worked out, it is suggested to not change the AA or Depthcue settings.

- **Axes** – A set of XYZ axes may be displayed at any one of five places on the screen (each of the corners or the center) or turned off. This is controlled by the Axes chooser.

- **Stage** – The Stage browser controls the stage, which is a checkerboard plane that can be located in any one of six places or turned off.

- **Detail** – The degree of detail in the displayed image. Nothing: show nothing; Points: show everything as points; Wireframe: show only wireframe representation for spheres, cylinders, etc.; Flat: show a flat image (no 3D effects); Full: show complete detail. The performance increases as the level of detail decreases.

- **Alt. Detail** – Same as Detail, but for the display during mouse operations on the displayed molecules, such as rotation, translation and scaling. Since VMD redraws the screen often while the molecule is rotated, it is sometimes beneficial to set lower level of detail for the time of rotation for better performance.

- **Perspective** – The view of the scene can be Perspective or Orthographic. In the perspective view (the default), objects which are far away are smaller than those near by. In the orthographic view, both objects appear at the same scale.

- **Stereo, Eye Sep, and Focal L** – These controls set the stereo mode and parameters; stereo is discussed fully in chapter §9. The Stereo chooser changes the stereo mode, while the Eye Sep and Focal L controls change the eye separation distance and the focal length, respectively.

- **Lights** – The graphics display window can use up to four separate light sources to add a realistic effect to displayed graphical objects. The Lights On browser turns these light sources on or off. If the number is highlighted, the light is on, and clicking on it turns the light off. See sectin §3.1.1 for more discussion regarding lights.

- **Clipping Planes (Near Clip and Far Clip)** – Clipping occurs when only the part of an image within a certain region of space is drawn. Only those parts of graphical objects which are between the near and far clipping planes are drawn. This feature is useful for viewing a slice of the molecule. The clipping planes also affect the depth cueing, if it is turned on. Objects at the near clipping plane are distinct while objects at the far clipping plane are indistinguishable from the background color.

  The positions of the clipping planes are changed with the Near Clip and Far Clip controls. It is not possible for the near clip to be farther away than the far clip. When using stereo, the near clip should be decreased, or even set to zero.

- **Screen Height (H) and Distance (D)** – The screen distance parameter determines the distance, in 'world' coordinates, from the origin to the display screen. If this is zero, the origin of the coordinate system in which molecules (and all other graphical objects) are drawn coincides with the center of the display. If it is less than 0, the origin is located between the viewer and the screen, while if it is greater than 0, the screen is located closer to the viewer than the origin. A negative value puts any stereo image in front of the screen, aiding the three-dimensional effect; a positive value results in a stereo image that is behind the screen, a less dramatic effect (but easier to see, for some people) stereo effect.

  The screen height, along with the screen distance, defines the geometry and position of the display screen relative to the viewer. The screen height is the vertical size of the display screen, in 'world' coordinates. Each molecule is initially scaled and translated to fit within a 2 x 2 x 2 box centered at the origin; so the screen height helps determine how large the molecule appears initially to the viewer. This parameter is used mainly to configure the VMD display to the dimensions and position of a large-screen display, such as a projector, that may be used as a stereo display.

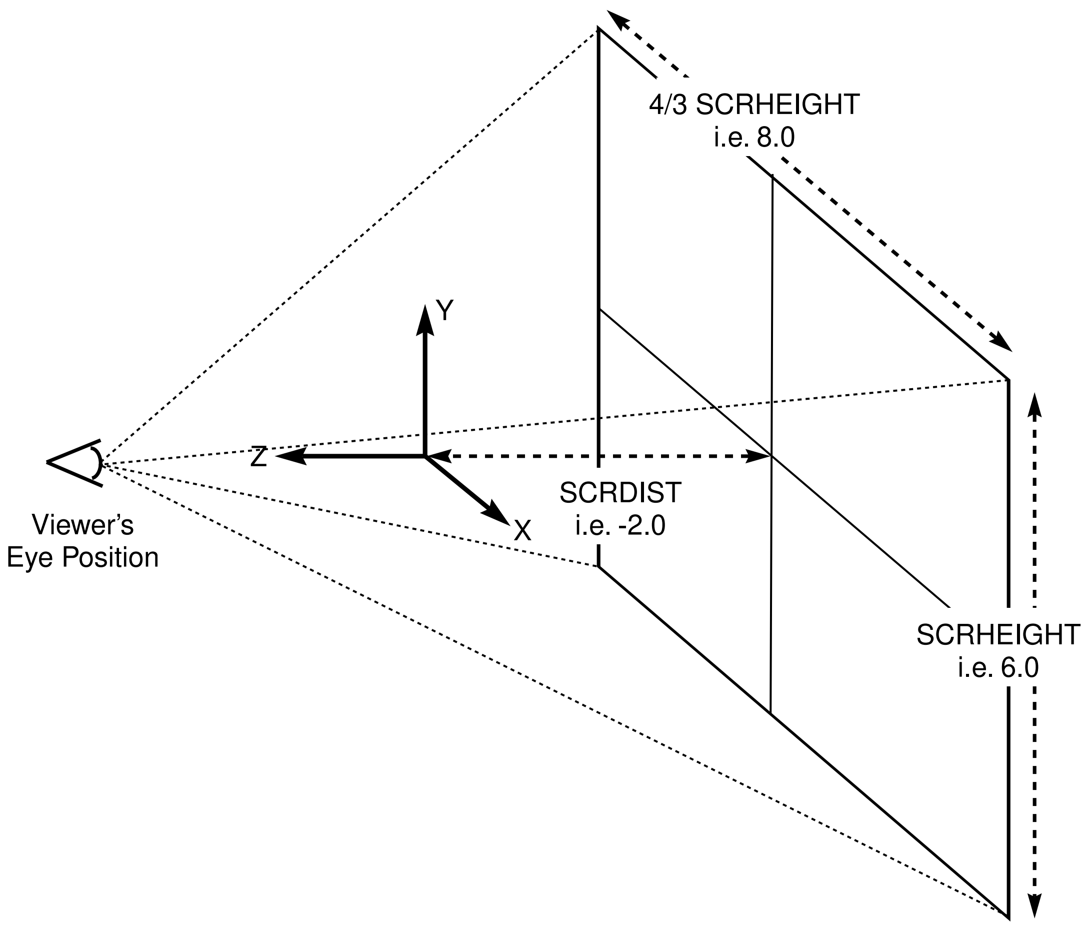


Figure 3.17: Relationship between screen height (SCRHEIGHT), screen distance to origin (SCRDIST), and the viewer

Figure 3.17 describes the relationship between the screen height, the screen distance, and the world coordinate space.

3.3.9 Color Form

VMD maintains a database of the colors used for the molecules and the other graphical objects in the display window. The database consists of several color *categories*; each color category contains a list of names, and each name is assigned a color. The assignment of colors to names can be changed with this form. There are 16 colors, as well as black (the VMD color *map*), and this form can also be used to modify the definitions of these 17 colors. For more about colors, see the chapter on Coloring [§6].

To see the names associated with a color category, click on the category in the Category browser located on the left side of the form. Click on the name to see the color to which it is mapped. To change the mapping, click on a new color in the browser to the right of the Category browser. For instance, to change the background to white, pick 'Display' in the left browser and 'Background' in the center one. The right browser will indicate the current color (which is initially `black` for the

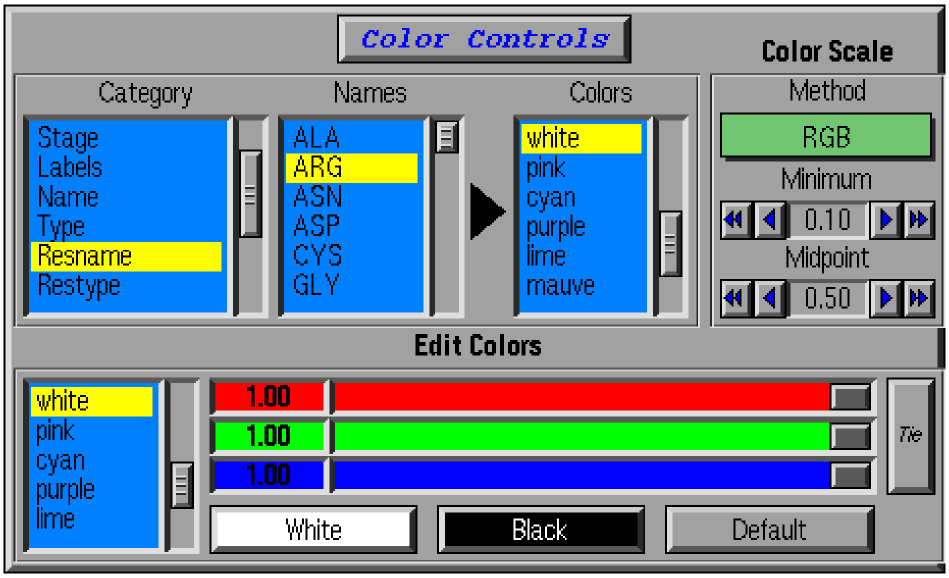


Figure 3.18: The Color form

background). Scroll through the right browser and select `white` to change the background.

Changing the RGB Value of a Color

Sometimes you may want to change the RGB value of a color in the color map, instead of changing which color is assigned to a particular name. For example, you may need to make a black and white picture and need to emphasize the contrast between a red oxygen and a yellow sulphur. This is done with the Edit Colors part of the form. First, choose the color you want changed in the browser. Then move the red, green, and blue sliders until you get the desired color. There are a few additional buttons to help you do this. The white button makes the color white (red = blue = green = 1.0) and the black color makes it black (red = blue = green = 0.0). The default button restores the color to its original RGB values. The tie button is used to make grey colors; when the button is depressed, as you move one slider the rest will follow. Press the tie button again to untie the sliders.

For example, suppose, we don't really like the appearance of the green color and we want to make it darker. To see what is happening we'll change the color of the background, so choose Display in the category field of the upper part of the Color form, then choose Background, and, finally, choose green. The background should become green. The (default) RGB values are 0.20, 0.70, 0.20. To make the color darker, let's bring the RGB values down by moving the red, green and blue sliders to the left. You can see the color changing as you move the sliders, so this way you can easily pick the color you prefer to be named 'green'. The definition can be immediately brought back to the default values by pressing the 'Default' button on the form. Note, that moving the sliders changes both solid and transparent ('green' and 'trans_green') colors (see Chapter § 6 for details).

Color Scale

Several of the coloring methods in the graphics form are used to color a range of values, as compared to a list of names. The actual coloring is determined by the color scale [§6.2.2].

There is only one color scale available at any one time (out of the ten possible) and it is changed with the Color Scale Method chooser. Changing the values of Minimum and Midpoint change some of the proportionality values used in making the scale.

The colors used by the color scale are not the colors in the color map, so the Edit Colors part of the form will not affect the color scale colors.

3.3.10 Render Form

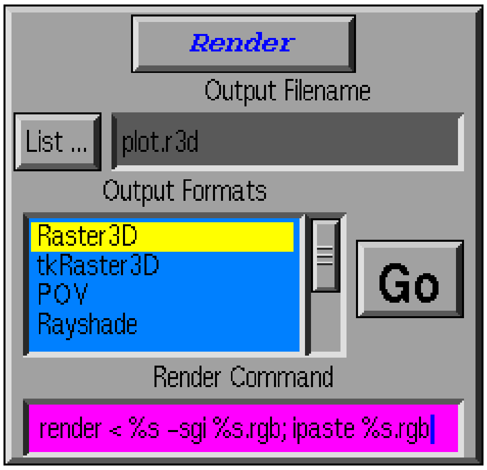


Figure 3.19: The Render form

This form is used to create a file with an image of the currently displayed graphics scene. VMD can write input script files for a number of image processing packages. These packages are listed in table 8.1. Once VMD creates an input script in one of the supported formats, the particular package can use this file to create a final output image. See Chapter § 8 for more information on how rendering is performed.

Rendering

The rendering process works in two stages. The first step writes an input file for the image processing program selected in the Output Formats browser, and the second (optionally) starts the rendering process. The file is given the name entered in the Output Filename field; a default name is given when a new format is selected, so it is best to hold off entering the filename until after the file format is selected. Another way to select the filename is available by pressing the List... button, which opens up a file browser.

Pressing the Go button writes the data file. After that, the Render Command is executed. The default command should start the appropriate rendering program if it is available.

The fastest of the currently supported programs is Raster3D. The rendering command for it has been set up to call the SGI program `ipaste` when the RGB output file is finished. VMD will wait for the rendering to finish, which causes VMD to freeze, so you may want to run the job in the background. This can be done by enclosing the existing text with ()'s and putting an & at the end. For example, the way to make the Raster3D render command run in the background is:

```
(render < %s -sgi %s.rgb; ipaste %s.rgb)&
```

Caution

There are some issues to consider when using the rendering commands, which can lead to discrepancies between the scene displayed in the VMD graphics display window and the image generated by the image processing application. These issues include:

- The near and far clipping planes are ignored;
- The colors or intensities may be slightly different, due to different colormaps, gamma values, or lighting models;
- The rendering commands do not currently support stereo output, so even if the display is currently in stereo mode, a non-stereo perspective will be used for the rendering program input script;
- Text is generally not available as a graphics primitive in the image processor input scripts, so label text will not appear, although the lines of bond, angle, etc. labels will be drawn. The only exception is in Postscript output, which supports text output.
- Dotted spheres are not drawn with dots.
- The background color may be black, as not all output formats support a background color other than black;
- Cylinders will look slightly different; in VMD cylinders are drawn as cylindrical polygons with N facets, while they may appear as smooth in the final output (which is generally looked upon as an improvement!)

3.3.11 Tracker Form

The following description assumes you have read the section on the 3D user interface and understand how the UNC trackers work in your installation. Since few outside this research group will have the required tracker hardware and the library, we will not describe this capability in any great detail. We suggest that if you want more information, please contact us via email to `vmd@ks.uiuc.edu`.

The Tracker form is used to control external 3D pointers and tools. This form is actually two forms, one for Trackers and the other for Tools, as selected by the appropriately labeled buttons on the second line.

Trackers

A *Tracker* is an external device capable of measuring the spatial (3D) position of one or more *Sensors*. The Available Trackers browser allows you to start a tracker with a given number of sensors. Most of the selections are for various UNC trackers, but only the Fastrak has been tested in the last two years. (The CAVE tracker is started automatically when the CAVE display is used.) The text emulation modes do not start a physical tracker but instead start a 'software' tracker with position and orientation values which are never updated. The `tracker` command can then be

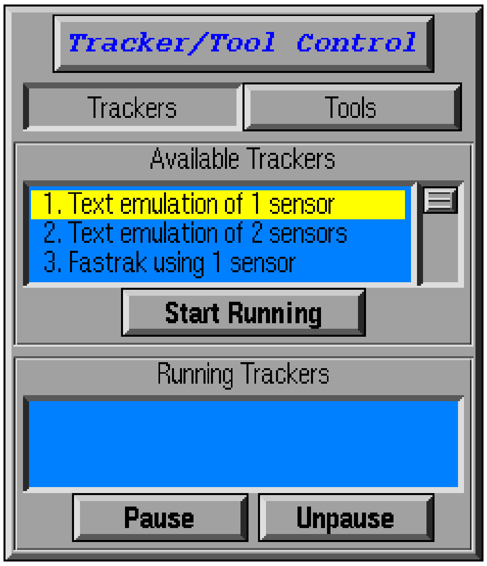


Figure 3.20: Tracker view of the Tracker form

used to override the current tracker coordinates and move the tool by hand. Indeed, that is how an external gesture user interface component has been tested with the program.

Press the Start Running button to start the tracker selected in the available browser. Be warned that there is a limitation in our interface to the UNC tracker library which does not allow you to try to start the same tracker more than once. Usually this isn't a problem, but sometimes the tracker doesn't work the first time (this usually happens when VMD previously exited accidentally while connected to the tracker. If the tracker doesn't start but you need to use it, the only thing to do in this case is quit VMD and try again.

The running trackers are listed in the lower browser. Only two things can be done to a tracker if selected: pause or unpause. When paused, none of the sensor coordinates will be updated. There is currently no way to delete a running tracker.

Tools

A *Tool* in VMD is a sensor configured to perform a particular task. The user can move the sensor about in space, and the tool connected to that sensor can then be used to perform an action. There are two types of tools: pointer (the default) or grabber. Each sensor can be configured as a different kind of tool. To make the first pointer into a grabber, select the tool in the browser and choose 'grabber' in the chooser. When you want to return to the pointer, choose 'previous'.

The standard VMD executable contains two three-dimensional components, called tools. Only two tools are available and they are hardwired to the first two sensors of the first enabled tracker. A tool can take on one of two types, a pointer or a grabber. In the future we will develop an external button input device to hook up to the trackers in order to provide controls separate from the keyboard. As it is now, we emulate the external device with the keys `F1` through `F4` for tool 0, while `F5` through `F8` are the equivalent keys for tool 1.

Each tool has a representation and several intrinsic properties: length, scale, detail, and offset. Changing the length controls the size of the tool's representation in only one direction, while

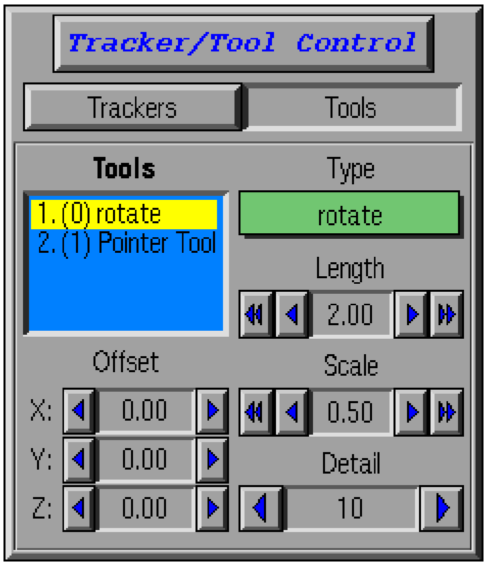


Figure 3.21: Tools view of the Tracker form

changing the size uniformly alters the whole object. If the detail is 1, the representation is very simple; often just a few lines. If the detail is 10, the object is drawn as a solid with many polygons and material characteristics. We have found that we only use a detail of 10 so we will not have this option in the future. Finally, the offset is used to translate the tool in the scene, perhaps to better locate the tool in the scene.

A pointer is just that; it points to different parts of the scene in 3D. It looks like a cylinder with a cone on the end. The pointer control can be in one of three states, length, size, and detail, which defines which value can be changed. Pressing `F2` loops through the list by one, e.g., from length to size to detail to length . . . . Pressing the `F1` key alternates between either increasing or decreasing the current value. When released and pressed again, the direction will be reversed.

3.3.12 Remote Form

As described in the chapter on Interactive Molecular Dynamics [§10], VMD has the capability to work with a molecular dynamics program running on another computer, in order to display the results of a simulation as they are calculated. The Remote form, along with the Sim form, are used to start up and connect to a molecular dynamics simulation on a remote computer, and to control the state and visual display characteristics of that simulation. VMD can display any number of different connections simultaneously. The Remote form is used to *establish* a connection, while the Sim form is used to *control* a connected simulation. To do all this, VMD uses the MDComm daemons and library, which have been developed by Rick Kufrin at the National Center for Supercomputing Applications. Before using the information in this section, however, you should first read the chapter on Interactive Molecular Dynamics [§10], which describes how this capability is installed and how it functions.

The Remote form has the following uses:

1. It is used to connect to a remote machine running the MDComm daemons.

2. It is used to submit a request to the daemon to start a new job running on the remote computer, or to attach to a job already running on the machine.

Establishing a Connection

The first step in starting a new simulation, or in attaching to a running simulation, is to establish a connection with the MDComm control daemon (called *rappd*) on the remote computer. Enter the machine name and username in the given text areas, and press the Connect button. This then directs VMD to connect to the remote MDComm daemon, which will return information about available applications, and running jobs, on that computer.

After the connection is established, two browsers will be shown:

1. A browser listing the *available applications*. These are the names of programs which can connect to VMD through this mechanism. If you are starting a new simulation, select the name of the program you wish to start, and press the Get Parameters button.

2. A browser listing the *running jobs*. These are jobs which were previously started by VMD, and left running (by using the *detach* ability in VMD). To reconnect to these jobs, select the appropriate job and press the large button at the bottom of the form.

Starting a New Simulation

After an application to start has been selected, and the parameters have been returned by pressing Get Parameters, a large browser will fill the form which is used to change the values for all the necessary simulation program parameters. At present, only the program NAMD is available to be used in this capacity, so the following discussion will assume this is the program you are using.

Each line of the browser lists a variable name, a short description, a flag indicating if the value is required or optional, and an indicator telling if the input parameter is a **s**tring, **i**nteger, or **f**loat. Additionally, the required options are shown in one color (hopefully in red, but your result may vary) while the optional ones are in another (like black). To change a line, select it and enter the new value in the text region which appears on the bottom of the form. Press Cancel to deselect the item without changing it.

When the simulation parameters browser is open but no items are selected, several options are available on the bottom of the form. Read from file allows you to set the options from a file while Write to file writes the file given the current parameter settings. Pressing Cancel returns the form to the mode which allows you to enter a new remote simulation host and a new user name.

Once all the parameters have been defined, run the simulation by pressing the Run Simulation button. You will need to go to the text window and enter the password for the remote account. This starts NAMD on the remote machine. If there are no problems, the new system will appear on the screen and be updated as each frame is computed.

Once a Connection is Made

After you have set up the parameters and pressed the Run Simulation button, or if you instead have chosen to attach to a running simulation, the structure and coordinates of the molecule being simulated will be transferred to VMD, and then VMD will continually update the molecule with new coordinates as soon as they are calculated by the remote computer. At this point, the Sim form is used to control the state of the remote simulation, to kill or detach the job, and to control how some aspects of the simulation are displayed by VMD. Even if other simulations are connected to VMD, you can use the Remote form after this to set up and start other connections.

When a connection is made, the molecular structure read from the remote simulation is added to VMD's list of loaded molecules just as if it had been read from a file. As new coordinate timesteps are calculated by the simulation and sent to VMD, they are added to the internal animation list of that molecule, and can be played back using the Animation form and controls just as a 'normal' molecule. In fact, you can read in new coordinate frames from PDB and DCD files and append them to the animation list of an on-line molecule just as with any other molecule. The main difference is that a molecule retrieved from a connected remote simulation will continue to get new data from that connection, until the connection is terminated. Such a molecule also has a few more options available for it, which are accessed through the Sim form (described next).

3.3.13 Sim Form

The Sim form allows you to control the behavior of a molecular dynamics simulations which has been previously connected to through use of the Remote form. This form contains controls to change parameters for the simulation and to affect how VMD displays the results of the simulation. The form also contains informative displays, which show the current status of the simulation connection, and such things as the current energy, temperature, and timestep of the molecular system being simulated.

At the top of the Sim form, there is a chooser which lists the currently- and previously-active simulation connects. Each time a connection is made, a new item is added to this list. As most of the controls in this form only affect the connection selected in this chooser, select there the simulation you wish to control. The informative displays in the Sim form will only reflect the state of the chosen connection

Below the connection chooser is a browser used to set modifiable parameters in the simulation. It will display the current state of these parameters, and can be used to modify them. To make a change, select the line with the relevant parameter, and in the text entry area that appears enter the new value and press `<return>`. After you enter a new value, a command will be sent to the remote simulation to change it, and there may be some delay between when the simulation gets the command, acts on it, and the results propagate back to VMD.

In the central portion of the form will be shown a *status message* for the chosen connection, and below this there is a chooser to turn on or off the display of *patches* for the simulation. The molecular dynamics program NAMD, which VMD is capable of communicating with, has the ability to run in parallel on several processors. It uses a *spatial decomposition* strategy to parallelize the task of computing the forces on each atom, where the volume of space occupied by the molecule is divided into uniform blocks, called *patches*. Each patch is assigned to a processor, and these patches may be moved between processors to balance the computational load. VMD can display a visual representation of these patches, and color the boxes according to different criteria. If you are using NAMD (or another program with similar parallel decomposition data that is being transmitted to VMD), you can turn on or off the display of these patches by using the chooser in the middle of the Sim form.

A large browser near the bottom of the Sim form displays the different energy values for the system being simulated (kinetic, electrostatic, etc.), as well as the current timestep and the temperature. It is automatically updated each time a new atomic coordinate set (timestep) is received and stored in the VMD animation loop.

At the bottom of the Sim form are two buttons:

- The Kill button, which when pressed will terminate the remote simulation which is currently selected in the connection chooser. This will not delete the molecule stored in VMD's internal lists, it will just stop the remote program from executing (and thus stop the transfer of new timesteps from that program to VMD). You must go to the Molecule form and delete the molecule from that list to completely remove the molecule from VMD memory. NOTE: If you use the Molecule form to delete a molecule which is currently connected to a running simulation, it will not only delete the molecule from VMD's memory, it will kill the remote simulation program (just as if the Kill button on the Sim form had been pressed). There will be no warning that it is doing this. Also, if you quit VMD while a simulation is running and connected, it will also kill the simulation without confirmation.
- The Detach button, which when pressed will sever the connection between VMD and NAMD, but will NOT kill the NAMD process. Instead, the simulation will be left running, and may be *attached* to later by using the Remote form.

As described for the Remote form, a molecule loaded via a remote connection acts just like any other molecule, with the extra abilities/options available via the Sim form. Once the connection is shut down, you can still use the Animate form to play the stored frames back (this can even be done while the simulation is active and connected).

Modifiable Parameters

There are some common parameters that may be modified while a simulation is running. They are changed using the browser at the top of the Sim form just below the connection chooser. These include:

- **Transfer Rate**: How often a timestep is transferred from the remote simulation program to VMD. By default, this is 1, which means every calculated timestep is sent. If this is set to some value N, then only every Nth step will send from the remote computer.
- **Keep Rate**: How often VMD saves the timestep in its animation list, instead of just discarding it after displaying it. By default, this is 0, which means that VMD does not save any frames. When this is 0, then when VMD receives a new frame it *replaces* the last frame in the animation list with the new frame, instead of *appending* it. When it is set to some number N larger than 0, then every Nth frame received from the remote simulation will be appended to the animation list, instead of being used to replace the last frame.

3.4 Other User Interfaces

VMD is the visualization component of MDScope which is an interactive environment for molecular modelling and dynamics. As part of that project we are experimenting with new modes of human-computer interaction. Most programs use the mouse and menu user interface components, or even a dial box (which, by the way, VMD does not currently support - anyone want to give us one?) but these do not fully support the full range of control needed for modelling. We envision a system where a ligand can be picked up and inserted directly into an active site, or where one can point to a molecule and say "which helix is that?"

3.4.1 External Interfaces

We have developed a simple mechanism to receive external text commands. It is based on the Tcl extension package, Tcl-DP, which adds commands to interface with standard socket communications. It allows other programs to send messages to VMD and be interpreted as if they originated from the keyboard. If the command returns information, that data is sent back to that calling process.

The external interface works by setting up VMD as a server for remote execution of a Tcl command (VMD is a Tcl-DP RPC server). Other processes can contact VMD by connecting to a port. The remote process sends a text command to VMD which interprets it and sends the result back to the remote process, which can do with it as it may.

The command `external on` starts VMD as a Tcl-DP RPC server. When an external process attempts to connect, the calling hostname is checked to see if it is allowed to run commands on the local machine. If so, the command is run. Note that `external off` does not disconnect currently attached processes, it only disallows new ones.

There is a simple security mechanism in the external command which derives from the standard Tcl-DP security. This allows or denies new connections based on the host name of the calling process and uses the command `external host`.

There are two ways to make a client that connects to VMD. The first is to use VMD itself. The command `external connect` <*hostname*> will connect to a VMD process on the given machine. The process that started the contact is the client VMD and the one that was contacted is the server VMD. The other way is to use a Tcl based shell which has Tcl-DP compiled in. Source (or look at) the file `$env(VMDDIR)/scripts/vmd/external.tcl` to see how the DP calls are made.

The client process sends a new command to the server with the command `external send` <*command*>.

Chapter 4

Loading A Molecule

The Files form is the primary means for loading a new molecule into VMD from one or more data files. VMD natively understands five molecular data file formats: PDB coordinate files, CHARMM and X-PLOR style PSF (topology) files, CHARMM and X-PLOR style DCD trajectory files, and Amber structure and trajectory files (i.e. PARM and CRD). These files contain some redundant information and can be loaded in different combinations.

The PDB file contains data about the atom, residue, and segment names, the occupancy and beta factor, and one coordinate set. The PSF and PARM files list the atom, residue, and segment names, the residue type, atomic mass and charge, and the bond connectivity. The DCD and CRD files contain only the atomic coordinates over multiple frames (timesteps).

When VMD loads a file it requires information about atom names and coordinates and tries to fill in the rest. Since the PDB file contains all this information, it does not need to be loaded with any other data files. However, the PDB file doesn't contain the atom types, masses, and charges, so these are guessed.

A PSF file does not contain coordinate information so it must be loaded along with a PDB or DCD file. If a PDB and PSF are given there is no missing data and VMD makes no assumptions. If a PSF and DCD are given then only the chain identifier and occupancy and beta values are missing so they are given a default value.

A PARM file is similar to a PSF in that it too contains no coordinate information. It must be loaded along with an CRD trajectory file. If a PARM and CRD file are loaded together, then only the segname and chain ID for the atoms are left blank. VMD fills in the remaining fields explicitly, making no assumptions.

A CRD or DCD file can be specified along with the PDB, in which case the PDB file will be read as normal, and then coordinate sets are read from the DCD or CRD until the end of the file is reached.

Additional coordinates from a PDB, CRD, or DCD file can be appended to the current coordinate set using the Edit form [§3.3.6].

4.1 Babel interface

VMD can use the program Babel, if installed, to translate a wide variety of different molecular data files into the PDB format. Babel, written by Pat Walters and Matt Stahl, can convert between many types of molecular data files. Not all of these have been tested for use with VMD, so your results may vary. Some data formats, such as XYZ, contain a series of coordinates. Babel converts these to a series of PDB files which VMD then reads as an animation. For more information about Babel, see `ftp://joplin.biosci.arizona.edu/pub/Babel` Presently VMD supports version 1.1b5 of Babel. Future versions of VMD will upgrade to support for version 2.0 of this program

VMD at present has no provision for using Babel to convert the output PDB CRD, or DCD files to anything else (see sections §3.3.6, § 11.3.1 and § 14.2 for information on writing output files).

The Babel-VMD interface uses two environment variables [§17.2], `VMDBABELBIN` and `VMDTMPDIR`. The first specifies the absolute location for the Babel binary (i.e. it must include the name of the executable itself, such as /usr/local/bin/babel). The second defines the location for the temporary PDB files made by Babel.

4.2 What happens when a file is loaded?

(See the Files form §3.3.3 for instructions on how to specify input files. See the the Edit form [§3.3.6] for instructions on how to save data as either a PDB, DCD, or CRD file. The Edit form can also be used to append [§3.3.6] PDB and DCD files to a loaded molecule.)

If only a coordinate file (i.e. just a PDB, with no PSF) is given, two passes are made through the file. The first pass determines how many atoms exist and the second reads them in. VMD then uses internal defaults to fill in the missing values. It then does a distance search to determine the bond connectivity, which will make some strange bonds if atoms are too close. If both a PSF and a PDB file are given, no approximations or guesses are made. For those interested in the details, when VMD is forced to guess the connectivity, it considers a bond to be formed whenever two atoms are within $R_1 * R_2 * 0.6$ of each other, where R_1 and R_2 are the respective radii of candidate atoms.

After the molecule is read, VMD checks to see if any new names are needed for the various coloring categories [§6.2.1]. If so, they are created and assigned a new color. (When the end of the color list is reached, the color starts again at the beginning.) Next, the bond connectivity is established and the molecule is analyzed to identify the various components, i.e., determine which residues are protein, nucleic acids, and waters, which atoms are backbone atoms, etc. A search is then made to connect these different types into larger fragments of the same type. As these searches take place, the information is printed to the screen. An example output for BPTI is:

```
Info 1) Analyzing structure ...
Info 1)    Atoms: 898   Bonds: 909
Info 1)    Backbone bonds: Protein: 231  DNA: 0
Info 1)    Residues: 58
Info 1)    Waters: 0
Info 1)    Segments: 1
Info 1)    Fragments: 1   Protein: 1   Nucleic: 0
```

There are actually several types of fragments. Protein and nucleic fragments are homogeneous; either all proteins or all nucleic acids. However, it is quite possible for a protein to be connected to a nucleic acid or some other non-protein. When this occurs, a warning message is printed, as in:

```
Warning 1) Unusual bond between residues 1 and 2
```

These warnings are known to occur with terminal amino acids, zinc fingers, myristolated residues, and poorly defined structures.

4.3 Raster3D file format

In addition to the molecular file formats, VMD can read the input file for Raster3D. (Raster3D converts an input file into a shaded raster image for use in making high quality pictures. It is often used with MolScript.) The ability to read Raster3D allows users to view MolScript files in 3D and incorporate special images into the display without having to edit the VMD code. If anyone has used this second option, we would be interested in knowing what was done.

The file format, which is part of the Raster3D documentation, describes a simple collection of triangles, spheres, and cylinders with either flat or spherical ends. Each shape is colored by an RGB triplet.

4.4 Raster3D Caveats

Nearly all of the header information is ignored (most notably, the viewing matrix).

VMD can read and display all the shapes in the format except the ends of the cylinders. (This was deliberately omitted since Raster3D uses many cylinders with spherical ends, which makes the resultant image very slow.)

Since VMD uses a palette of 16 colors, each triplet is converted into its "nearest" indexed color. This may cause images to be colored slightly differently than expected.

Chapter 5

Molecular Drawing Methods

Each molecule in VMD is drawn as several *representations*, or *views*, of the molecule. A view is just one particular way of drawing the molecule, and consists of three characteristics:

- A *rendering* method (representation *style*), which determines what shape to draw the atoms, bonds, and other components of the molecule. This chapter describes the different rendering methods available in VMD.
- A *coloring* method, which determines how to color each of the atoms and bonds included in the view. The Graphics form contains controls to set the coloring method at the right of the form. Chapter § 6 describes the different coloring methods.
- An *atom selection*, which determines which of the atoms in the molecule will be included in the view. This selection is entered in the text input field at the bottom of the Graphics form. Chapter § 7 describes the syntax used to select atoms.

A molecule can contain any number of different representations, and complex pictures of the molecule can be generated by creating views with different selections, coloring schemes, and rendering methods. For example, the protein backbone can be drawn as a smooth tube in one view, and important residues in the protein can be drawn as spheres or licorice bonds in other views. When a molecule is first loaded, it is given a 'default' view, which will draw all the atoms as lines and points, coloring each atom by what kind of element it is.

All of the different rendering methods have various parameters which determine how they are drawn. For each method, there are controls in the Graphics form which modify the associated parameters, such as the line width and sphere resolution (the graphical controls are described in section §3.3.4). Table 5.1 lists the available rendering methods, and the following sections describe these methods and the parameters which modify their appearance.

5.1 Lines

The default representation is 'Lines', which is also known as 'wireframe'. It draws a line between each atom and the atoms to which it is bonded. Both atoms have to be selected before the bond will be drawn. The first half of each bond is colored appropriately for the first atom, while the color of the final half corresponds to the second atom.

The only parameter for this option is the line thickness. On some SGIs you will only notice a difference between settings of 0, 1, and 2. Anything beyond 2 looks the same as 2. To understand

Representation styles	Description
Lines	simple lines for bonds, points for atoms
Bonds	lighted cylinders for bonds
CPK	scaled VDW spheres, with cylinders for bonds
Points	just points for atoms, no bonds
VDW	solid van der Waal spheres for atoms, no bonds
Dotted	dotted van der Wall spheres for atoms, no bonds
Solvent	dotted representation of the solvent accessible surface
Trace	connected cylindrical segments through C_α atoms
Licorice	spheres for atoms, cylinders for bonds, same radius
Ribbons	flat ribbon through the C_α atoms
Tube	smooth cylindrical tube through the C_α atoms
Cartoon	cartoon diagram (cylinders and ribbons) based on secondary structure
MSMS	molecular surface as determined by the program MSMS
HBonds	display hydrogen bonds
Surf	molecular surface as determined by SURF
Off	do not draw anything

Table 5.1: Molecular view representation styles.

why, you should see the man page for `linewidth` (look at an excerpt from that man page in section §13.1).

5.2 Bonds

Nearly everything about this option is the same as lines except that instead of drawing a bond as a line between two atoms, a cylinder is drawn instead. To be more specific, it draws an n-sided prism, where the number of sides is determined in the Graphics form by the "Bond Res" control and the radius is given by the value of "Bond Rad," in Angstroms. If the radius or number of sides gets too small, the bonds are drawn as lines.

In order to fine tune the bond representation, VMD does a small amount of trickery to the prisms. That is, imagine two hollow cylinders coming together so that the center of the face of one cylinder is in the same position as the center of the face of the other cylinder. Also suppose these two cylinders come together at 90 degrees. Although most of these two cylinders will overlap, there will appear to be a gap at their intersection.

To correct for this problem, VMD extends both cylinders somewhat so that the far ends touch. If one looks closely, this produces more of an overlap, but it is much nicer looking than the gap. When three or more bonds join at one atom, VMD chooses the lowest numbered bond and extends all other bonds to meet with that one. It then extends that lowest numbered bond to meet with the second lowest numbered one. A bit technical, but not too difficult to do.

5.3 CPK

'CPK' is a combination of both 'Bonds' and 'VDW' in that it draws the atoms as spheres and the bonds as cylinders. The resolution and radius can be modified independently. The radius of the sphere drawn in CPK mode is by default smaller than the sphere drawn in VDW mode, but this radius can be made larger. Since a sphere is drawn for each atom, it will always be slower than the VDW option, but we will work on performance for future versions. If the values for a sphere or bond attribute are too small, then those objects will not be drawn.

5.4 Points

'Points' draws each atom as a point, and does not draw any of the bonds. This option is not terribly useful.

5.5 VDW

'VDW' draws the atoms as spheres. The radius used is the van der Waals radius multiplied by a user-selectable scaling factor. The sphere resolution determines how finely to tessellate the spheres that are drawn. Drawing spheres takes some time, since they are built from a collection of triangles produced by a sphere library external to GL/OpenGL.

For those of you interested in the details, what happens is as follows: At the most primitive level, the sphere drawing algorithm starts with a shape, such as a square bipyramid (two pyramids with a square base joined base-to-base). Then a recursive bisection is applied to this shape, where at each level of recursion, given a triangular face, the endpoints of the centers are computed and scaled so as to be on the surface of the sphere. Given these endpoints and centers, one can construct four sub-triangles, which themselves are subject to bisection on the next level of recursion. A diagram illustrating these concepts is given below.

```
Subdivide each triangle in the old approximation and normalize
 the new points thus generated to lie on the surface of the unit
 sphere.
Each input triangle with vertices labeled [0,1,2] as shown
 below will be turned into four new triangles:

                      Make new points
                          a = (0+2)/2
                          b = (0+1)/2
                          c = (1+2)/2

                      Normalize a, b, c
                          a = (0+2)/2
                          b = (0+1)/2
                          c = (1+2)/2
       1
      /\              Normalize a, b, c
     /  \
   b/____\ c          Construct new triangles
   /\    /\               [0,b,a]
  /  \  /  \              [b,1,c]
 /____\/____\             [a,b,c]
0      a     2            [a,c,2]
```

The “Sphere Res” setting is actually controlling the number of levels of recursion being applied.

Note:: Due to variations in atom naming conventions, in rare instances VMD may improperly assign VDW radii to specific atoms, since VMD determines each atom type based on the first letter forming its name. For example, VMD would assume an atom named “HG” to be a hydrogen rather than a mercury. If this happens, you are always free to redefine the radii, using a syntax much like that below:

```
set sel [atomselect top ‘‘name HG’’]
$sel set radius 1.9
```

5.6 Dotted

Same as ‘VDW’ except that the spheres are drawn dotted instead of solid. That is, a dot is placed at each of the vertices of the triangle making up each sphere. This can be used, for instance, to imitate a surface representation.

5.7 Solvent

This method is similar in spirit to the Dotted representation in that it gives a quick estimate of the molecular surface with a collection of dots. However, it goes above and beyond the Dotted option by giving a more uniform coverage of the surface. The method that VMD uses to check for overlaps isn’t technically correct, but it is fast and works quite well. A technical description of the algorithm is as follows:

For each point of the surface distribution (of radius r = atom radius + probe radius) of atom i, check each of the atoms j to which it is covalently bound. If the point is too close to j, don’t display it. Also, if the point is too close to any neighbor k of j ($k \neq i$) then don’t draw it. This is fast since there aren’t that many neighbors to check, but it doesn’t omit parts of the surface in contact with atoms which aren’t one or two bonds away. This can be considered a good thing since you might get a better idea of the contact surface.

There are three parameters for this option. One is the probe radius, which was mentioned in the description. If the probe radius is too large, the problem of over-lapping surfaces between non-connected atoms becomes more apparent. The second is ”Detail”, which should probably be renamed ”Density” as it determines the surface density of the distributions. The higher the detail, the higher the density. The final option is the ”Method”. By default the surface is drawn as a collection of points, but a point is a pixel in size regardless of the scale of the molecule, so when scaled small the surface density appears high, and when scaled large, the density appears low. Method 2 draws little plus signs instead of points, which does scale better so the density appears more contant. Method 3 draw lines between the surface points that are on the same atom, but makes no attempt to connect the two spheres.

Thanks to Jan Hermans for implementation pointers and thanks again to Jon Leech for the code to compute the uniform point distributions. That code was included as part of the 1.x distribution.

5.8 Tube

There are two ways to draw a 'Tube' representation, one for proteins and the other for nucleic acids. The protein tube is a smooth curve through the selected C_α positions, and the nucleic acid tube is a smooth curve through the backbone phosphates.

The protein tube is a spline curve that passes through all the C_αs in a protein fragment. Five evenly spaced interpolation points are found along the curve to break the curve connecting the two C_αs into six line segments. If the first C_α is selected, the first three segments are colored by the color assigned to that C_α. If the second C_α is selected, the last three segments are colored by the color of the second C_α. The nucleic acid tube is constructed in the same manner except that the phosphate atoms are used.

The two controls set the spline radius and resolution and have the same meaning as they did in the 'Bond' control. However, if the bond radius becomes 0 or the bond resolution is 2 or less then the spline is drawn as a simple line. This make moving and rotation the image much faster.

It is possible to pick with the mouse the C_α which defines the tube by clicking near the middle of the six tube segments which are associated with that atom.

5.9 Trace

This representation applies much of the procedure used to construct the Tube. In the end, it connects the alpha-carbon atoms of successive residues by cylindrical segments with adjustable width. In the case of nucleic acids, it is the P backbone atoms which are connected. As always, the segment pieces are colored according to the atom they are associated with. If the cylinder radius is made 0.00, then the cylinder segments are replaced with lines.

Note: the Trace option is useful for people doing threading or protein folding work who only look at the C_α coordinates and residue names, for then they don't have to build the sidechains necessary to see their structure. Also, people working on polymers can fake their structure by naming everything "CA." in the PDB file and then using Trace.

5.10 Licorice

'Licorice' draws the atoms as spheres and the bonds as cylinders. The difference between this and 'CPK' is that the sphere radius is not controllable; instead, it is made the same size as the bond. This makes for a nice, smooth transition and is one of the most often used representations. It can be rather slow for large molecules.

5.11 Ribbon

The 'Ribbon' representation is similar to 'Tube' in that it follows the same spline curve for both the protein and nucleic acids. However, it uses additional information (the O of the protein backbone or some of the phosphate oxygens for nucleic acids) to find a normal for drawing the oriented ribbon. (There may be some problems with the ribbon definition for nucleic acids as it is possible for the nucleic acid detection routine to label a residue as a nucleic acid even though it does not have phosphate oxygens.)

Given the coordinates of each atom and the offset vector for the ribbon vector, the drawing code finds the spline curves for the top and bottom of the ribbon. The two splines are connected by triangles and both splines are drawn as small tubes. As with the 'Tube' representation, the six ribbon segments nearest the given atom are drawn with the color assigned to that atom and the atom can be selected by clicking near the center of those six elements.

Bond radius and resolution modify the tubes that make up the top and bottom of the ribbon. If the radius or resolution get too small, the tubes are not drawn (this speeds up drawing time by an appreciable amount). The line thickness controls the width of the ribbon and make it look like everything from vermicelli to lasagna. Additionally, the sugars are drawn filled in with triangles. This helps highlight the pucker.

Thanks to Ethan Merrit for the ribbon drawing algorithm taken from Raster3D.

5.12 Surf

This option uses the molecular surface solver written by Amitabh Varshney when he was at the University of North Carolina. When this option is used, the radii and coordinates are written to a temporary file and the 'surf' executable is run with the probe radius as a parameter. When finished, the output is written to another temporary file which is then read by VMD and colored and displayed. The value of the probe radius is controlled by the sphere radius, and this is identical to the probe size in Å.

This surface is rather slow in both generation and display for systems over several hundred atoms. The SURF calculation is quite exact and will show complete detail even when it isn't needed. The use of disk space as a communications medium takes up about half the time, and the method used to draw the triangles has not been optimized.

There are two environment variables which can affect the Surf display option:

- `SURF_BIN` – location of the SURF binary (defaults to `SURF_$ARCH` as defined in the vmd startup script)
- `SURF_TMPDIR` – location of the directory to use for temporary files (defaults to the value of the environment variable `VMDTMPDIR`, which defaults to `/tmp`)

A helpful trick when constructing surfaces is to use the "Apply Changes Automatically" toggle button on the graphics form wisely. That is, since surfaces often take a long time to build, changing viewing parameters such as the probe radius can cause long delays. By default, each time you hit the probe radius button, VMD rebuilds the surface. If you want to reduce or enlarge the probe radius by several increments, then you would end up rebuilding the surface multiple times. By toggling the afore-mentioned button, you can force VMD to update on your command only. This trick is sometimes helpful with other representations as well.

For a faster though arguably more imprecise surface rendering method, see the description of MSMS below.

5.13 Cartoon

The 'Cartoon' option produces a simplified representation of a protein based on its secondary structure. Helices are drawn as cylinders, beta sheets as solid ribbons, and all other structures (coils and turns) as a tube. If the secondary structure has not yet been determined, it will be calculated automatically by the program STRIDE.

A helix cylinder is constructed by finding the least squares linear fit along the coordinates of the helix's C_α atoms. If a given residue's C_α is selected, the small cylinder (found by linear interpolation along the line of best fit) is drawn with radius determined by the ??? parameter. Because this method computes a best fit, a helix must have at least 3 residues before it is drawn (those helicies with one or two residues are drawn as a coil). It is possible to pick the C_α for each cylinder segment, but they are at the location of the C_α, which is not near the axis cylinder. Interesting results occur when the whole protein is defined to be a helix and drawn as a cartoon.

The solid beta ribbon is constructed by building a spline along the center points between each beta sheet residue. Again, the spline is linearly interpolated to find the start and end points for each residue. Those are extended to construct the corners for a ribbon with rectangular cross section (the amount of extension is determined with the ??? parameter). A ribbon segment is used if the corresponding C_α atom is selected. Note that since this method assumes the protein is in a beta conformation, it draws a much smoother ribbon than the standard Ribbon option, which draw the ribbon with an oscillation along the sheet.

The other conformations are drawn as a tube. Since the endpoints of the helix cylinder and cartoon sheet are not at the C_α coordinate, the tube method was slightly changed to make the tube go to the new locations. This does not always work, resulting in a tube which does not quite connect to a cylinder.

5.14 MSMS

Another molecular surface renderer is MSMS, a program written by Michael Sanner of Olsen's lab at Scripps. This program is much faster than the Surf code in both generating the initial surface and then in manipulating the surface after it is built. See the web page http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html for more details. Available options include

- All Atoms – should the surface be of the selection (0) or of the contribution of this selection to the surface of all the atoms? (1)
- Density – triangle density on the surface (typical values are 1.0 for molecules with more than one thousand atoms and 3.0 for smaller molecules)
- Probe – Probe radius used to construct the molecular surface

5.15 HBonds

The HBond representation will draw a dotted line between two atoms if there is a possible hydrogen bond between them. A possible hydrogen bond is defined by the following criteria:

```
Given an atom D with a hydrogen H bonded to it and an atom
A with no hydrogen bonded to it, a hydrogen bond exists between
A and H iff the distance ||D-A|| < dist and the angle D-H-A < ang,
where ang and dist are user defined.
```

Only the selected atoms are searched, so both the donor and acceptor must be selected for the bond to be drawn. Also, you'll note that the above doesn't check the atom type of the donor or acceptor; the only criterion is if it already has or doesn't have a hydrogen.

One downfall of the current implementation is that it does an n^2 search of the selected atoms so you probably don't want to show all the HBonds of a very large structure. Look for performance improvements in future versions of VMD.

If you choose an HBonds representation but fail to see any hydrogen bonds, it may be because the default angle and distance criterion in VMD are too small, so you might want to try increasing the angle value from 20 to 30 degrees and the distance value from 3 to 4.

The HBonds are drawn as dotted lines of a given width. The default is 1 but you should probably increase that to 2. On most SGIs you can't make it any wider than that, as described in the man page for linewidth. The bond is colored by the color associated with the acceptor.

5.16 Off

The 'Off' representation draws nothing. It is used in place of delete to hide a selection so it can be redisplayed quickly later.

Chapter 6

Coloring Molecules and Objects

VMD maintains a database of the colors used for the molecules and other graphical objects which are visible in the display window. It keeps track of

- color name definitions - RGB, alpha value, etc.
- mappings from a color category to color name - so residue name MET is colored yellow
- the current color scale - red to white to blue, and several related parameters

There are 98 colors available in VMD, with color ids ranging from 0 to 97. Of these, 34 are named. The first 17 (with values 0 to 16) are, in order: blue, red, gray, orange, yellow, tan, silver, green, white, pink, cyan, purple, lime, mauve, ochre, iceblue, and black. The next 17 (with color indices from 17 to 33) are translucent (partially transparent) versions of the same terms. The names for these are the same as the solid color, prefixed with the word "trans_". For example, the translucent version of "yellow" is "trans_yellow".

The next group of 32 colors (from 34 to 65) are solid colors used in the color map, followed by the 32 colors (65 to 97) which make up the translucent forms of the solid colors. These can be set to one of several ranges with the Color form or the `color` text command: red→green→blue, red→white→blue, or black→white, etc. There are no names for the specific colors. The color map will be discussed in more detail in a section to follow.

Each color has six properties which consist of floating point values. Some describe the color, others the transparency, and still others how the color interacts with the surrounding light. For the most part, you will not need to alter the values given to these parameters by default. Alterations will usually only arise when you want to tweak values to output rendering files used by VRML and POV readers. Another application of changing the default color settings might be defining your own color palette or making black-and-white pictures. The six options are:

- RGB: a triplet of numbers (0.0 to 1.0) describing the color (red, green, blue); 0 indicates none of that color (also see Diffuse).
- Alpha: a number (0.0 to 1.0) describing the transparency; 1 is solid, 0 is transparent. By default, transparent objects are drawn with alpha set to .3
- Ambient: a triplet of numbers (0.0 to 1.0) describing RGB values for the ambient light for the given color. Noticeable in absence of diffuse light. Ambient light provides a uniform illumination of objects with a background lighting of the specified color. This means that if

you have ambient for the color named 'cyan' set to 1 0 0, the object colored in 'cyan' will appear red in the absence of any light sources and will have a red component in the 'shaded' areas in the presence of light sources. Default is (0.0, 0.0, 0.0).

- Diffuse: same as RGB; This property determines the color of the object. Diffuse reflections are independent of the viewing direction, but depend on the direction of the light source with respect to the surface of the displayed object.
- Specular: a triplet of numbers (0.0 to 1.0) describing the color of specular reflections. Produces highlights, the higher the value, the smaller and the brighter the highlight. Default is (1.0, 1.0, 1.0). You can reduce the reflectivity of the surface by reducing all three values (say, to 0.5 0.5 0.5). This would give white light reflections with less intensity. However, you can use any settings to produce effect of a colored reflection.
- Shininess: a number (1 to 100) describing how large is the angle of the specular reflections. The smaller the number the wider the angle and the more shiny objects appear. Default is 40.

For details regarding these material properties, consult an elementary graphics book such as Foley & Van Dam (Computer Graphics).

6.1 Color categories

VMD maintains a database of the colors used for the molecules and the other graphical objects in the display window. The database consists of several color *categories*; each color category contains a list of names, and each name is assigned a color. For example, there is a Resname color category, and within this category there are many names; one for each of the available residue names. Some of these are `ALA`, `CYS`, and `PRO`. Each name can be assigned a color from a list of 17 available colors called the *color map*. The RGB value of each color can be modified directly in the Color form [§3.3.9]. Also, each color has a semi-transparent version used when the transparent button [§ 3.3.4] is pressed in the Graphics form. To color items in a gradation manner, there are additional 32 colors used in the color scale [§6.2.2].

The different color categories in VMD are listed in table 6.1. The Color form can be used to change the assignment of colors to the names in each of these categories. For example, to change the color used to draw Arginine residues when molecules are colored by residue, you would use the Color form, select the 'Resname' category, select the 'Arg' name there, and then pick the color to use for Arginine's from the list of colors next to the names.

6.2 Coloring Methods

As described in chapter 5, each representation for a molecule has a specific *coloring method*. The coloring method determines how the color for each atom in the representation (view) is determined. These different methods use the colors assigned to the names in the categories listed above, and use those names to color the atoms. Molecular drawing methods which also draw the bonds between atoms will always color each half of the bond separately, using the color of the nearest atom for each half. Table 6.2 lists the different coloring methods available. The description for each method explains the source of the information used to determine the color.

Category	Contents
Display	The background color
Axes	The components of the axes
Stage	The colors for the checkboard stage
Name	The available atom names (color by Name)
Type	The available atom types (color by Type)
Resname	The residue names (color by ResName)
Chain	The one-character chain identifier.
Segname	The segment names (color by SegName)
Molecule	The names assigned to each molecule (color by Molecule)
Highlight	The protein, nucleic, and non-backbone colors
Restype	The residue types (color by ResType)
Structure	The secondary structure type (helix, sheet, coil) (color by Structure)
Labels	The different labels (atoms, bonds, etc.)

Table 6.1: Color categories used in VMD.

6.2.1 Coloring by color categories

The default method is to color by the atom name. The way it works is that there is a color category called 'Name' which contains a list of all the atom names (e.g., CA, N, O5', and H) that have been loaded into VMD. Each name is assigned one of the 16 main colors (e.g., cyan, blue, red, and white). When the drawing representation needs a color for a specific atom, it looks in the appropriate color category and finds that CA is colored cyan, N is blue, and so on.

Most of the coloring methods are based on color categories, so coloring by 'ResName' colors each residue name differently, 'SegName' colors each segment differently, and so on. The mapping between a given item in a color category and a color can be changed using the Color form [§3.3.9].

This allows users to make atoms with the name CA be black and the residue CYS be yellow. Some attention was given to making the colors reasonable, so that oxygens are red, nitrogens blue, sulphur and cysteines yellow, etc.

6.2.2 Color scale

Several of the coloring methods, including 'Beta', 'Charge', and 'Occupancy', describe a range of floating point values rather than a set of names. These are colored via the *color scale*, which is a list of 32 smoothly changing colors. There are many color gradations available. All of them consist of transformations of three colors. For instance, "RGB" colors the smallest value red, values near the middle of the scale are green, and the largest values are blue. Colors in-between are linear mixes of the two colors. The list of available gradations is given below.

The minimum of the range of values is linearly scaled and shifted to start at 0 and end at 1. Assume the color scale is RGB. For a given value of x in the scale range [0..1], the RGB value is found first from a linear scaling based on the midpoint. If x = 0, R is 1 (for maximum red). This continues linearly until x = midpoint, at which point, R is 0 and stays 0. The green component is 0 at both x = 0 and x = 1 and is 1 at the midpoint. Linear scaling occurs in between. The blue component is 0 for x <= midpoint, and 1 for x = 1.

An additional term, "min", is added to each of the component terms before they are merged. This shifts the final colors more towards white or black. Min can take on values from -1 to 1.

Method	Description
Name	Atom name, using the Name category
Type	Atom type, using the Type category
ResName	Residue name, using the Resname category
ResType	Residue type, using the Restype category
ResID	Residue identifier, using the resid mod 16 for the color
SegName	Segment name, using the Segname category
Molecule	Molecule all one color, using the Molecule category
Structure	Helix, sheet, and coils are colored differently
Chain	The one-character chain identifier, using the Chain category
ColorID	Use a user-specified color index (from 0 to 15)
Beta	Color scale based on beta value of the PDB file
Occupancy	Color scale based on the occupancy field of the PDB file
Mass	Color scale based on the atomic mass
Charge	Color scale based on the atomic charge
Pos	Color scale based on the distance of each atom to the center of the molecule. This is an interesting way to view globular systems.
Index	Color scale is based on the atom index. Due to the way the PDB file is organized, this actually looks nice.
Backbone	Backbone atoms green, everything else is blue

Table 6.2: Molecular coloring methods.

There is only one color scale used at a time so it is impossible to display objects colored by multiple different color scales.

6.3 Transparency

Making objects semi-transparent is a potentially powerful means of viewing multiple layers of the molecule simultaneously. Imagine a protein on the surface of, and extending part way into, a membrane. One way to visualize the extent of the penetration is to represent the lipids as 'Bonds' and make them transparent. That will show the membrane without completely obstructing the view of the protein.

Transparency is currently not implemented in the OpenGL version of VMD. Thus, the following suggestions apply only to versions of VMD compiled with the GL option for IRIX5.x and IRIX6.x platforms.

If transparent views of a molecule seem to be obscuring other views, make sure all the transparent views in the Graphics form are farther down the list of views than non-transparent views.

Also, VMD will sometimes have problems displaying transparent objects when they are not drawn in a front-to-back order. If you click on the `Transparent` button on the Graphics form and rotate the molecule around you will see instances of this problem. VMD provides a script command to correct for this, but it tends to slow down the display update considerably. The command to type into the console window is "display depthsort on". This will force VMD to arrange the objects in a front-to-back order by means of a depth sort. When many objects are on the screen, this sort

Method	Description
RGB	small=red, middle=green, large=blue
BGR	small=blue, middle=green, large=red
RWB	small=red, middle=white, large=blue
BWR	small=blue, middle=white, large=red
RWG	small=red, middle=white, large=green
GWR	small=green, middle=white, large=red
GWB	small=green, middle=white, large=blue
BWG	small=blue, middle=white, large=green
BlkW	small=black, large=white
WBlk	small=white, large=black

Table 6.3: Available Color Scale Gradations.

can result in a substantial amount of calculation. Thus, it is best to first rotate the view around until you find an angle and style that you are happy with, and then turn depth-sorting on to enable correct transparency. The command to turn depth sorting off is simply "display depthsort off."

The transparent colors change their definitions automatically when the corresponding solid colors are redefined through the the Color form [§3.3.9]. Using text interface, however, requires changing the settings for the transparent colors explicitly.

6.4 VMD Script Commands for Colors

In order to fine tune color parameters, one typically needs more sophisticated controls than those offered in the GUI. For this reason, VMD provides a number of scripting level commands for color access. These commands will be discussed in detail in chapter §11, but to give you a flavor for their use, here are a couple of examples that you may find useful right away. Most things can be done with `color` [§11.3.3] and `colorinfo` [§11.3.4] commands.

6.4.1 Adjusting the degree of transparency

To change an object's transparency value from the default of .3 to a smaller value of .1, one would type the following Tcl script into the console, or write it to a text file and source it (see **Using text commands** [§11]

```
proc change_transparency {new_alpha} {
        # This will always get the correct colors even if VMD
        # gains new definitions in the future
        set color_start [colorinfo num]
        set color_end [expr $color_start * 2]
        # Go through the list of colors (by index) and
        # change their transp. value
        for {set color $color_start} {$color < $color_end} {incr color} {
                color change alpha $color $new_alpha
        }
}
```

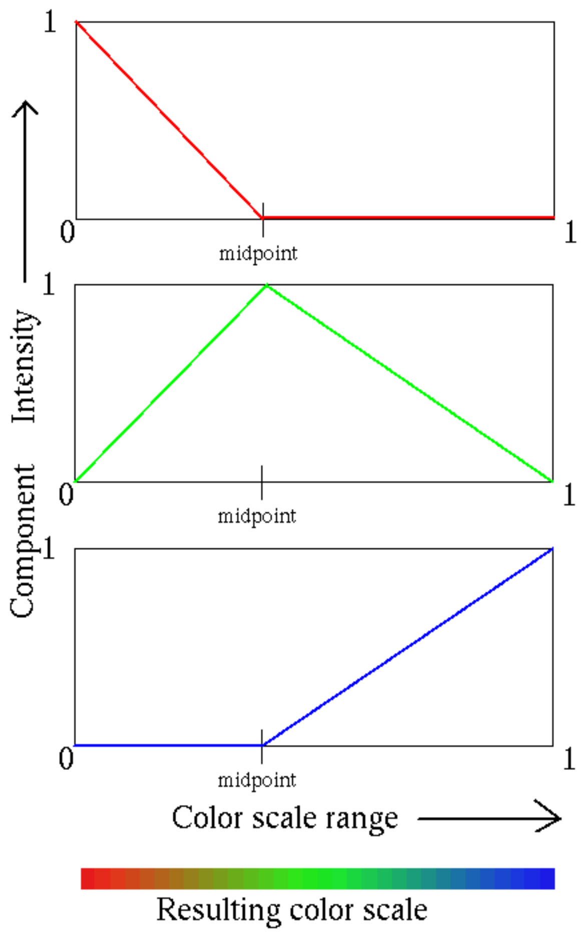


Figure 6.1: RGB color scale: the three plots shows the contributions of each color, and the resulting colors are on the bottom.

```
# to use:
change_transparency .1
```

You can even make a new popup window to do this:

```
user add menu Transp
for {set i 0} {$i < 10} {incr i} {
   set f [expr $i / 10.0]
   user add subitem Transp $f "change_transparency $f"
}
```

6.4.2 Changing the color scale definitions

Suppose that of the 32 colors, the first 15 should be red, then 2 whites, and finally 15 blues. You can use the 'color' command to modify the color scale values accordingly. The only sticky point is that you must update the transparent definitions as well or things might look strange.

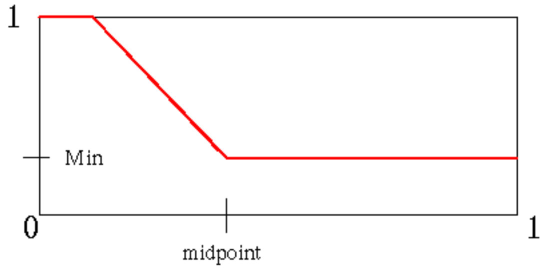


Figure 6.2: The shift to the red component of the RGB scale caused by the value of "min".

```
proc tricolor_scale {} {
  set color_start [expr [colorinfo num] * 2]
  for {set i 0} {$i < 32} {incr i} {
    if {$i == 0} {
      set r 1;  set g 0;  set b 0
    }
    if {$i == 15} {
      set r 1;  set g 1;  set b 1
    }
    if {$i == 17} {
      set r 0;  set g 0;  set b 1
    }
    color change rgb [expr $i + $color_start     ] $r $g $b
    # get the transparent version as well
    color change rgb [expr $i + $color_start + 32] $r $g $b
  }
}

tricolor_scale
```

6.4.3 Creating a set of black-and-white color definitions

To map grayscale on the color ids 0-16 (0=black; 16=white):

```
proc make_grayscale {} {
  display update off
  for {set i 0} {$i < 17} {incr i} {
    set val [expr $i / 16.0]
    color change rgb $i $val $val $val
  }
  display update on
}
```

Note that the display updates are switched off for the time of redefinition, so that the screen would not be redrawn every time one color is changed. This way the procedure works faster. The only bad thing about this idea is that black becomes white, and white changes too, so the names of the colors (yellow, orange, etc.) become useless.

6.4.4 Revert all RGB values to defaults

After some of the color definitions have been changed and you want to restore the default definitions, the following procedure might be useful.

```
proc revert_colors {} {
  display update off
  foreach color [colorinfo colors] {
    color change rgb $color
  }
  display update on
}
```

6.4.5 Setting the transparent colors according to solid

Once you have set the RGB values for solid colors, you may want to set the transparent colors to the same RGB values. Here is how one could do it:

```
proc set_trans_colors {} {
   set numofcolors [colorinfo num]
   for {set i 0} {$i < $numofcolors} {incr i} {
     lassign [colorinfo rgb $i] r g b
     color change rgb [expr $i + $numofcolors] $r $g $b
   }
}
```

6.4.6 Making the molecule to show up slowly

The only application of this that I know of is producing a movie for demonstration and having parts of the molecule appear and disappear slowly. These are very crude scripts and they can be improved.

```
   # change alpha values for all solid colors.
   proc change_alpha {newval} {
     set num [colorinfo num]
     for {set i 0} {$i < $num} {incr i} {
       color change alpha $i $newval
     }
   }

   # vary the alpha value from 0 to 1 thus forcing an object to slowly
   # appear on the screen.
   proc fade_in {} {
     for {set i 0} {$i < 1} {set i [expr $i + 0.05]} {
```

```
    change_alpha $i
  }
}
```

6.4.7 Querying VMD for Color Information

To obtain a list of the available color names:
`colorinfo colors`
To obtain a listing of the attributes of the color red:

```
foreach attrib {rgb alpha shininess ambient specular} {
   puts "$attrib [colorinfo $attrib red]"
}
```

To change the ambient value of the color red to (1.0 .5 .33):
`color change ambient red 1.0 .5 .33`

Chapter 7

VMD Atom Selection Language

VMD has a rather powerful atom selection language available. It is based around the assumption that every atom has a set of associated with it values which can be accessed through keywords. These values could be boolean (is this a protein atom?), numeric (as in the atom index or atomic mass), or string (the atom name). The values can even be referenced via a Tcl array.

To start off, here are some examples of valid selection commands in VMD. Following these will be a more in depth description of how selections work.

```
name CA
resid 35
name CA and resname ALA
backbone
not protein
protein (backbone or name H)
name 'A 1'
name 'A *'
name "C.*"
mass < 5
numbonds = 2
abs(charge) > 1
x < 6 and x > 3
sqr(x-5)+sqr(y+4)+sqr(z) > sqr(5)
within 5 of name FE
protein within 5 of nucleic
same resname as (protein within 5 of nucleic)
protein sequence "C..C"
name eq $atomname
protein and @myselection
```

There are two types of selection modes. The first is a keyword followed by a list of either values or a range of values. For example,

```
name CA
```

selects all atoms with the name CA (which could be a C_α or a calcium);

```
resname ALA PHE ASP
```

selects all atoms in either alanine, phenylalanine, or asparagine;

```
index 5
```

selects the 6th atom (in the internal VMD numbering scheme).

VMD can also do range selections, similar to X-PLOR's ':' notation:

```
mass 5 to 11.5
```

selects atoms with mass between 5 and 11.5 inclusive,

```
resname ALA to CYS TYR
```

selects atoms in alanine, arginine, asparagine, aspartic acid, cystine, and also tyrosine.

The keyword selection works by checking each term on the list following the keyword. The term is either a single word (eg, `name CA`) or a range (eg `resid 35 to 90`).

The method for determining the range checking is determined from the keyword data type; numeric comparisons are different than string comparisons. The comparison should work as expected so that "8" is between "1" and "11" in a numeric context but not in a string one. This may lead to some peculiar problems. Some keywords, such as `segname`, can take on string values but can also be used by some people as a number field. Suppose someone labeled the `segname` field with the numbers 1 through 12 on the assumption that they are numbers. That person would be rather confused to find that `segname 1 to 11` only returns two segments. Also, strings will be converted (via `atof()`) to a number so if the string isn't a number, it will be given the value of 0. It is possible to force a search to be done in either a string or numeric context using the relational operator discussed in §7.6

Selections can be combined with the boolean operators `and` and `or`, collected inside of parenthesis, and modified by `not`, as in

```
(name CA or name CB) and mass 12 to 17
```

which selects all atoms name CA or CB and have masses between 12 and 17 amu (this could be used to distinguish a C-alpha from a calcium). VMD has operator precedence similar to C so leaving the parentheis out of the previous expression, as in:

```
name CA or name CB and mass 12 to 17
```

actually selects all atoms named CA or those that are named CB and have the appropriate mass.

7.1 Definition of Keywords and Functions

The keywords available for selecting atoms in VMD are listed in tables 7.1 and 7.2 at the end of this chapter. If a keyword definition is followed by *bool*, it is either on or off. If followed by *str* it takes a value in the string context. If followed by *num* it takes a value in the number context.

Table 7.3 lists the built-in functions which may be used in atom selection expressions with keywords which take on a numeric value.

7.2 Boolean Keywords

Some selections take no values. For example, `backbone` selects the backbone atoms of the protein and nucleic acids and `protein` selects protein atoms. Giving options to these selections is an error. The selections can be used in the same way as other selections, as in:

```
protein and backbone
nucleic or protein
```

In addition, if neither `and` nor `or` are located after a boolean keyword, then an implicit `and` is inserted, so that the following are valid:

```
protein name CA         (same as: protein and name CA)
nucleic backbone
```

7.3 Short Circuiting

The boolean logic in VMD does *short circuit* evaluation on an element-wise basis. For instance, given one atom, if `X` is true then `X or Y` will be true regradless of the value of `Y`, so there is no need to evaluate it. Similarly, if `X` is false, then `X and Y` will also be false, so `Y` again need not be evaluated.

Knowing how short circuit selections work can speed up several types of selections. Consider a system with a large number of waters and a protein. The expression `protein and segname < 10` is faster than `segname < 10 and protein` since in the first selection only the atoms which are proteins have the `segname` converted to a number, while in the second selection, all the segment names are converted.

The `within` selection has its own form of short circuiting. The command can be interpreted as "find the atoms of A which are withing a given distance from B," and if A isn't given, search all the atoms. The search done in VMD takes a time roughly proportional to the number of atoms in A multiplied by the number of atoms in B, so reducing the number of atoms in A (i.e., by not testing every atoms) make the search faster.

Using the system with a lot of water and a protein, compare the selection
`protein within 5 of resid 1`
to `(within 5 of resid 1) and protein`.
The first is very fast as it does a distance search between all the protein atoms and all the atoms in resid 1. However, the second selection searches through all the atoms for those which are within 5 Å of resid and then finds which of those are protein atoms.

7.4 Quoting with Single Quotes

VMD allows two types of quoting mechanisms, single and double quotes. Single quotes are used to include spaces and other non-alphanumeric characters. Believe it or not, there are some residue names with a space in them, so they can be referenced as, for example,

```
resname 'A 1'
```

More importantly, ribose atoms can be given names like `C5'` or `C5*` (depending on the age of the PDB record). The lexer in VMD has been modified so that `C5'`, `O"`, and `N''` can be used without quotes, but it cannot handle an unquoted asterisk (`*` conflicts with multiplication and the parser is not able to resolve the difference). Some examples are:

```
name 'O5*'
segname 'A *'
name O5'
```

Quotes may also be used to get around a reserved selection word, like `x`. The selection command `segname x` will give an error because `x` is another keyword. Instead, use `segname 'x'`. There is an escape mechanism for including single quotes inside a single quoted string which uses a backslash ('\') before the single quote. This allows unusual names like `C '` to be quoted as `'C \''`.

```
segname x      <---- error; conflicts with the 'x' keyword
segname 'x'
name 'O5\''
```

Also, double quotes (discussed in the next section) can be used, as in `"C '"` or `"C \*"`.

7.5 Double Quotes and Regular Expressions

Double quotes around a string are used to specify a regular expression search (as defined by the GNU regular expression library). Regular expressions are a very powerful concept but rather hard to explain from scratch. If you don't know how to use them, you might have some luck with the unix man pages for `ed`, `egrep`, `vi`, or `regex`. If not, ask someone, or get any one of a number of books including the O'Reilly and Associates Sed and Awk book. The following should given an idea of how they work.

Regular expressions allows selection of all atoms with a name starting with `C` as:

```
name "C.*"
```

or segment names containing a number as

```
segname ".*[0-9]+.*"
```

As expected, multiple terms can still be provided on the list of matching keywords, as in

```
resname "A.*" GLY ".*T"
```

to select residues starting with an `A`, the glycine residues, and residues ending with a `T`. Kind of silly, but it is just to demonstrate. As with a string, a regular expression in a numeric context gets converted to an integer, which will always be zero.

In brief, a regular selection allows matching to multiple possibilities, instead of just one character. Table 7.4 shows some of the methods that can be used.

So there are many ways to do some selections. For example, choosing atoms with a name of either CA or CB can be done in the following ways:

```
name CA CB
name "CA\|CB"
name "C[AB]"
name "C(A|B)"
```

Several caveats for those who already understand regular expressions. VMD automatically prepends "`^\(`" and appends "`\)$`" to the selection string. This makes the selection `O` match only `O` and not `OG` or `PRO`. On the other hand, putting `^` and `$` into the command won't really affect anything, selections that match on a substring must be preceded and followed by ".*", as in `.*O.*`, and some illegal selections could be accepted as correct, but strange, as in `C)|(O` , which gets converted to `^\(C)|(O)$` and matches anything starting with a C or ending with an O.

A regular expression is similar to wildcard matching in X-PLOR. Table 7.5 is a list of conversions from X-PLOR style wildcards to the matching regular expression.

7.6 Comparison selections

Comparisons can be used in VMD to do atom selections like `mass < 5`, which selects atoms with mass less than 5 amu, and `name eq CA`, which is another way of choosing the CA atoms. The underlying idea for the comparison selection is also based on the concept that every atom has a property as specified by a keyword. When the keyword is given in the expression, the array (or vector) of the corresponding values is constructed, and the size of the array is the same as the number of atoms in the molecule. (If a single number or string is given instead of a keyword, the array consists of copies of that given value.) The operations, like addition, multiplication, string matching, and comparison, are then applied element-wise along the array. This type of selection is similar to the vector statement in X-PLOR.

Take the example `mass < 5` when applied on water, which has an oxygen of mass 15.9994 and two hydrogens of mass 1.008. VMD sees the keyword **mass** and constructs the array [15.9994, 1.008, 1.008], then sees the "5" and makes the array [5, 5, 5]. It then compares each term of the array and returns with the boolean array [False, True, True] (since 15.9994 is not less than 5, but 1.008 is). This final boolean array is then used to determine which atoms are selected; in this case, the hydrogens.

More complicated comparison selections can be constructed, either from arithmetic operations or by using some of the standard math functions (the functions are listed in Table 7.3). Probably the most often used function will be `sqr`, which squares each element of the array. Thus, the command to select all atoms within 5 Å of a point (x,y,z) = (3,4,-5) in space is:

```
sqr(x-3)+sqr(y-4)+sqr(z+5) <= sqr(5)
```

7.7 Comparison Operators

There are two types of comparison operators — numeric and string — which allow the user to specify the appropriate comparison function. Suppose the segment name, which takes on a string value, contains the names '11', and '8'. VMD cannot figure out if '8' should be less than '11' (in the numeric sense) or greater than '11' (in the lexographical sense). Instead of trying to resolve this problem through some sort of internal heuristics, VMD leaves it up to the user so that `8 < 11` but `11 lt 8`. (Perl users should recognize this solution.)

The numeric comparisons are the standard ones: `<`, `<=`, `=` or `==`, `>=`, `>`, and `!=`. The corresponding string comparisons are: `lt`, `le`, `eq`, `ge`, `gt`, and `ne`. As in perl there is a "match" operator, `=~`, so that

```
'CA' =~ "C.*"
segname =~ "VP[1-4]" (matches VP1, VP2, VP3, and VP4, present in some
                      virus structures)
```

are valid. No distinction is made between single and double quotes.

7.8 Other selections

7.8.1 sequence

VMD supports selection based on the one-letter amino acid sequence with the `sequence` selection keyword. This allows selections of the form

```
sequence APD
sequence "C..C"       (might be used to pick out zinc fingers)
sequence AATCGGAT
```

Unlike the other string selection commands which take one of three types of strings, all the strings for `sequence` are taken as regular expressions (though strings with non-alphanumerics must still be quoted to get past the input parser). The method works by taking each of the protein and nucleic acid fragments (pfrag and nfrag) in turn and constructing the one-letter amino acid sequence. If a regular expression matches any of the sequence, the atoms in the matching residues are selected. Multiple matches are allowed, though they cannot overlap. As is usual with regular expressions, the largest possible match is made, so take care with expressions like `C.*C`.

7.8.2 `within` and `same`

Two useful types of selection mechanisms available in VMDare: `within <number> of <selection>` and `same <keyword> as <selection>`. The first selects all atoms within the specified distance (in Å) from a selection, including the selection itself. Therefore, the command:

```
within 5 of name FE
```

selects all atoms within 5 Å of atoms named FE. One common use for this command is to limit the region of atoms shown on the screen. Another is to find atoms that may be involved in interactions. For instance:

```
protein within 5 of nucleic
```

finds the protein atoms that are nearby nucleic acids. On a cautionary note, the distance search method is not very efficient, making these selections rather slow. Some selections may be sped up by short circuiting [§7.3].

The `same <keyword> as <selection>` finds all the atoms which have the same 'keyword' as the atoms in the selection. This can be used for selections like

```
same fragment as resid 35
```

which finds all the atoms attached to residue id 35. Any keyword can be used, so selections like

```
same resname as (protein within 5 of nucleic)
```

are fine, although weird. The perhaps the most useful keyword for this command is `residue`, so you can say `same residue as ...`.

7.8.3 annmm

ANNMM is an proposed standard for defining how to get molecular data and how to reconstruct a given viewpoint. It contains a simple selection language which has been incorporated directly into VMD; partially to support the full ANNMM format, and partially to show how extensible VMDś selection language really is. We will not describe the format here.

7.9 Referencing Tcl data

Tcl variables (with either scalar or array data) can be used in the selection in one of two possible ways.

7.9.1 $references

If a keyword starts with a `$`, the rest of the keyword is used as the Tcl variable name. The Tcl interpreter is queried for the associated data, which is then used as it would be for any other keyword. The variable can be either an array, in which case element 0 is for the first atom, 1 for the second, ...; or a scalar value, in which case that value is used for all the atoms. For an array example, suppose the Tcl array "foo" contained the values foo(n) = 2*n. Then the statement `$foo < 10` selects atom with indices 0 to 4, while `$foo 20 to 30 6` selects indices 10, 11, 12, 13, 14, 15, and 3. If foo has a scalar value, such as PRO, the statement `resname eq $foo` selects the proline residues. Because of a limitation in Tcl, the only way VMD recognizes if a variable is an array is if element 0 exists.

7.9.2 @references

The other way to reference a Tcl variable is as a boolean array. This is most often used when there is a set of atoms which must be referenced often or when the selection is very complicated. These start with a @ sign and specify if the given atom should be on or off. This notation is used in the same place as a boolean keyword (like `protein` or `water`). As with the $ variables, the Tcl variable can be either a scalar or an array. If the value is false, (or 0) the corresponding atom is not selected. If true (or non-zero), it is turned on. Additionally, if the scalar value is an atom selection, the data from that selection is used. Here are some examples of @variables:

if the array foo(n) is defined as 1 of n is odd, 0 if n is even, then the selection text `@foo` turns on atoms with an odd numbered atom index.

Let "my_selection" be defined as set my_selection [atomselect top resid 35] then the selections "$my_selection and name CA" and "sidechain $my_selection" are also valid.

7.9.3 The differences

The difference between the two notations ($ and @) is somewhat subtle and occurs because the original selection language is not as dynamic as one would expect. The key thing to remember about Tcl references is they are used only where keywords are allowed. $variables are for keywords that have a value while @variables are for boolean keywords. That means variable references cannot be in the parameter list for the value keywords (e.g., `resname $name` is not allowed). Instead, the comparison operators must be used (as in `resname eq $name`).

Keyword	Arg	Description
all	*bool*	everything
none	*bool*	nothing
name	*str*	atom name
type	*str*	atom type
index	*num*	the atom number, starting at 0
chain	`str`	the one-character chain identifier
residue	*num*	a set of connected atoms with the same residue number
protein	*bool*	a residue with atoms named `C, N, CA,` and `O`
nucleic	*bool*	a residue with atoms named `P, O1P, O2P` and either `O3', C3', C4', C5', O5'` or `O3*, C3*, C4*, C5*, O5*`. This definition assumes that the base is phosphorylated, an assumption which will be corrected in the future.
backbone	*bool*	the `C`, `N`, `CA`, and `O` atoms of a protein and the equivalent atoms in a nucleic acid.
sidechain	*bool*	non-backbone atoms and bonds
water, waters	*bool*	all atoms with the resname `H2O, HHO, OHH, HOH, OH2, SOL, WAT, TIP, TIP2, TIP3` or `TIP4`
fragment	*num*	a set of connected residues
pfrag	*num*	a set of connected protein residues
nfrag	*num*	a set of connected nucleic residues
sequence	*str*	a sequence given by one letter names
numbonds	*num*	number of bonds
resname	*str*	residue name
resid	*num*	residue id
segname	*str*	segment name
x, y, z	*num*	x, y, or z coordinates
radius	*num*	atomic radius
mass	*num*	atomic mass
charge	*num*	atomic charge
beta	*num*	temperature factor
occupancy	*num*	occupancy
at	*bool*	residues named `ADA A THY T`
acidic	*bool*	residues named `ASP GLU`
acyclic	*bool*	"protein and not cyclic"
aliphatic	*bool*	residues named `ALA GLY ILE LEU VAL`
alpha	*bool*	atom's residue is an alpha helix
amino	*bool*	a residue with atoms named `C, N, CA`, and `O`
aromatic	*bool*	residues named `HIS PHE TRP TYR`
basic	*bool*	residues named `ARG HIS LYS`
bonded	*bool*	atoms for which numbonds¿0
buried	*bool*	residues named `ALA LEU VAL ILE PHE CYS MET TRP`
cg	*bool*	residues named `CYT C GUA G`
charged	*bool*	"basic or acidic"
cyclic	*bool*	residues named `HIS PHE PRO TRP TYR`

Table 7.1: Atom selection keywords.

Keyword	Arg	Description
hetero	*bool*	"not (protein or nucleic)"
hydrogen	*bool*	name "[0-9]?H.*"
large	*bool*	"protein and not (small or medium)"
medium	*bool*	residues named `VAL THR ASP ASN PRO CYS ASX PCA HYP`
neutral	*bool*	residues named `VAL PHE GLN TYR HIS CYS MET TRP ASX GLX PCA HYP`
polar	*bool*	"protein and not hydrophobic"
purine	*bool*	residues named `ADE A GUA G`
pyrimidine	*bool*	residues named `CYT C THY T URI U`
small	*bool*	residues named `ALA GLY SER`
surface	*bool*	"protein and not buried"
helix	*bool*	atom's residue is an alpha helix
helix_3_10	*bool*	atom's residue is an alpha helix
extended_beta	*bool*	atom's residue is a beta sheet
bridge_beta	*bool*	atom's residue is a beta sheet
annmm	*annmm string*	translates an annmm selection syntax to VMD
rasmol	*Rasmol string*	translates Rasmol selection syntax to VMD
alpha_helix	*bool*	atom's residue is an alpha helix
pi_helix	*bool*	atom's residue is a pi helix
helix	*bool*	atom's residue is an alpha or pi helix
sheet	*bool*	atom's residue is a beta sheet ???
turn	*bool*	atom's residue is in a turn conformation
coil	*bool*	atom's residue is in a coil conformation
structure	*str*	single letter name for the secondary structure
within	*str*	selects all atoms within a specified distance of a selection (i.e `within 5 of name FE`).
same	*str*	selects all atoms which have the same keyword as the atoms in a given selection (i.e. `same segname as resid 35`)
ux, uy, uz	*num*	force to apply in the x, y, or z coordinates

Table 7.2: Atom selection keywords (continued).

Function	Description
sqr(x)	square of x
sqrt(x)	square root of x
abs(x)	absolute value of x
floor(x)	largest integer not greater than x
ceil(x)	smallest integer not greater than x
rint(x)	the integer nearest to x
sin(x)	sine of x
cos(x)	cosine of x
tan(x)	tangent of x
atan(x)	arctangent of x
asin(x)	arcsin of x
acos(x)	arccos of x
sinh(x)	hyperbolic sine of x
cosh(x)	hyperbolic cosine of x
tanh(x)	hyperbolic tangent of x
exp(x)	"e to the power x"
log(x)	natural log of x
log10(x)	log base 10 of x
erf(x)	error function of x
erfc(x)	compliment of the error function of x

Table 7.3: Atom selection functions.

<table>
<tr><th>Symbol</th><th>Example</th><th>Definition</th></tr>
<tr><td>.</td><td>. , A.C</td><td>match any character</td></tr>
<tr><td>[]</td><td>[ABCabc] , [A-Ca-c]</td><td>match any char in the list</td></tr>
<tr><td>[~]</td><td>[~Z] , [~XYZ] , [^x-z]</td><td>match all except the chars in the list</td></tr>
<tr><td>^</td><td>^C , ^A.*</td><td>next token must be the first part of string</td></tr>
<tr><td>$</td><td>[CO]G$</td><td>prev token must be the last part of string</td></tr>
<tr><td>*</td><td>C* , [ab]*</td><td>match 0 or more copies of prev char or regular expression token</td></tr>
<tr><td>+</td><td>C+ , [ab]+</td><td>match 1 or more copies of the prev token</td></tr>
<tr><td>\|</td><td>C\|O</td><td>match either the 1st token or the 2nd</td></tr>
<tr><td>\(\)</td><td>\(CA\)+</td><td>combines multiple tokens into one</td></tr>
</table>

Table 7.4: Regular expression methods.

X-PLOR Wildcard	Description	Regular Expression
*	matches any string	.*
%	matches a single character	.
+	matches any digit	[0-9]
#	matches any number	[0-9]+

Table 7.5: Regular expression conversions.

Chapter 8

Creating Output Raster Images

VMD currently has no provision for making a raster image file (like a GIF, TIFF, or RGB) directly, other than using an external program to do a screen capture. In many cases it is sufficient to use the SGI program `snapshot` to get the image from the screen. This saves the image to an RGB file which can then be converted to other formats with the programs `/usr/sbin/to*` (e.g., `togif` converts RGB files to GIF). Since RGB is the default SGI image format, many other programs can be used to view, edit, and manipulate RGB files.

To use `snapshot` in this sense, simply open the Render form [§3.3.10[and choose the `snapshot` option. VMD will automatically pass the window coordinates to this program and the capture will take place and the resulting RGB file will be posted to the display. After a little practice with this option, you will recognize that it is important not to have other windows in front of the VMD display when doing this type of capture, since `snapshot` will then include part of these windows in the output RGB picture.

The `snapshot` rendering option is presently not available to those using the HP or Linux version of VMD

Sometimes the screen image isn't good enough; you may want a very large, high quality picture, or maybe a picture with shadows. Instead of making these images directly, VMD writes files which can be used as input to several of the more popular ray shading programs. Table 8.1 lists the currently supported output formats, and, if applicable, where they may be obtained.

8.1 Creating an Output Image File

Making the raster image is a two step process. First you must make the output file suitable for the image processing program, and then execute the program using the new file as input to produce the raster output. The problem is that each of the programs write a different output file format, which will probably have to be converted to something more appropriate for you. It is impossible to predict what that might be, so we'll say how to convert the different file types to RGB and let you use the tools listed in Table 8.1 to get what you need. Raster3D produces an RGB file, so you don't need to do anything. POV and POV3 produce TARGA files, which can be converted on SGI machines with the program `/usr/sbin/fromtarga`. Rayshade creates RLE image files, which can be converted on SGI machines with `/usr/sbin/fromutah`. Radiance generates an .oct file, which can be converted with the `rview` and `rpict` commands in the Radiance distribution.

The free program `display` from ImageMagick – see
`http://www.wizards.dupont.com/cristy/ImageMagick.html` –

Name	Description	Default Render Command
Raster3D[1]	Fast raster file generator	`render < %s -sgi %s.rgb; ipaste %s.rgb`
tkRaster3D[2]	Tk interface to Raster3D	`render < %s -sgi %s.rgb; ipaste %s.rgb`
POV3[3]	POV-Ray 3.0 ray tracer	`povray3 +H500 +W400 -I%s -O%s.tga +D +X +A +FT +C`
POV[3]	POV-Ray ray tracer	`povray +H250 +W200 -I%s -O%s.tga +P +X +A +FT`
Rayshade[4]	Rayshade ray tracer	`rayshade < %s > %s.rle`
Radiance[5]	Radiosity ray tracer	`oconv %s > %s.oct; rview -pe 100 -vp -3.5 0 0 -vd 1 0 0 %s.oct`
ART[6]	VORT ray tracer	`render < %s -sgi %s.rgb; ipaste %s.rgb`
PostScript	Direct PostScript Output	`ghostview %s &`
STL	Stereolithography Format	`true` - Renders Triangles Only
VRML-1	Virtual Reality Markup Language	`true` - view this with VRML viewers like webspace
Token	Text Dump of Graphical Objects	`true`

[1]See `http://www-bio.unizh.ch/visualization/raster3d/raster3d.html` for more info.
[2]Contact ???
[3]See `http://www.povray.org/` for more info.
[4]See `http://www-graphics.stanford.edu/ cek/rayshade/rayshade.html` for more info.
[5]See `http://radsite.lbl.gov/radiance/HOME.html` for Radiance
[6]Available from ftp://gondwana.ecr.mu.oz.au/pub along with the rest of VORT package
[7]See `http://vrml.sgi.com/intro.html`

Table 8.1: Supported output rendering formats.

should be able to read and interconvert all of these formats.

We suggest using either Raster3D or Radiance as they are the fastest of the programs. The first is the easiest to understand and the second has many, many options (different types of lights, textures, ...).

The image processing program input formats are straight text so they are easy to modify. This is most often done to create a larger raster file, though some have other global options which you may wish to change. For instance, by default the Raster3D file turns shadows on. We suggest you consult the relevant documentation to determine what can be modified in the file.

To actually render the current image into an output file, first set up the graphics in VMD just as you wish the output to appear. Then, either use the Render form [§ 3.3.10], or the following text command, to create the input file and start the rendering program going:

- **render** ***method filename*** [***render command***]

method is one of the names listed in the first column of table 8.1, and *filename* is the name of the file which will contain the resulting image processing program script. Any text following this will be used as a command to be run to process the file. If `%s` appear in the command string, they will be replaced with the name of the script file.

8.2 Known Problems

When VMD creates the output file it will try to match the current view and screen size. For the most part it does a good a job but there are some problems. The colors in the final raster image seem to be faded when compared to the screen. We may be able to correct this in future versions by artificially increasing the color intensity. In the meantime, this can be remedied with a program like `xv` or `display` which will brighten and sharpen the image.

The eye position seems to be slightly different between the screen and raster images. This occurs because the Screen Height and Screen Width (see §3.3.8) values are not propagated to the output file, so the programs have slightly different transformation matrices. We have added an extra scaling factor, determined by trial and error, which reduces this effect for the default screen sizes, but the factor is only valid if the parameters are not changed.

The lights are supposed to be positioned in the output file as they are on the screen but not all formats support the number of lights available in VMD. In addition, there seems to be a bug in the code that determines the correct light positions. This can be changed by editing the output file by hand.

A new option in VMD draws cylinders with flat disks covering the ends. Some of the renderers may not draw the capped ends.

We have primarily tested the Raster3D output and have little experience with the other programs. The only problem we have found with Raster3D is if there are too many objects to render (this could easily occur when using 'Tube' or 'Ribbon' on large molecules), it will not render anything.

8.3 One Step Printing

A frequently asked question is "How can I quickly get a printout of the VMD Display?" There are several one step solutions to this problem

- Choose the PostScript output option in the Render form and type `lpr %s` in the render command box.
- Choose the `snapshot` option and type `convert %s eps:%s.ps; lpr %s.ps` in the render command box. This assumes that you have the ImageMagick tools available in your PATH setting.
- As an example of how to directly print a Raster3D file, choose the Raster3D option in the Render form and type `render < %s -sgi %s.rgb; convert %s.rgb eps:%s.ps; lpr %s.ps` in the render command box.

Chapter 9

Viewing Modes

There are many different viewing modes available. These show the scene in orthographic or perspective views, and in several mono- and stereo- graphic displays. The stereo mode can be changed using the popup menu available from the graphics display window. It can also be changed by using the `stereo` entry in the Display form or the text command `display stereo` *`mode`*.

9.1 Perspective/Orthographic views

In the perspective view (the default), objects which are far away are smaller than those nearby. In the orthographic view, all objects appear at the same scale. Since some prefer one over the other, both options are available. Perspective viewpoints give more information about depth and are often easier to view because you use perspective views in real life. Orthographic viewpoints make it much easier to compare two parts of the molecule, as there is no question about how the viewpoint may affect the perception of distance.

9.2 Monoscopic Modes

When you normally look at objects, your two eyes see slightly different images (because they are located at different viewpoints). Your brain puts the images together to generate a stereoscopic viewpoint. When generating a single image for the computer display, the default calculations (mode Stereo Off) assume there is one eye centered between where two eyes would be. Sometimes, as when generating ray tracing input files, the left and right eye views need to be generated independently. Choosing mode Left produces the left eye viewpoint, while Right produces the right eye viewpoint.

9.3 Stereoscopic Modes

Molecules may be rendered in stereo, which can greatly enhance the appearance and visual content of the displayed systems. There are several stereo formats available:

1. Side-by-side cross-eyed stereo;
2. Side-by-side wall-eyed stereo;
3. Crystal Eyes stereo (requires stereo-capable monitor, stereo sync emitters and special stereo glasses equipped with liquid crystal lenses).

9.3.1 Side-By-Side and Cross-Eyed Stereo

Side-by-side stereo means that the normal display is divided into two halves, a left view and a right view, each occupying one-half of the original display area. Each view displays the current molecules from a slightly different perspective, corresponding to the left and right eye of the viewer. The images are separated, however, so to actually see a 3D object you must direct your eyes until the two images are on top of each other, and then focus on the resulting image until you can see it as three-dimensional.

There are two ways of placing the images. In wall-eyed stereo, the left eye's image is located on the left side of the display, and the right eye's image is on the right. This is the standard method for displaying stereo images in publications as it works well when the display (in this case, the piece of paper) is close to the eyes. It is called wall-eyed because your eyes are directed the same way they would be if looking at a distant wall. In VMD, this method is referred to as "SideBySide" stereo.

In cross-eyed stereo, the left eye's image is located on the right side of the display, and the right eye's image is on the left, and hence the name cross-eyed. This is mostly used for distant displays (such as overhead projections) as it is much easier to cross eyes at that range than use the wall-eyed method – you are already looking at the wall. In VMD, this method is referred to as "CrossEyes" stereo.

9.3.2 Crystal Eyes Stereo

Crystal Eyes stereo refers to a special hardware option available on Silicon Graphics workstations that allows one to view three-dimensional objects through the use of a special display monitor mode and special stereo viewing glasses. Stereographics Corp. is one such supplier of these glasses. This mode also requires special infrared emitters to synchronize the stereo glasses with the display monitor.

There are presently two types of Crystal Eyes stereo modes available on SGI machines; which one is used depends on the available graphics hardware (see the man page `stereo(7)`). Follow the outline below for the relevant graphics option:

Stereo in a window (aka new-style stereo): Available on workstations equipped with Reality Engine2-style graphics subsystems only (such as Onyx or Power Onyx machines), this mode provides separate left and right eye frame buffers. It allows the user to have a window display in stereo, and the other windows appear as normal. Using this mode, however, requires the monitor to be in a lower-resolution 960 by 680 display mode. The monitor must be set in this mode before starting VMD. To do so, do one of the following:

- If superuser access is not available, and the monitor is in the normal 1280x1024 mode, execute the command:

  ```
  /usr/gfx/setmon -n 960x680_108s
  ```

  This will change the display characteristics to the lower resolution mode, with a higher display frequency (108 Hz), but will not change the managed size of the X-Window display screen. To change back to the normal mode (after running VMD), execute the command:

  ```
  /usr/gfx/setmon -n 1280x1024_60
  ```

When the computer is in the lower resolution mode, run VMD as normal, but note that the lower resolution means the windows will appear larger and may end up sometimes positioned off-screen. If this is a problem, a system superuser should set the video mode permanently with the commands listed in the next item.

- If superuser access is available, do the following, which will change the video mode and restart the window server with the lower resolution settings. (WARNING: executing this command will log out anyone on the console)

  ```
  /usr/gfx/setmon -x 960x680_108s; /usr/gfx/stopgfx; /usr/gfx/startgfx &
  ```

  To reset the computer to regular 1280x1024 use, execute the command:

  ```
  /usr/gfx/setmon -x 1280x1024_60; /usr/gfx/stopgfx; /usr/gfx/startgfx &
  ```

  It may be useful, if stereo-in-a-window will be used often, to set the monitor to the lower resolution mode, and leave it that way.

Once set in the proper display mode, start VMD as normal, and select 'Crystal Eyes stereo' from the display window pop-up menu. The image should switch to two images nearly superimposed, but slightly offset.

Regular stereo mode: All stereo-equipped SGI workstations can use this display mode, including Reality Engine 2 workstations. VMD uses two programs `onstereo` and `offstereo` to switch the monitor display mode. You will need to install these programs somewhere in your Unix path in order to allow VMD to use them. Regular stereo mode will change the display window to occupy the entire monitor screen, as opposed to stereo-in-a-window which can independently set the stereo mode of just the display window. Versions of `onstereo` and `offstereo` which will run on SGI IRIX 5.X workstations may be obtained via anonymous ftp from `ftp://ftp.ks.uiuc.edu/pub/mdscope/vmd`.

9.3.3 Problems with stereo on Indigo2 machines

SGI machines running versions 5.X or early versions of 6.X of IRIX may require patches for their OS in order to run crystal eyes stereo.

9.3.4 Stereo Parameters

A stereo image is generated by drawing two images from two different perspectives, one from the left eye and one from the right. The images are made by finding the view that would be seen by someone located inside the scene. The method uses two parameters to find the view; the *eye separation* and the *focal length*. The first defines the distance between the eyes and gives the parallax effect. Setting the separation to 0 will result in a flat 2D image, while setting it too large will give most people a headache.

The graphics model used by VMD assumes the eyes looking in front of the viewer and focusing at the same point the focal length away. If the focal length is 0, the viewer's eyes are crossed and looking at each other. A larger focal length will often help in creating a viewable image.

The two parameters can be changed with the text commands `display focallength` and `display eyesep`, or using the Display form [§3.3.8].

In general, try to make the eye separation as large as possible without giving the viewer a migrane, and try to vary the focal length to cut down on double images. It may often help to translate the molecule forward or backward and also adjust the scaling, since there is typically an optimum position for a molecule for a given set of stereo parameters.

9.4 Making Stereo Raster Images

As discussed in Chapter §8, VMD can create output files of the current scene in the format needed for input by various image processing packages. These formats do not always natively support the ability to draw stereo images. In principle, it is possible to write the scene to the file twice with the appropriate transformations applied to make the view correct for each eye, but then the shadows would be incorrect.

Instead, we suggest making one image of the current scene, then shift the molecules to the left (or right) to make the other image. (Note that neither the stage nor the axes will move, so they will not be in stereo.) The text commands for this are something like:

```
render Raster3D left.r3d
trans by -.1 0 0
render Raster3D right.r3d
```

The two files must then be rendered to produce the rgb file. As it turns out, this method makes it easy to produce stereo images of ordinary Raster3D files. Since VMD can read the Raster3D format, all you have to do is read the file and then execute the commands listed above. The text commands for generating left or right views also have equivalents in the GUI under the `Stereo` option of the Display form.

Chapter 10

Interactive Molecular Dynamics Display Using MDComm

VMD has the capability to work with a molecular dynamics program running on another computer, in order to display the results of a simulation as they are calculated. As new atomic coordinates are generated by the simulation process, they can be transferred directly over the network to VMD, which can then animate the molecule. A major new feature in VMD is the ability to add perturbative steering forces to a running simulation, which are incorporated directly into the dynamics calculation.

VMD uses a set of daemons and library routines to provide the communication and coordination functions necessary to allow connection to a remote application program. This set of daemons and library routines is known as MDComm, written by Rick Kufrin of the National Center for Supercomputing Applications (NCSA)[1]. A version of the MDComm library and daemons are provided in the standard VMD distribution for use on SGI workstations. The MDComm source code and documentation are also available from the MDComm anonymous ftp directory[2]. See also the MDComm web page[3].

In order for VMD to work in this fashion as a graphical front end and control console for a remote molecular dynamics simulation, several things are necessary:

- A molecular dynamics program capable of communicating with VMD must be installed properly on another computer (although it can also be installed on the same computer used to run VMD, if desired). Right now, there is one MD program supported by the Theoretical Biophysics group for this purpose, called NAMD. See the NAMD WWW home page[4] for information on obtaining a copy of NAMD. NAMD is a parallel molecular dynamics program written in C++, which implements the CHARMM energy function. It is compatible with X-PLOR style PSF, PDB, and parameter files. The rest of the discussion in this chapter assumes you are using NAMD.
- A master control daemon (the *rappd* daemon) must be running on the computer which will be used to run NAMD. This daemon stores information about what applications are available

[1]MDComm is a particular usage of the more general remote application library *RAPP*, also developed by Rick Kufrin

[2]ftp://ftp.ks.uiuc.edu/pub/mdscope/mdcomm/

[3]http://www.ks.uiuc.edu/Research/mdcomm/

[4]http://www.ks.uiuc.edu/Research/namd/

on that computer to be started, and also keeps a list of what jobs are currently running on that computer. VMD communicates with this daemon at the start to find out how to start up a new job, and then when all parameters have been set up this daemon will actually launch a new application-specific daemon which will start NAMD. *rappd* will also provide to both VMD and NAMD the information required for them to start talking to each other. Basically, it is the manager for the whole process.

- A NAMD-specific communication daemon (*namdd* or *simple_namdd*) must be available and installed on the same computer as will run NAMD. When a new MD job is started by *rappd*, both a NAMD and a *namdd* process are started up. *namdd* is responsible for buffering the dynamic data transferred between NAMD and VMD, to make things faster and to help prevent deadlocks.

- A VMD-specific communication daemon (*namd_consumer*) must be available and installed on the same computer which is running VMD. When a new MD job is started by *rappd*, at the same time a new *namd_consumer* process will be started. Just as *namdd* is used, this daemon buffers dynamic data and helps prevent deadlocks.

For a full description of how these processes interact, we again direct you to the MDComm home page.

Once all these components are in place, you can use the graphical user interface tools in VMD to start up new MD jobs on a remote computer, view the results as they are calculated, and control the simulation parameters via VMD. You can choose to then kill the remote job, or else you can detach from simulation and leave it running. If you detach from a job, you can later reattach to that same job and resume visualizing the data at the current point in the simulation. See sections on Remote form [§3.3.12], Sim form [§3.3.13], `remote` command §11.3.20], and MDScope home page at MDScope WWW home page for more information.

10.1 How the Connection Works

Figure 10.1 shows schematically how the components used by MDComm, NAMD, and VMD work together. There are two computers involved: the Graphics workstation (where VMD runs), and the Compute workstation (where NAMD runs). They may be completely separate computers, but it is certainly possible for both of these machines to be the same (particularly if you are working on a multiple-processor computer such as an SGI Onyx). To begin, just VMD and *rappd* need to be running. When you display the 'Remote Connection Setup' menu in VMD, and enter a username and hostname and ask to connect to that computer, VMD uses MDComm routines to contact the *rappd* process running on the requested computer, and if it is present VMD gets back two lists of data:

1. The names of all MD programs which *rappd* knows how to start.

2. The ID's of all MD simulation jobs which have been previously started on that computer by *rappd*, and which are still running.

After contacting *rappd* and getting these lists, VMD displays them and lets you either resume a connection to an already-running job (by selecting an item from the second list), or lets you set up and start a new simulation (by selecting an MD application from the first list).

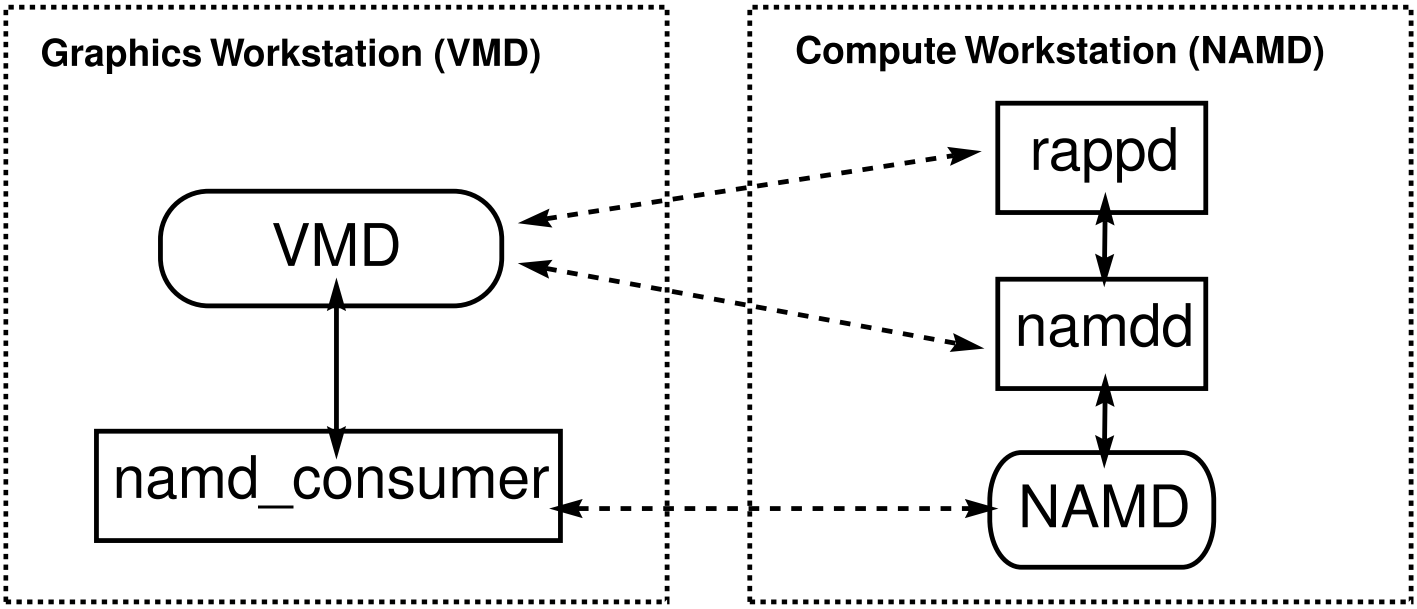


Figure 10.1: Schematic view of remote simulation connections.

To start up a new simulation, you choose the name of the MD program, and press the button to get the parameters. VMD then requests *rappd* to send over the list of parameters required to start up the simulation, such as the name of the molecular coordinate file, the temperature, the timestep, etc. When these have all been entered, you press the Start Simulation button; VMD returns the completed parameter list to *rappd*, which starts the following events in motion:

1. *rappd* verifies that all the parameters are correct, and then starts up the *namdd* daemon. *namdd* is given the parameter set, and knows how to start NAMD running. *namdd* is used also to buffer data moving between NAMD and VMD.

2. *namdd* starts up NAMD.

3. On the Graphics end, the daemon *namd_consumer* is started, which does for VMD what *namdd* does for NAMD — it buffers the data flowing between the applications. Note that VMD can maintain multiple connections, and can display and control several simulations at the same time. A separate *namd_consumer/namdd* pair is started for each connection.

4. After NAMD starts up and reads in its coordinates, it starts sending the coordinates back via *namdd* to VMD.

5. *rappd* keeps note of the fact that this instance of NAMD is running, but does little else. Later, if the connection between VMD and NAMD is dropped, *rappd* knows how to re-establish this link.

10.2 Installing VMD Remote Connection Components

To get all this to work, certain files must be installed in the proper location, and some data files must be edited with site-specific data for your system. This section describes how to get VMD set up to work with NAMD over the network. The MDComm files and libraries mentioned in this section can be found in the `lib/mdcomm` directory of the VMD distribution.

First, VMD must have been compiled with the REMOTE option specified to the configure script, as described in the Installation Guide. If you are using the pre-compiled version of VMD for SGI workstations supplied in the distribution, the REMOTE option has been included, and you are OK. If not, see the Installation Guide about how to compile with the REMOTE option. At present, this has only been tested for IRIX version 5.X or HP-UX version 9.X workstations, though the complete code is available for porting to other platforms.

Second, the *namd_consumer* program must be installed in `VMDDIR/mdcomm/bin_ARCH`, where `VMDDIR` is the directory where VMD is installed, such as `/usr/local/lib/vmd` and `ARCH` is the type of OS (such as IRIX5 or HPUX9).

Third, VMD must be told where to find the *namd_consumer* executable. The environment variable `MDC_CONSUMER` must be set to the name of the *namd_consumer* program, and include the full path. The shell script provided in the VMD distribution which is used to start VMD should do this automatically; however, if things are not working, try setting it by hand before starting VMD. Also, if you install *namd_consumer* in some place other than as described in the previous paragraph, you need to change the `vmd` startup script to set `MDC_CONSUMER` to this different location.

Once this has been set up, you can run VMD and use it to connect to another computer which has *rappd* running and which can run NAMD. But before this will work, NAMD and *rappd* must be set up properly, using the instructions in the following section.

10.3 Installing NAMD Remote Connection Components

The MDComm files mentioned in this section should be available in both the VMD and NAMD distributions. In the VMD distribution, they are in `lib/mdcomm`; in the NAMD distribution, they are in the `mdcomm` directory.

The important thing that must be done to configure another computer to run NAMD and to allow it to connect to VMD is to start *rappd* on this computer, and set up a configuration file for *rappd* which describes where all the various programs and daemons are located. This discussion assumes NAMD has already been installed in some location, i.e. `/usr/local/bin/namd` or really anywhere.

You should create a globally-executable directory to install the MDComm daemons and *rappd* data files; for example, `/usr/local/lib/mdcomm`. Then, copy the programs *rappd* and *namdd* from the NAMD distribution directory `mdcomm/bin_ARCH` (*ARCH*=HPUX9 if using HP-UX workstations, IRIX5 for SGI, etc) to `/usr/local/lib/mdcomm/bin` (or wherever the MDComm files are being installed). Finally, copy the entire directory `mdcomm/rappd` to `/usr/local/lib/mdcomm`, so that there is the a directory `/usr/local/lib/mdcomm/rappd`. This directory contains files with `.conf` extensions which are used to configure *rappd*. There should be at least one file there, `namd.conf`.

Once `/usr/local/lib/mdcomm` is created and the files are copied over, the last step is to edit the files in `/usr/local/lib/mdcomm/rappd` for your personal setup. There is a directory for each ARCH available; go to the one for the ARCH on which *rappd* and NAMD will run, and edit the file `namd.conf`. This file tells *rappd* where to find the NAMD executable, the *namdd* daemon, and what all the parameters are for NAMD. You will need to edit the following lines:

- `:daemon-prog "/directory/namdd"`
  Change the file in quotes to the full path to the location of *namdd*; in the example given here, this would be `/usr/local/lib/mdcomm/bin/namdd`.

- `("EXE" "STR" "Full pathname of the molecular dynamics executable" "namd"`
  Change the final item in quotes to the full path to the location of the NAMD executable; in the example given here, this would be `/usr/local/bin/namd`.

Finally, once the configuration file is set up, *rappd* must be started. You specify to *rappd* a directory which will be searched for files ending in a `.conf` extension, by using a command-line argument of the form `-d <directory>`. Each file of this form in the specified directory should contain the configuration data for a different application that *rappd* can possible start. In this way, there can be several different MD programs installed on a remote computer, and VMD can provide menu options for the user to choose which program to start. (Note, that version 1.1 of rappd reads *all* files in the specified directory, and not only those which end in `.conf`.) A useful script to use to start up *rappd* is the following (assuming ARCH=HPUX9, and *rappd* files are in `/usr/local/lib/mdcomm`; if not, change this to the proper path and ARCH):

```
#!/bin/sh
#
# brief script to start up rappd, killing the old rappd if necessary

# start up new rappd
if [ -x /usr/local/lib/mdcomm/bin/rappd -a -d /usr/local/lib/mdcomm/rappd ]
then
  cd /usr/local/lib/mdcomm/bin
  ./rappd -d ../rappd &
else
  echo "Cannot start new rappd."
fi

# end of script
```

Once *rappd* is running, you should be able to use VMD to connect to it and get a list with `namd` as an available MD program.

10.4 When Problems Occur

There are many, many things which can go wrong in this procedure, so if you have problems please send a note to `vmd@ks.uiuc.edu` describing exactly how things are set up (i.e. where VMD and NAMD and MDComm files are installed, what script is being used to start VMD, and what are the symptoms of the problem). Please note that a new version of this is being worked out and should be available very soon. Here are a couple quick notes about things that can go wrong:

- **You connect to NAMD, NAMD starts up, but nothing ever gets displayed and VMD just sits there.**
  Quite often this is because *namd_consumer* could not be started on the graphics workstation. Typical reasons for this are:
  - *namd_consumer_ARCH* was never installed.
  - `MDC_CONSUMER` was not set to the location of *namd_consumer_ARCH*.

- the *namd* stdout and stderr information is saved in the files `~/rapp.stdout.$PID` and `~/rapp.stderr.$PID` — you will have to delete these yourself when the simulation(s) finish. These files may also help to identify the problem since the occurring error message appears in one of them.

- **You make a connection to *rappd*, but the lists of available programs and jobs are both empty.**
  The configuration files were not set up properly, and when (or if) *rappd* was started it could not figure out where any of the applications are located.

Chapter 11

Text User Interface

The text interface provides complete access to all the VMD commands. In its basic form it can be used to load molecules, rotate them, add and alter representations, and anything else that can be done with the Forms and mouse interface. The standard distribution is compiled with Tcl and the TclX and Tcl-DP extensions, which add a complete scripting language including variables, loops, and conditionals along with a standard method for communicating with other programs via standard TCP/IP sockets. Versions 1.2 and later also include the Tk toolkit and the TkX extension. This allows, for example, to quickly create a form with buttons bound to one's favorite actions (similar to popup menus, but much more flexible).

This section describes the basics of the text interface as well as the core VMD commands. A few of the Tcl commands are mentioned here, but are so noted.

11.1 Using text commands

Text commands are entered by typing them at the VMD prompt in the text console window. This window normally contains the prompt `vmd >` . When other text (e.g., from a mouse pick) is displayed to the screen, it will scroll the screen up so the prompt is not at the last line of the screen. To make it reappear, press enter. When entering multi-line commands, an alternate prompt appears, `?` , and will not disappear until the command is finished. Sometimes it is waiting for a close to a double quote, open brace, or open bracket, while at other times it is waiting for a line that doesn't end in a backslash. Please read a Tcl manual to better understand what constitutes the end of a statement.

Since you may not want to retype all the data in every time, there are two ways to read the data in from a text file. The preferred method is the `play` VMD core command. This reads a line from the file, executes it, then updates the screen and checks for any changes in the mouse or forms input. Using this command you can modify the display options while the script is being read. The other option uses the Tcl command `source`. This reads the whole file before allowing the mouse and forms to respond to new input.

There are two other ways to `play` a file. If the file `.vmdrc` (see section §17.3.4) exists during startup, it is played. Similarly, at startup the `-e` command line flag can be used to specify an input file.

11.2 Tcl/Tk

Tcl (short for Tool Command Language, developed by John Ousterhout) is an embeddable and extensible scripting language. In other words, Tcl sits inside VMD as a language interpreter where it can execute its standard language commands or the various VMD specific extensions. There are several reasons for using Tcl rather than writing our own language, the most important being that it is easy to use, it was easy to modify our code to use it, it has few bugs, and documentation is available at many bookstores. Many other packages use it, including Quanta. It is not necessary that you know Tcl to use VMD. However, it is useful for some occasions, like making movies or scripts.

VMD uses Tcl version 7.6 and Tk version 4.2. Since Tcl is extensible, many extension packages have been written to improves current features and add new ones. We have included two of these, TclX (version 7.6), which adds many useful command for script writing, and Tcl-DP (version 4.0), which adds an interface to Unix socket commands. This latter extension is used by the External interfaces [§3.4.1]. The TkX version 4.2.0 is also included. We refer you to `http://www.scriptics.com/` for more information about Tcl, and `http://www.sco.com/Technology/tcl/Tcl.html` for references to documentation on Tcl/Tk and TclX. For the documentation on Tcl-DP see the Tcl-DP home page at `http://www.cs.cornell.edu/Info/Projects/zeno/Projects/Tcl-DP.html`.

11.3 Core Text Commands

Most of VMD was originally written without the Tcl interface. The text interpreter used (which is still available as a compile time option) treated the commands in a slightly different form than Tcl does. Because we did not rewrite the old commands when Tcl was added, there are slight differences in usage between the old style interface (the "core" commands) and the new Tcl functions. For the most part, these are subtle and not important. For more information see the section on "writing scripts."

All text commands in VMD are composed of one or more words or phrases separated by white space, and terminated by a newline. Since the parser now uses Tcl, a "phrase" is text surrounded by double quotes or by a matching set of open and close braces. (Please read the Tcl manual to better understand what constitutes the end of a statement.) The first word of each command indicates the general purpose for the command, and the following words specify the exact type of command to execute. Table 11.1 summarizes the text commands in VMD by listing the first words, and describing the general purpose for commands starting with those words. Since VMD can be compiled with optionally included components and features, commands labeled *optional* may not be available in your version of VMD.

The commands described in the following sections are listed by name, and followed by a list of the available arguments. If an argument is optional, it is enclosed in `[]`s. If only one of a list of arguments is needed, the list is enclosed in `<>`s and the items are separated by `|`. Words in italics indicate a string or value to be specified by the user.

11.3.1 animate

These commands control the animation of a molecular trajectory and are used to read and write animation frames to/from a file or Play/Pause/Rewind a molecular trajectory.

- **dup** [**frame** *number*] *molId*: Duplicate the given frame (default "now") of molecule *molId* and add the new frame to this molecule.
- **forward**: Play animation forward.
- **for**: Same as forward.
- **reverse**: Play animation backward.
- **rev**: Same as reverse.
- **pause**: Pause animation.
- **prev**: Go to previous frame.
- **next**: Go to next frame.
- **skip** *n*: Set stride to *n*+1 frames.
- **delete all**: Delete all frames from memory.
- **speed** *n*: Set animation speed to *n*.
- **style once**: Set to play animation once.
- **style loop**: Set to loop through animation continuously.
- **style rock**: Set to play animation forward and back continuously.
- **styles**: Return a list of the available styles.
- **goto start**: Go to first frame.
- **goto end**: Go to last frame.
- **goto** *n*: Go to frame *n*.
- **read** *file type filename* [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [*molecule_number*]: Read data for *molecule_number* from *filename*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*+1.
- **write** *file type filename* [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [*molecule_number*]: Write data from *molecule_number* to *filename*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*+1.
- **delete** [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [*molecule_number*]: Delete data for *molecule_number*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*+1.
- **readdel** *filename* [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [*molecule_number*]: Read data for *molecule_number* from *filename*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*+1, overwriting existing data.

11.3.2 axes

The axes (orthogonal vectors pointing along the x, y, and z directions) can be placed in any of 5 locations on the screen, or turned off.

- **locations**: Return a list of possible locations.
- **location**: Get the current location.
- **location < off | origin | lowerleft | lowerright | upperleft | upperright >**: Position axes.

Also, though this may seem like a likely command for changing the color of the axes, this function can only be performed from the Colors form or by the `color` command (see below). Future implementations of VMD may change this.

11.3.3 color

Change the color assigned to molecules, or edit the color scale. All color values are in the range 0 ... 1. Please see the section on coloring [§ 6] for a full description of the various options.

- *category name color*: Set color of object specified by *category* and *name* to *color*.
- **scale method <** *scale_name* **>**: Set type of scale to use for coloring objects by values. They are:
  - RGB – Red to green to blue.
  - BGR – Blue to green to red.
  - RWB – Red to white to blue.
  - BWR – Blue to white to red.
  - RWG – Red to white to green.
  - GWR – Green to white to red.
  - GWB – Green to white to blue.
  - BWG – Blue to white to green.
  - BlkW – Black to white.
  - WBlk – White to black.
- **scale midpoint** x: Set midpoint of color scale to x, in the range 0 ... 1.
- **scale min** x: Set minimum of color scale to x, in the range 0 ... 1.
- **scale max** x: Set maximum of color scale to x, in the range 0 ... 1.
- **change < alpha | shininess | rgb | ambient | diffuse | specular >** *color*: Reset given property of *color* to default value.
- **change < alpha | shininess >** *color value*: Set given (scalar) property of color to *value*.
- **change < rgb | ambient | diffuse | specular >** *color r g b*: Set given (3-valued) property of *color* to *r g b*.

See also Chapter § 13 on how to change color of a user-defined graphics object.

11.3.4 colorinfo

(Tcl) This command provides access to the color definitions. For information on the color properties see the chapter on Coloring [§6].

- **colorinfo categories**: returns a list of available categories
- **colorinfo category** *category*: returns a list of names for the given category
- **colorinfo num**: returns the number of base solid colors (17)
- **colorinfo max**: returns the total number of colors available (98)
- **colorinfo colors**: returns a list of the named solid colors
- **colorinfo index < name | colorid >**: returns the index of the given name or color id. Useful for the cases when you do need the integer value and don't want to use lsearch on `colorinfo colors`.
- **colorinfo ¡rgb | alpha | shininess | ambient | specular¿ < name | colorid >**: returns the appropriate values for the given name or color id
- **colorinfo scale < method | methods | midpoint | min | max >**: returns the information about the color scales

Examples:

```
# find out what color corresponds to which id:
set i 0
foreach color [colorinfo colors] {
  puts "$i $color"
  incr i
}


# also get a list of RGB values
set i 0
foreach color [colorinfo colors] {
  lassign [colorinfo rgb $color] r g b
  puts "$i $color   \{$r $g $b\}"
  incr i
}
```

11.3.5 debug

Turn on/off printing of debugging messages. This will have no effect if VMD was compiled without the debugging option (the standard distribution was not compiled with debugging).

- **< on | off >**: Turn debug on or off.
- **level** *n*: Set debug level to *n*.

11.3.6 display

Change various aspects of the graphical display window. For information about the options, see the section describing the Display form [§3.3.8].

- **update**: Force the display update. Makes sense if the display update is off. This does not necessarily take care of resizing the display window or using the Forms GUI while the display update is turned off.
- **update on**: Turn display update on.
- **update off**: Turn display update off. By default VMD does the display updates constantly. Sometimes it is beneficial to turn the turn the display updates off. This prevents VMD from redrawing the scene as a response to every change, thus saving time while doing changes of representations. See sections §16.6.2, § 6.4.3 and § 6.4.4 for examples of use.
- **update status**: Return the display update status (on or off).
- **update ui**: Similar to `display update`, but also forces updates of the GUI forms. The Forms interface is subject to the following behavior: if the display update is set to `off` and actions (such as, e.g., iconify/deiconify) have been performed to the Forms, the Form windows do not get updated by just `display update` command, whereas `display update ui` forces both updates to happen. Tk does not seem to have this problem, so this option will become obsolete after switching to Tk graphics user interface.
- **reshape**: Reshape the display. If the display update is turned off, and the display window has been resized command `display update` updates the display, but does not necessarily respond to the resizing of the window.
- **resetview**: Reset the view.
- **eyesep** *value*: Set the eye separation to *value*.
- **focallength** *value*: Set the focal length to *value*.
- **height** *value*: Set the screen height to *value*.
- **distance** *value*: Set the screen distance to *value*.
- **antialias < on | off >**: Turn antialiasing on or off.
- **depthcue < on | off >**: Turn depth cueing on or off.
- **depthsort < on | off >**: Turn depth sorting on or off (useful in transparent mode)
- **detail < full | flat | lines | points | none >**
- **altdetail < full | flat | lines | points | none >**
- **stereo** *mode*: Set the stereo mode to *mode*.
- **nearclip < set | add >** *value*: Add or set near clipping plane position to it value.
- **farclip < set | add >** *value*: Add or set far clipping plane position to *value*.

- **get < eyesep | focallength | height | distance | antialias | depthcue | stereo | projection | detail | altdetail | nearclip | farclip >**: Return information about the given option.
- **get < stereomodes | projections | details >**: Return a list of the availalble values for the given options. (See section § 3.3.8 and chapter § 9 for more information.)

11.3.7 echo

Turn on/off echoing of text commands to the console. When this is turned on, text commands read from a file or from the VMD prompt are echoed to the screen before they are executed. Do not confuse this with the standard Unix `echo` command.

- **< on | off >**: Turn echoing on or off.

11.3.8 exit

Quit VMD.

- **confirm**: Use a form to verify with the user before quitting.
- **[now]**: Quit without asking.

11.3.9 external

(Tcl) Communicate Tcl commands between different processes based on the Tcl-DP RPC functions (optional). This command can set the machine up as a server (listening for commands) or client (sending commands) or both.

- **on**: Start VMD as a server on port 8639 (the number can be changed by changing the variable `vmd_external_port` before starting VMD as a server). This lets **any** process connect to VMD and send it new text commands. By default, only processes from the same machine can connect to it, but that can be changed with the `host` option. See the sections on external interfaces [§3.4.1] and remote control of VMD [§15.8] for more information on how to use this feature.
- **off**: Deny new external connections to be made. Old connections will still exist.
- **host** *machinename (machinename ...)*: Allow or deny new connections based on the address of the requesting machine. The format for each machine on the list is the same as `dp_Host`, which is based on `xhost` so `external host +` allows new connections from anyone, `external host -` denies all new connections, `external host +foo` allows connections from `foo` and `external host -bar +128.186.7.*` will allow any machine from the Florida State physics department to connect.
- **connect** *machinename*: Establish connection with a VMD process on the given machine. The remote process is the *server* and the process establishing the connection is the *client*.
- **send** *command*: Evaluate the command on the server process.
- **close**: Detach from the server.

11.3.10 help

Display the on-line help file with an HTML viewer. See Chapter § 17 for information on how to change the default viewer (which is Netscape).

- `[` *subject*`]`: Jump to help corresponding to *subject*.

Presently, "subject" can be any one of the following words, which launches the associated URL. To guarantee that the help system will work correctly, you will probably want to start up your web browser before choosing one of these options. After you do this, VMD will properly direct the browser to the pages mentioned below.

11.3.11 label

Turn on or off labels for the four categories: atoms, bonds, angles, or dihedral angles. Once a label is created (given the list of associated atoms) it can be turned on or off until it is deleted. Also, the value of the label over the trajectory can be saved to a file and viewed with an external program such as `xmgr`. In the following, *category* implies one of `[`Atoms—Bonds—Angles—Dihedrals`]`.

- **list**: Return a list of available categories.
- **list** *category*: List all labels in the given category.
- **add** *category molID1/atomID1* `[` *molID2/atomID2*... `]`: Add a label involving the atom(s) *atomID* of the molecule *molID* to the given category.
- **show** *category* < **all** | *label_number* >: Turn on labels in the given category.
- **hide** *category* < **all** | *label_number* >: Turn off labels in the given category.
- **delete** *category* < **all** |*label_number*>: Delete labels in the given category.
- **graph** *category label_number* `[`*command*`]`: Show a graph of a label from the given category. The data will be written to a temporary file, and the specified command will be run to graph the data. %s in the *command* string will be replaced by the temporary filename.

11.3.12 light

There are four light sources, numbered 0 to 3, which are used to illuminate graphical objects. They are point sources located at infinity so the only controls are to rotate a light or turn it on or off. A dotted line from the light to the axis can be turned on to help show where the light is located.

- **num**: Return the number of lights available.
- *light_number* **on**: Turn a light on.
- *light_number* **off**: Turn a light off.
- *light_number* **highlight**: Display a line indicating the position of a light source.
- *light_number* **unhighlight**: Hide the line indicating the position of a light source.

- *light_number* **status**: Return the pair on/off highlight/unhighlight
- *light_number* **rot < x | y | z >** *angle*: Rotate a light (at infinity) *angle* degrees about a given axis.

11.3.13 logfile

Turn on/off logging a VMD session to a log file. This will create a log file with commands for all the actions taken during the session. The log file may be played back later by using the 'play' command or the Tcl 'source' command. The only actions recorded are those which change the state of the VMD display, so straight Tcl commands are not saved. All of the core VMD commands will write to the log.

- *filename*: Turn on logging to *filename*.
- **off**: Turn off logging.

To write log information to the file 'off', use the file name './off'. To have log information appear on the text console, use `logfile /dev/tty`.

11.3.14 menu

The menu forms of the GUI Control or query the on-screen GUI menu forms.

- **list**: Return a list of the available menus
- *menu_name* **on**: Turn a menu on.
- *menu_name* **off**: Turn a menu off.
- *menu_name* **status**: Return `on` if on, `off` if off.
- *menu_name* **loc**: Return the x y location.
- *menu_name* **move** x y: Move a menu to the given (x, y) location

where *menu_name* is one of the following:

- main
- mol
- animate
- edit
- graphics
- labels
- render
- display

- color
- sim
- tracker
- remote

Note that in the case when VMD is compiled without the `REMOTE` option, Sim and Remote menus are not defined and do not appear on the Main form [§3.3.1].

11.3.15 mol

Load, modify, or delete a molecule in VMD. In the following, *molecule_number* is a string describing which molecules are to be affected by the command. It is one of the following: `all`, `top`, `active`, `inactive`, `displayed`, `on`, `off`, `fixed`, `free`, or one of the unique integer ID codes assigned to the molecules when they are loaded (starting with 0). The codes (molIDs) are not reused after a molecule is deleted, so if you, for example, have three molecules loaded (numbered 0, 1, 2), delete molecule with molID equal to 0, and then load another molecule, the new molecule will have molID 3. Thus, the list of available molecule IDs becomes (1 2 3). The index of the molecule on this list is, among many other things, accessible through the `molinfo` command [§14]. In the above case, for example, molecule that was loaded the last has molID equal to 3, however, it is the third on the list of molecules, so it has the index equal to 2 (since we start countin from 0).

The molecule representations (views) are assigned integer number (starting with 0 for each molecule), which appear in the list on the Graphics form [§3.3.4]. The representations can be added, deleted or changed with the `mol` command. See also sections on `molinfo` command [§ 14] for more ways of retrieving information about the representations. See Chapter § 13 for information on how to use `mol` command to load user-defined graphics.

- **< new | load >** *structure_file_type structure_file* [*coordinate_file_type coordinate_file*] : Load a new molecule from *filename(s)* using the given *format*.
- **urlload <file type> <URL>**: Load a molecule from a given URL address
- **pdbload <four letter accession id>**: ftp molecule from PDB.
- **list**: Print a one-line status summary for each molecule.
- **list** *molecule_number*: Print a one-line status summary for each molecule matching the *molecule_number*. If only one molecule matches the *molecule_number*, also print the representation status for this molecule, i.e., number of representations as well as the representation number, coloring method , representation style and the selection string for each of the representations.
- **color** *coloring_method*: Change the default atom coloring method setting.
- **representation** *rep_style*: Change the default rendering method setting.
- **selection** *select_method*: Change the default atom selection setting.
- **modcolor** *rep_number molecule_number coloring_method*: Change the current coloring method for the given representation in the specified molecule.

- **modstyle** *rep_number molecule_number rep_style*: Change the current rendering method (style) for the given representation in the specified molecule.
- **modselect** *rep_number molecule_number select_method*: Change the current selection for the given representation in the specified molecule.
- **addrep**: Using the current default settings for the atom selection, coloring, and rendering methods, add a new representation to the top molecule.
- **delrep** *rep_number molecule_number*: Deletes the given representation from the specified molecule.
- **modrep** *rep_number molecule_number*: Using the current default settings for the atom selection, coloring, and rendering methods, changes the given representation to the current defaults.
- **delete** *molecule_number*: Delete molecule(s).
- **active** *molecule_number*: Make molecule(s) active.
- **inactive** *molecule_number*: Make molecule(s) inactive.
- **on** *molecule_number*: Turn molecule(s) on (make drawn).
- **off** *molecule_number* : Turn molecule(s) off (hide).
- **fix** *molecule_number*: Fix molecule(s).
- **free** *molecule_number*: Unfix molecule(s).
- **top** *molecule_number*: Set the top molecule.

11.3.16 molecule

Same as mol.

11.3.17 mouse

Change the current state (mode) of the mouse.

- **mode 0**: Set mouse mode to rotation.
- **mode 1**: Set mouse mode to translation.
- **mode 2**: Set mouse mode to scaling.
- **mode 3** *N*: Set mouse mode to rotate light *N*.
- **mode 4** *N*: Set mouse mode to picking mode *N*, where *N* is one of the following:
  - 0: query item
  - 1: pick center

- 2: pick atom
- 3: pick bond
- 4: pick angle
- 5: pick dihedral
- 6: move atom
- 7: move residue
- 8: move fragment
- 9: move molecule
- 10: force on atom
- 11: force on residue
- 12: force on fragment

11.3.18 play

Start executing text commands from a specified file, instead of from the console. When the end of the file is reached, VMD will resume reading commands from the previous source. This command may be nested, so commands being read from one file can include commands to read other files.

- *filename*: Execute commands from *filename*.

11.3.19 quit

Same as exit.

11.3.20 remote

Set up VMD to connect to or start a remote simulation program (optional).

- **cancel**: Cancel a remote simulation setup.
- **run**: Start a new remote simulation running, using the current setup parameters.
- **list**: Print the current configuration status in a human readable form.
- **list** [**all** | **jobs** | **apps** | **parameters** | **connection**]: List properties of remote simulations.
- **initialize** *computer* [*username*]: Make an initial connection to the provided remote computer, using the given username or the current username if it is not specified. This will contact the MDComm daemon there and retrieve a list of remote applications and running jobs.
- **new** *application*: Begin setting the parameters in order to run a new simulation using the specified program (application).

- **attach** *job*: Attach to a remote job which has been previously started by VMD, and left running.
- **readoptions** *filename*: Read parameter settings from the given file for the current remote simulation setup.
- **writeoptions** *filename*: Write out the current parameter settings to a file, for user to use later by the 'readoptions' command.
- **set** *option* **[value]**: Change the current setting for the given option to the specified value. This is used when setting the initial parameters before starting the simulation.

11.3.21 render

Output the currently displayed image (scene) to a file.

- **list**: List the available rendering methods.
- *method filename*: Render the global scene to *filename* using *method* and execute the default command, where *method* can be one of the following:
  - Raster3D
  - tkRaster3D
  - snapshot
  - POV3
  - POV
  - RayShade
  - Radiance
  - ART
  - PostScript
  - STL
  - VRML-1
  - Token
- *method filename command*: Render the global scene to *filename*, then execute '*command*'. Any %s in '*command*' are replaced by the filename (up to 5).
- *options method*: Get the default command string.
- *options method command*: Set new default command.
- *default method*: Get the original default command.

11.3.22 rock

Rotate the current scene continually at a specified rate.

- **off**: Stops rocking.
- **< x | y | z > by** *step*: Rock around the given axis at a rate of *step* degrees per redraw.
- **< x | y | z > by** *step n*: Rock around the given axis at a rate of *step* degrees per redraw for *n* steps, reverse, and repeat.

11.3.23 rotate

Rotate the current scene around a given axis by a certain angle. This does not change atom coordinates.

- **stop**: Stop all rotation, same as rock off.
- **< x | y | z > by** *angle*: Rotate around the given axis *angle* degrees.
- **< x | y | z > to** *angle*: Rotate the given axis to the absolute position *angle*.
- **< x | y | z > < by | to >** *angle step*: Rotate at a rate of *step* degrees per redraw.

11.3.24 scale

Scale the current scene up or down. This does not change atom coordinates.

- **by** *f*: Multiply scene scaling factor by *f*.
- **to** *f*: Set scene scaling factor to *f*.

11.3.25 sim

Control the state of a remote simulation after it is connected (optional).

- **list**: Return information about remote simulations. Each element returned contains three pieces of information; the molecule id, the remote machine name, and the current status. The status is one of:
  - No Connection: remote connection has been disconnected or stopped
  - Running Simulation: connected to an active simulation
- **top** *sim_number*: Set the top simulation.
- **top**: Get the molecule id of top simulation.
- **modify** *parameter value*: Change a parameter for the top simulation. Possible parameters are

  - rate n: Send only every n simulation timesteps.
  - keep n: Keep only every n received timesteps, appended to the animation.

- **patch < off | byatom | byload | bynode >**: Display the patches from a simulation (if any), coloring them by the specified criteria, for the top simulation.
- **detach** *sim_number*: Detach from a simulation, leaving it running on the remote computer.
- **kill** *sim_number*: Kill a simulation.

11.3.26 simulation

Same as sim.

11.3.27 stage

Position a checkerboard stage on the screen.

- **location < off | origin | bottom | top | left | right | behind >**: Set the location.
- **location**: Get the current location.
- **locations**: Get a list of possible locations.
- **panels** n: Set number of panels in stage, up to 30.
- **panels**: Get the number of panels in use

11.3.28 tracker

Initialize and control external spatial tracking devices.

- **list**: List running trackers.
- **list avail**: List available trackers.
- **start** *tracker_number*: Start a tracker.
- **pause** *tracker_number*: Pause a tracker.
- **unpause** *tracker_number*: Unpause a tracker.
- **location** *tracker_number sensor*: Print current location of *sensor*.

11.3.29 translate

Translate the objects in the current scene. This does not change the atom coordinates.

- **by** x y z: Translate by vector (x, y, z) in screen units (note, that this does not change the atom coordinates).
- **to** x y z: Translate to the absolute position (x, y, z) in screen units.

11.3.30 user

Add user-customized commands to the graphics display popup menu.

- **add menu** *name*: Add a new submenu *name* to the main popup menu. The submenu will be initially empty, and will not appear until items are added to it.
- **add item** *name command*: Add a new menu item named *name* to the 'User Command' submenu of the main menu. If a *command* is specified, then when the menu item is selected by the user, that command will be executed as if it had been typed at the console.
- **add** *separator*: Add a separator to the end of the 'User Command' submenu of the main menu.
- **add subitem** *menu name command*: Add an item *name* to the given submenu *menu*, which must have been added already with an `add menu` command. The given command will be executed when the item is selected by the user.
- **add subseparator** *menu*: Add a separator to the end of the given submenu.
- **add key** *key command*: Assign the given text command to the hot key *key*. When *key* is pressed while the mouse is in the display window, the specified command will be executed.
- **print < menus | keys >**: Print out the current definition of the user-defined menus, or the hot keys.

See section § 15.1 for examples of the use of the `user` command.

11.3.31 vmdlog

(Tcl) By default this procedure does not exist. If it is created, (nearly) every command that affects the display state of VMD is passed to it as its single parameter. This can be used as an interface alternative to the `logfile` [§11.3.13] command.

11.3.32 vmdinfo

(Tcl) Returns information about this version of VMD.

- **version**: Returns the version number;
- **versionmsg**: Full information about this version;
- **authors**: List of authors;
- **arch**: architecture type (in case you couldn't tell);
- **options**: options used to compile VMD;
- **www**: VMD home page;
- **wwwhelp**: VMD help page.

This function is available without Tcl and the information is displayed to the screen.

11.3.33 wait

Specify a number of seconds to wait before reading another command. Animation *continues* during this time.

- *time*: wait *time* seconds.

11.3.34 sleep

Specify a number of seconds to sleep before reading another command. Animation *stops* during this time.

- *time*: sleep *time* seconds.

First Word	Description
animate	Play/Pause/Rewind a molecular trajectory.
axes	Position a set of XYZ axes on the screen.
color	Change the color assigned to molecules, or edit the colormap.
colorinfo	(Tcl) Obtain color properties for various objects
debug	Turn on/off printing of debugging messages.
display	Change various aspects of the graphical display window.
echo	Turn on/off echoing of text commands to the console.
exit, quit	Quit VMD.
external	(Tcl) Start an external VMD command source *(optional)*.
help	Display an on-line help file with an HTML viewer.
logfile	Turn on/off logging a VMD session to a log file.
label	Turn on/off labels for atoms, bonds, angles, or dihedral angles.
light	Control the light sources used to illuminate graphical objects.
menu	Control or query the on-screen GUI menu forms.
molecule or mol	Load, modify, or delete a molecule in VMD.
mouse	Change the current state (mode) of the mouse.
play or run	Start executing text commands from a specified file.
remote	Set up VMD to connect to or start a remote simulation *(optional)*.
render	Output the currently displayed image (scene) to a file.
rock	Rotate the current scene continually at a specified rate.
rotate	Rotate the current scene around a given axis by a certain angle.
scale	Scale the current scene up or down.
simulation or sim	Control the state of a remote simulation after it is connected *(optional)*.
stage	Position a checkerboard stage on the screen.
tracker	Initialize and control external spatial tracking devices.
translate	Translate the objects in the current scene.
user	Add new commands to the graphics display pop-up menu.
vmdinfo	(Tcl) Get information about this version of VMD
wait	Wait a number of seconds before reading another command. Animation continues.
sleep	Sleep a number of seconds before reading another command. Animation is frozen.

Table 11.1: Summary of core text commands in VMD.

Source	Associated URL
faq	http://www.ks.uiuc.edu/Research/vmd/allversions/vmd_faq.html
raster3d	http://www.bmsc.washington.edu/raster3d
stride	http://www.embl-heidelberg.de/stride/stride_info.html
mdscope	http://www.ks.uiuc.edu/Research/mdscope/
mdcomm	http://www.ks.uiuc.edu/Research/mdcomm/
homepage	http://www.ks.uiuc.edu/Research/vmd/
quickhelp	http://www.ks.uiuc.edu/Research/vmd/vmd_help.html
babel	http://www.eyesopen.com/babel.html
radiance	http://radsite.lbl.gov/radiance/HOME.html
maillist	http://www.ks.uiuc.edu/Research/vmd/mailing_list/
namd	http://www.ks.uiuc.edu/Research/namd/
vrml	http://www.vrml.org/
rayshade	http://www-graphics.stanford.edu/~cek/rayshade/rayshade.html
povray	http://www.povray.org/
tcl	http://www.scriptics.com/
software	http://www.ks.uiuc.edu/Research/vmd/allversions/related_programs.html
userguide	http://www.ks.uiuc.edu/Research/vmd/vmd-1.3/ug/ug.html

Table 11.2: On-line Help Sources

Chapter 12

Vectors and Matrices

Tcl does not handle mathematical expressions very well. It is slow at evaluating expressions, and provides no facility for handling vectors or matrices. Since the latter two are needed for structure analysis, we have added routines to manipulate them. The routines themselves are independent of VMD, but do require TclX for a couple functions, such as `lassign`. The routines are loaded during the evaluation of the `$env(VMDDIR)/scripts/vmd/vmdinit.tcl` script.

A vector in VMD is a list of numbers. All of the vector routines but one will work with vectors of any length; `veccross` will only use vectors of three numbers. A matrix is a 4x4 collection of numbers stored as a list of 4 vectors of 4 numbers, in row-major form.

Following are descriptions and examples of all the commands. For more examples of vectors, though without much documentation, the script used to test the vectors implementation is located at `$env(VMDDIR)/scripts/vmd/test-vectors.tcl`. See also the chapter on writing scripts for VMD[§16].

Since Tcl is slow at math, some of these commands have been reimplemented in C++. (The original definition is in the vmd script distribution, but it is redefined later on inside VMD). At times, the speedup is a factor of 40 or more. These commands are noted by (C++).

12.1 Vectors

- **veczero** – Returns the zero vector, {0 0 0}

```
Example:
vmd > veczero
Info) 0 0 0
```

- (C++) **vecadd** *v1 v2* [*v3 ... vn*] – Returns the vector sum of all the terms.

```
Examples:
vmd > vecadd {1 2 3} {4 5 6} {7 8 9} {-11 -11 -11}
Info) 1 4 7
vmd > vecadd {0.1 0.2 0.4 0.8} {1 1 2 3} {3 1 4 1}
Info) 4.1 2.2 6.4 4.8
vmd > vecadd 4 5
Info) 9
```

- (C++) **vecsub** *v1 v2* – Returns the vector subtraction of the second term from the first

```
Examples:
vmd > vecsub 6 3.2
Info) 2.8
vmd > vecsub {10 9.8 7} {0.1 0 -0.1}
Info) 9.9 9.8 7.1
vmd > vecsub {1 2 3 4 5} {6 7 8 9 10}
Info) -5 -5 -5 -5 -5
```

- (C++) **vecscale** *c v* –
- (C++) **vecscale** *v c* – Returns the vector of the scalar value c applied to each term of v

```
Examples:
vmd > vecscale .2 {1 2 3}
Info) 0.2 0.4 0.6
vmd > vecscale {-5 4 -3 2} -2
Info) 10 -8 6 -4
vmd > vecscale -2 3
Info) -6
```

- **vecdot** *v1 v2* – Returns the scalar dot product of the two vectors

```
Examples:
vmd > vecdot {1 -2 3} {4 5 6}
Info) 12
vmd > vecdot {3 4} {3 4}
Info) 25
vmd > vecdot {1 2 3 4 5} {5 4 3 2 1}
Info) 35
vmd > vecdot 3 -2
Info) -6
```

- **veccross** *v1 v2* – Returns the vector cross product of the two vectors.

```
Examples:
vmd > veccross {1 0 0} {0 1 0}
Info) 0 0 1
vmd > veccross {2 2 2} {-1 0 0}
Info) 0 -2 2
```

- **veclength** *v* – Returns the scalar length of v ($\|v\|$)

```
Examples:
vmd> veclength 5
Info) 5.0
vmd > veclength {5 12}
```

```
Info) 13.0
vmd > veclength {3 4 12}
Info) 13.0
vmd > veclength {1 -2 3 -4}
Info) 5.47723
```

- **veclength2** *v* – Returns the square of the scalar length of v ($\|v\|^2$)

```
Examples:
vmd > veclength2 5
Info) 25
vmd > veclength2 {5 12}
Info) 169
vmd > veclength2 {3 4 12}
Info) 169
vmd > veclength2 {1 -2 3 -4}
Info) 30
```

- **vecnorm** *v* – Returns the vector of length 1 directed along v

```
Examples:
vmd > vecnorm -10
Info) -1.0
vmd > vecnorm {1 1 }
Info) 0.707109 0.707109
vmd > vecnorm {2 -3 1}
Info) 0.534522 -0.801783 0.267261
vmd > vecnorm {2 2 -2 2 -2 -2}
Info) 0.408248 0.408248 -0.408248 0.408248 -0.408248 -0.408248
```

- **vecdist** *v1 v2* – Returns the distance between the two vectors ($\|v_2 - v_1\|$)

```
Examples:
vmd > vecdist -1.5 5.5
Info) 7.0
vmd > vecdist {0 0 0} {3 4 0}
Info) 5.0
vmd > vecdist {0 1 2 3 4 5 6} {-6 -5 -4 -3 -2 -1 0}
Info) 15.8745
```

- **vecinvert** *v* – Returns the additive inverse of v ($-$v).

```
Examples:
vmd > vecinvert -11.1
Info) 11.1
vmd > vecinvert {3 -4 5}
Info) -3 4 -5
vmd > vecinvert {0 -1 2 -3}
Info) 0 1 -2 3
```

12.2 Matrix routines

Because matrices are rather large when expressed in text form, the following definitions are used for the examples.

- **transidentity** – Returns the identity matrix.

```
Example:
vmd > transidentity
Info) {1.0 0.0 0.0 0.0} {0.0 1.0 0.0 0.0} {0.0 0.0 1.0 0.0} {0.0 0.0 0.0 1.0}
```

- **transtranspose** *m* – Returns the matrix transpose of the given matrix

```
Example:
vmd > transtranspose {{0 1 2 3 4} {5 6 7 8} {9 10 11 12} {13 14 15 16}}
Info) {0 5 9 13} {1 6 10 14} {2 7 11 15} {3 8 12 16}
```

- (C++) **transmult** *m1 m2* [*m3 ... mn*] – Returns the matrix multiplication of the given matrices

```
Examples:
vmd > set mat1 {{1 2 3 4} {-2 3 -4 5} {3 -4 5 -6} {4 5 -6 -7}}
vmd > set mat2 {{1 0 0 0} {0 0.7071 -0.7071 0} {0 0.7071 0.7071 0} {0 0 0 1}}
vmd > set mat3 {{0.866025 0 0 0} {0 1 0 0} {-0.5 0 0.866025 0} {0 0 0 1}}
vmd > transmult $mat1 [transidentity]
Info) {1.0 2.0 3.0 4.0} {-2.0 3.0 -4.0 5.0} {3.0 -4.0 5.0 -6.0}
   {4.0 5.0 -6.0 -7.0}
vmd > transmult $mat1 $mat2 $mat3
Info) {0.512475 3.5355 0.612366 4.0} {0.7428 -0.7071 -4.28656 5.0}
   {-0.58387 0.7071 5.5113 -6.0} {7.35315 -0.7071 -6.73603 -7.0}
```

- **transaxis** <**x**|**y**|**z**> *amount* [**deg**|**rad**|**pi**] – Returns the transformation matrix needed to rotate around the specified axis by a given amount. By default, the amount is specified in degrees, though it can also be given in radians or factors of pi.

```
Examples:
vmd > transaxis x 90
Info) {1.0 0.0 0.0 0.0} {0.0 -3.67321e-06 -1.0 0.0} {0.0 1.0 -3.67321e-06 0.0}
        {0.0 0.0 0.0 1.0}
vmd > transaxis y 0.25 pi
Info) {0.707107 0.0 0.707107 0.0} {0.0 1.0  0.0 0.0}
        {-0.707107 0.0 0.707107 0.0} {0.0 0.0  0.0 1.0}
vmd > transaxis z 3.1415927 rad
Info) {-1.0 -2.65359e-06 0.0 0.0} {2.65359e-06 -1.0 0.0 0.0} {0.0 0.0 1.0 0.0}
        {0.0 0.0 0.0 1.0}
```

- **transvec** *v* – Returns the transformation matrix needed to bring the x axis along the v vector. This matrix is not unique, since a final rotation is allowed around the vector. The matrix is made from a rotation around y, then one about z.

```
Examples:
vmd > transvec {0 1 0}
Info) {-3.67321e-06 -1.0 0.0 0.0} {1.0 -3.67321e-06 0.0 0.0} {0.0 0.0 1.0 0.0}
        {0.0 0.0 0.0 1.0}
vmd > vectrans [transvec {0 0 2}] {1 0 0}
Info) 0.0 0.0 1.0
```

- **transvecinv** *v* – Returns the transformation needed to bring the vector v to the x axis. This produces the inverse matrix to transvec, and is composed of a rotation about z then one about y.

```
Examples:
vmd > transvecinv {0 -1 0}
Info) {-3.67321e-06 -1.0 0.0 0.0} {1.0 -3.67321e-06 0.0 0.0} {0.0 0.0 1.0 0.0}
    {0.0 0.0 0.0 1.0}
vmd > vectrans [transvecinv {-3 4 -12}] {-3 4 -12}
Info) 13.0 -1.8e-05 5.8e-05
vmd > transmult [transvec {6 -5 7}] [transvecinv {6 -5 7}]
Info) {0.999999 2.29254e-07 -6.262e-09 0.0} {2.29254e-07 0.999999
    -4.52228e-07 0.0} {-6.262e-09 -4.52228e-07 1.0 0.0} {0.0 0.0 0.0 1.0}
```

- (C++) **transoffset** *v* – Returns the transformation matrix needed to translate by the given offset

```
Examples:
vmd > transoffset {1 0 0}
Info) {1.0 0.0 0.0 1} {0.0 1.0 0.0 0} {0.0 0.0 1.0 0} {0.0 0.0 0.0 1.0}
vmd > transoffset {-6 5 -4.3}
Info) {1.0 0.0 0.0 -6} {0.0 1.0 0.0 5} {0.0 0.0 1.0 -4.3} {0.0 0.0 0.0 1.0}
```

- **transabout** *v amount* **[deg|rad|pi]** – Generates the transformation matrix needed to rotate by the given amount counter-clockwise around axis which goes through the origin and along the given vector. As with transvec, the units of the amount of rotation can be degrees, radians, or multiples of pi.

```
Examples:
# this is a rotation about x by 180 degrees
vmd > transabout {1 0 0} 180
Info) {1.0 0.0 0.0 0.0} {0.0 -1.0 -2.65359e-06 0.0} {0.0 2.65359e-06
        -1.0 0.0} {0.0 0.0 0.0 1.0}
# a rotation about z by 90 degrees
# (compare this to "transaxis z 90"
vmd > transabout {0 0 1} 1.5709 rad
Info) {0.999624 -0.027414 0.0 0.0} {0.027414 0.999624 0.0 0.0}
        {0.0 0.0 1.0 0.0} {0.0 0.0 0.0 1.0}
vmd > transabout {1 1 1} 1 pi
Info) {-0.333335 0.666665 0.666669 0.0} {0.666668 -0.333334 0.666666
        0.0} {0.666666 0.66667 -0.333332 0.0} {0.0 0.0 0.0 1.0}
```

- **trans** [**center** {*x y z*}] [**origin** {*x y z*}] [**offset** {*x y z*}] [**axis x** *amount* [**rad|deg|pi**]] [**axis y** *amount* [**rad|deg|pi**]] [**axis z** *amount* [**rad|deg|pi**]] [**x** *amount* [**rad|deg|pi**]] [**y** *amount* [**rad|deg|pi**]] [**z** *amount* [**rad|deg|pi**]] [**axis** {*x y z*} *amount* [**rad|deg|pi**]] [**bond** {*x1 y1 z1*} {*x2 y2 z2*} *amount* [**rad|deg|pi**]] [**angle** {*x1 y1 z1*} {*x2 y2 z2*} {*x3 y3 z3*} *amount* [**rad|deg|pi**]] –

  This command can do almost everything the other ones can do, and then some. It is designed to be the main function used for generating transformation matrices.

Using it correctly calls for understanding how it works internally. There are three matrices: centering, rotation, and offset. The centering matrix determines where the center of rotation is located. By default, this is the origin, but it can be changed to pivot about any point. The rotation matrix defines the rotation about that centering point, and the offset matrix defines the final translation after the rotation.

For example, to rotate around a given point, the transformations would be 1) the centering matrix to bring that point to the origin, 2) the rotation about the center, and 3) the final offset to return the origin back to its original location.

The different options for the `trans` command modify the matrices in various ways.

- **center** {*x y z*} – Sets the centering matrix so that point x y z is brought to the origin
- **offset** {*x y z*} – Sets the offset matrix so that the origin is brought to x y z
- **origin** {*x y z*} – Sets both the centering and offset matrices to x y z
- **axis x** *amount* [**rad|deg|pi**] – Adds a rotation about the x axis by the given amount to the rotation matrix
- **axis y** *amount* [**rad|deg|pi**] – Adds a rotation about the y axis by the given amount to the rotation matrix
- **axis z** *amount* [**rad|deg|pi**] – Adds a rotation about the z axis by the given amount to the rotation matrix
- **axis** {*x y z*} *amount* [**rad|deg|pi**] – Adds a rotation of the given amount about the given vector to the rotation matrix
- **bond** {*x1 y1 z1*} {*x2 y2 z2*} *amount* [**rad|deg|pi**] – Sets the center and offset transformations to the first point, and defines a rotation about the bond axis by the given amount.
- **angle** {*x1 y1 z1*} {*x2 y2 z2*} {*x3 y3 z3*} *amount* [**rad|deg|pi**] – Sets the center and offset transformation to the second point, and defines a rotation about the axis perpendicular to the plane made by the three points (the vector is computed from the cross product of the vector connecting the first two points with that connected the last two).

12.3 Multiplying vectors and matrices

There are two commands to multiply a matrix and a vector, `vectrans` and `coordtrans`. They assume the vector is in column form and premultiply the matrix to the vector. If the vector contains four numbers, the two commands are identical. If the vector has three elements, a fourth is added; a 0 for `vectrans` and a 1 for `coordtrans`. The difference is that vectors are not affected by translations during transformations, while coordinates are.

- (C++) **vectrans** *m v* – Multiple the matrix m with the vector v (length 4); returns a vector
- **coordtrans** *m v* – Multiple the matrix m with the coordinate v (length 3); returns a vector

```
Examples:
vmd > vectrans [transaxis z 90] {1 0 0}
Info) -3.67321e-06 1.0 0.0
vmd > vectrans [transvecinv {-3 4 -12}] {-3 4 -12}
Info) 13.0 -1.8e-05 5.8e-05
```

12.4 Misc. functions and values

Several other terms are added to the vectors package. The first is the variable M_PI, which contains the value of pi.

```
Examples:
vmd > set M_PI
Info) 3.14159265358979323846
vmd > expr 90 * ($M_PI / 180)
Info) 1.5708
```

The functions `trans_from_rot`, `trans_to_rot`, `trans_from_offset`, and `trans_to_offset` are used to get or set a transformation matrix from either a 3x3 rotation matrix or offset vector. As currently designed, these assume there is no scaling in the matrix. The `trans_from_offset` is identical to `transoffset` and is present for completeness.

The last is `find_rotation_value varname`, which takes a variable name and extracts from the beginning of it those terms which describe an amount of rotation. The rest of the data in the variable remains, and the amount of rotation, in radians, is returned. This is used by those functions which need a rotation. The valid values are: a number, followed by one of `rad`, `radian`, or `radians` for a value in radians, the word `pi` to give the rotation in factors of pi, or one of `deg`, `degree`, or `degrees` for a value in degrees. If no units are given, the value is assumed to be in degrees.

Examples:

```
vmd > set a "180 deg north"
Info) 180 deg north
vmd > find_rotation_value a
Info) 3.14159
vmd > set a
Info) north
vmd > set a "1 pi to eat"
Info) 1 pi to eat
vmd > find_rotation_value a
Info) 3.14159
vmd > set a
Info) to eat
vmd > set a 45
Info) 45
```

```
vmd > find_rotation_value a
Info) 0.785398
vmd > expr $M_PI * 3.0 / 2.0
Info) 4.71239
vmd > set a "4.71238 radians"
Info) 4.71238 radians
vmd > find_rotation_value a
Info) 4.71238
```

Chapter 13

User-Defined Graphics

Sometimes you may want to display graphics other than the standard molecular graphics. You may want to draw a box around a molecule, or an arrow between two atoms, or place a text label somewhere in space, or test a new method for visualizing a molecule. Because of the wide range of objects which can be displayed, VMD cannot be pre-programmed to display them all. Instead, it does offer a way to build up a new object from simple graphical items, including: points, lines, cylinders, cones, spheres, triangles (both with and without specified normals), and text. Since these are displayed in the scene just like all other graphics, they can also be saved to the various ray tracing formats.

This chapter describes how to use the text interface to add new graphics to VMD. It is broken up into four parts. The first is an introduction to the basic graphics available, the second is a tutorial which demonstrates many of the ways to use the graphics feature, the third describes how the base `graphics` command works, and the last describes the `draw` extension which can handle larger, more complicated objects.

13.1 Introduction

There are 10 types of basic graphical items built into VMD, which are used with `graphics` and `draw` commands. These are:

- **point** {*x y z*} –
  draws a point at the given position
- **line** {*x1 y1 z1*} {*x2 y2 z2*} [**width** *w*] [**style** <**solid**|**dashed**>] –
  draws either a solid or dashed line of the given width from the first point to the second. By default, this is a solid line of width 1. A caution about the line width from the SGI man page for `linewidth(3G)`:

```
The maximum width of aliased lines is 255.

IRIS-4D VGX model supports antialiased line widths 1.0 and 2.0, and
aliased line widths one through 255.  IRIS-4D G, GT, and GTX models, the
Personal Iris, Indy, Iris Entry, XL, XS, XS24, XZ, Elan and Extreme
systems support antialiased line width 1.0, and aliased line widths one
through 255.
```

Note that there is a known bug in VMD which causes it to mix up and lose track of the anti-alias and depth-cue modes.

- **cylinder {***x1 y1 z1***} {***x2 y2 z2***} [radius** *r***] [resolution** *n***] [filled <yes|no>]** –

  draws a cylinder of the given radius (default `r`=1) from the first point to the second. The cylinder is actually drawn as an `n` sided polygon. If the `filled` option is true (default), the ends are capped with flat disks, otherwise the cylinder is hollow.

- **cone {***basex basey basez***} {***tipz tipy tipz***} [radius** *r***] [resolution** *n***]** –

  draw a cone with the center of the base at the first point and the tip at the second. The radius(default `r`=1) determines the width of the base. As with `cylinder`, the resolution determines the number of polygons used in the approximation.

- **triangle {***x1 y1 z1***} {***x2 y2 z3***} {***x3 y3 z3***}** –

  draws a triangle with endpoints at each of the three vertices

- **trinorm {***x1 y1 z1***} {***x2 y2 z3***} {***x3 y3 z3***} {***nx1 y1 z1***} {***nx2 ny2 nz3***} {***nx3 ny3 nz3***}** –

  draws a triangle with endpoints at each of the first three points. The second group of three values specify the normals for the three points. This is used for making a smooth shading across the triangle.

- **sphere {***x y z***} [radius** *r***] [resolution** *n***]** –

  draws a sphere of the given radius (default `r`=1) centered at the vertex. The resolution (default `n`=6) determines how many polygons are used in the approximation of a sphere.

- **text {***x y z***} "***text string***"** –

  displays the text string with the bottom left of the string starting at the given vertex.

- **color** *colorId* –

- **color** *name* –

- **color** *trans_name* – Each of the above geometrical objects are drawn using the current color. Initially, that color is blue, which has the colorid of 0. The `color` command changes the current color, and stays info effect until the next `color` command. Thus, to draw a red cylinder then a red sphere, first use the command `color red` command to change the color, then use the `cylinder` and `sphere` commands.

- **materials <on|off>** – Material properties are used to make the graphical objects (lines, cylinders, etc.) be affected by the light sources. These make the objects look more realistic, but are slower on machines which don't implement materials in hardware (see chapter § 6 and sections on color [§ 11.3.3] and colorinfo [§ 11.3.4] commands for the information on how to turn off material characteristics for all objects in VMD). One surprising effect of material characteristics is that lines are affected. In some lighting situations, the lines can even appear to disappear. Thus, you may want to turn off materials before drawing lines.

13.2 Tutorials and Examples

We'll start with some basic examples. Load VMD and enter:

```
draw cylinder {0 0 0} {5 0 0} radius 0.2
```

VMD will print:

```
Info) Loading new molecule ...
```

because of the way graphics are implemented (don't worry about the details yet) and draw a blue cylinder on the screen.

Now change the color to red and draw two more cylinders:

```
draw color red
draw cylinder {4 1 0} {5 0 0} radius 0.1
draw cylinder {4 -1 0} {5 0 0} radius 0.1
```

You might want to reset the view. Since the original scene had no data, a default size was chosen. Now that there is data, a reset view will center the graphics just as if it was a regular molecule (in fact, the implementation is a molecule with no atoms and only graphics). You now have a simple arrow. Let's make a nicer looking one now. First, remove all the graphics:

```
draw delete all
```

and make a cylinder with a cone on the end

```
draw cylinder {0 0 0} {4 0 0} radius 0.1
draw cone {4 0 0} {5 0 0} radius 0.15
```

Note that the widest part of the cone is at the first point. As long as we're at it, let's add some text to the end of the arrow in green

```
draw color green
draw text {5 0 0} "This way"
```

Cool, eh? Of course, more things are available than a cylinder, cone, and text. How about a sphere?

```
draw sphere {0 0 0} radius 0.3
draw sphere {0 2 0} radius 0.2
```

The color of the spheres is green because that was the previously assigned color. The color doesn't change until explicitly specified with the color option. The default color is blue.

And some lines, first, a dotted line connecting the spheres

```
draw line {0 0 0} {0 2 0} style dashed
```

then a solid one from the second sphere to the end of the arrow

```
draw line {0 2 0} {5 0 0}
```

Now, reset the view and move things around a bit. You may be surprised to see the lines appear and disappear. This is because the lines have a "material property"; which means they are affected by the lights. For some things, this may be useful, but usually if you want to draw lines, you'll need to turn materials off.

```
draw materials off
```

13.2.1 Drawing a graph

Here's an example of how to draw a 2-D graph. First, get rid of the current graphics (since there may be graphics from the previous tutorial):

```
draw delete all
```

Define a function to graph, in this case, a parabola.

```
proc f {x} {
 expr $x * $x
}
```

The following function draws a graph given the function name, the start and end values, and the step size.

```
proc draw_graph {function start end step} {
  # do some error checking
  if {$step < 0} {set step -$step}
  if {$step == 0} {error "draw_graph: cannot have step size of zero"}
  if {$start > $end} {
      set tmp $start
      set start $end
      set end $tmp
  }
  # turn materials off
  draw materials off
  # draw the data in green
  draw color green

  # calculate and save the initial coordinates
  set y0 [$function $start]
  set x0 $start
  set miny $y0
  set maxy $y0

  # go through the coordinates $start+$step to $end, $step at a time
  for {set x1 [expr $start + $step]} {$x1 <= $end} {
                                     set x1 [expr $x1 + $step]} {
      # calculate the function value for this point
      set y1 [$function $x1]
      # and draw a line to connect the previous point to the current one
      draw line "$x0 $y0 0" "$x1 $y1 0"
      # save the min/max values of y, and copy the x1,y1 into x0,y0
      if {$y1 < $miny} {set miny $y1}
      if {$y1 > $maxy} {set maxy $y1}
      set y0 $y1
      set x0 $x1
   }
```

```
    # draw a red box around everything
    draw color red
    draw line "$start $miny 0" "$end   $miny 0"
    draw line "$start $miny 0" "$start $maxy 0"
    draw line "$end   $maxy 0" "$end   $miny 0"
    draw line "$end   $maxy 0" "$start $maxy 0"
    # and put some labels down
    draw text "$end   $miny 0" "x ->"
    draw text "$start $maxy 0" "f(x)"
}
```

Copy these two definitions into VMD's text console (this script is available from the VMD script library at http://www.ks.uiuc.edu/Research/vmd/script_library/), then enter

```
draw_graph f -5 4 0.2
```

and reset the view. Presto, a nice little parabola.

13.2.2 Triangles

If you want to draw the surface of a solid object you can build it out of triangles. The graphics interface to VMD supports two types of triangles, simple triangles with just the corners defined, or "trinorms" which have normals defined for each corner.

First, a simple red triangle

```
draw delete all
draw color red
draw triangle {1 0 0} {0 1 0} {0 0 1}
```

Not bad, so let's put some more in

```
draw triangle {1 0 0} {0 1 0} {0 0 -1}
draw triangle {1 0 0} {0 -1 0} {0 0 1}
draw triangle {1 0 0} {0 -1 0} {0 0 -1}
draw triangle {-1 0 0} {0 1 0} {0 0 1}
draw triangle {-1 0 0} {0 1 0} {0 0 -1}
draw triangle {-1 0 0} {0 -1 0} {0 0 1}
draw triangle {-1 0 0} {0 -1 0} {0 0 -1}
```

And you have yourself an octahedron. See how the colors on the faces don't go smoothly from one side to the next? This is because the surface normals for one face are quite different from another, so the lights are reflected differently. The colors can be made smoother by defining the normals for the corners such that the normals of the edges are the same. One caution about defining the vertices and normals: they must be given in counter-clockwise order or the shading will be wrong.

As an example, here's the above octahedron with the normals specified:

```
draw delete all
draw color red
draw trinorm  {1 0 0} {0 1 0} {0 0 1} {1 0 0} {0 1 0} {0 0 1}
```

```
  draw trinorm  {1 0 0} {0 0 -1} {0 1 0} {1 0 0} {0 0 -1} {0 1 0}
  draw trinorm  {1 0 0} {0 0 1} {0 -1 0} {1 0 0} {0 0 1} {0 -1 0}
  draw trinorm  {1 0 0} {0 -1 0} {0 0 -1} {1 0 0} {0 -1 0} {0 0 -1}
  draw trinorm  {-1 0 0} {0 0 1} {0 1 0} {-1 0 0} {0 0 1} {0 1 0}
  draw trinorm  {-1 0 0} {0 1 0} {0 0 -1} {-1 0 0} {0 1 0} {0 0 -1}
  draw trinorm  {-1 0 0} {0 -1 0} {0 0 1} {-1 0 0} {0 -1 0} {0 0 1}
  draw trinorm  {-1 0 0} {0 0 -1} {0 -1 0} {-1 0 0} {0 0 -1} {0 -1 0}
```

13.2.3 Draw a surface plot

The following will plot the surface of a given function from $-5 <= x, y <= 5$. (This is a bit more complicated, so I don't want to include as much error checking for the inputs.)

The graphics here will be the function

```
proc g {x y} {expr sqrt(abs($x*$y))}

proc draw_surface { f } {
  set minx -5
  set maxx 5
  set miny -5
  set maxy 5
  set step 0.5
  # first, get the data (this isn't the most data efficient way of
  # doing things)
  for {set i $minx} {$i <= $maxx} {set i [expr $i + $step]} {
    for {set j $miny} {$j <= $maxy} {set j [expr $j + $step]} {
      set data($i,$j) [$f $i $j]
    }
  }
  # make another pass through to plot it
  for {set i $minx} {$i <= [expr $maxx - $step]} {set i [expr $i + $step]} {
    for {set j $miny} {$j <= [expr $maxy - $step]} {set j [expr $j + $step]} {
      # get the next two corners
      set i2 [expr $i + $step]
      set j2 [expr $j + $step]
      # find the middle
      set imiddle [expr $i + $step/2]
      set jmiddle [expr $j + $step/2]
      set kmiddle [expr ($data($i,$j) + $data($i2,$j) + $data($i2,$j2) \
+ $data($i,$j2)) / 4.0]
      # use a cool coloring scheme (this depends on the graph having a min
      # value of 0 and max of 5, or at least less than about 30)
      draw color [expr 34 + [int [expr  6 * $kmiddle]] % 32]
      # make 4 triangles
      draw triangle "$i $j $data($i,$j)" "$imiddle $jmiddle $kmiddle" \
                "$i2 $j $data($i2,$j)"
      draw triangle "$i $j $data($i,$j)" "$i $j2 $data($i,$j2)" \
```

```
                "$imiddle $jmiddle $kmiddle"
      draw triangle "$i2 $j2 $data($i2,$j2)" "$i2 $j $data($i2,$j)" \
                 "$imiddle $jmiddle $kmiddle"
      draw triangle "$i2 $j2 $data($i2,$j2)" "$imiddle $jmiddle $kmiddle" \
                "$i $j2 $data($i,$j2)"
    }
  }
}
```

And graph it with the command:

```
draw_surface g
```

Looks like it should be on a quilt, doesn't it?

13.2.4 Drawing a box around a molecule

Here's a quick function to draw a box around a molecule. Delete any graphics you may have loaded and enter this script.

```
proc box_molecule {molid} {
      # get the min and max values for each of the directions
      # (I'm not sure if this is the best way ... )
      set sel [atomselect top all]

      set coords [lsort -real [$sel get x]]
      set minx [lindex $coords 0]
      set maxx [lindex [lsort -real -decreasing $coords] 0]

      set coords [lsort -real [$sel get y]]
      set miny [lindex $coords 0]
      set maxy [lindex [lsort -real -decreasing $coords] 0]

      set coords [lsort -real [$sel get z]]
      set minz [lindex $coords 0]
      set maxz [lindex [lsort -real -decreasing $coords] 0]

      # and draw the lines
      draw materials off
      draw color yellow
      draw line "$minx $miny $minz" "$maxx $miny $minz"
      draw line "$minx $miny $minz" "$minx $maxy $minz"
      draw line "$minx $miny $minz" "$minx $miny $maxz"

      draw line "$maxx $miny $minz" "$maxx $maxy $minz"
      draw line "$maxx $miny $minz" "$maxx $miny $maxz"

      draw line "$minx $maxy $minz" "$maxx $maxy $minz"
      draw line "$minx $maxy $minz" "$minx $maxy $maxz"
```

```
      draw line "$minx $miny $maxz" "$maxx $miny $maxz"
      draw line "$minx $miny $maxz" "$minx $maxy $maxz"

      draw line "$maxx $maxy $maxz" "$maxx $maxy $minz"
      draw line "$maxx $maxy $maxz" "$minx $maxy $maxz"
      draw line "$maxx $maxy $maxz" "$maxx $miny $maxz"
}
```

If you don't already have a molecule loaded, load one, e.g.,
`mol load pdb $env(VMDDIR)/proteins/alanin.pdb`
and make it the `top` molecule. Then execute the command:

```
  box_molecule top
```

(you could also enter the molecule id instead of `top`).

13.2.5 Adding a label

Here's a quick way to add your own label to an atom selection [§7]. This function take the selection text and the labels that atom (in the `top` molecule) with the given string. It returns with an error if more anything other than one atom is selected.

```
proc label_atom {selection_string label_string} {
    set sel [atomselect top $selection_string]
    if {[$sel num] != 1} {
        error "label_atom: '$selection_string' must select 1 atom"
    }
    # get the coordinates of the atom
    lassign [$sel get {x y z}] coord
    # and draw the text
    draw text $coord $label_string
}
```

13.2.6 Interface to picking

When an atom is picked with the mouse, the Tcl variable `vmd_pick_mol` gets set to the moleule id of the picked molecule, and `vmd_pick_atom` contains the atom index. The Tcl command `trace` can be used to call a function when those values change.

As an example of what can be done with this, the following statements will put a sphere around each atom when it has been picked with the mouse. To turn it off, execute
`trace vdelete vmd_pick_atom w sphere_pick`.

```
proc sphere_pick {name element op} {
     # get the atom and molecule picked
     global vmd_pick_atom vmd_pick_mol
     # get the coordinates
     set sel [atomselect $vmd_pick_mol "index $vmd_pick_atom"]
     lassign [$sel get {x y z}] coords
```

```
    draw sphere $coords radius 1.0
}
trace variable vmd_pick_atom w sphere_pick
```

13.2.7 Animation

One last example of what can be done with the user-defined graphics. While VMD is not designed for animation, it can be made to animate objects by repeating the cycle of drawing the objects then calling the `display update` command. Here's an example which has several balls swinging (non-physically) at the end of a chain. If you want to stop it before it finishes, use `<Ctrl-C>`, but if you catch it in the wrong place and nothing moves, you'll need to execute the command `display update on` (see section § 11.3.6 for details about display updates.

```
proc swing {} {
    # get rid of everything else (if there is anything)
    if [expr [molinfo top] != -1] { mol off all }
    # create a new graphics molecule to handle just this
    mol load graphics "swing"
    set mol [molinfo top]
    set center {0 0.5 0}
    set radius 0.125
    set offset ".25 0 0"
    set length 1
    set firsttime 1
    display resetview
    axes location off
    stage location off
    for {set i 0} {$i < 1000} {incr i} {
        display update off
        scale by 1.003
        rotate y by 5
        display update on
        set top1 [vecsub $center $offset]
        set top2 $center
        set top3 [vecadd $center $offset]
        # compute the bottom location
        set bot1 [vecsub $top1 "0 $length 0"]
        set bot2 [vecsub $top2 "0 $length 0"]
        set bot3 [vecsub $top3 "0 $length 0"]
        set xdiff [expr sin($i/10.0)]
        set ydiff [expr 1.0 - abs(cos($i/10.0))]

        if [expr $xdiff < 0] {
            set bot1 [vecadd $bot1 "$xdiff $ydiff 0"]
        } else {
            set bot3 [vecadd $bot3 "$xdiff $ydiff 0"]
        }
```

```
        if $firsttime {
    set firsttime 0
            display resetview
   } else {
            graphics $mol delete all
    }
        # draw the three different balls/strings in different colors
        graphics $mol color red
        graphics $mol sphere $bot1 radius $radius
        graphics $mol color green
        graphics $mol sphere $bot2 radius $radius
        graphics $mol color blue
        graphics $mol sphere $bot3 radius $radius
graphics $mol materials off
        graphics $mol color red
graphics $mol line $top1 $bot1
        graphics $mol color green
graphics $mol line $top2 $bot2
        graphics $mol color blue
graphics $mol line $top3 $bot3
display reshape
    }
    display update on
}
```

13.3 Graphics

This section describes technical aspects of how the graphics commands are implemented and used. It should be read by those who want to understand how to build script-level extensions and those who like knowing more about how things work.

The user controlled graphics are implemented as if they are molecules of zero length which contains only a list of graphical objects. (Indeed, internally the class is called "MoleculeGraphics" and is derived from the Molecule class.) Thus, all the normal controls are applicable, and this is why the graphics "molecules" are on the Mol form. This simplifies matters as many of the controls, such as the mouse controls, do not need to be duplicated. However, some of the commands that apply only to molecules then make no sense, such as the graphics form.

Multiple graphics lists can be created. Since they are just specializations of a normal molecule, they are created with the `mol load` command, just like any other molecule. The full command is:

- **mol load graphics** *name*

The name parameter should be a distictive string related to the molecule. For instance, if the purpose of a new graphics list is to read in a graphics file format, the name should be the file name. If it is a method of viewing data from another, loaded molecule, the name should contain the name of the method and the molecule Id of the molecule from which it was derived. VMD reserves the name "graphics" for use with the `draw` command as that command always adds its graphics to the last graphical molecule with that name.

Once the graphics list is loaded and on the list of molecules, it can be distinguished from a normal molecule by using the `molinfo <molID> get source` command, which returns "Graphics" for graphics molecules.

The information pertaining to a given graphics list is accessed via the `graphics` command, which is of the following form:

- **graphics <molecule id> command** [*options ...*]

where the "molecule id" is the id of the graphics molecule on the molecule list. The *command* field can be one of the graphical items listed in section §13.1, or one of the commands `exists`, `delete`, `replace`, `list` or `info`. The `graphics` command returns a value, which depends on the command given.

As graphical primitives are added to the list they are assigned a unique, increasing value. The first object added is assigned 0, the second is assigned 1, etc. The commands which add an item return its value. A list of the available object ids is available with the `list` command. To test if a specific primitive exists, use the `exists` command followed by the object index. This returns a 1 if the index exists, and 0 if it does not. Information can be retrieved about a given item with the `info` command. This takes one option, which is the number of graphical information from which information is desired.

A given primitive is deleted with the `delete` command followed by the index of the object to delete. It does not have a return value. If the parameter is the string `all`, then all the primitives are deleted and the index count reset to 0.

Another deletion method allows one primitive to be replaced by another; suspiciously enough, this is the `replace` command. What it does is delete the given primitive and set things up so the next primitive is put in the same spot on the list and given the same index. This is designed for situations where the graphics are complicated to build, but where the colors might want to be easily changed. The color definition can be replaced without needing to rebuild the full list.

Here are examples of all the above commands:

```
vmd > mol load graphics testing
Info) Loading new molecule ...
vmd > graphics top sphere {0.1 0.2 0.3} radius 0.4
Info) 0
vmd > graphics top cylinder {-1 -1 -1} {2.6 2.5 2.4} resolution 3
Info) 1
vmd > graphics top line {0 0 0} {3 0 0} style dashed
Info) 2
vmd > graphics top delete 1
vmd > graphics top list
Info) 0 2
vmd > graphics top exists 0
Info) 1
vmd > graphics top exists 1
Info) 0
vmd > graphics top info 0
Info) sphere {0.100000 0.200000 0.300000} radius 0.400000 resolution 6
vmd > graphics top replace 0
vmd > graphics top point {3.3 2.2 1.1}
```

```
Info) 0
vmd > graphics top info 0
Info) point {3.300000 2.200000 1.100000}
vmd > graphics top list
Info) 0 2
```

13.4 Draw and Drawing Extensions

The `draw` command is a straight Tcl function which is meant to simplify the interface to the `graphics` command as well as provide a base for extensions to the standard graphics primitives.

The format of the `draw` command is:

- **draw** *command* [*arguments*]

Unlike the `graphics` command, `draw` does not take a molecule index by default. Instead, it searches for the last graphical molecule with the name "graphics" and uses that. (If such a molecule doesn't exist, it is created.) The basic commands are identical to those used by `graphics`. However, if the command doesn't work or doesn't exists, VMD tries to find an extension draw function.

This is done by searching for a function of the form `vmd_draw_$command`. If the function exists, it is called with the first parameter the molecule index and the rest are the arguments from the original `draw` call.

Here's an example which extends the draw command to include an "arrow" primitive.

```
proc vmd_draw_arrow {mol start end} {
    # an arrow is made of a cylinder and a cone
    set middle [vecadd $start [vecscale 0.9 [vecsub $end $start]]]
    graphics $mol cylinder $start $middle radius 0.15
    graphics $mol cone $middle $end radius 0.25
}
```

and here's one which adds the "color" option to the "cylinder primitive. It works because the draw command tries the "graphics" command and fails, because the "color" option doesn't exist. At that point, the extensions are checked.

```
proc vmd_draw_cylinder {mol start end args} {
    if [expr [llength $args] % 2] {
        error "draw: cylinder: incorrect argument list"
    }
    # start the construction of the graphics command
    set opts [list graphics $mol cylinder]
    lappend opts $start $end
    # search for the "color" option
    while {[string compare $args {}]} {
        # get the parameter and value
        set a [lvarpop args]
        set b [lvarpop args]
        # check if it is the color command
        if [string compare $a "color"] {
```

```
            # if it isn't, save the options
    lappend opts $a $b
} else {
            # otherwise, save the color
    set color $b
        }
    }
    # call the graphics commands
    graphics $mol color $color
    eval $opts
}
```

Chapter 14

Molecular information: molinfo and atomselect

This Chapter covers how to extract information about molecules and atoms using the VMD text commands `molinfo` and `atomselect`.

14.1 molinfo

The `molinfo` command is used to get information about a molecule (or loaded file) including the number of loaded atoms, the filename, the graphics selections, and the viewing matrices. It can also be used to return information about the list of loaded molecules.

14.1.1 Using molinfo to access the molecule list

The molecule list contains information about all the loaded molecules, including those which are not properly called molecules, such as a Raster3D file or "graphics" molecule (see section §13). Each molecule has a unique id, which is assigned to it when it is first loaded. These start at zero and increase by 1 for each new molecule. When a molecule is deleted, the number is not used again. There is one unique molecule, called the `top` molecule `top` molecule [§3.3.2], which is used to determine some parameters, such as the center of view, the data in the animation form, etc.

The list of available molecule ids is available with the command `molinfo list`, and the number of loaded molecules is found with `molinfo num`. The command `molinfo top` returns the id of the `top` molecule. One other command, `molinfo <num> get index`, returns the num'th molecule index on the list, starting from 0. (See section §11.3.15). In all cases, a molecule id of -1 is returned when no other valid id exists.

```
Examples:
vmd > molinfo list
Info) 0 1 2
vmd > molinfo top
Info) 2
vmd > mol top 1
vmd > molinfo top
Info) 1
vmd > molinfo num
```

```
Info) 3
vmd > mol delete 1
Info) Deleted 1 molecules.
vmd > molinfo list
Info) 0 2
vmd > molinfo top
Info) 2
vmd > molinfo index 1
Info) 2
```

14.1.2 Using `molinfo` to access information about a molecule

The `molinfo` command can also be used to access and, in some cases, modify information specific to a given molecule. A query is in the form:

- **molinfo** *molecule_id* **get** {*list of keywords*}

and the result is a list of elements, one for each keyword.

```
Examples:
vmd > molinfo top get numatoms
Info) 568
vmd > molinfo 0 get {source filename}
Info) File /home/dalke/pdb/pti.pdb
```

This next example is a bit more complicated. It loops through all the graphical representation (`numreps`) and, for each one, gets the representation used (`rep`), the selection text (`selection`), and the coloring method (`color`).

```
vmd > for {set i 0} {$i < [molinfo top get numreps]} {incr i} {
? lassign [molinfo top get "{rep $i} {selection $i} {color $i}"] a b c
? puts "view $i:"
? puts " representation: $a"
? puts " selection: $b"
? puts " coloring method: $c"
? }
view 0:
 representation: Tube 0.300000 8.000000
 selection: protein backbone
 coloring: method Structure
view 1:
 representation: Lines 2.000000
 selection: same residue as name "S.*"
 coloring: method ResName
view 2:
 representation: VDW 1.000000 6.000000
 selection: name "S.*"
 coloring: method Name
```

A similar behavior can be achieved by using `mol list top` , in which case, however, there is no direct access to the data. A complete list of keywords is given in Table 14.1.2.

The `molinfo` command, contrary to its name, can also be used to set some keyword values, such as the current frame number and the display state flags. This duplicates some of the functionality of the `mol` command, though there are distinct differences in the implementation. Specifically, the `mol` command uses the internal command queue which, among other things, notifies the appropriate forms that a change occured, redraws the graphics, and logs the commands to the log file, of logging is enabled. In future versions of VMD there will be only one command; for now we suggest only using the `molinfo` command to get information and to set the frame value and the various viewing matrices.

Examples:

Two functions, one to save the current view position, the other to restore it. The position of the axis is not changed by these operations.

```
proc save_viewpoint {} {
   global viewpoints
   if [info exists viewpoints] {unset viewpoints}
   # get the current matricies
   foreach mol [molinfo list] {
      set viewpoints($mol) [molinfo $mol get {
center_matrix rotate_matrix scale_matrix global_matrix}]
   }
}
proc restore_viewpoint {} {
   global viewpoints
   foreach mol [molinfo list] {
      puts "Trying $mol"
      if [info exists viewpoints($mol)] {
         molinfo $mol set {center_matrix rotate_matrix scale_matrix
   global_matrix} $viewpoints($mol)
      }
   }
}
```

Cycle through the list of displayed molecules, turning each one on one at a time. At the end, return the display flags to their original state.

```
# save the current display state
foreach mol [molinfo list] {
  set disp($mol) [molinfo $mol get drawn]
}
# turn everything off
mol off all
# turn each molecule on then off again
foreach mol [molinfo list] {
  if $disp($mol) {
     mol on $mol
     sleep 1
```

```
    mol off $mol
  }
}
# turn the original ones back on
foreach mol [molinfo list] {
  if $disp($mol) {mol on $mol }
}
```

The last loop, which turns the originally drawn molecules back on, doesn't turn them on at the same time. That's because some commands (those which use the command queue) redraw the graphics when they are used. This can be disabled with the `display update` (see section § 11.3.6 for more information). Using this, the final loop becomes

```
#turn the original ones back on
display update off
foreach mol [molinfo list] {
  if $disp($mol) {mol on $mol }
}
display update on
```

Alternatively, since the `drawn` option is settable, you could do:

```
foreach mol [molinfo list] {
  if $disp($mol) {molinfo $mol set drawn 1}
}
```

However, that won't set the flag to redraw the scene so you need to force a redraw with `display redraw`.

14.2 Atom information

Atom selection is the primary method to access information about the atoms in a molecule. It works in two steps. The first step is to create a selection given the selection text, molecule id, and optional frame number. This is done by a function called `atomselect`, which returns the name of the new atom selection. the second step is to use the created selection to access the information about the atoms in the selections.

Atom selection is implemented as a Tcl function. The data returned from `atomselect` is the name of the function to use. The name is of the form `atomselect%d` where '%d' is a non-negative number (such as 'atomselect0', atomselect26', ...).

To clear up a few points (even though this is covered elsewhere), the "molecule identifier" is a non-negative integer which uniquely identifies one of the loaded molecules. Each molecule gets a new number when it is loaded, and the numbers are generated in numeric order. Thus, the first molecule loaded has a molecule id of 0, the second of 1, and so on. To see the molecule id, open the "Molecule" window and look for the number to the right of the colon (":"). This has been padded with zeros to make it a 3 digit number, so molecule id 0 is expressed as 000. The id can also be the word "top", which indicates the top molecule.

The "selection text" is the same as the selection language used in the Graphics form [§ 3.3.4] and described in Chapter §7. It is used to pick a given subset of the atom. The text cannot be changed once a selection is made. Some of the terms in the selection depend on data that change during a trajectory (so far only the keywords 'x', 'y', and 'z' can change over time). For these, the optional 'frame value' is used to determine which specific frame to use. The frame number can be a non-negative integer, the word "now" (the current frame), the word "first" (for frame 0) and "last" (for the last frame).

In terms of usage, the form of the atom selection command is:

- **atomselect** *molecule_id selection_text* [**frame** *frame_number*]

Some examples are:

```
vmd> atomselect top "name CA"
atomselect0
vmd> atomselect 3 "resid 25" frame last
atomselect1
vmd> atomselect top "within 5 of resname LYR" frame 23
atomselect2
```

The name returned is a function name, and you can use it like any other functions. There are several parameters to these, as shown in the following:

```
# Number of atoms in the selection
vmd> atomselect1 num
13
# The selection text used to create the selection
vmd> atomselect1 text
{resid 25}
# list of atom indicies in the selection
vmd> atomselect0 list
0 5 11 17 23 29 35 41 47 53 59 65
```

The "resid 25" is in braces because it is one element of a Tcl list. This is an important point to notice, for reasons to be discussed after multiple attributes are introduced.

The way to use the function created by the `atomselect` command is to store the name into a variable, then use the variable to get the name when needed.

```
vmd> set sel [atomselect top "water"]
atomselect3
vmd> $sel text
water
```

This is equivalent to saying

```
vmd> atomselect3 text
```

The easiest way of thinking about this is that the `atomselect` command creates an object. To get information from the object you have to send it a command. Thus, in the example above (`atomselect1 num`) the object "atomselect1" was sent the command "num", which asks the object to return the number of atoms in the selection.

These derived object functions (the ones with names like atomselect3) take many options. They are listed in Table 14.2.

You already saw some examples of the first few of these commands. The second half of the commands allows you to get or set atom information. The information that can be returned are the same keywords available in the selection command (eg, "resid", "x", "segname") as well as the boolean words like "water" and "backbone" (these are returned as 1 if true and 0 if false).

For instance, given the selection

```
vmd> set sel [atomselect top "resid 4"]
atomselect4
```

you can get the atom names for each of the atoms in the selection with

```
vmd> $sel get name
{N} {H} {CA} {CB} {C} {O}
```

(which, remember, is the same as

```
vmd> atomselect4 get name
```

)

Multiple attributes can be requested by submitting a list, so if you want to see which atoms are on the backbone,

```
vmd> $sel get {name backbone}
{N 1} {H 0} {CA 1} {CB 0} {C 1} {O 1}
```

and the atom coordinates with

```
vmd>  $sel get {x y z}
{0.710000 4.211000 1.093000} {-0.026000 3.700000 0.697000} {0.541000
4.841000 2.388000} {-0.809000 4.462000 2.976000} {1.591000 4.371000
3.381000} {2.212000 5.167000 4.085000}
```

It is very important to note that the information returned by one of these commands is a list of lists. Specifically, it is a list of size n where each element is itself a list of size i, where n is the number of atoms in the selection and i is the number of attributes requested.

One quick function you can build with the coordinates is a method to calculate the geometrical center (not quite the center of mass; that's a bit harder). This also uses some of the vector commands discussed in the section about vectors and matrices [§12], but you should be able to figure them out from context.

```
proc geom_center {selection} {
        # set the geometrical center to 0
        set gc [veczero]
        # [$selection get {x y z}] returns a list of {x y z}
        #    values (one per atoms) so get each term one by one
        foreach coord [$selection get {x y z}] {
           # sum up the coordinates
           set gc [vecadd $gc $coord]
        }
        # and scale by the inverse of the number of atoms
        return [vecscale [expr 1.0 /[$selection num]] $gc]
}
```

With that defined you can say (assuming $sel was created with the previous atomselection example)

```
vmd> geom_center $sel
0.703168 4.45868 2.43667
```

I'll go through the example line by line. The function is named `geom_center` and takes one parameter, the name of the selection. The first line sets the variable "gc" to the zero vector, which is 0 0 0. On the second line of code, two things occur. First, the command

```
$selection get {x y z}
```

is executed, and the string is replaced with the result, which is

```
{0.710000 4.211000 1.093000} {-0.026000 3.700000 0.697000} {0.541000
4.841000 2.388000} {-0.809000 4.462000 2.976000} {1.591000 4.371000
3.381000} {2.212000 5.167000 4.085000}
```

This is a list of 6 terms (one for each atom in the selection), and each term is a list of three elements, the x, y, and z coordinate, in that order.

The "foreach" command splits the list into its six terms and goes down the list term by term, setting the variable "coord" to each successive term. Inside the loop, the value of $coord is added to total sum.

The last line returns the geometrical center of the atoms in the selection. Since the geometrical center is defined as the sum of the coordinate vectors divided by the number of elements, and so far I have only calculated the sum of vectors, I need the inverse of the number of elements, which is done with the expression

```
expr 1.0 / [$selection num]
```

The decimal in "1.0" is important since otherwise Tcl does integer division. Finally, this value is used to scale the sum of the coordinate vectors (with vecscale), which returns the new value, which is itself returned as the result of the procedure.

The center of mass function is slightly harder because you have to get the mass as well as the x, y, z values, then break that up into to components. The formula for the center of mass is $\sum m_i x_i / \sum mass_i$

```
proc center_of_mass {selection} {
        # some error checking
        if {[$selection num] <= 0} {
                error "center_of_mass: needs a selection with atoms"
        }
        # set the center of mass to 0
        set com [veczero]
        # set the total mass to 0
        set mass 0
        # [$selection get {x y z}] returns the coordinates {x y z}
        # [$selection get {mass}] returns the masses
        # so the following says "for each pair of {coordinates} and masses,
#  do the computation ..."
```

```
        foreach coord [$selection get {x y z}] m [$selection get mass] {
           # sum of the masses
           set mass [expr $mass + $m]
           # sum up the product of mass and coordinate
           set com [vecadd $com [vecscale $m $coord]]
        }
        # and scale by the inverse of the number of atoms
        if {$mass == 0} {
                error "center_of_mass: total mass is zero"
        }
        # The "1.0" can't be "1", since otherwise integer division is done
        return [vecscale [expr 1.0/$mass] $com]
}

vmd> center_of_mass $sel
Info) 0.912778 4.61792 2.78021
```

The opposite of "get" is "set". A few of the keywords (most notably, "x", "y", and "z") can be set to new values. This allows, for instance, atom coordinates to be changed, the occupancy values to be updated, or user forces to be added. Right now the only things that can be changed are the coordinates, the beta and occupancy values and user forces.

```
  set sel [atomselect top "index 5"]
  $sel get {x y z}
{1.450000 0.000000 0.000000}
  $set set {x y z} {{1.6 0 0}}
```

Note that just as the get option returned a list of lists, the set option needs a list of lists, which is why the extra set of curly braces were need. Again, this must be a list of size n containing elements which are a list of size i. The exeception is if n is 1, the list is duplicated enough times so there is one element for each atom.

```
  # get two atoms and set their coordinates
  set sel [atomselect top "index 6 7"]
  $sel set {x y z} { {5 0 0} {7.6 5.4 3.2} }
```

In this case, the atom with index 6 gets its (x, y, z) values set to 5 0 0 and the atom with index 7 has its coordinates changed to 7.6 5.4 3.2.

It is possible to move atoms this way by getting the coordinates, changing them (say by adding some offset) and replacing it. Following is a function which will do just that:

```
proc moveby {sel offset} {
  set newcoords {}
  foreach coord [$sel get {x y z}] {
    lvarpush newcoords [vecadd $coord $offset]
  }
  $sel set $newcoords
}
```

And to use this function (in this case, to apply an offset of (x y z) = (0.1 -2.8 9) to the selection "$movesel"):

```
moveby $movesel {0.1 -2.8 9}
```

However, to simplify matters some options have been added to the selection to deal with movements (these commands are also implemented in C++ and are much faster than the Tcl versions). These functions are `moveby`, `moveto`, and `move`. The first two take a position vector and the last takes a transformation matrix.

The first command, `moveby`, moves each of the atoms in the selection over by the given vector offset.

```
$sel moveby {1 -1 3.4}
```

The second, `moveto`, moves all the atoms in a selection to a given coordinate (it would be strange to use this for a selection of more than one atom, but that's allowed). Example:

```
$sel moveto {-1 1 4.3}
```

The last of those, `move`, applies the given transformation matrix to each of the atom coordinates. This is best used for rotating a set of atoms around a given axis, as in

```
$sel move [trans x by 90]
```

which rotates the selection 90 degrees about the x axis. Of course, any transformation matrix may be used.

A more useful example is the following, which rotates the side chain atoms around the CA-CB bond by 10 degrees.

```
# get the sidechain atoms (CB and onwards)
set sidechain [atomselect top "sidechain residue 22"]
# get the CA coordinates -- could do next two on one line ...
set CA [atomselect top "name CA and residue 22"]
set CAcoord [lindex [$CA get {x y z}] 0]
# and get the CB coordinates
set CB [atomselect top "name CB and residue 22"]
set CBcoord [lindex [$CB get {x y z}] 0]
# apply a transform of 10 degrees about the given bond axis
$sidechain move [trans bond $CAcoord $CBcoord 10 deg]
```

14.3 Analysis scripts

Following are some more examples of routines that could be used for analysing molecules. These are not the best routines to used since many of these are implemented with the `measure` command, which calls a much faster built-in function.

Get the total mass given a selection.

```
proc total_mass {selection} {
  set sum 0
  foreach mass [$selection get mass] {
```

```
    set sum [expr $sum + $mass]
  }
  return $sum
}
```

Here's another (slower) way to do the same thing. This works because the mass returned from the selection is a list of lists. Putting it inside the quotes of the eval makes it a sequence of vectors, so the vecadd command will work on it.

```
proc total_mass1 {selection} {
  set mass [$selection get mass]
  eval "vecadd $mass"
}
```

Find the min and max coordinate values of a given molecule in the x, y, and z directions (see also the `measure` command 'minmax'). The function takes the molecule id and returns two vectors; the first contains the min values and the second contains the max.

```
proc minmax {molid} {
  set sel [atomselect $molid all]
  set coords [$sel get {x y z}]
  set coord [lvarpop coords]
  lassign $coord minx miny minz
  lassign $coord maxx maxy maxz
  foreach coord $coords {
    lassign $coord x y z
    if {$x < $minx} {set minx $x} else {if {$x > $maxx} {set maxx $x}}
    if {$y < $miny} {set miny $y} else {if {$y > $maxy} {set maxy $y}}
    if {$z < $minz} {set minz $z} else {if {$z > $maxz} {set maxz $z}}
  }
  return [list [list $minx $miny $minz] [list $maxx $maxy $maxz]]
}
```

Compute the radius of gyration for a selection (see also `measure rgyr`). The square of the radius of gyration is defined as $\sum_i m_i(\vec{r}_i - \vec{r}_c)^2 / \sum_i m_i$. This uses the center_of_mass function defined earlier in this chapter; a faster version would replace that with `measure center`.

```
proc gyr_radius {sel} {
  # make sure this is a proper selection and has atoms
  if {[$sel num] <= 0} {
    error "gyr_radius: must have at least one atom in selection"
  }
  # gyration is sqrt( sum((r(i) - r(center_of_mass))^2) / N)
  set com [center_of_mass $sel]
  set sum 0
  foreach coord [$sel get {x y z}] {
    set sum [vecadd $sum [veclength2 [vecsub $coord $com]]]
  }
  return [expr sqrt($sum / ([$sel num] + 0.0))]
}
```

Applying this to the alanin.pdb coordinate file

```
vmd > mol load pdb alanin.pdb
vmd > set sel [atomselect top all]
vmd > gyr_radius $sel
Info) 5.45443
```

Compute the rms difference of a selection between two frames of a trajectory. This takes a selection and the values of the two frames to compare.

```
proc frame_rmsd {selection frame1 frame2} {
  set mol [$selection molindex]
  # check the range
  set num [molinfo $mol get numframes]
  if {$frame1 < 0 || $frame1 >= $num || $frame2 < 0 || $frame2 >= $num} {
    error "frame_rmsd: frame number out of range"
  }
  # get the first coordinate set
  set sel1 [atomselect $mol [$selection text] frame $frame1]
  set coords1 [$sel1 get {x y z}]
  # get the second coordinate set
  set sel2 [atomselect $mol [$selection text] frame $frame2]
  set coords2 [$sel2 get {x y z}]
  # and compute the rmsd values
  set rmsd 0
  foreach coord1 $coords1 coord2 $coords2 {
    set rmsd [expr $rmsd + [veclength2 [vecsub $coord2 $coord1]]]
  }
  # divide by the number of atoms and return the result
  return [expr $rmsd / ([$selection num] + 0.0)]
}
```

The following uses the frame_rmsd function to list the rmsd of the molecule over the whole trajectory, as compared to the first frame.

```
vmd > mol load psf alanin.psf dcd alanin.dcd
vmd > set sel [atomselect top all]
vmd > for {set i 0} {$i < [molinfo top get numframes]} {incr i} {
?   puts [list $i [frame_rmsd $sel $i 0]]
? }
0 0.0
1 0.100078
2 0.291405
3 0.523673
 ....
97 20.0095
98 21.0495
99 21.5747
```

The last example shows how to set the beta field. This is useful because one of the coloring methods is 'Beta', which uses the beta values to color the molecule according to the current color scale. (This can also be done with the occupancy field.) Thus redefining the beta values allows you to color the molecules based on your own definition. One useful example is to color the molecule based on the distance from a specific point (for this case, coloring a poliovirus protomer based on its distance to the center of the virus (0, 0, 0) helps bring out the surface features).

```
proc betacolor_distance {$sel point} {
  set newbeta {}
  # get the coordinates
  foreach coord [$sel get {x y z}] {
   # get the distance and put it in the "newbeta" list
    set dist [veclength2 [vecsub $coord $point]]
    lvarpush newbeta $dist
  }
  # set the beta term
  $sel set beta $newbeta
}
```

And here's one way to use it:

```
# load pdb2plv.ent using anonymous ftp to the PDB
# (this assumes the url_get program was set up correctly (see ...)
vmd > mol pdbload 2plv
vmd > set sel [atomselect top all]
vmd > betacolor_distance $sel {0 0 0}
```

Then go to the graphics menu and set the 'Coloring Method' to 'Beta'.

Keyword	Aliases	Arg	Set	Description
id		*int*	N	molecular id
index		*int*	N	index on the molecule list
numatoms		*int*	N	number of atoms
source		*str*	N	one of File, Graphics, Remote, or Sigma
name		*str*	N	the name of the molecule (usually the name of the file)
filename		*str*	N	full filename (if two files, the topology file)
filetype		*str*	N	corresponding file type (PSF, PDB, XYZ, ...)
filename2		*str*	N	full filename of coordinate file (if two files)
filetype2		*str*	N	corresponding file type (DCD or PDB)
active		*bool*	Y	is/make the molecule active
drawn	displayed	*bool*	Y	is/make the molecule drawn
fixed		*bool*	Y	is/make the molecule fixed
top		*bool*	Y	is/make the molecule top
center		*vector*	Y	get/set the coordinate used as the center
center_matrix		*matrix*	Y	get/set the centering matrix
rotate_matrix		*matrix*	Y	get/set the rotation matrix
scale_matrix		*matrix*	Y	get/set the scaling matrix
global_matrix		*matrix*	Y	get/set the global (rotation/scaling) matrix
view_matrix		*matrix*	N	get/set the overall viewing matrix
numreps		*int*	N	the number of representations
selection *i*		*string*	N	the string for the i'th selection
rep *i*		*string*	N	the string for the i'th representation
color *i*	colour	*string*	N	the string for the i'th coloring method
numframes		*int*	N	number of animation frames
frame		*int*	Y	current frame number
remote		*boolean*	N	is this a remote molecule (via MDComm)
bond		*float*	N	the bond energy (for the current frame)
angle		*float*	N	the angle energy
dihedral		*float*	N	the dihedral energy
improper		*float*	N	the improper energy
vdw		*float*	N	the van der Waal energy
electrostatic	elec	*float*	N	the electrostatic energy
hbond		*float*	N	the hydrogen bond energy
kinetic		*float*	N	the total kinetic energy
potential		*float*	N	the total potential energy
energy		*float*	N	the total energy
temperature	temp	*float*	N	the overall temperature

Table 14.1: `molinfo` keywords

Option	Description
num	return the number of atoms in the selection
list	return a list of the atom indicies in the selection (BTW, this is the same as "get index")
text	return the text used to create this selection
molid	returns the molecule id used to create this
type	returns the string 'atomselect'
delete	delete this object (removes the function)
global	moves the object into the global namespacee
uplevel	moves the object to another level
get	given a list of attributes, return the list containing the list of attribute values (see examples)
set	the complimentary functions (currently only works for a few terms)
move	move the selection by a a 4x4 tranformation matrix
moveby	move all the atoms by a given offset
lmoveby	move each atom by an offset given in the list
moveto	move all the atoms to a given location
lmoveto	move each atom to a point given by the appropriate list element
writepdb	write the selected atoms in a pdb file

Table 14.2: `atomselect` keywords

Chapter 15

Tips and Tricks

This chapter is a "how to" collection of some of the various ways to use VMD in different situations.

15.1 Customizing the Popup Menu and the Hot Keys

When the mouse is in the graphics display window, you can perform commands by selecting items from the VMD popup menu (accessed by pressing the right mouse button), and by using keyboard *hot keys*. This section describes how to customize the popup menu and hot keys to your individual preference.

15.1.1 Customizing the popup menu

The popup menu in VMD has a number of default menu items and submenus, which cannot be changed. These are described in an earlier section § 3.1. But you can add new submenus and items to this menu, and assign text commands to those items. When the items are selected by the user, the assigned text command will be executed as if it had been typed in at the console. In fact, with Tcl you can assign arbitrarily complex commands (such as scripts) to a single menu selection.
To add a menu to the main pop-up menu, use:

- **user add menu** *name*

where *name* is the name of the menu entry as it will appear in the Popup menu.
For example, lets suppose we want to make a menu of our most often used structures. Then the command to accomplish this is:

```
user add menu Favorites
```

Since there isn't anything in the menu, it doesn't appear in the popup menu yet. To add items to the menu, use:

- **user add subitem** *menu name command*

where *command* can either be a list of Tcl commands to execute or the name of a script that is to be processed when the menu item is chosen. For instance, we often load the structures "alanin.pdb" and "brH.pdb" and remotely download the 9pti structure from the PDB FTP site, so we'll add:

```
user add subitem Favorites BPTI {mol pdbload 9pti}
user add subitem Favorites alanin {mol load pdb alanin.pdb}
user add subitem Favorites bR {mol load pdb brH.pdb}
```

Enter these three lines and then bring up the popup menu. You should see the new menu item just above the "Help" lines. Unless you also happen to have these molecules available, the last two items probably won't work, but the first one should. Try it out.

You can also add a separator in the menu, using:

- **user add subseparator** *menu*

So if you wanted a separation between the first three molecules and a fourth, say the poliovirus 2PLV structure, then you would type:

```
user add subseparator Favorites
user add subitem Favorites Polio {mol pdbload 2plv}
```

As a second complete example, the code below will produce a cascading menu which allows you to quickly adjust the degree of transparency for transparenent objets (i.e. the default is .3, where 1.0 is opaque and 0.0 is invisible).

```
# Set the alpha value of the "transparent" materials
proc set_transparency_value {alpha} {
        display update off
        # change the opaquicity of the transparent colors
        for {set color 17} {$color < 34} {incr color} {
                color change alpha $color $alpha
        }
        # and the transparent color scale (66 to 97)
        for {set color 66} {$color < 98} {incr color} {
                color change alpha $color $alpha
        }
        display update on
}
user add menu Transparency
for {set i 1.0} {$i >= 0.0} {set i [expr $i - 0.1]} {
        eval "user add subitem Transparency $i {
                set_transparency_value $i}"
}
```

Unfortunately, the user modifiable popup menus are not hierarchical. There is no way to add a submenu to one of your own menus. They can only be added to the main popup menu. However, there is another place to add new menu items, the 'User Commands' submenu of the main menu. "Where is it?" you ask? It doesn't contain anything, so it isn't displayed. You can add items to it just as you can to the main menu, but the commands are slightly different. Admittedly this is not an optimal solution. Future versions of VMD will use Tk menus to alleviate this problem. Until then though, here is a specification of the syntax.
To add an item to the 'User Commands' menu:

- **user add item** *name command*

To add a separator to the 'User Commands' menu:

- **user add separator**

The usage is the same as their related commands, so we won't give a detailed example, but to see how it works, try this:

```
# Turn off the axis
user add item {Axes Off} axes location off
user add item {Axes On} axes location LowerLeft
# a break
user add separator
# delete all the loaded molecule
user add item {Zap Everything} {
        foreach molecule [molinfo list] {
                mol delete $molecule
        }
}
```

Finally, if you wish to see the current definitions of the user-customized menu items, use the command

- **user print menus**

This will print out to the console the names and associated text commands for all menu items added by the user.

15.1.2 Customizing the Hot Keys

In a similar fashion, VMD keeps a list of user hot keys, which when pressed will result in an associated text command being executed as if it had been typed at the console. There are a number of predefined hot keys, as listed in tables 3.1, 3.2, 3.3, and 3.4. The current state of these can be printed out with the command

- **user print keys**

And in the same fashion as for the popup menu, the commands attached to these hot keys can be customized by the user.

To add/modify a hot key, use the command

- **user add key** *key command*

key must be a single character. When that key is pressed while the mouse cursor is in the graphics display window, the associated command will be executed.

15.1.3 Automatically loading customization commands

Once you have discovered a set of commands which are particularly useful and familiar for you, you will want to have these popup menu commands and hot key commands automatically available every time you run VMD. This can be done by placing the commands to add these items in your `.vmdrc` file, which is a file containing VMD text commands that is executed every time VMD starts up. The basic method for setting up this file is described in section §17.3.4. Once you have such a file, just put the `user add` commands in it, and you will have your own specialized version of VMD.

15.2 Using VMD as a WWW Client (for chemical/* documents)

Mosaic, Netscape, and possibly other browsers can be configured to use VMD as a helper application for viewing some `chemical/*` documents.

15.2.1 MIME types

When a web browser receives a document from a server it actually gets two pieces of information: the header and the body. The header contains information about the message and body. One of the most important pieces of data, called the *MIME type* specifies what the body of text describes. For instance, a GIF image is given the MIME type of `image/gif`, a JPEG image is `image/jpg`, and postscript is `application/postscript`. A class of types, `chemical/*`, has been created for chemical models so the MIME type for PDB files is `chemical/pdb`, for XYZ is `chemical/xyz`, etc.

Helper Applications

The web browser uses the MIME type to determine how to view the body of the message. Some of the documents are viewed by the browser itself, like `text/html` which describes HTML documents. In other cases, the browser has to start up another application. From here on, we'll describe how Mosaic and Netscape do this. First, it saves the incoming message body to a temporary file. It then scans the global and local *mailcap* files to determine which application is used to view the given MIME type. The application, which must take a file name on the command line, is then executed. When the application exits, the temporary file is deleted.

15.2.2 Setting up your `.mailcap`

In the VMD installation directory ($VMDDIR/scripts/vmd/) there is a perl script called chemical2vmd which will create a VMD command file. Since

then start VMD with the -e command line option to read that file. The file contains the necessary commands to convert

It is also possible to install the previous script in the global `.mailcap` file to make it accessible to everyone. You will have to consult the documentation for your web browser(s) to find out how.

15.2.3 Example sites

Some web sites that send `chemical/pdb` types are the Protein Data Bank at `http://www.rcsb.org/` and "Molecules R US" at `http://www.nih.gov/htbin/pdb`.

15.3 Making a Movie

It is possible to make movies with VMD, though the interface is not well developed. This section is not written for the casual user!

The following Tcl script uses the mpeg encoder available from ftp.cs.berkeley.edu, the image converter `toppm`, and successive Raster3D runs (though it is possible to do screen grabs as well) to make a movie of a spinning molecule. The script rotates the system 35 times by 10 degrees each time. For each orientation, it saves the image as a Raster3D input file, runs Raster3D on it to get an RGB raster image, then converts the RGB file to a ppm graphics file for use by the mpeg encoder. Once all the files are made, the mpeg is created. The temporary files are saved in

`./images` and the full process could take up a lot of disk space, depending on the size of the VMD graphics window. This script does not automatically delete the files or the directory.

The encoder input file is given after the Tcl script.

VMD script to make a movie of a rotating system

```
for { set i 0 } { $i < 360 } { set i [ expr $i + 1 ] } {
  render Raster3D out.r3d
  catch { exec render < out.r3d -sgi out.rgb }
# get the right format for the number (only works up to 99)
  if { $i < 1 } {
    set nm "00"
  } elseif { $i < 10 } {
    set nm "0$i"
  } else {
    set nm $i
  }
# convert from RGB to PPM
  catch { exec toppm out.rgb $nm.ppm }
# delete the RGB file
  catch { exec rm out.r3d out.rgb }
# and rotate by 10 degrees
  rot y by 10
}
# now make the mpeg
catch { exec mpeg_encode-1.3.sgi.bin mpeg.input }
```

The mpeg encoder script follows, but be careful as we guessed at most of the values.

```
PATTERN I
OUTPUT spin.mpg
INPUT_DIR .
INPUT
*.ppm [00-89]
END_INPUT
BASE_FILE_FORMAT PPM
INPUT_CONVERT cat *
GOP_SIZE 4
SLICES_PER_FRAME 2
PIXEL FULL
RANGE 5
PSEARCH_ALG LOGARITHMIC
BSEARCH_ALG SIMPLE
IQSCALE 1
PQSCALE 1
BQSCALE 1
REFERENCE_FRAME DECODED
```

You would use this by first loading a molecule, changing to the directory you wish to use to store the image files, and entering the command

```
play <Tcl script filename>
```

It does work. Honest.

15.4 Coloring Trick - Override a Coloring Category

There is currently no user-defined coloring method. This makes it hard to color residues by property "X" if X is not already defined in VMD. It is possible to get around this limitation somewhat by overriding one of the values in the PDB or PSF. For instance, suppose you wanted to color the atoms by the distance of the atom from a given point. One way is to compute the distance and put it in either the occupancy or beta field of the PDB file. Then when the molecule is colored by occupancy it is actually coloring by distance.

You could also override, say, the segment name field or even the residue name. Don't override the atom name unless you are really desperate as VMD uses it to determine which residues are proteins and nucleic acids, and hence which residues can be drawn as a tube or ribbon.

15.5 Some Nice Represenations

The following views are quite nice for displaying proteins and nucleic acids:

```
selection: all
drawing method: tube
coloring method: segname (or chain)
why?  This show the backbone of the protein and nucleic acid strands

selection: protein and (name CA or not backbone)
drawing method: lines
coloring method: segname (or chain)
why? shows where the side chains are located, but they are thin so the
  backbone is still visible and the scene is quickly drawn

selection: (numbonds = 0) and not waters
drawing method: vdw
coloring method: name
why? shows ions.  The "not waters" omits cases where a water's oxygen is
  known but not the hydrogen.

selection: not (waters or protein or nucleic)
drawing method: lines
coloring method: name
why? shows whatever is left; usually ligands and crystallizing agents
```

15.6 Finding Contact Residues

Suppose you want to view the atoms in "A" which are in contact with "B". Use the `within <distance> of <selection>` selection command. For purposes of demonstration, let A be protein, B be nucleic, and define contact as an atom in A which is within 2 Å of an atom in B. Then the selection command is

```
protein within 2 of nucleic
```

If you want to see all the residues of A which have at least one atom in contact with B, use

```
same residue as (protein within 2 of nucleic)
```

15.7 Tcl Logging

Every issued command which changes the state of VMD (loading a molecule, rotation, opening a form, etc.) can be saved to a file via the log command. In addition, if the Tcl command *vmdlog* exists, it is called with the issued command as its only term. One use for this is to filter out some of the commands.

One practical use of this feature is to filter out the `menu` commands so they don't constantly disappear and reappear on playback. In addition, this adds `wait` command if the time between succesive commands was more than a second so the playback will emulate the timings of original actions.

```
# set things up to record commands to the file ``session_log.vmd''
proc start_recording {} {
     global recording_fileid
     set recording_fileid [open session_log.vmd w]
}

# set up the vmdlog proc

proc vmdlog s {
     if {[regexp {^menu } $s] != 1} {
             global recording_fileid
             global recording_time
             set now [getclock]
             set delay [expr $now - $recording_time]
             if {$delay > 1} {
               puts $recording_fileid "wait $delay"
             }
             puts $recording_fileid $s
             flush $recording_fileid
             set recording_time $now
         }
}

proc stop_recording {} {
     global recording_fileid
     unset recording_fileid -1
     rename vmdlog {}
}
```

15.8 Remote Control of VMD

As part of our research on developing new types of user interfaces for molecular modelling, we have added a simple way for other programs to send information to a VMD process via the Tcl-DP RPC mechanism. (See sections § 3.4.1 and §11.3.9). As an example of use, here's how to control one VMD session from another program.

For now, I'll assume that both programs are on the same computer. Start VMD and execute `external on`. This tells VMD to start listening for commands from one of the machine ports. (For those interested in the details, this function disables acceptance of commands from other machines (`dp_Host -; dp_Host +[exec hostname]`) then starts the server as `dp_MakeRPCServer 8639`.)

SECURITY WARNING: Once the external option is enabled, VMD will allow any program to connect to it and execute a Tcl script as you. By default, VMD limits its connections to other processes on the same machine, so you are reasonably safe if you trust the other users. However, it is easy to modify the settings to allow anyone in the world to connect and do whatever they wanted. It is possible to make the connection more secure, but since VMD is designed for an academic environment where different users trust each other, we have not implemented it. (See the Tcl-DP distribution for documentation of how to do so.)

Start up another program that uses Tcl-DP, like `dpwish` or another instance of VMD. (Remember, you can start VMD without a display with `vmd -dispdev none`.) Execute the command

```
set vmd_server [dp_MakeRPCClient [exec hostname] 8639]
```

This should return a name like "file7" if successful. To send a command (in this case, `rotate x by 90`) to VMD, enter

```
dp_RPC $vmd_server rotate x by 90
```

To simplify matters you might want to implement a new procedure to handle these messages

```
proc tell_vmd {args} {
    global vmd_server
    dp_RPC $vmd_server eval $args
}
tell_vmd rotate x by 90
```

Where the last line is an example of how this procedure is used.

If you want to control VMD on from machine, on the controlling process, replace the `[exec hostname]` with the name of the machine with VMD.

15.9 Controlling One VMD With Another

The previous example showed how to control VMD from another process and the one before that showed how to get access to all the VMD display commmands. These can be combined to have one instance of VMD control another. Here's how.

On the VMD to be controlled (on the machine named `foo`), allow external communications by executing `external on`. On the controlling VMD enter these lines:

```
external connect foo
proc vmdlog s {
      external send $s
}
```

Chapter 16

Advanced Script Writing

The biggest improvement in VMD 1.1 has been the addition of commands for analyzing atomic and molecular information. This chapter describes through examples how to put those commands together to write scripts. We assume at this point you already read through the other chapters which describe the various commands, including:

- how to use Tcl (see Chapter §11)
- which commands are/are not "core" commands (see section §11.3)
- the Tcl vector and matrix extensions (see Chapter §12)
- the atomselect, molinfo, and colinfo commands (see Chapter § 14 and section §11.3.4)

The examples used will not be described in detail, but will only explain some of the more unusual or novel features of the VMD scripting language. The source of the examples can be found in the VMD script library at `http://www.ks.uiuc.edu/Research/vmd/script_library/`.

Many of these scripts use the `trace` command from Tcl. This does not mean all good scripts must use that command, but rather that putting a trace on a variable is a nice way to add an automatic action.

16.1 Drawing a distance matrix

This example creates a distance matrix made of the distance between the CAs of two residues. The only input value is selection used to find the CAs. The distances are stored in the 2D array "dist", which is indexed by the atom indices of the CAs (remember, atom indices start at 0).

The distance matrix is a new graphics molecule. It consists of two triangles for each element of the matrix (to make up a square). The color for each pair is one of the 32 elements in the color scale and is determined so the range of distance values fits linearly between 34 and 66 (excluding 66 itself). The name of the graphics is "CA_distance(n)" where "n" is the molecule id of the selection used to create the matrix.

The first graphics command, "materials off" is to keep the lights from reflecting off the matrix (otherwise there is too much glare). At the end, the corners of the matrix are labeled in yellow with the resid values of the CAs.

One extra procedure, "vecdist", is created. This computes the difference between two vectors of length 3. It is not as general as the normal method (`vecnorm [vecsub $v1 $v2]`) but is almost twice as fast, which speeds up the overall subroutine by almost 10%. The script is not very fast; after all, for a 234 residue protein, 27495 distance calculations are made and 54756*2 triangles generated. Nearly all of that is done in Tcl. In terms of times, about 1/3 is spent in the distance calculations, another 1/3 in the math to make the triangles, and another 1/3 in the three "graphics" calls. The residue 234 example protein took 70 seconds to do everything on a Crimson.

Example usage:

```
mol load pdb $env(VMDDIR)/proteins/alanin.pdb
set sel [atomselect top all]
ca_distance $sel

# Input: two vectors of length three (v1 and v2)
# Returns: ||v2-v1||
proc vecdist {v1 v2} {
    lassign $v1 x1 x2 x3
    lassign $v2 y1 y2 y3
    return [expr sqrt(pow($x1-$y1,2) + pow($x2-$y2,2) + pow($x3-$y3, 2))]
}


\index{atom!coordinates}
\index{atomselect!command}
# Input: a selection
# Does: finds the CAs in the selection then computes and draws the
#   CA-CA distance grid with colors based on the color scale
# Returns: the id of the new graphics molecule containing the grid
proc ca_distance {main_sel} {
    # get the CA atoms from the selection
    set mol [$main_sel molindex]
    set sel [atomselect $mol "@$main_sel and name CA"]

    # find distances between each pair
    set coords [$sel get {x y z}]
    set max 0
    set list2 [$sel list]

    foreach atom1 $coords id1 [$sel list] {
        foreach atom2 $coords id2 $list2 {
            set dist($id1,$id2) [vecdist $atom2 $atom1]
            set dist($id2,$id1) $dist($id1,$id2)
            set max [max $max $dist($id1,$id2)]
        }
        lvarpop list2
        lvarpop coords
    }
```

```
# draw the pretty graphic
puts "Distances calculated, now drawing the distance map ..."
mol load graphics "CA_distance($mol)"
set gmol [molinfo top]
# turn material characteristics off
graphics $gmol materials off
# i1 and j1 are i+1 and j+1; this speeds up construction of x{01}{01}
set i 0
set i1 1
# preset the scaling factor (31.95 instead of 32 ensures there will be
# no color 66 (for the max color), which is transparent)
set scale [expr 31.95 / ($max + 0.)]
set list2 [$sel list]
foreach id1 [$sel list] {
   set j 0
   set j1 1
   set x00 "$i $j 0"
   set x10 "$i1 $j 0"
   set x11 "$i1 $j1 0"
   set x01 "$i $j1 0"
   foreach id2 $list2 {
      set col [expr int($scale * $dist($id1,$id2)) + 34]
      graphics $gmol color $col
      graphics $gmol triangle $x00 $x10 $x11
      graphics $gmol triangle $x00 $x11 $x01
      incr j
      incr j1
      set x00 $x01
      set x10 $x11
      set x11 "$i1 $j1 0"
      set x01 "$i $j1 0"
   }
   incr i
   incr i1
}
# put some numbers down to give an idea of what is where
set resids [$sel get resid]
set num [llength $resids]
set start [lindex $resids 0]
set end [lindex $resids [expr $num - 1]]
graphics $gmol color yellow
graphics $gmol text "0 0 0" "$start,$start"
graphics $gmol text "$num 0 0" "$end,$start"
graphics $gmol text "0 $num 0" "$start,$end"
graphics $gmol text "$num $num 0" "$end,$end"
return $gmol
```

```
}
```

16.2 Analysis of PDB files

The VMD atom selection commands were prototypes in two in-house programs developed previously, `pdblang`, which showed the need for an easy-to-use language for manipulating structures, and `parse` which tested the usefulness of Tcl for analyzing large numbers of structures.

Specifically, the goal of the second project was to find what features were needed to write a script to analyze the propensity of various residues to be located near metal ions. Such a script would need to do the following:

- given a list of representative PDB files, get them from the PDB
- find if a metal ion is present
- find the residues within 3, 5, and 7 Å from the ion
- keep track of the results

The hardest part of the script is determining if a metal ion is present, as it is hard to distinguish between a CA calcium and a CA alpha carbon. That still hasn't been solved, though the method below should work for nearly all cases, except when ions are inadvertently bonded to other atoms. The new PDB definition has a new field for element type, but VMD does not yet recognize it.

There are several uncommon methods used here. They are described in the comment statements. Note that this uses the 'pdbload' command to ftp the structure files directly from the PDB ftp site.

```
# adds the given (floating point) value to the value
# if the value doesn't exist, sets it to 0
# This procedure is used because "incr" fails if the variable doesn't exist
proc myincr {var val} {
  regexp {^[^(]*} $var prefix
  global $prefix
  if {![info exists $var]} {
    set $var 0
  }
  set $var [eval "expr $var + $val"]
}

# given the atom index, find the ions within the given distance
# return
proc find_nearby_residues {index ion distance} {
  set nearby [atomselect top "(within $distance of index $index) \
and not index $index"]

 # I need to count each residue once, but I need to distinguish
 # two successive residues, so using just the residue name is not
 # enough.  "resname residue" is unique and, since atoms on the
```

```
 # same residue have successive indices, the luniq gets just one
 # of them.
 foreach res_pair [luniq [$nearby get {resname residue}]] {
   lassign $res_pair resname
   myincr count($ion,$distance,$resname) 1
 }
}


proc analyze_ion_propensity {pdblist metals} {
  global count
  # get each of the entries from the list of PDB files
  foreach entry $pdblist {
    # load them from the PDB ftp site
    mol pdbload $entry
    # go through the search list of metal names
    foreach ion $metals {
      set sel [atomselect top "name $ion and numbonds == 0"]
      foreach atom [$sel list] {
        # find neighbors for each of the test ranges
        find_nearby_residues $atom $ion 3
        find_nearby_residues $atom $ion 5
        find_nearby_residues $atom $ion 7
      }
      # save memory space by forcing the deletion of the
      # temporary selection.  (Otherwise they wouldn't be purged
      # until the end of the procedure.)
      $sel delete
    }
    mol delete top
  }
  # the array ``count'' contains the data in the form
  # (ion name,distance,residue name)
  # For now just print the values for the normal residues within
  # 7A of a Zn.  Use a histogram of '*'
  set resnames {ALA ARG ASN ASP CYS GLN GLU GLY HIS ILE LEU LYS MET \
          PHE PRO SER THR TRP TYR VAL}
  foreach resname $resnames {
    puts -nonewline "$resname :"
    myincr count(ZN,7,$resname) 0
    puts [replicate * $count(ZN,7,$resname)]
  }
}
```

For the example, the files 1TRZ, 1LND, and 1EZM contain zincs.

```
vmd> unset count
```

```
vmd> analyze_ion_propensity {1trz 1lnd 1ezm} ZN
ALA :****
ARG :***
ASN :****
ASP :***
CYS :**
GLN :
GLU :******
GLY :
HIS :***********
ILE :
LEU :*
LYS :**
MET :
PHE :*
PRO :
SER :****
THR :
TRP :
TYR :***
VAL :****
```

As expected, histidines were one of the most common zinc neighbors. Of course, there will still be problems of missampling (for instance, overcounting molecules with zinc finger dimers) so you should be very sure of what you are doing when using this type of automated analysis.

16.3 save/load VMD state information

One of the more frequently requested options is the ability to save the current state of VMD to a file so it can be recovered. It is possible to log the core commands to a file and play them back later, and reconstruct everything that way. However, that is quite cumbersome and slow, so a better way is needed.

Because we are making much of the VMD data accessible at the Tcl script level, we decided to implement the save/restore option as a Tcl command. This is actually a rather complicated process and some of the things to watch out for are:

- A loaded molecule could be from a coordinate file or a coordinate and a topology file
- The molecule could have been converted with Babel
- The molecule could be a "graphics" molecule (see Chapter §13)
- Each molecule has a list of selections, colors, and drawing methods
- The mapping of a color category item to color can be changed
- Color definitions can be redefined
- Each molecule has its own viewing matrix

- Loading a new molecule resets all the viewing matrices

Ivo Hofacker has graciously struggled through all the above caveats to write a set of Tcl commands which will save nearly all of the current VMD state to a file, and restore it back again. The commands are

- **save_state** *file_name* – saves the vmd state in the form of a script in file *file_name*. Restarting vmd as `vmd -e` *file_name* should restore most of the previous session. Essentially everything that can be changed from the graphics and mol and color forms will be saved, but nothing else. In particular, the script will restore:
  - molecules read in from files
  - graphics objects (spheres, cylinders...)
  - all representations
  - the transformation matrices for each molecule
  - which color to use for which representation, definitions of colors 0..33

  It will not restore:
  - trajectories loaded via the edit menu
  - labels
  - interactive MD sessions
  - anything in the display form, such as lighting, axes, stage, ...
- **save_viewpoint** – saves the center, rotation, scaling and global matrices (in this order) in a global variable `viewpoints`
- **restore_viewpoint** – restores the viewpoint saved in variable `viewpoints`
- **save_reps** – saves the current molecule(s) representations in a global array `representations`.
- **restore_reps** – replaces current representations by the representations stored in the global array `representations`. Note, that `save_reps` saves molecule Ids along with the representations, so if the molId of a currently loaded molecule is different from that saved, this command will not work properly.
- **save_colors** *file_descriptor* – writes color definitions to a file specified by a file_descriptor. This file can later be sourced to restore the definitions. To save the color definitions, the following procedure can be used.

```
# save the color definitions in a file
# argument: file name
proc savcol {file} {
  save_colors [open $file w]
  close $filedesc
}
```

  This procedure takes the file name as a parameter (as opposed to file descriptor in `save_colors`).

If you plan to write scripts which interact with VMD itself (not just the analysis commands), you are highly advised to look at the commands, which are defined in `$VMDDIR/scripts/vmd/save_state.tcl` (The code is too long to be included in this section.)

16.4 Currently picked molecule/atom

Every time an atom is picked, the Tcl variables `vmd_pick_mol` and `vmd_pick_atom` are set to the molecule id and atom index of the picked atom. This is useful for scripts that need to act on used defined atom.

For example, the following procedure takes the picked atom and finds the molecular weight of residue it is on.

```
proc mol_weight {} {
      # use the picked atom's index and molecule id
      global vmd_pick_atom vmd_pick_mol
      set sel [atomselect $vmd_pick_mol "same residue as index $vmd_pick_atom"]
      set mass 0
      foreach m [$sel get mass] {
            set mass [expr $mass + $m]
      }
      # get residue name and id
      set atom [atomselect $vmd_pick_mol "index $vmd_pick_atom"]
      lassign [$atom get {resname resid}] resname resid
      # print the result
      puts "Mass of $resname $resid = $mass"
}
```

Once an atom has been picked, run the command `mol_weight` to get output like:

```
Mass of ALA 7 : 67.047
```

16.5 Trace on the pick variables

Tcl has the "trace" command which registers a procedure to be called when a variable is read, changed, or deleted. (Please see one of the various Tcl books for examples on how to use this.)

Since VMD sets the vmd_pick_atom and vmd_pick_mol variables, they can be traced.

16.5.1 Information about the picked atom

As a simple example, the following procedure calls the "mol_weight" command (in the previous section).

```
proc mol_weight_trace_fctn {args} {
      mol_weight
}
```

(This function is needed because the functions registered with trace take three arguments, but "mol_weight" only takes one.)

The trace function is registered as:

```
trace variable vmd_pick_atom w mol_weight_trace_fctn
```

And now the residue masses will be printed automatically when an atom is picked. To turn this off,

```
trace vdelete vmd_pick_atom w mol_weight_trace_fctn
```

16.5.2 Making a sphere appear when an atom is picked

Similarly, you can use the callback to generate a sphere when an atom is picked.

```
proc pick_sphere {args} {
      global vmd_pick_atom vmd_pick_mol
      # get the coordinates
      lassign [[atomselect $vmd_pick_mol "index $vmd_pick_atom"] \
            get {x y z}] x y z
      # draw the sphere
      draw sphere "$x $y $z" radius 1
}
```

and establish the trace:

```
trace variable vmd_pick_atom w pick_sphere
```

Whenever you click on an atom, a sphere will appear at the same location. Since the graphics and the molecule aren't the same graphics object, you may need to reset view to make them aligned.

To turn the trace off:

```
trace vdelete vmd_pick_atom w pick_sphere
```

16.5.3 Drawing a line from the eye to the picked atom

This last example of a trace on the picked atom draws a line from the picked atom to the user's eye. It is a good example of how to use matrices in VMD. The limitation to the following procedure is that it doesn't understand perspective viewing, so to make it work, use the orthographic mode.

This was used to find the direction to pull a ligand from its bound position out of the protein. The molecule was rotated until the user could look straight down to the ligand. The user then picked an atom on the ligand, causing a line (actually, a cylinder) to be drawn from the atom past the eye location, and the start and end points of the cylinder were printed for later use.

```
proc eye_line {} {
  global vmd_pick_atom vmd_pick_mol
  set sel [atomselect $vmd_pick_mol "index $vmd_pick_atom"]

  # coordinates of the atom
  set coords [lindex [$sel get {x y z}] 0]

  # position in world space
  set mat [lindex [molinfo $vmd_pick_mol get view_matrix] 0]
  set world [vectrans $mat $coords]

  # since this is orthographic, just get the projection on z
  lassign $world x y
  # get a coordinate behind the eye
  set world2 "$x $y 5"
```

```
  # convert back to molecule space
  # (need an inverse, which is only available with the
  #  measure command)
  set inv [measure inverse $mat]
  set coords2 [vectrans $inv $world2]

  # and draw the line
  draw cylinder $coords $coords2 radius 0.3

  puts "Start: $coords"
  puts "End: $coords2"
}
```

16.6 Trajectory frames

This section shows examples of how to use `vmd_frame($molecule)` and `vmd_timestep($molecule)`. They both depend on trajectory information, but one is set during playback of an animation while the other is set only when a new coordinate set has been received from the remote simulation.

16.6.1 Animating the secondary structure

The secondary structure definitions for the molecules in VMD don't change during an animation but they can be made to do so with a trace on the `vmd_frame($molecule)` Tcl variable. The simplest way is to call `vmd_calculate_structure(molecule)` every time the frame changes, e.g.,

```
proc structure_trace {name index op} {
      vmd_calculate_structure $index
}

trace variable vmd_frame w structure_trace
```

but this isn't the fastest solution for a couple of reasons. First off, the secondary structure code is somewhat slow (and about 2/3 of the time is spent in the Tcl interface; mostly in writing a temporary PDB file). If you don't plan to modify the coordinates, it is possible to store, or *cache*, the secondary structure from one frame to the next. Second, if there are multiple molecules loaded and animated, the secondary structures of all of them are computed.

The following script, sscache (for "secondary structure cache") addresses those two problems. It is turned on with the command `start_sscache` followed by the molecule number of the molecule whose secondary structure should be saved (the default is "top", which gets converted to the correct molecule index). Whenever the frame for that molecule changes, the procedure `sscache` is called.

`sscache` is the heart of the script. It checks if a secondary structure definition for the given molecule number and frame already exists in the Tcl array sscache_data(molecule,frame). If so, it uses the data to redefine the "structure" keyword values (but only for the protein residues). If not, it calls the secondary structure routine to evaluate the secondary structure based on the new coordinates. The results are saved in the sscache_data array.

Once the secondary structure values are saved, the molecule can be animated rather quickly and the updates can be controlled by the animate form.

To turn off the trace, use the command `stop_sscache`, which also takes the molecule number. There must be one `stop_sscache` for each `start_sscache`. The command `clear_sscache` resets the saved secondary structure data for all the molecules and all the frames.

```
# Cache secondary structure information for a given molecule

# reset the secondary structure data cache
proc reset_sscache {{molid top}} {
    global sscache_data
    if {! [string compare $molid top]} {
      set molid [molinfo top]
    }
    if [info exists sscache_data($molid)] {
        unset sscache_data
    }
}

# start the cache for a given molecule
proc start_sscache {{molid top}} {
    if {! [string compare $molid top]} {
      set molid [molinfo top]
    }
    global vmd_frame
    # set a trace to detect when an animation frame changes
    trace variable vmd_frame($molid) w sscache
    return
}

# remove the trace (need one stop for every start)
proc stop_sscache {{molid top}} {
    if {! [string compare $molid top]} {
      set molid [molinfo top]
    }
    global vmd_frame
    trace vdelete vmd_frame($molid) w sscache
    return
}


# when the frame changes, trace calls this function
proc sscache {name index op} {
    # name == vmd_frame
    # index == molecule id of the newly changed frame
    # op == w
```

```
    global sscache_data

    # get the protein CA atoms
    set sel [atomselect $index "protein name CA"]

    ## get the new frame number
    # Tcl doesn't yet have it, but VMD does ...
    set frame [molinfo $index get frame]

    # see if the ss data exists in the cache
    if [info exists sscache_data($index,$frame)] {
        $sel set structure $sscache_data($index,$frame)
        return
    }

    # doesn't exist, so (re)calculate it
    vmd_calculate_structure $index
    # save the data for next time
    set sscache_data($index,$frame) [$sel get structure]

    return
}
```

16.6.2 Viewing selections which change during an animation

One of the researchers here needed a way to see which waters bridged between a protein and a nucleic acid during a trajectory. The specific waters change during the simulation, so the static method used in the graphics form doesn't work. Instead, the `vmd_frame` variable and a caching method similar to the sscache of the previous section was used for the following solution:

The cache data is stored in the array `bridging_waters`. Unlike `sscache`, this array is indexed only by the frame number; you'll have to modify it to analyze multiple changing selections.

When the frame number changes, the cache is checked and, if no data exists, the selection is rebuilt. Since the selection is created in a procedure, it must be given by the `global` command to make it exist outside that procedure's context.

The global Tcl variable `bridging` is set to the selection for this frame.

The first text selection in the graphics form is set to `@bridging`. This is one of the special extensions to the selection language which enable the selections to reference a Tcl variable. Note also that the display updates are temporarily turned off. This is used to keep the display from being drawn an additional time by the call to `mol modselect`. This is done every time the frame changes since that's the only way to tell VMD the graphics have changed; forcing it to recalculate that representation's display.

```
# start the trace
proc start_bridging {{molid top}} {
 global vmd_frame bridging_molecule
 if {![string compare $molid top]} {
    set molid [molinfo top]
```

```
  }
  trace variable vmd_frame($molid) w calc_bridging
  set bridging_molecule $molid
}

# stop the trace
proc stop_bridging {} {
  global bridging_molecule vmd_frame
  trace vdelete vmd_frame($bridging_molecule) w calc_bridging
}

# do the actual calculation
proc calc_bridging {args} {
  global bridging_waters bridging_molecule bridging
  # get the current frame number
  set frame [molinfo $bridging_molecule get frame]

  # has the selection already been made?
  if {! [info exists bridging_waters($frame)]} {
    puts "Calculating frame $frame for $bridging_molecule"
    set bridging_waters($frame) [atomselect $bridging_molecule
       {(water within 4 of segname DNA) and (water within 4 of segname PRO1)}
       frame $frame]
  }
  # set things up for the graphics form to use the precomputed selection
  set bridging $bridging_waters($frame)
  # do this since otherwise the selection is deleted when the proc ends
  $bridging global
  # update the 0th element of the graphics molecule
  # Note: if the display wasn't turned off, there would be an extra
  # update of the animation ... very bad
  display update off
  mol modselect 0 $bridging_molecule {@bridging}
  display update on
}
```

Since the selections are available in the global scope, you can analyze the results at any time. This simple function prints how many atoms are in each selection of the cache.

```
proc num_bridging {} {
  global bridging_waters
  set nums [lsort -integer [array names bridging_waters]]
  foreach num $nums {
    set num_atoms [$bridging_waters($num) num]
    puts "Frame $num has $num_atoms atoms"
  }
```

```
}
```

It would be nice to change the text selection used, support multiple selections, have the "@" variables in more complicated expressions in the graphics selections, and be able to use this script for multiple molecules. These are left to the student as an exercise `:)`.

16.6.3 Simulation frames

When a new simulation timestep arrives, the Tcl variable `vmd_timestep(molecule)` is set to the value of the new frame. You guessed it – this can be used along with the `trace` command.

One simple case prints the temperature of the new timestep.

```
proc timestep_temp_trace {name id op} {
      set temp [molinfo $id get temperature]
      puts "Temp. of molecule $id is $temp"
}

trace variable vmd_timestep w timestep_temp_trace
```

Of course, why do textually what you can do graphically? The following makes a plot of the last ten temperatures. For simplicity, only one plot is allowed.

The `begin_temp_plot` sets up the temperature plot variables and calls the `trace`. The trace calls `timestep_temp_trace` which adds the new temperature to the storage list and then calls the graphics routine. (A list was used here because, while slower than an array, the lvarpop was handy.) There is nothing new or unusual about the plotting procedure (`draw_temps_plot`) or the routine to finish the real-time plotting (`end_temp_plot`).

```
proc begin_temp_plot {molid} {
   # make sure the molecule exists
   molinfo $molid get frame

   global temps temps_mol temps_trace_mol vmd_timestep
   # create a new graphics molecule
   mol load graphics "Temp.of_$molid"
   set temps_mol [molinfo top]
   set temps [ldup 10 0]
   # the molecule to watch
   set temps_trace_mol $molid
   # start the trace
   trace variable vmd_timestep($molid) w timestep_temp_trace
}

proc end_temp_plot {} {
   global temps_trace_mol
   # remove the trace
   trace vdelete vmd_timestep($temps_trace_mol) w timestep_temp_trace
}
```

```
proc draw_temps_plot {} {
   global temps temps_mol
   graphics $temps_mol delete all
   graphics $temps_mol color red
   set tmp $temps
   set pt0 [lvarpop tmp]
   set x0 0
   set x1 10
   foreach pt1 $tmp {
      graphics $temps_mol line "$x0 $pt0 0" "$x1 $pt1 0"
        set x0 $x1
        incr x1 10
        set pt0 $pt1
   }
   graphics $temps_mol color green
   graphics $temps_mol line {0 0 0} {100 0 0}
   graphics $temps_mol line {0 0 0} {0 500 0}
}

proc timestep_temp_trace {args} {
   global temps temps_mol temps_trace_mol
   # delete old and add newest value to temps
   lvarpop temps
   set new_t [molinfo $temps_trace_mol get temp]
   set temps "$temps $new_t"
   draw_temps_plot
}
```

Needless to say, many more options can be added to this for plotting different variables, autoscaling, adding text, etc.

By the way, though it has not yet been tested out, we envision that a trace on the `vmd_timestep` variable could be used to modify the user forces as the simulation progresses. This makes the linear force controls emulate a harmonic well, or let you apply the forces along a path. You could even make two atoms come towards each other, or draw apply the forces to the atoms on a selection such that the center of mass of the selection is the important term. If you want to try it out, good luck!

16.7 RMSD and best-fit alignments

Calculation of RMSD and best-fit alignment of two sets of atoms using the mouse controls were described in section §3.1.1. The same actions can be taken on the scripting level. Text interface also gives you more flexibility through the atom selection mechanism allowing to choose the atoms to fit/compare.

16.7.1 RMSD Computation

There are two atom selections needed to do an RMSD computation, the list of atoms to compare in both molecules. The first atom of the first selection is compared to the first atom of the second selection, fifth to fifth, and so on. The actual order is identical to the order from the input PDB file.

Once the two selections are made, the RMSD calculation is a matter of calling the `measure rmsd` function. Here's an example:

```
      set sel1 [atomselect 0 "backbone"]
      set sel2 [atomselect 1 "backbone"]
      measure rmsd $sel1 $sel2
Info)  10.403014
```

This prints the RMSD between the backbone atoms of molecule 0 with those of molecule 1. You could also use a weighting factor in these calculations. The best way to understand how to do this is to see another example:

```
      set weighted_rmsd [measure rmsd $sel1 $sel2 weight mass]
Info)  10.403022
```

In this case, the weight is determined by the mass of each atom. Actually, the term is really one of the standard keywords available to an atom selection. Other ones include index and resid (which would both be rather strange to use) as well as charge, beta and occupancy. These last terms useful if you want to specify your own values for the weighting factors.

16.7.2 Computing the Alignment

The best-fit alignment is done in two steps. The first is to compute the 4×4 matrix transformation that takes one set of coordinates onto the other. This is done with the `measure fit` command. Assuming the same selections as before:

```
        set transformation_matrix [measure fit $sel1 $sel2]
Info) {0.971188 0.00716391 0.238206 -13.2877}
{0.0188176 0.994122 -0.106619 3.25415} {-0.23757 0.108029 0.965345 -2.97617}
{0.0 0.0 0.0 1.0}
```

As with the RMSD calculation, you could also add an optional `weight <keyword>` term on the end.

The next step is to apply the matrix to a set of atoms using the move command. So far you have two coordinate sets. You might think you could do something like `$sel1 move $transformation_matrix` to apply the matrix to all the atoms of that selection. You could, but that's not the right selection.

The thing to recall is that `$sel1` is the selection for the backbone atoms. You really want to move the whole fragment to which it is attached, or even the whole molecule. (This is where the discussion earlier comes into play.) So you need to make a third selection containing all the atoms which are to be moved, and apply the transformation to those atoms.

```
        # molecule 0 is the same molecule used for $sel1
        set move_sel [atomselect 0 "all"]
        $move_sel move $transformation_matrix
```

As a more complicated example, with the mouse you cannot do something like "align all of molecule 1 with molecule 9 using only the backbone atoms of residues 4 to 10 in both systems" with the mouse. However, you can do it with the text command. Here's how:

```
# compute the transformation matrix
set reference_sel  [atomselect 9 "backbone and resid 4 to 10"]
set comparison_sel [atomselect 1 "backbone and resid 4 to 10"]
set transformation_mat [measure fit $comparison_sel $reference_sel]

# apply it to all of the molecule 1
set move_sel [atomselect 1 "all"]
$move_sel move $transformation_mat
```

16.7.3 A simulation example script

Here's a longer script which you might find useful. The problem is to compute the RMSD between each timestep of the simulation and the first frame. Usually in a simulation there is no initial global velocity, so the center of mass doesn't move, but because of angular rotations and because of numerical imprecisions that slowly build up, the script aligns the molecule before computing its RMSD.

```
# Prints the RMSD of the protein atoms between each timestep
# and the first timestep for the given molecule id (default: top)
proc print_rmsd_through_time {{mol top}} {
        # use frame 0 for the reference
        set reference [atomselect $mol "protein" frame 0]
        # the frame being compared
        set compare [atomselect $mol "protein"]

        set num_steps [molinfo $mol get numframes]
        for {set frame 0} {$frame < $num_steps} {incr frame} {
                # get the correct frame
                $compare frame $frame

                # compute the transformation
                set trans_mat [measure fit $compare $reference]
                # do the alignment
                $compare move $trans_mat
                # compute the RMSD
                set rmsd [measure rmsd $compare $reference]
                # print the RMSD
                puts "RMSD of $frame is $rmsd"
        }
}
```

To use this, load a molecule with an animation (for example, `$VMDDIR/proteins/alanin.DCD` from the VMD distribution). Then run `print_rmsd_through_time`. Example output is shown here:

```
vmd > print_rmsd_through_time
```

```
RMSD of 0 is 0.000000
RMSD of 1 is 1.060704
RMSD of 2 is 0.977208
RMSD of 3 is 0.881330
RMSD of 4 is 0.795466
RMSD of 5 is 0.676938
RMSD of 6 is 0.563725
RMSD of 7 is 0.423108
RMSD of 8 is 0.335384
RMSD of 9 is 0.488800
RMSD of 10 is 0.675662
RMSD of 11 is 0.749352
 [...]
```

If you wanted you could do all sorts of things, like graph this value through time.

Chapter 17

Customizing VMD Sessions

There are a number of ways to change the behavior of VMD from the default settings, both in how the program starts up and in how the program behaves during a session. This Chapter describes the data files, command-line options, and environment variables which are used to customize a VMD session.

These files control the initial appearance and behavior of VMD at the start, and may be customized to suit each user's particular tastes. Default versions of these files are placed in the VMD installation directory (usually `/usr/local/lib/vmd`). Each user may specify their own versions of some of these files, but unless this is done the commands and values in the default files are used. In this way, an administrator may customize the default behavior of VMD for all users, while giving each user the option to change the default behavior however they choose.

Several configurable parameters may also be set in a number of ways, including use of command-line options or environment variables. The order of precedence of these methods is as follows (highest precedence to lowest):

1. Command-line options.
2. Environment variable settings.
3. Values specified in the initialization file (`.vmd_init`, section 17.3.3).
4. Built-in defaults, as specified by compilation configurable parameters. These are used only if no other values are specified by the other methods mentioned in this list. The Installation Guide describes how to change these default values when compiling VMD.

17.1 Command-Line Options

When started, the following command-line options may be given to VMD. Note that if a command-line option does not start with a dash (-), and is not part of another option, it is assumed to be a PDB filename. Thus, the command

```
vmd molecule.pdb
```

will start VMD and load a molecule from the file `molecule.pdb`.

- `-h | -?` : Print a summary a command-line options to the console.

- `-e filename` : After initialization, execute the text commands in filename, and then resume normal operation.
- `-psf filename` : Load the specified molecule (in PSF format) at startup. The PSF file only contains the molecular structure; a PDB or DCD file must also be specified when this option is used.
- `-pdb filename` : Load the specified molecule (in PDB format) at startup.
- `-dcd filename` : Load the specified trajectory file (in binary DCD format) at startup. The DCD file only contains atomic coordinates; a PDB or PSF file must also be specified when this option is used.
- `-dispdev < win | text | cave | caveforms | none >` : Specify the type of graphical display to use. The possible display devices include:
  - `win`: a standard graphics display window.
  - `text`: do not provide any graphics display window.
  - `cave`: use the CAVE virtual environment for display, Forms disabled.
  - `caveforms`: use the CAVE virtual environment for display and with Forms enabled. This is useful with `-display machine:0` for remote display of the Forms when the CAVE uses the local screen.
  - `none`: same as text.

  It is possible to use VMD as a filter to convert coordinate files into rendered images, by using the `-dispdev text` and `-e` options.
- `-pos x y` : Specify the position for the graphics display window. The position (x,y) is the number of pixels from the lower-left corner of the display to the lower-left corner of the graphics window.
- `-size x y` : Specify the size for the graphics display window, in pixels.
- `-nt` : Do not display the VMD title at startup.
- `-startup filename` : Use filename as the VMD startup command script, instead of the default file.
- `-init filename` : Use filename as the file containing VMD initialization data, instead of the default file `.vmd_init`.
- `-debug [level` : Turn on output of debugging messages, and optionally set the current debug level (1=few messages ... 5=many verbose messages). Note this is only useful if VMD has been compiled with debugging option included.

17.2 Environment Variables

Several environment variables are used by VMD to determine the location of certain files and directories. These variables are accessible to text interface through array `env`. These variables include:

- `VMDDIR` : The directory which contains the VMD data files (such as this help file) and architecture-specific executables. By default, this is /usr/local/lib/vmd.
- `VMDTMPDIR` : The directory which VMD should use for temporary data files. By default, this is /tmp.
- `VMDBABELBIN` : The complete path and filename for the program babel, which is used by VMD to convert molecular structure/coordinates files into PDB files which VMD can actually understand. If this is not set explicitly, the VMD startup script will attempt to find babel in the current path. If Babel cannot be found or is not installed, VMD will not be able to read molecular file formats other than PDB, PSF, and binary DCD files.
- `VMDHTMLVIEWER` : The name of the HTML viewer (Netscape, Mosaic, whatever you prefer) that VMD should use to display HTML documents (such as this help file). By default, this is Netscape.

17.3 Startup Files

17.3.1 Core Script Files

In the following, the value of `$VMDDIR` is the vmd installation directory. During the original installation this is the value of `INSTALLLIBDIR`. It can also be found by looking at the first few lines of the vmd startup script (`head 'which vmd'`) or by starting VMD and using the command `set env(VMDDIR)`.

As mentioned elsewhere, VMD uses the Tcl interpreter with two language extensions, TclX and Tcl-DP. Each of these three packages reads Tcl scripts at initialization, which are contained in VMD distribution; with a different directory for each package. The locations of the directories is determined by the environment variables defined in the `vmd` script. In order they are `TCL_LIBRARY`, `TCLX_LIBRARY`, and `DP_LIBRARY`, which are set in the vmd startup script to `$VMDDIR/scripts/tcl`, `$VMDDIR/scripts/tclX`, and `$VMDDIR/scripts/dp`, respectively. In addition, VMD has its own directory of core Tcl routines.

The most important of these is tt $VMDDIR/scripts/vmd/vmdinit.tcl. This file sets up the basic Tcl initialization commands including the commands that catch `Ctrl-C` interrupts, define some environment variables, and add the vmd script directory to the Tcl autoindex path. Most of the other files are referenced through the `auto_path`.

There are a few non-Tcl scripts in this directory. Currently these are perl scripts used for the `urlload` command and `web client` startup (see section § 11.3.15 and section §15.2).

17.3.2 User Script Files

Two user configuration files are available: an initialization file `.vmd_init`, and a run-time command file `.vmdrc` with a list of initial VMD text commands to process. These files may be changed to customize individual user's initial screen appearance and to set the proper display characteristics for displaying in stereo. If they do not exist, default values are used.

17.3.3 `.vmd_init` File

This file is used to define several configuration variables which must be set before several key objects (e.g., the display device) can be created. (This is actually a limitation of the VMD implementation

— future versions will not use this file.) Thus this is the first file read by VMD; before even the vmdinit.tcl file. VMD searches for this file in three locations; `$VMDDIR/.vmd_init`, `$HOME/.vmd_init`, and `./.vmd_init`. **Each** copy of the file found will be read in and processed in that order. This allows the administrator to establish a complete set of default values for all users, and then for each user to selectively change the settings required without having to set ALL parameters.

The lines in the initialization file should be either blank lines, comment lines (beginning with the `#` character), or parameter setting lines of the form

```
<keyword> = <value>
```

with any amount of whitespace allowed before and after the equal sign. The possible keywords that may be entered here are:

- `BABEL = `<*filename*>
  The name (including complete path) of the `babel` program executable, which is used to convert various molecular file formats to PDB files which may be read by VMD. See section §4.1 This parameter should not be used; instead, use environment variable `VMDBABELBIN` (see section §17.2)

- `DISPLAY = <WIN | CAVE | TEXT>`
  The type of display device which VMD should use initially. This can be set to either `WIN`, `CAVE`, or `TEXT`. By default, this is `WIN`. It can be overridden with the `-dispdev` command line option.

- `HTMLVIEWER = `<*filename*>
  The name of the HTML browser which should be used to display HTML files (such as the VMD on-line help file). Typical examples are Mosaic or Netscape; by default this is Netscape.

- `SCRDIST = `<*value*>
  The distance, in 'world' coordinates, from the origin to the display screen. If this is zero, the origin of the coordinate system in which molecules are drawn coincides with the center of the display. If it is < 0, the origin is located between the viewer and the screen, while if it is > 0, the screen is located closer to the viewer than the origin. A value < 0 puts any stereo image in front of the screen, aiding the three-dimensional effect; a value > 0 results in a stereo image that is behind the screen, a less dramatic (but easier to see, for some people) stereo effect when stereo display is used. Figure 3.17 describes the relationship between this parameter and the world coordinate space.

- `SCRHEIGHT = `<*value*>
  This parameter, with `SCRDIST`, defines the size and distance of the display screen. `SCRHEIGHT` is the default value for the screen height, which is the vertical size of the display screen in 'world' coordinates. Each molecule is initially scaled and translated to fit within a 2 x 2 x 2 box centered at the origin; so the height of the screen helps determine how large the molecule appears initially. If VMD is being displayed on a workstation monitor only, it is best not to change this value from the default setting. This parameter is used mainly to configure the VMD display to the dimensions and position of a large-screen display, such as a projector, that may be used as a stereo display. As for `SCRDIST`, figure 3.17 describes the relationship between this parameter and the world coordinate space.

- `SCRPOS = `<*xpos*> <*ypos*>
  The position of the lower-left corner of the window used to display images, if the display type is `WIN`. Both *xpos* and *ypos* are in pixels. If they are not specified, and are not given as command-line options, the window will have to be interactively placed with the mouse.

- `SCRSIZE = `<*xsize*> <*ysize*>
  The size of the window used to display images, if the display type is `WIN`. Both *xpos* and *ypos* are in pixels. If they are not specified, and are not given as command-line options, the window will have to be interactively sized with the mouse.

- `TITLE = <ON | OFF>`
  A flag indicating whether to display the VMD title in the main display when the program starts. This can have the value of `ON` or `OFF`. By default, `TITLE` is set to be `ON`. If the title is turned on, it will remain displayed until a molecule is loaded, after which it will be hidden from view. This can also be set with the command line option `-nt`.

Here is an example of an initialization file:

```
display = win
title = on
scrheight = 6.0
scrdist = -2.0
scrpos  = 596 190
scrsize = 669 834
```

17.3.4 `.vmdrc` File

After everything is initialized, VMD reads the *startup* file using the equivalent of the command `play .vmdrc`. This file contains text commands for VMD to execute just as if they had been entered at the VMD text console command prompt. The file can contain any number of commands, including blank lines and comment lines (which begin with the `#` character). If an error is encountered while reading this file, the command in error is skipped and processing of the file continues.

VMD searches for this file in three locations; `./.vmdrc`, `$HOME/.vmdrc` and `$VMDDIR/.vmdrc`. Only the first file found will be read in and processed.

See chapter § 11 for a description of the VMD text commands which may be put in this file. Also, section § 15.1 discusses how to put commands into the `.vmdrc` file to customize the behavior of the popup menu and hot keys.

Here is an example of a startup file:

```
# add personalized keyboard shortcuts
user add key E echo on
user add key e echo off
user add key g display reset
user add key A stage location bottom
user add key m mol list

# position the stage and axes
axes location lowerleft
stage location off
```

```
# position and turn on menus
menu main move 5 196
menu display move 386 90
menu animate move 124 7
menu edit move 125 196
menu graphics move 5 455
menu files move 5 496
menu mol move 5 745

menu main on

# start the scene a-rockin'
rock y by 1
```

Chapter 18

Future Plans

Following is a list of features we would like to add. They will be implemented as they are needed, but some will not be done until the next major version.

- Improve PDB reader to read multiple models and the new ”element” field (see ??? for a program to convert CONECT records to a PSF file)
- element type based on atom name and residue type, if information isn't given in the PDB
- append and delete specific atoms
- text for raster outputs
- improve surface rendering speeds
- improve sphere rendering speeds
- save more information about current setup (see section § 16.3 for the script commands to save the current setup)
- standardize nomenclature and usage of keywords and other definitions
- add more selection definitions, like `purine`
- expand the ability of the mouse to pick atoms
- add atom coloring methods
- rewrite the forms using Tk
- faster calculation of hydrogen bonds
- implement "correct" transparency
- be able to select bonds
- draw molecular and isoelectric surfaces (a la GRASP) ???
- develop the 3D interface
- improve the interactive steering

- will need to add a new input/output coordinate file (probably GROMOS) because we are approaching limits to atom and residue number

- read in the electron density maps made by X-PLOR and display isosurfaces ??? get the right name (based on Joel's comments)

- work on an interface to use X-PLOR scripts as subroutines, perhaps via Tcl's "expect"

Index