VMD User's Guide

Version 1.8.4

April 17, 2006

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

Description

The VMD User's Guide describes how to run and use the molecular visualization and analysis program VMD. This guide documents the user interfaces displaying and grapically manipulating molecules, and describes how to use the scripting interfaces for analysis and to customize the behavior of VMD.

[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. VMD runs on all major Unix workstations, Apple MacOS X, and Microsoft Windows. Online information about VMD is available from:
`http://www.ks.uiuc.edu/Research/vmd/`

List of key VMD features:

- **General molecular visualization**
  At its heart, VMD is a general application for displaying molecules containing any number of atoms and is similar to other molecular visualization programs in its basic capabilities. VMD reads data files using an extensible plugin system, and supports Babel for conversion of other formats. User-defined atom selections can be displayed in any of the standard molecular representations. Displayed graphics can be exported to an image file, to a scene file usable by ray tracing programs, or to a geometry description file suitable for use with 3-D printers.

- **Visualization of dynamic molecular data**
  VMD can load atomic coordinate trajectories from AMBER, Charmm, DLPOLY, Gromacs, MMTK, NAMD, X-PLOR, and many other simulation packages. The data can be used to animate the molecule or to plot the change in molecular properties such as angles, dihedrals, interatomic distances, or energies over time.

- **Visualization of volumetric data**
  VMD can load, generate, and display, volumetric maps. Supported map formats include CryoEM maps, electrostatic potential maps, electron density maps, and many other map file formats.

- **Interactive molecular dynamics simulations**
  VMD can be used as a graphical front-end to a live molecular dynamics program running on a remote supercomputer or high-performance workstation. VMD can interactively apply and visualize forces in an MD simulation as it runs.

- **Molecular analysis commands**
  Many commands are provided for molecular analysis. These include commands to extract information on sets of atoms and molecules, vector and matrix routines for coordinate manipulation, and functions for computing values such as center of mass and radius of gyration.

- **Tcl and Python scripting languages**
  VMD uses the freely available Python and Tcl scripting languages for processing text commands. These popular languages which contain variables, loops, subroutines, and much more. VMD also uses the Tk Toolkit - a simple user interface toolkit that interfaces with Tcl.

- **Easy to extend**
  VMD is written in C and C++ and employs object-oriented design. VMD implements a plugin interface for extending its file format support and for general purpose extensions in functionality.

- **Support for multimodal input and various display systems**
  A number of different visual display and control systems are supported in addition to the usual monitor, keyboard, and mouse. The VRPN library is used to get position and orientation information from a wide variety of spatial input devices, including magnetic trackers, haptic (force feedback) devices, Spaceballs, etc. VMD works with WireGL and Chromium on tiled display walls, and immersive VR environments via compiled-in CAVE and FreeVR support.

1.1 Contacting the authors

The current developer of VMD is John E. Stone. The list of individuals that made signficant contributions to this version of VMD in the form of patches, bug fixes, and completely new plugins includes Anton Arkhipov, Michael Bach, Jordi Cohen, Markus Dittrich, John Eargle, Peter Freddolino, Luis Gracia, Justin Gullingsrud, Konrad Hinsen, Axel Kohlmeyer, John Mongan, Jim Phillips, Jan Saam, Alexander Spaar, Marcos Sotomayor, and Kirby Vandivort.

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`.

1.2 Registering VMD

VMD is made available free of charge for all interested end-users of the software (but please see the Copyright and Disclaimer notices). Please check the current VMD license agreement for details. Registration is part of our software download procedure. Once you've filled out the forms on the VMD download area and have read and agreed to the license, you are finished with the registration process.

1.3 Citation 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 and Computational Biophysics Group at the University of Illinois and the Beckman Institute. 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 Acknowledgments

The authors would particularly like to thank those individuals who have contributed suggestions and improvements, particularly those contributing new features. Special thanks go to Anton Arkhipov, Andrew Dalke, Michael Bach, Alexander Balaeff, Ilya Balabin, Eamon Caddigan, Jordi Cohen, Markus Dittrich, John Eargle, Peter Freddolino, Todd Furlong, Luis Gracia, Paul Grayson, Justin Gullingsrud, Konrad Hinsen, Barry Isralewitz, Sergei Izrailev, Axel Kohlmeyer, John Mongan, Jim Phillips, Jan Saam, Alexander Spaar, Charles Schwieters, Marcos Sotomayor, John E. Stone, and Kirby Vandivort. The entire VMD user community now benefits from your contributions.

The authors would like to thank individuals who have indirectly helped with development by making suggestions, pushing for new features, and trying out buggy code. Thanks go to Aleksei Aksimentiev, Daniel Barsky, Axel Berg, Tom Bishop, Robert Brunner, Ivo Hofacker, Mu Gao, James Gumbart, Xiche Hu, Tim Isgro, Dorina Kosztin, Ioan Kosztin, Joe Landman, Ilya Logunov, Clare Macrae, Amy Shih, Lukasz Salwinski, Stephen Searle, Charles Schwieters, Ari Shinozaki, Svilen Tzonev, Emad Tajkhorshid, Michael Tiemann, Elizabeth Villa, Raymond de Vries, Simon Warfield Willy Wriggers, Dong Xu, and Feng Zhou.

Many external libraries and packages are used in VMD, and the program would not be as capable without them. The authors of VMDwish to thank the authors of FLTK; the authors of Tcl and Tk; the authors of Python; the authors of VRPN; Jon Leech for uniform point distributions; Amitabh Varshney for SURF; Dmitrij Frishman for developing STRIDE; Jack Lund for the url_get perl script; Brad Grantham for the ACTC triangle consolidation library; John E. Stone for the Tachyon ray tracer and Spaceball drivers; and Ethan Merrit for one of the ribbon drawing algorithms.

1.5 Copyright and Disclaimer Notices

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:
`http://www.ks.uiuc.edu/Research/vmd/current/LICENSE.html`
Some of the code and executables used by VMD have their own usage restrictions:

- ACTC

3. All advertising materials mentioning features or use of this software must display the following acknowledgment: This product includes software developed by Brad Grantham and Applied Conjecture.
4. Neither the name Brad Grantham nor Applied Conjecture may be used to endorse or promote products derived from this software without specific prior written permission.
5. Notification must be made to Brad Grantham about inclusion of this software in a product including the author of the product and the name and purpose of the product. Notification can be made using email to Brad Grantham's current address (grantham@plunk.org as of September 20th, 2000) or current U.S. mail address.

- Python
  Python is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python may be obtained from a proxy server on the Internet using the following URL: `http://hdl.handle.net/1895.22/1012`

- PCRE
  The Perl Compatible Regular Expressions (PCRE) library used in VMD was written by Philip Hazel and is Copyright (c) 1997-1999 University of Cambridge.
  Permission is granted to anyone to use this software for any purpose on any computer system, and to redistribute it freely, subject to the following restrictions:
  1. This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
  2. The origin of this software must not be misrepresented, either by explicit claim or by omission.
  3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software.
  4. If PCRE is embedded in any software that is released under the GNU General Purpose License (GPL), then the terms of that license shall supersede any condition above with which it is incompatible.

- STRIDE
  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`

- SURF
  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 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.

- Tachyon
  The Tachyon multiprocessor ray tracing system and derivative code built into VMD is Copyright (c) 1994-2006 by John E. Stone. See the Tachyon distribution for redistribution and licensing information.

1.6 For information on our other software

VMD is part of a suite of tools developed by the Theoretical and Computational Biophysics group at the University of Illinois.

- BioCoRE
  BioCoRE is a web-based collaborative environment for structural biology which provides tools to allow collaboration between researchers down the hall or around the world. Anyone with access to the internet and a standard web browser can join BioCoRE and create or be added to research projects, and information about a particular project is shared among all members of that project. More information is available at the BioCoRE home page[1]
- NAMD
  A parallel, object-oriented molecular dynamics code designed for high-performance simulation of large biomolecular systems. NAMD uses the CHARMM force field and file formats compatible with both CHARMM and X-PLOR. NAMD supports both periodic and non-periodic boundaries with efficient full electrostatics, multiple timestepping, constant pressure and temperature ensemble simulation methods. More information is available at the NAMD home page[2]
- MDTools
  MDTools is a collection of programs, scripts, and utilities provided for researchers to make various modeling and simulation tasks easier. More information is available at the MDTools home page[3]

For more information on our software efforts, see the Theoretical and Computational Biophysics Group home page[4].

[1]http://www.ks.uiuc.edu/Research/biocore
[2]http://www.ks.uiuc.edu/Researach/namd
[3]http://www.ks.uiuc.edu/Development/MDTools
[4]http://www.ks.uiuc.edu/

Chapter 2

Tutorials

2.1 Rapid Introduction to VMD

For those of you who don't like reading manuals, here is a quick introduction to VMD. The molecules and data files used in this tutorial can be downloaded from the VMD home page from the documentation area associated with this version and is clearly labeled as User's Guide tutorial data. The rest of this tutorial assumes that you have downloaded and unpacked this data set.

To start VMD type `vmd` on the command line of your shell (Unix), or start it by clicking the VMD icon in your desktop or Start menu (Apple MacOS X and Microsoft Windows). VMD should start up with a window titled vmd console, a display window entitled VMD OpenGL Display, and a main menu entitled VMD. Text commands are typed in the console window, molecules are displayed and manipulated in the graphics window, and other interfaces and extensions are available from the menu interface. All of the forms can be closed or minimized, using your computer's standard windowing controls or the `menu` command [§8.3.18] in text console. Most functions can be performed with both the menu interface and the text console, though some of the more sophisticated scripting features are only available as text commands.

2.2 Viewing a molecule: Myoglobin

In our quick tour of VMD, we'll start out by demonstrating a few of its visualization features. To load a new molecule, select New Molecule... from the File menu in the main form, this will open the Files form [§4.4.4]. We will load a PDB (Protein Data Bank) file containing the coordinates of the atoms in myoglobin (compliments of Joel Berendzen of Los Alamos National Laboratory). Select the Browse... button in the files form to bring up a file browser. Go into the `proteins/` directory of the tutorial data set that you have downloaded from the VMD web site. Once there, select the file `mbco.pdb` in the file browser, and press the Load button in the molecule file browser. button in the Files form. 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 [§4.1.1]: rotation, translation, and scaling. The mode can be changed from the Mouse menu in the main form, or by pressing `r`, `t`, or `s` on the keyboard while the mouse is in the graphics window. While experimenting, note how the cursor changes to indicate the mouse 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

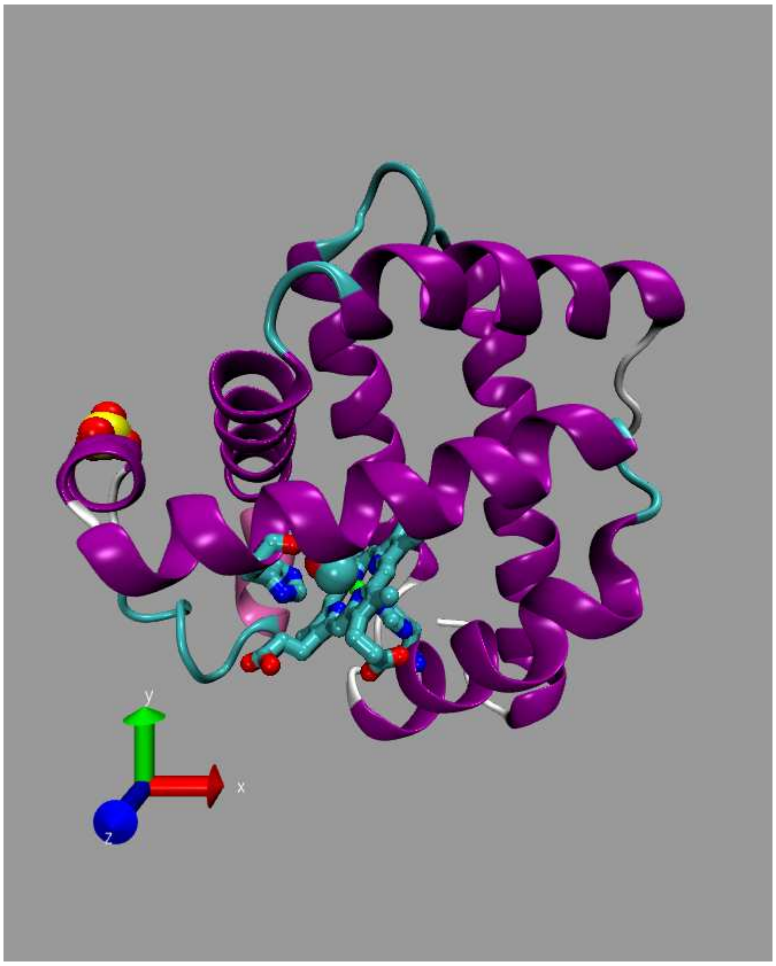

Figure 2.1: Sample VMD session displaying myoglobin.

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.

By default molecules are displayed in a “lines” representation, colored by atom type. 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, open the Graphics form [§4.4.7] by selecting the Representations item in the the graphics menu of the VMD main form. Type `backbone` in the Selected Atoms text entry area and press 'enter' to select the myoglobin backbone. All of the protein except for the backbone will disappear. Choose NewCartoon in the drawing method chooser to display the backbone as a tube, and choose Structure in the coloring method chooser to color the tube with the predefined secondary structure color. Press the Create Rep button. This creates a new representation in the browser, identical to the original one. The new representation can be changed without affecting others, so clear the atom selection text area and enter `resname HEM` to select the heme. At this point the heme isn't visible because it cannot be drawn as a cartoon, 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 Rep 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 nice image of myoglobin! With further experimentation you should be well on your way to learning how to use VMD.

2.3 Rendering an Image

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 often 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 Chapter 7 on rendering for a further description of how this works.

Open the Render form [§ 4.4.11] and select 'Tachyon' from the Render Using menu. Both of the text boxes will be filled with default values which should not need to be changed for the purposes of this tutorial. Press the Start Rendering button. After a few moments of processing, you sould see the message

```
Info) Rendering complete.
```

in the VMD text console. If everything worked correctly, you will end up with an image file named plot.dat.tga (on MacOS X or Unix) or plot.dat.bmp (on Windows) in your current working directory. This image is in either Windows BMP or Targa graphics format, and can be read by many programs (such as `display`, `ipaste`, `xv`, Gimp or Photoshop).

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, open the molecule file browser as described previously. Next click on the Browse button and select the `alanin.psf` file in the file browser. Once selected, press the Load button to load the structure file. Next, select the `alanin.dcd` file and load it as well. This will read the DCD trajectory frames into the same molecule with the previously loaded `alanin.psf` file.

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, animation will stop. To see it again or to fine- tune playback, use the animation controls [§4.4.3] found at the bottom of the main VMD form. Press the button that looks like > to play the animation. Use the Speed 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. The animation controls are generally similar to what you'd find on a DVD or CD player.

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 section 2.2 created. If this is not true, go back 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 [§ 4.4.7] 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, enter `all` and press the Create Rep 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 [§ 5.7] 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 5.3 for a full description of the selection command.

2.6 Comparing Two Structures

Let's start from scratch by deleting everything: use the text console and tye the command `mol delete all` and press enter. This deletes all loaded molecules and is often more convenient then selecting them and deleting them all one by one. Alternatively, you could highlight each molecule in the molecule browser, and use the Delete Molecule item in the Molecule menu to remove them one by one.

Begin 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 picking it with the mouse. Do this by changing the "Object Mode" pull-down to "Pick", and selecting "Atoms" for the pick mode in the Mouse menu. The label `HEM154:FE` should appear both on the display and in the text console.

Change the pick mode in the Mouse menu to "Bonds". To get the distance between the iron and the oxygen of the CO, click with the left mouse button first on the iron and then on the oxygen. The first click turned the FE label on 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 Mouse menu, change the pick mode to "Angles". This should cause the cursor to become a red crosshair. Click on each of the three atoms using the left mouse button. 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 [§4.4.9] 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 main 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 [§4.1.2]).

By the way, unfix the molecules and do a 'Reset View' from the Display 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.

2.7 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
```

2.8 Saving your work

After creating a set of attractive and informative representations of your molecule, you may want to save your work so that you can regenerate the scene later. There are two ways to do this in VMD:

- In the main menu, press the Save State button found in the File menu; this will bring up a browser window where you can enter a file name in which to save your work.
- In the text console, type `save_state` *filename*, where *filename* is the name of the file in which to save your work.

To restore your scene, you also have three choices:

- Use the Load State item in the File manu to select and load a previously saved VMD session.
- From the command line, start VMD with the options `vmd -e` *filename*, where *filename* was the name of the file you saved before.
- After starting VMD, from the text console, type `play` *filename*.

The most common source of problems is when VMD can't find the files you used to load the molecule. If this happens, try changing to the directory you were in when you first loaded the molecule, or edit the state file and use the full path names where you see `mol new`, `mol addfile`, or `mol load` commands.

Chapter 3

Loading A Molecule

The File menu is the primary means for loading molecules and other data into VMD. The built-in file readers will load molecular structures from combinations of topology files, coordinate files, and trajectory files. Readers are also included for data such as potential maps, electron density maps, Grasp surface data, and arbitrary 3-D geometric data from Raster3D scene files. VMD can load structures directly from Protein Data Bank over the internet, provided that a network connection is present. Entering the four-character PDB accession code in the molecule file broswer form will retrieve and load the structure over the network.

3.1 Notes on common molecular file formats

VMD natively understands several popular molecular data file formats: PDB coordinate files, CHARMM, NAMD, and X-PLOR style PSF topology files, CHARMM, NAMD, and X-PLOR style DCD trajectory files, NAMD binary restart (coordinate) files, AMBER structure (PARM) and trajectory (CRD) files including both the old format and the new formats used by AMBER 7.0, and Gromacs (e.g. GRO, G96, XTC, TRR) structure and trajectory files. These files may contain some redundant information and can be loaded in different combinations.

PDB files contains data about atoms, residues, segment names, occupancy and beta factor, and one coordinate set. PSF and PARM files contain atoms, residues, segment names, residue types, atomic mass and charge, and the bond connectivity. VMD supports four file formats used by Gromacs: GRO, G96, TRR and XTC. GRO and G96 files contain structure information including atoms, residue and segment data, and one coordinate set. CRD, DCD, TRR and XTC files contain only coordinate data (frames). It should be noted that while PDB, GRO and G96 files were designed to contain only one coordinate set, multiple files can be concatenated into one larger file to create a makeshift trajectory file which can be loaded by VMD.

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 or assigned default values. In particular, charges will be assigned a value of 0.0 if the file does not contain explicit charge information.

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 a 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. 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. Gromacs GRO and G96 files can be loaded on their own since they contain the necessary atom data and coordinates. They can also be loaded along with TRR and XTC files to obtain trajectory data. Additional coordinates from a PDB, CRD, or DCD file can be appended to the current coordinate set using the Molecule File Browser form.

3.2 What happens when a file is loaded?

When a coordinate file is loaded by itself (i.e. just a PDB, no PSF), VMD uses heuristics to replace missing values that would normally be provided by a structure file. If necessary, VMD does a distance-based bond search to determine connectivity. A bond is 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. If both structure and coordinate files are loaded, no approximations or guesses are made.

After the molecule is read in, new names are added to the coloring categories [§5.2.3], and assigned colors. Next, bond connectivity is established and the molecule is analyzed to identify its components, i.e., to determine which residues are protein, nucleic acids, and waters, etc. A search is then made to connect these into larger fragments of the same type, and summary 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 several types of fragments. Protein and nucleic fragments are homogeneous; either all proteins, or all nucleic acids. However, it is 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 will occur with terminal amino acids, zinc fingers, myristolated residues, and poorly defined structures.

3.3 Babel interface

VMD can use the program Babel, if installed, to translate a wide variety of different molecular data files into the PDB format. Not all of these have been tested for use with VMD, so your results may vary. VMD only uses Babel to read files and does not allow the use of Babel to save files to other formats. The `VMDBABELBIN` environment variable [§12.2] is used to specify the absolute path to the the Babel executable (including the executable name). For more information about Babel, see `http://smog.com/chem/babel/`. VMD currently supports version 1.6 of Babel.

3.4 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. 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.

Certain newer Raster3D objects are ignored, such as quadrics. Also, nearly all of the header information is ignored—most notably, the viewing matrix. Raster3D uses many cylinders with spherical (rounded) ends. VMD deliberately omits these rounded ends since the resultant image would be very slow to render interactively. VMD uses a fixed size palette of colors, each triplet is converted into its "nearest" indexed color. This may cause images to be colored slightly differently than expected.

Chapter 4

User Interface Components

VMD provides several methods for the user to control and interact with the molecular display. The primary methods are by using the mouse, either in the graphics window or in the different graphical user interface (GUI) *forms* provided by the program. In addition to the mouse, VMD also supports a number of more advanced input devices such as the Spaceball, Magellan, and Phantom, which provide the ability to manipulate molecules with six degrees of freedom. Some devices such as the Phantom can also provide haptic (sense of touch) force feedback. VMD also provides a text console interface for executing built-in commands or running scripts. This chapter describes how to use the mouse-based user interfaces, and some of the advanced input devices supported in VMD. The the text and scripting interface is described fully in chapter 8.

4.1 Using the Mouse in the Graphics Window

The graphics window is labeled VMD OpenGL Display and contains a view of the molecules and other objects which make up the scene. When the mouse is in the graphics display window, it may be used to perform the following actions such as:

- Rotate, translate, or scale the displayed molecules
- Select, or 'pick' atoms or other objects in order to move them, or label them
- Translate and rotate a set of atoms
- Apply a force (acceleration) to a set of atoms
- Move the lights

User-defined keyboard accelerators, or *hot keys*, are also available when the mouse is in the graphics display window. These keys are bound to VMD text commands, which are executed when the key is pressed. VMD has many built-in default hot key commands (see Tables 4.1, 4.2, 4.3 and 4.4). Users can add new hot keys, overriding default settings if desired.

4.1.1 Mouse Modes

The mouse is in one of several *modes* at any time; the current mouse mode determines the effect of pressing and releasing mouse buttons or the mouse wheel while the mouse is in the graphics window.

Each mouse mode, except the lights mode (see below), sets the mouse cursor to a characteristic shape. The mouse mode is selected via the Mouse menu.

The available mouse modes are as follows:

- **Rotate Mode** (hot key 'r')
  When the mouse is in rotate 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 or right.

  You can leave 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 select 'Stop Rotation' in the Mouse menu. Also, pressing the rotation hot key `r` or any of the other mouse mode hot keys causes rotation to stop.

- **Translate Mode** (hot key 't')
  When the mouse is in translate mode, holding the left button down allows you to move the molecules parallel to the screen plane (left, right, up, and down). To move the molecule towards or away from you, hold the middle button down and move the mouse right or left, respectively.

- **Scale Mode** (hot key 's')
  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. Scaling can also be accomplished with the mouse wheel (irrespective of the current mode setting) on computers equipped with an appropriate mouse.

- **Move Light**
  VMD provides four directional lights to illuminate the molecular scene. The lights provide diffuse lighting and specular highlights and help the user perceive surface shape in rendered objects. You can use the mouse to rotate each of the light source directions to a new position. If the light isn't on, moving it will not affect the displayed image. To turn a light on or off, use the Lights item within the Mouse menu.

- **Add/Remove Bonds**
  When the mouse is in add/remove bonds mode, clicking on atoms in a molecule will add a bond between those atoms if one is not already present, or remove the bond between those atoms if there is already a bond. The two atoms must belong to the same molecule.

4.1.2 Pick Modes

Mouse picking can be 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. Picking modes are selected from the Mouse menu.

The available pick mode actions are:

- **Center** (hot key ’c’)
  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 Mouse menu.

- **Query** (hot key ’0’)
  Clicking on an item will print out the name of the item (e.g. the atom name) to the text console window.

- **Label → Atom** (hot key ’1’)
  Clicking on an atom will toggle on/off a label for the atom.

- **Label → Bond** (hot key ’2’)
  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).

- **Label → Angle** (hot key ’3’)
  Clicking on three atoms in a row will toggle on/off a label showing the angle formed by the three atoms.

- **Label → Dihedral** (hot key ’4’)
  Clicking on four atoms in a row toggles on/off a label showing the dihedral angle formed by the four atoms.

- **Move → Atom** (hot key ’5’)
  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.

- **Move → Residue** (hot key ’6’)
  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. Holding down the ¡shift¿ key and the left mouse button while moving the mouse will rotate the atoms in the residue about the selected atom. If the middle mouse button is held down instead, the atoms in the residue will rotate about a line drawn through the picked atom and parallel to a line coming directly out of the screen. This behavior is similar to the usual Rotate mode, except that coordinates of atoms are changed.

- **Move → Fragment** (hot key ’7’)
  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. Holding down the ¡shift¿ key and the left mouse button while moving the mouse will rotate the atoms in the fragment about the selected atom. If the middle mouse button is held down instead, the atoms in the fragment will rotate about a line drawn through the picked atom and parallel to a line coming directly out of the screen. This behavior is similar to the usual Rotate mode, except that coordinates of atoms are changed.

- **Move → Molecule** (hot key '8')
  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. Holding down the ¡shift¿ key and the left mouse button while moving the mouse will rotate the atoms in the molecule about the selected atom. If the middle mouse button is held down instead, the atoms in the molecule will rotate about a line drawn through the picked atom and parallel to a line coming directly out of the screen. This behavior is similar to the usual Rotate mode, except that coordinates of atoms are changed.

- **Move → Rep** (hot key '9')
  This mode may be used to move all the atoms in a selected representation at the same time. You select a representation by clicking on one of the reps in the browser window of the Graphics form. In order to move the atoms in this rep, the atom you pick with the mouse must be selected by that rep.

  When you have clicked on an atom in the rep, move the mouse to a new position while keeping the mouse button pressed. All the atoms selected by the highlighted rep will be moved the same amount. Holding down the ¡shift¿ key and the left mouse button while moving the mouse will rotate the atoms in the rep about the selected atom. If the middle mouse button is held down instead, the atoms in the rep will rotate about a line drawn through the picked atom and parallel to a line coming directly out of the screen. This behavior is similar to the usual Rotate mode, except that coordinates of atoms are changed.

4.1.3 Hot Keys

When the mouse is in the graphics window, many commands are accessible via programmable hot keys. Hot keys allow you to do things like change mouse modes or advance the animation by a frame by simply pressing a key. There are a number of predefined hot keys, as listed in tables 4.1, 4.2, 4.3, and 4.4. They can be printed out with the command `user print keys`. The commands listed are the text commands which are executed when the hot key is pressed; these text commands are explained in section 8.3.

To add or modify a hot key, use the command `user add key` *key command*. The *key* parameter must be a single character. If *command* contains more than one word, it must be enclosed in braces so that the subsequent command words are not ignored. When that key is pressed while the mouse cursor is in the graphics display window, the associated command will be executed. Once you have a set of commands which are particularly useful and familiar for you, you will want these 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 12.3.3. Once you have such a file, put the `user add` commands in it.

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`	assign rotation 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
`9`	`mouse mode 4 13`	move highlighted rep
`%`	`mouse mode 4 10`	apply force on atom
`^`	`mouse mode 4 11`	apply force on residue
`&`	`mouse mode 4 12`	apply force on fragment

Table 4.1: Mouse control hot keys.

4.2 Using the Spaceball in the Graphics Window

VMD provides optional support for Spaceball six-degree-of-freedom input devices. The Spaceball may be used to rotate, translate, and scale molecules, using up to 6 control axes simultaneously (3 axes in translation, 3 in rotation). The Spaceball can be used independently and simultaneously with the mouse. With the spaceball in one hand and the mouse in the other, a user can perform complex picking and identification operations more efficiently, since the mouse can be left in pick mode (for example) while the Spaceball is used to perform rotations, translations, and scaling operations with the other hand.

The Spaceball can be run in one of several modes within VMD. The Spaceball interface currently provides two methods of rotation and translation, and a scaling mode. The Spaceball interface currently uses *Button 1* (known as *Function 1* in the SpaceWare driver) to reset the view, and *Button 2* to cycle through the available Spaceball interface modes.

4.2.1 Spaceball Driver

VMD interfaces to the Spaceball in one of two ways; either by communicating directly with the Spaceball using built-in serial interface software, or vendor provided drivers. Unix and Mac OS X versions of VMD use the built-in serial Spaceball driver. At startup, VMD checks for the existence of an environment variable *VMDSPACEBALLPORT*. This environment variable must be set to the Unix device name of the serial port to which the Spaceball is attached. The serial port device permissions must be set to allow the VMD user to open the device for reading and writing. In typical usage, this usually requires performing a `chmod 666 /dev/somettyname` on the appropriate device as root. One restriction with the use of the built-in Spaceball driver is that only one VMD process may safely use the Spaceball at a time. If multiple VMD sessions are started on the same machine and all are set to open the Spaceball, it will behave very erratically.

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 4.2: Rotation & scaling hot keys.

Hot Key	Command	Purpose
`Alt-M`	`menu main off;menu main on`	Show main menu
`Alt-f`	`menu files off;menu files on`	Show files 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
`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 4.3: Menu control hot keys.

The Windows version of VMD uses the vendor-provided SpaceWare driver exclusively, and operates somewhat differently from the serial interface software used on Unix. The SpaceWare software runs as a separate process from VMD and must be started and fully operational before VMD is run. At startup time VMD attempts to open the SpaceWare interface, displaying the success or failure of initialization as it occurs, with applicable diagnostic information. The SpaceWare driver provides detailed control over the sensitivity and configuration of the Spaceball. In order to use the Spaceball function keys with VMD the SpaceWare driver must be set to send button events as *Function 1* and *Function 2* at a minimum. Once set, it should be possible to cycle through the various VMD Spaceball operational modes as described below.

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 4.4: Animation hot keys.

4.3 Using the Joystick in the Graphics Window

The Windows version of VMD provides support for the Windows joystick driver, and will enumerate all available joystick devices at startup time. The joystick interface employed in VMD is quite simple, allowing the use of three control axes to translate, rotate, and scale the molecule. The joystick interface assumes a device with at least two buttons. The first joystick button resets the view in the display window, and the second button cycles through each of the available joystick modes. When VMD first attaches to each of the joysticks, they are initially disabled so that miscalibrated joysticks do not adversely affect the VMD session. Each joystick is initially enabled by pressing its second button to switch modes. All joysticks are independently controlled such that multiple joysticks can control different control axes, and multiple users could interact with the program with separate controls.

4.4 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). 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.

4.4.1 Main Form

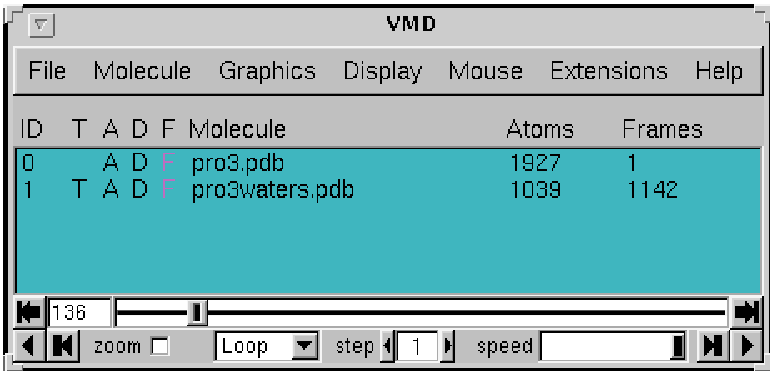

Figure 4.1: The Main form

The Main form is the main way to access other forms, load and save files, control trajectory playback, change various global program settings, access help, and to quit the program. Many of these actions can also be performed with the menu shortcut keys described in Table 4.3.

The Quit menu item exits 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.

Help

The Help menu items each start a web browser to display on-line VMD help documents. The browser is designated by the environment variable `VMDHTMLVIEWER` [§ 12.2]. Selecting a help item multiple times may start multiple browsers. The default web browser is Mozilla for Unix systems, and the built-in Explorer shell for Windows systems. The menu contains items for the VMD Quick Help page, as well as the current User's Guide, FAQ, and links to various helpful information and programs.

4.4.2 Main Form Molecule List browser

The Main 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 or fixed in place (e.g., prevented from being affected by mouse rotation commands). The form contains controls to change the status of the molecules individually or in groups.

The browser displays information about each molecule. A unique integer ID is assigned to each molecule by VMD when it is loaded. The Molecule is the file name which contained the topology information. Atoms shows the number of atoms in the molecule, and Frames gives the number of frames associated with the file.

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 specifically specified otherwise, will do their action for all the *active* molecules. The primary use for this control is to prevent some molecules from being animated. Inactive molecules will not animate when the play button is pressed.

- **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 double-clicking the appropriate flag. Only one molecule can be top at any one time, so the previous top molecule will change status when another is toggled.

Saving Trajectory Frames

Using the Save Coordinates... menu item, you can write trajectory frames to a file in one of several file formats including PDB, DCD, Amber CRD, etc. This feature may be used to write out a new trajectory in a single file after assembling many frames from different sources (such as PDB CRD, DCD or Gromacs files, or even from a remote simulation). You can also use this, in combination with the molecule file browser 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 [§ 4.4.4]. Then select the appropriate output file type in the File Type chooser, and press the Save 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 Save button in the browser, the file will be written without further confirmation. See the section on the `atomselect writexxx` [§ 8.3.2] command for information on how to write atom coordinates for an atom selection in a PDB file.

Deleting Trajectory Frames

You can delete frames from memory through a dialog box. To bring it up, start by selecting a molecule and choosing the Delete Frames... from the Molecule menu, or by double-clicking on the Frames column for that molecule in the Molecule Browser. On this is done, choose the range of frames you wish to delete with the First and Last controls, and then press the Delete button. There is no confirmation of deletions.

The Stride control allows you to keep some frames in the range using the specified interval. For example, if your range contains 10 frames labeled 0 through 9, and you use a stride of 4, the frames numbered 0, 4 and 8 will be kept. A stride of 0 (zero) implies that all frames will be deleted.

Deleting a Molecule

The Delete Molecule menu item deletes all the 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.

GUI Shortcuts

There are a few useful mouse-based shortcuts that can be used in the Molecule List browser. Here is a list:

- Double-clicking on a molecule's name brings up the Rename Molecule dialog box.
- Double-clicking on a molecule's number of frames brings up the Delete Frames dialog box.
- Triple-clicking on the T (top) in front of a molecule focusses on that molecule by making it the only molecule to be displayed (D) and active (A). Furthermore, the view is reset and the molecule gets selected in the Representations form.

4.4.3 Main Form Animation Controls

Figure 4.2: The Main form animation controls

Each molecule in VMD can contain multiple sets of atomic coordinates, which may be animated to show its motion over time. The coordinate sets can come from a molecular dynamics simulation, or simply multiple versions of the same molecular structure. The main form contains controls for animated playback of these trajectories. The controls contains several buttons which act like the buttons on a VCR or DVD player. The buttons provide a way to play the trajectory, step forward, stop, go to a specific frame, and go to the beginning or end. The status and frame counters shown in the animation control reflects the state of the *top* molecule. Commands entered via this control, however, affect all active molecules [§4.4.2], not just the top molecule, allowing concurrent animation of multiple molecules.

Animation Speed

The rate of playback can be controlled in two ways. The Step control changes the animation step size. By default, the frame step is 1, so each step of the playback increases (or decreases) the animation frame number by one. If the frame step is 5 then the animation proceeds five times faster because only a fifth 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. The start button 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 frame counter text area next to the start button 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. The animation controls are all relative to the top molecule [§4.4.2].

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.

4.4.4 Molecule File Browser Form

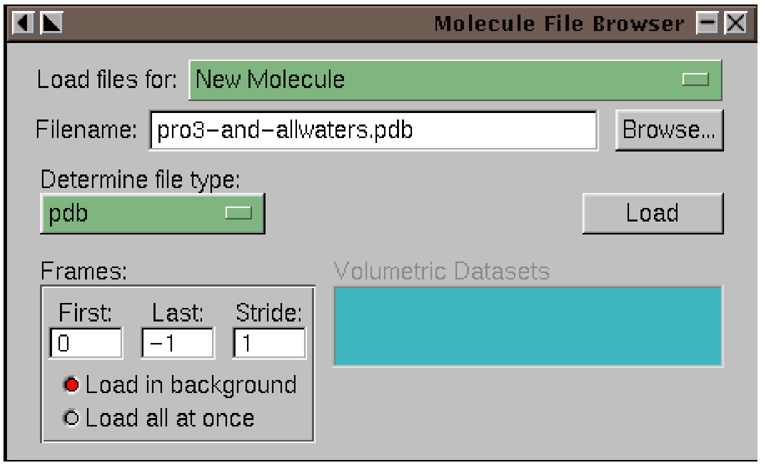

Figure 4.3: The Molecule File Browser form

The Files form is used to load a file from disk into a new or existing VMD molecule. It can be brought up by choosing New Molecule... from the File menu, or by hilighting a molecule in the

Main form4.4.1 and choosing the Load Data Into Molecule... menu item. Once the form appears, select the file you want by using the file browser or by typing the filename into the text entry area. By default VMD will try to guess the type of file you are loading by matching the filename extension with one of the file reader plugins in the file type list (the available file types are described in Chapter 3). If VMD is unable to guess the appropriate file type or guesses incorrectly, you must select it from the list manually.

You can control into which VMD molecule you want to load your data by selecting it from the Load files for: popup menu at the top of the form. If the file being loaded is intended for a new molecule, select New Molecule instead. If the file being loaded contains additional coordinate frames, electron density map, or other ancillary data for an existing molecule, choose the appropriate molecule from the selection list at the top of the form. If the file being loaded contains trajectory frames , you have the option of loading a subset of the trajectory skipping ranges or strides of frames rather than the whole thing. You can also select for VMD to load all frames before continuing on, or to load them in the background so that you may continue to interact with the menus and forms while it loads additional frames . If the file being loaded contains multiple volumetric data, you may select which data sets you would like to load.

Once you have selected the file to be loaded, the appropriate file type, and the way it will be loaded, press the Load button and VMD will being loading the selected file. Any informational messages, errors or warnings which occur while loading the file will appear in the text window.

Reading Trajectory Frames

VMD can read in new coordinate sets from one of several file formats such as PDB, CRD, DCD, or Gromacs files. The new coordinate sets are appended to the end of the stored frames for the selected molecule. Loading coordinate data is like loading any other file, select it with the file browser make sure the file type is set correctly for the file being loaded, and then press the Load button.

By default, VMD will load all of the frames contained in a coordinate or trajectory file.

Sometimes you may not want to read in a whole coordinate or trajectory file. For example, you may only want the last frame, or every tenth frame. You can do this by changing the options in the Frames control of Files form. The Frames controls consist of three numeric input fields labeled First, Last, and Stride. These make it possible to use a subset of the frames, starting at frame First and selecting every Stride frames until the Last is reached. For instance, to select every fifth frame between frames 14 and 98, set:

- First to 14
- Last to 98
- Stride 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 First to -1 is the same as starting at the first frame, Last = -1 is the same as ending at the last frame, and Stride = -1 is the same as taking one step.

4.4.5 Mouse Menu

The Mouse menu indicates and controls the behavior of the mouse when the mouse moves and clicks within the graphics window. Mouse clicks and drags can affect VMD in one of two ways. It can change the *view* of the scene, either by rotating, translating, or scaling. It can also *pick* objects in the scene, causing some further action to be taken. These behaviors are all reflected in the state of the Mouse menu.

Below, we describe the main parts of the Mouse menu.

Mouse modes

The top three menu items select whether the mouse will rotate, translate, or scale the scene when the user clicks and drags with the left mouse button.

Pick modes

These modes, located right below the mouse modes in the Mouse menu, control how the mouse affects objects in the scene (as opposed to how the mouse changes the *view* of these objects). Note that any time you choose a new pick mode, the current mouse mode changes to "Rotate".

- **Center** changes how VMD rotates and scales the scene. To get a feel for how this works, select "Center" from the Mouse menu, then click on an atom in the scene. If you now rotate the scene by clicking and dragging with the left mouse button, the scene should rotate about the picked atom. If you change the view mode to "Scale" using the "View Mode" pulldown menu, the scene will expand while keeping the picked atom in view. The picked atom will remain the center atom until a new atom is selected as "Center", the "Reset View" button is pressed, or a new molecule is loaded.
- **Query** prints information about the item (e.g. the atom name) on the text console window.
- **Label** adds labels to atoms in the scene. Labels include atoms, bonds, angles, and dihedrals. These labels require, respectively, one, two, three, and four atoms to be picked. For the latter three label types, the numerical value of the geometric label is displayed, along with a stippled line connecting the picked atoms. The units for "Bonds" corresponds to whatever units the coordinate file is written in. "Angles" and "Dihedrals" are measure in degrees.

  Labels can then be manipulated through the Labels form.
- **Move** changes the actual coordinates of atoms in the scene. Note that this is different from simply changing the view. Clicking on one of the buttons in the Mode Mode menu selects what group of atoms to move. "Atom" moves only the selected atom. "Residue" moves all atoms in the same residue (e.g., amino acid or nucleotide) as the selected atom. "Fragment" moves all atoms connected by a bond to the picked atom. "Molecule" moves every atom in the molecular structure. "Highlighted Rep" is the most flexible; it moves all atoms in the highlighted representation in the browser window of the Graphics form.

  Atoms are moved by clicking and dragging with the left mouse button. If the shift key is held while the mouse is moved, the affected atoms are *rotated* about the selected atom. Rotating atoms with the left button rotates about the x or y axis of the screen; rotating with the middle or right button rotates about an axis perpendicular to the screen.

  Note that there is currently no way to undo Move operations, so the atom coordinates should first be saved to a file.

- **Force** applies a force to selected atoms in a running simualtion. These forces will be visible only if an IMD connection has been established. Clicking and dragging with the left mouse button will apply a force to the selected Atom, Residue, or Fragment, as in Move Mode. Clicking with the middle or right button will cancel the force on the selected atoms.

- **Move Light** allows the lights to be positioned around the scene. Individual lights are turned on or off in the Display form. Selecting one of the lights in the Move Light menu rotates the selected light about the origin. The Move Light Mode can also be cancelled by changing into any other pick mode or mouse mode.

- **Add/Remove Bonds** adds a bond between two clicked atoms if there is not one present, and removes the bond otherwise. Both atoms must belong to the same molecule.

4.4.6 Display Menu

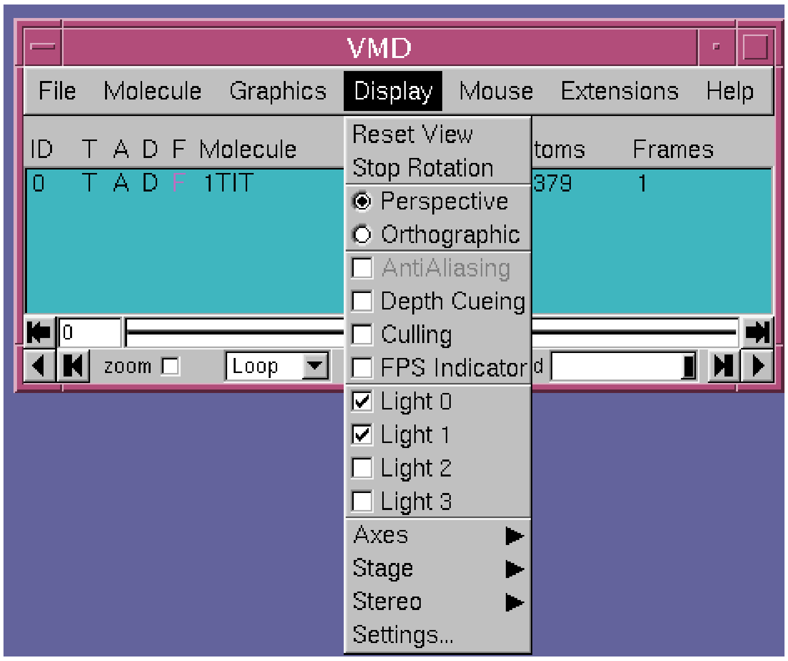

Figure 4.4: The Display menu

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

- **Reset View** – This menu item can be used to force VMD to reset the scene back to the default viewing orientation and scale as is done when a molecule is first loaded.

- **Stop Rotation** – This menu item stops autorotation of the scene. The scene can be autorotated by quickly dragging the mouse while briefly depressing and releaseing the mouse button, leaving the scene spinning until it is stopped either by this menu item or by further mouse interactions.

- **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. Note that several of the supported external rendering programs do not support orthographic rendering. As such, it may be necessary to "fake it" by translating the scene far away from the camera, and apply a zoom factor. This has the effect of significantly reducing the perspective, while not truly an orthographic view.

- **Antialiasing** – Turns antialiasing on or off. Antialiasing helps smooth out the jagged appearance of displayed geometry resulting from the inherently discrete pixels on the display device. The antialiasing feature is only available on platforms which support full-screen antialiasing, sometimes known as "multisample antialiasing". Several SGI and Sun graphics systems fully support this feature. On platforms lacking the multisample capability, there may be alternate ways to perform full-screen antialiasing by selecting an option in the display driver setup. Windows machines most commonly place these controls in the display driver configuration panel.

- **Depth Cueing** – Turns depth cueing on or off. Depth cueing causes distant objects to blend into the background color, in order to aid in 3-D depth perception. The depth cueing settings controlled in the Display Settings window. The **Cue Mode** parameter controls which type of fog equation is used. The **Linear** depth cueing mode provides a simple depth gradient with a defined starting point and endpoint. The **Exp** and **Exp2** depth cueing modes take a density parameter, and generally blend into the background color much more sharply than the linear depth cueing mode. Scaling up the molecule will increase the amount of depth cueing effect that is visible, since it will occupy a larger depth range. Scaling the molecule size down decreases the depth cueing effect. Translating the molecule into and out of the screen will cause it to blend into and out of the background color.

- **Culling** – Turns backface culling on or off. This feature is primarily used to accelerate rendering performance on software based implementations of OpenGL, such as Mesa. Backface culling actually reduces performance on some hardware renderers, so you'll have to use your own best judgement on whether or not it is helpful to use on your specific computer system.

- **FPS** – This option enables or disables on-the-fly display of the achieved VMD rendering frame rate. The frame rate is displayed in the upper right hand corner of the graphics window when it is enabled.

- **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 section 4.1.1 for more discussion regarding lights.

- **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.

- **Stereo, Eye Sep, and Focal Length** – These controls are found in the Display Settings form. These controls set the stereo mode and parameters; stereo is discussed fully in chapter

6. The Stereo chooser changes the stereo mode, while the Eye Sep and Focal Length controls change the eye separation distance and the focal length, respectively.

- **Cachemode** – The Cachemode toggle controls whether or not VMD uses a display list caching mechanism to accelerate rendering of static geometry. This feature can be extremely beneficial for achieving good interactive display performance on tiled display walls, and for remote display over a network. Caching cannot be performed while animating trajectories, so the performance benefit is only possible interactive rotation and zooming of static molecular structures.

- **Rendermode** – The Rendermode chooser controls which low-level rendering method VMD uses. The **Normal** rendering mode is the default VMD rendering algorithm based on standard fixed-function OpenGL. The **AlphaBlend** rendering mode is an alternate implementation which requires the Sun Global Alpha OpenGL extension. This feature allows VMD to use a high quality alpha-blended transparency mode with no performance loss compared to the screen-door method it normally uses. On machines lacking the Sun Global Alpha extension, this rendering mode will not be available. The **GLSL** rendering mode uses OpenGL Programmable Shading Language to implement real-time ray tracing of spheres, alpha-blended transparency, and high-quality per-pixel lighting for all geometry. On machines with high performance graphics boards supporting programmable shading, the **GLSL** rendering mode provides quality on par with many of the external software renderers supported by VMD but at interactive display rates.

- **Clipping Planes (Near Clip and Far Clip)** – These controls are found in the Display Settings form. Only those parts of the scene between the near and far clipping planes are drawn. The display clipping planes also set the depth cueing start and endpoints. Objects at the near clipping plane are distinct and crisp, objects at the far clipping plane will be blended into the background. Clipping planes positions 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, it may be useful to set the near clip plane much lower than the default value. This makes the geometry "pop out of the screen" a bit more, and can be used for greater dramatic effect.

- **Screen Height (Hgt) and Distance (Dist)** – These controls are found in the Display Settings form. 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.

  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 distance is negative the origin is located between the viewer and the screen, if it is positive, the screen is 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.

  Figure 4.5 describes the relationship between the screen height, the screen distance, and the world coordinate space.

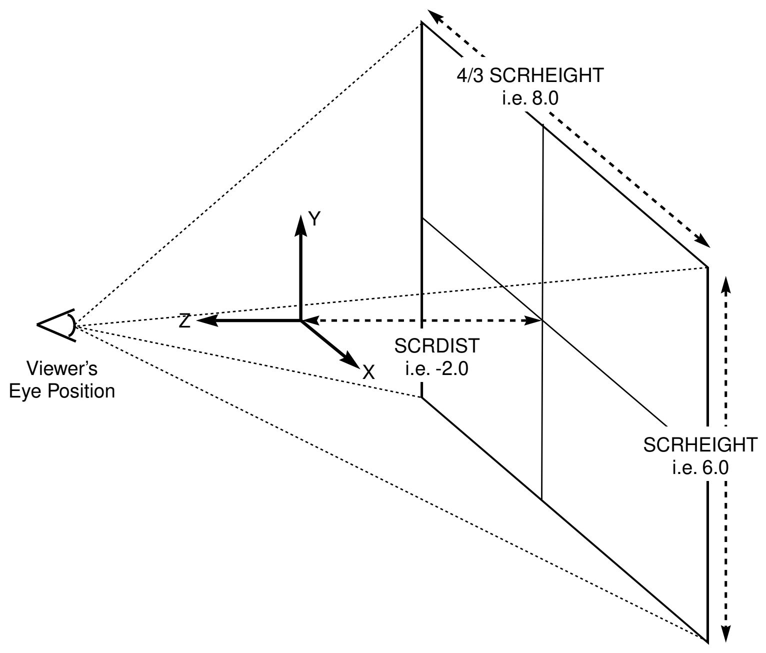

Figure 4.5: Relationship between screen height (SCRHEIGHT), screen distance to origin (SCRDIST), and the viewer

4.4.7 Graphics Form

The Graphical Representations or "Graphics" form controls how molecules are drawn. Molecules are represented by *reps*, which are defined by four main parameters: the selection [§ 5.3], the drawing method [§ 5], the coloring method [§ 5.2], and the material [§ 5.2.5]. The selection determines which part of the molecule is drawn, the drawing method defines which graphical representation is used, the coloring method gives the the color of each part of the representation, and the material determines the effects of lighting, shading, and transparency on the representation.

Draw Style Tab

Select a molecule for editing using the 'Selected Molecule' chooser at the top of the form. The browser below this chooser lists the reps available for the molecule. Each line of the browser summarizes information about the drawing method, the coloring method, and the selection. Below this browser, choosers and a text input filed reflect the current state of the rep, and provide controls for changing the properties of the rep. Each drawing method has specific controls which will appear when it is selected. When the 'ColorID' coloring method is selected, 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 16.

Changing a rep. To change a representation, select it in the representation browser. The atom selection for that rep will appear in the Selected Atoms text area and the controls will update to reflect the current settings. Changing the settings will immediately affect the displayed representation if the Apply Changes Automatically check box is selected. When it is disabled updates will only

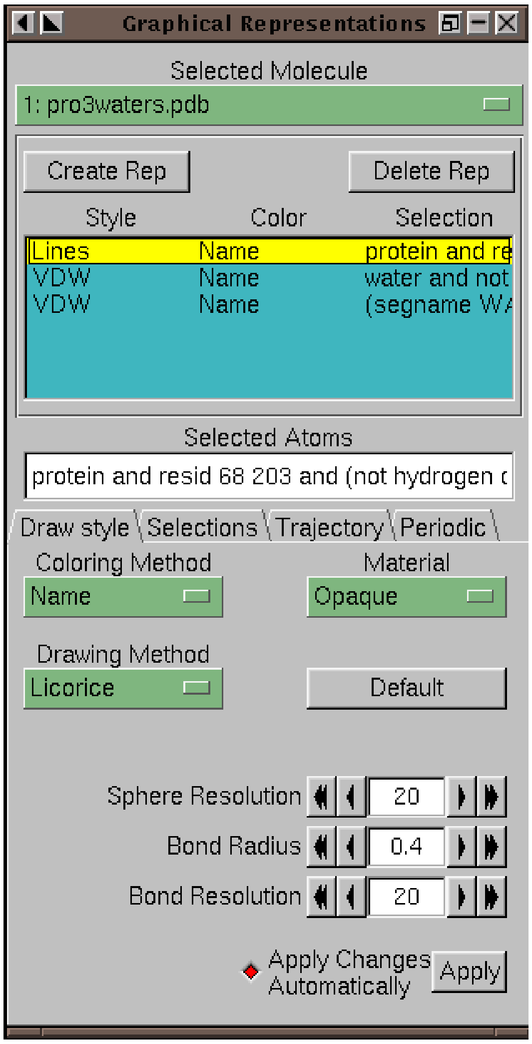

Figure 4.6: The Graphics form (in Draw Style mode)

occur when the Apply button is pressed. Changing the drawing method brings up method-specific controls and defaults. If you go back to the previous draw style, VMD restores any changes that you may have made to the settings. Pressing the Default button will restore the default settings. The display will be updated after every change.

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

Deleting a rep. To delete a representation, select the representation in the browser and press the Delete button. Bear in mind that this does not delete the molecule, it only deletes one of its graphical representations.

Hiding a rep. To hide a rep, double-click its entry in the browser. The text will turn pink to indicate that the rep is hidden. Turn the rep back on by double-clicking again on the same line. Hidden reps will not recalculate their geometry if the animation frame changes until the rep is turned back on.

Selections Tab

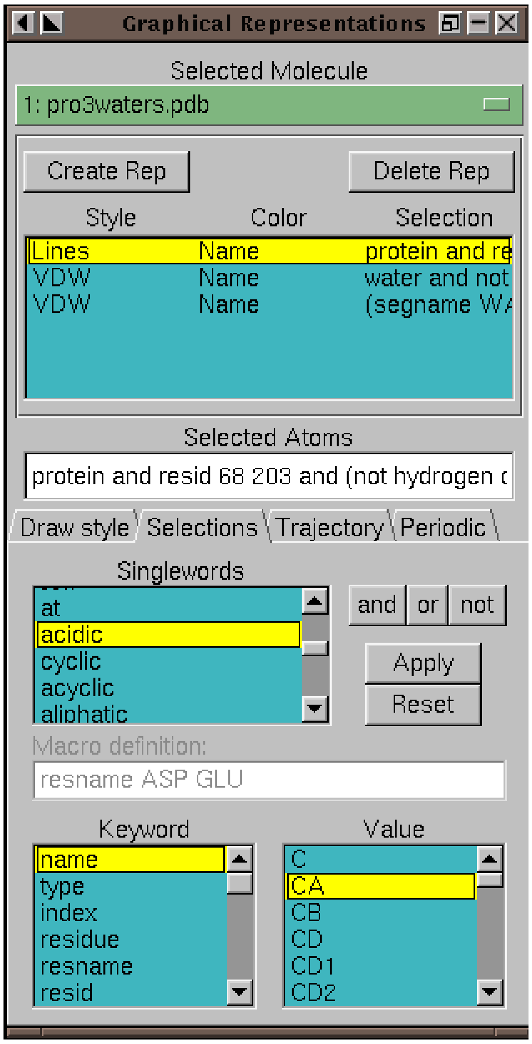

Figure 4.7: The Graphics form (in Selections mode)

The Selections tab provides access to browsers which display the lists of atom names, residue names, and so forth for the selected molecule. When the Selections tab is pressed, several browsers appear in place of the drawing and coloring method controls. These are used to list the available keywords, macros, and values for use in selecting atoms for the associated representation. The top browser lists singlewords and macros such as `all`, `water`, and `hydrophobic`. The botton left browser contains a list of the keywords and functions understood by the selection command [§5.3]. 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 bottom-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. Press Apply to actually change the atom selection for the current rep. Press Reset to restore the atom selection to its original value.

The Selections tab also shows the atom selection macros that have been defined. These macros let you define a commonly used atom selection as a single word so that it can be inserted into a rep more conveniently. Atom selection macros can currently be defined only through the Tcl [§8.3.2] or Python [§9.3] text interfaces; see these sections for details.

Trajectory Tab

Selection and Color auto-update. When an atom selection such as `water within 3 of protein` is made, the atoms in the selection are computed for the current animation frame. When the animation frame changes, the selection is not normally recalculated; thus the displayed atoms may not correspond to those that would be selected if the atom selection were performed for the new animation frame. If the Update Selection Every Frame checkbox is highlighted by clicking on the checkbox, then the atom selection for the current rep will be recalculated every time the animation frame changes. Similarly, if the Update Color Every Frame checkbox is activated, the color will be recalculated for every frame.

Color Scale Data Range. Several of the coloring methods available in Draw Style tab operate over data fields that have no specifically implied range of values. It is often useful to highlight a very specific range of data values, in order to accomplish this the color scale range can be manually set to a specific starting and ending values, overriding the default behavior which is to autoscale from the minimum value to the maximum value. This feature is particularly useful when displaying trajectories, since the range of values of interest may be quite different from the autoscaled range for a single frame or all frames.

Draw Multiple Frames. Draw multiple trajectory frames or coordinate sets simultaneously. This setting allows the user to select one or more ranges of frames to display simultaneously. The frame specification takes one of the following forms **now**, *frame_number*, *start:end*, or *start:step:end*.

Trajectory Smoothing. The Trajectory Smoothing Window Size is used to control the application of a per-representation windowed-averaging smoothing function. This simple smoothing feature can be used to eliminate much of the thermal noise inherent in a molecular dynamics trajectory so that one can more easily see structural changes occuring over a wider time scale. The window size parameter controls how many frames are averaged together to produce the coordinates which are actually displayed. One important consideration when using the trajectory smoothing feature is that VMD does not take periodic boundary conditions into consideration when smoothing trajectory coordinates, so any atoms which wrap around within the span of the window will cause erratic motions in the displayed representation. This can be avoided by unwrapping trajectory coordinates prior to loading into VMD or by using atom selections to eliminate atoms which wrap around.

Periodic Tab

The Periodic tab controls the display of periodic images of a molecule. In order to display periodic images, a molecule must have unit cell information set for `a`, `b`, `c`, `alpha`, `beta`, and `gamma`, which are discussed in section 8.3.21. When the proper unit cell information is present, the periodic display feature can show periodic images of the unit cell by transforming and rendering additional copies of the structure. The current implementation of this feature doesn't provide for complex crystallographic symmetry operations. Unit cells that can be replicated by translation along the three unit cell axes are the only ones supported presently. The periodic images to be drawn are selected by enabling images in one or more of the six faces of the unit cell. The Self image selects the untranslated unit cell itself, so that one my render a representation consisting of only replica images. This feature allows the unit cell and its periodic images to be displayed using different materials, for cases where it is desirable to draw more attention to the original unit cell or to one ore more of the replicas. The Number of Images counter controls how many replicas are made in each of the six directions. Some file formats read by VMD may not include unit cell information, in such cases you can use the scripting interface to set the unit cell information manually. PDB files containing CRYST1 records are an example of a file format that provides unit cell information.

4.4.8 Labels Form

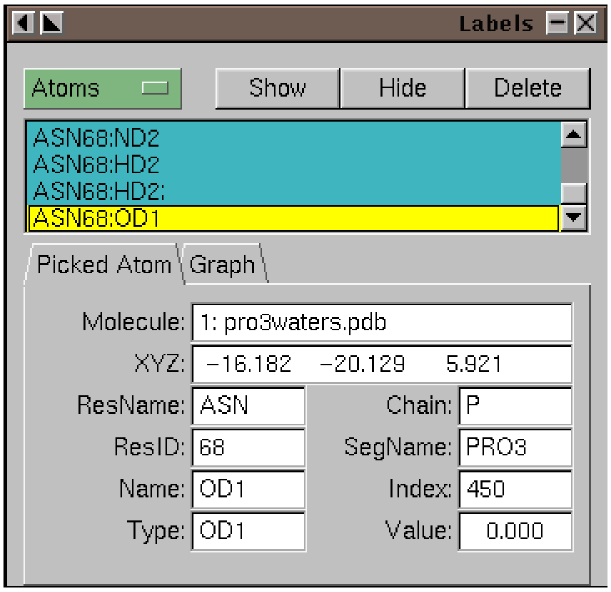

Figure 4.8: The Labels form

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 a mouse, as discussed in section 4.1.2. 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.
- Springs, which are shown as dotted lines between the atoms with the bond length displayed at the bond midpoint;

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.)

Pick information

The Picked Atom tab displays information about the last atom picked by the mouse. This information is also echoed to the vmd console. The data in the will remain in until a new label is selected by the mouse. Information about the following fields is identified:

- Molecule - the name of the molecule referenced
- XYZ - the position of the atom in 3D space
- 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.

- 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
- 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
- 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).
- Value - the calculated length of bonds, angles, or geometric measurements performed by the selected label

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 calls a Tcl script to plot the data for the selected labels. You can create your own script to handle label plotting simply by creating a Tcl proc named `vmd_labelcb_user`. The proc should accept three arguments. Have a look at the default scripts in the VMD scripts directory, found in the VMD installation directory under `scripts/vmd/graphlabels.tcl`. If no supported graphing program is available, a dialog box will be presented which will allow you to save the values of the labels to a file.

4.4.9 Color Form

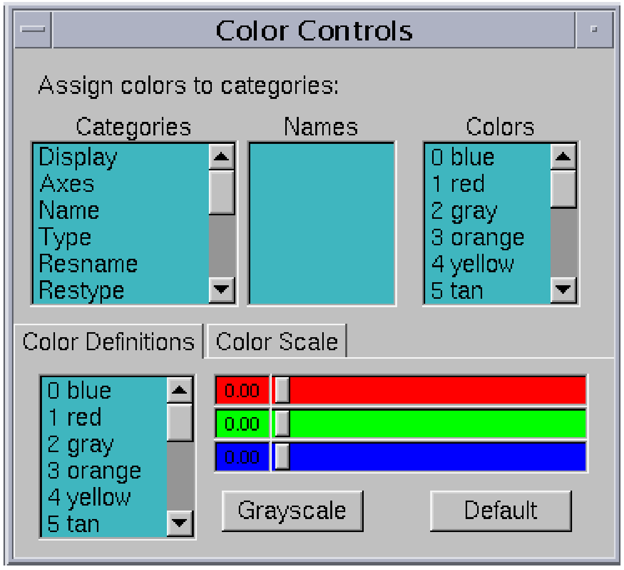

Figure 4.9: The Color form

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 [§5.2].

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

Changing the RGB Value of a Color

The Color Definitions tab at the bottom of the Color menu lets you change the RGB definition of the 17 palette colors. Select a color to edit using the browser at the bottom left corner of the menu, then slide the three sliders to set the amount of each red, green and blue component. Default restores the original color definition, and Grayscale toggles whether or not the three sliders will move together as a unit. Color definitions are immediately updated in the graphics window, so you can see the result of your editing right away.

Color Scale

Several of the coloring methods in the graphics form (e.g., Beta, Index, Position) are used to color a range of values, as opposed to a list of names. The actual coloring is determined by the color scale [§5.2.4].

The color scale used to assign these colors is set in the Color Scale tab of the color menu. Choose one of the ten color scales from the chooser, and adjust the Offset and Midpoint sliders until the color scale shown at the bottom of the tab is as desired.

4.4.10 Material Form

This form is used to create and modify material definitions. The material definitions created here will show up in the pulldown menu in the Graphics Form, allowing you to apply a material to a given representation.

The upper left corner of the Materials Form contains a browser listing all the currently defined materials. Below this browser is a set of five sliders which indicate the current materials settings for the material highlighted in the browser. Highlighting a different material in the browser by clicking with the mouse will update the settings of the sliders. Conversely, moving the sliders will change the definition of the the currently highlighted material in the browser. Pressing the "Default" button will restore either of the first two materials, "Opaque" and "Transparent", to their original settings.

To create a new material, press the "Create New" button in the upper right corner of the Form. A new material with a default name will be created and displayed in the browser window. This name can be changed at any time to something more descriptive by typing in the input box to the right of the material browser and pressing "enter" (note that the names of "Opaque" and "Transparent" cannot be changed). You can now edit the properties of this material using the

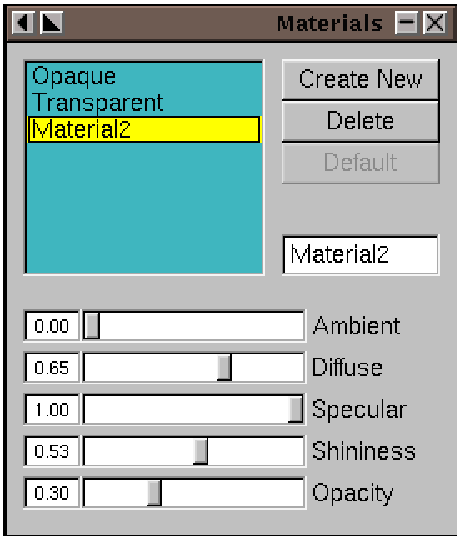

Figure 4.10: The Material Form

sliders at the bottom of the form. All materials in the materials browser, including those you create, will appear in the Material pulldown menu in the Graphics form.

To experiment with the material settings, first create a new material so that you can edit its values. Next, load any molecule, change its drawing method to VDW representation, and using the Material pulldown menu in the Graphics form, change the representation's material to the material you just created. Now, go back to the Materials form, highlight the new material in the browser, and change some of the values in the sliders. The effect of changing shininess should be especially dramatic.

4.4.11 Render Form

This form is used to export the currently displayed graphics scene to an image file or to a geometric scene description file suitable for use by one of several external renderers, which can produce a final image. The supported rendering packages are listed in table 7.1. See Chapter 7 for detailed information on how rendering is performed using external programs, as well as information on 3-D printing and other uses of the exported scene description files.

The rendering process works in two stages. The first stage exports the displayed VMD scene to a text or image file in the selected format. The second (optional) stage renders the exported file, potentially displaying the results when complete. The exported file is named in the 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 Browse button, which opens up a file browser. Pressing the Start Rendering 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.

Some of the rendering commands have been set to call a display program on the rendered image when it is completed. VMD will wait for the display program to finish, which causes VMD to freeze until the display program closes, so you may want to run the job in the background. This can be done (on Unix) by enclosing the existing text with `()`'s and putting an `&` at the end. For example,

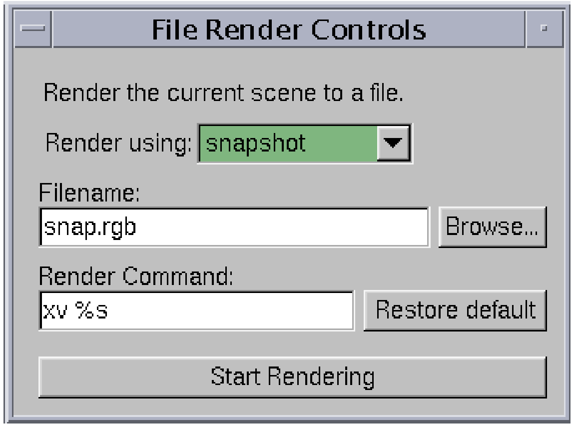

Figure 4.11: The Render form

the way to make the Raster3D render command run in the background is:

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

4.4.12 Tool Form

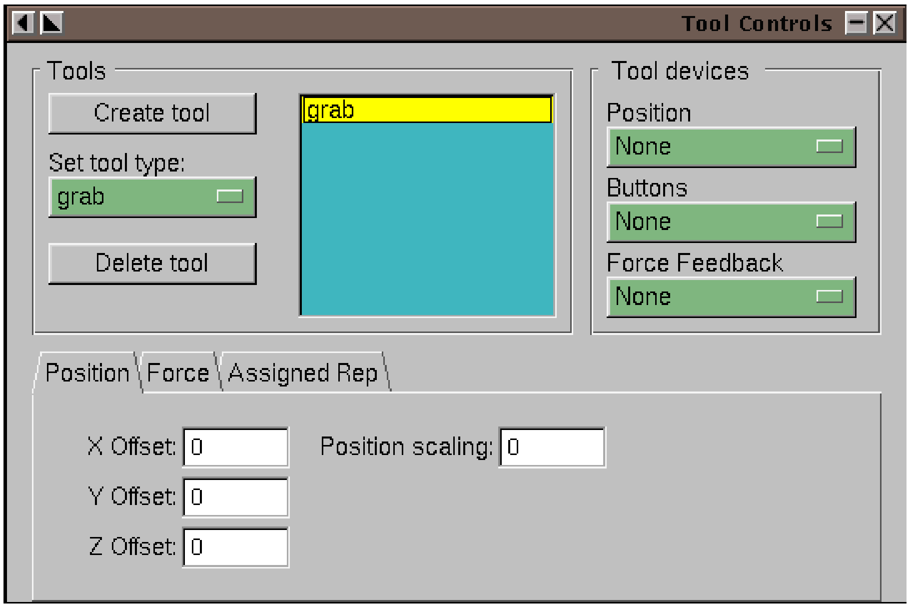

Figure 4.12: The Tool form

The Tool form is used to set up external 3D pointers, buttons, force-feedback devices, and the VMD "tools" that they control. VMD communicates with input devices through CAVElib, FreeVR, or via Virtual Reality Peripheral Network (VRPN), or with direct operating system interfaces. Since VRPN provides networked device abstraction, VMD doesn't have to be running on the same computer that VRPN devices are attached to. With VRPN, you may use buttons, trackers, and also force-feedback (haptic) devices such as the PHANToM. In the CAVE or FreeVR, VMD recognizes two types of devices: buttons and trackers. The built-in Spaceball driver can also be used to control tools.

Configuring input devices

To use input devices with VMD "tools", you need a *sensor configuration file*, in your home directory called `.vmdsensors` (see the VMD Installer Guide). In this file, any number of devices can be specified, using a *universal sensor locator* (USL). The format for a USL is as follows:
USL – type`://`*place*`/`*name*`:`*nums*

- *type* – the type of sensor (`vrpntracker`, `vrpnbuttons`, `vrpnfeedback`, `cavetracker`, `cavebuttons`, or `sballtracker`)
- *place* – the machine that controls it. Devices that cannot yet be used on arbitrary computers over the network must have the keyword `local` here to be compatible with future versions.
- *name* – the name of the device within that machine. If multiple devices can't currently exist, such as with the CAVE, then a standard name should be used, such as `cave`, so that the same USL will make sense in the future, when multiple devices are allowed.
- *nums* – a comma-separated list of numbers of devices belonging to that names (optional, defaults to zero). Some devices demand only one number or a specific number but button devices should work correctly now.

The lines of a sensor configuration file come in four flavors:

- *Comments* begin with `#` and are ignored.
- *Empty lines* are also ignored.
- *Device lines* have the form `device` *name USL*, where *name* is the name that VMD will use to refer to the device, and *USL* is the device's USL.
- *Options* tell VMD how to use the most recently listed device. Currently, there are four supported options:
  - "`scale` x" scales the position of a tracker by a factor x.
  - "`offset` x y z" adds a constant vector to the position of a tracker.
  - "`rot right—left` A_{00} $A_{01} \ldots A_{33}$" multiplies the orientation matrix returned by a tracker on either the right or the left by the matrix A.
  - "`forcescale` x" multiplies the force applied to a force-feedback device by the amount x.

Here is a simple example, showing some of the things you can do with a sensor configuration file, for a more complete example, please refer to the .vmdsensors file that came with your VMD distribution:

```
### Sensable PHANTOM via VRPN
### http://www.sensable.com/
### The Phantom haptic device connected to the computer "odessa"
device phantomtracker   vrpntracker://odessa/Phantom0
```

```
scale 10
rot left 0 0 -1 0 1 0 1 0 0
device phantombuttons   vrpnbuttons://odessa/Phantom0
device phantomfeedback vrpnfeedback://odessa/Phantom0
```

Using Tools

There are several different "tools", each of which can be used with any of the input devices[1]:

- The **Grab Tool** mimics a pair of tweezers, and can be used to move molecules around on the screen without any keyboard or mouse commands. Pressing a button connects the 3d cursor to the nearest molecule. Then, moving or rotating the tracker will cause the molecule to move or rotate around on the screen.

- The **Rotate Tool** is a tool for precisely rotating molecules with haptic devices. When a button is pressed and released, the cursor is again connected to the molecule. With this tool, however, the center of the molecule is fixed, and the end of the haptic pointer is forced to lie on the surface of a sphere about this center. Moving the device around the surface of the sphere rotates the molecule, and another button click releases the molecule. There are detentes — like the clicks commonly felt in a 2d dial — on the surface of the sphere, arranged so that the user can rotate the molecule to precise 90-degree points. If the user holds down the button for a while initially, he can feel the sphere and the detentes, but do not affect the molecule. This "preview mode" allows the user to find a good point from which to start the rotation.

- The **Joystick Tool** is the three-dimensional equivalent of a Joystick, for haptic devices. Pressing the button creates a virtual "spring," holding the device to its current location. If it is pushed away from this point in some direction, the selected molecule starts sliding in that direction, with a velocity that is proportional to the displacement of the device. The joystick tool shows how a three dimensional input device can be used to supply relative (differential) coordinates instead of absolute coordinates.

- The **Tug Tool** is a tool that allows interaction with running molecular dynamics simulations. Pressing the button connects the device with a simulated spring to the nearest atom, and pulling on it adds a force to the simulation. If a haptic device is being used, the user will feel a force on his hand that is proportional to this force. In this way, the tug tool implements something like the click-and-drag that is commonly used with windowing systems.

  If an atom selection is assigned to the Tools, the the Tug Tool will apply a force to all the atoms in the selection. The force applied will be proportional to the masses of the atoms in the selection, so that all atoms experience the same acceleration. When a Tool Selection has been assigned, the Tug Tool will always affect that selection, even if the button is pressed far from any atoms in the selection; this is intended to make it easier for the user to apply forces only on those atoms he/she intends to steer.

- The **Spring Tool** also allows interaction with running molecular dynamics simulations. It works like the Tug Tool *except* that when the button on the tracker is released near an atom,

[1] The tools have been designed to allow VMD to use haptic devices. Most of the tools can give force-feedback to the user, but none of them require haptic devices in order to operate.

the simulated spring is connected to it. See section 4.4.8 for information on viewing and modifying the list of active springs.

- The **Pinch Tool** is similar to the Tug Tool, except that force is applied only along the axis defined by the orientation of the tracker.
- The **Print Tool** is meant to be used as a debugging aid when one first sets up VMD for use with VRPN, the CAVE, or other 3-D input devices. When enabled, this tool prints text messages to the VMD console indicating the current position of the tool in question. This tool is useful when calibrating the various transformation matrices that operate on tracker position and orientation data (whether in VMD or in VRPN, CAVElib, etc).

To add a new tool to a VMD session, open the Tool form and click the **Create Tool** button. The tool's number and type are displayed in the list to the left. Devices can be added to the tool by selecting them from the **Add Device** menu, or removed with the **Delete Device** button. Some of the options that can be specified in the sensor configuration file can be edited in using the controls below, and the tool's type can be changed with the **Type** menu.

4.4.13 Simulation Form

VMD has the ability to work with a molecular dynamics program running on another computer, to interact with and display the results of a simulation as they are calculated. A major feature in VMD is the ability to add perturbative steering forces to a running simulation, which are incorporated directly into the dynamics calculation; we refer to this capability as Interactive Molecular Dynamics (IMD). In order to run and IMD simulation it is necessary to have a molecular dynamics program that supports the IMD communication protocol. To date, two such programs exist; NAMD, developed at University of Illinois, and Protomol, developed at Notre Dame. The rest of the discussion in this chapter assumes you are using NAMD. See the NAMD home page[2] for information on obtaining NAMD.

Interactive Molecular Dynamics

IMD works by establishing a TCP connection between VMD and the molecular dynamics simulation program. NAMD, or whichever MD program is being used, acts as the server. In order to prepare NAMDto accept VMD's IMD connection request, NAMDmust be configured to listen for incoming connections on a network port. Once NAMD has started up, may wait for the user to connect through that port. When VMDconnects to NAMDsuccesfully, the simulation commences. Before connecting to the remote simulation, the VMDuser must first load a molecule corresponding to the system being simulated. The structure file should correspond to the same structure file used by NAMD. Once the molecule is loaded and NAMD has been started and is listening for a connection, you are ready to connect to the simulation and start receiving coordinates. To establish a connection, open the Sim form, enter the hostname on which NAMD is running and the port on which NAMD is listening for incoming connections, then press the Connect button to establish the connection. If NAMD is running on several distributed nodes, VMD must connect to the root node on which NAMD initially started out.

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

IMD Using the 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 form are two entry fields and a button for establishing a connection to a running MD simulation. Enter both the hostname on which the simulation is running, and the port on which the simulation is listening, then press the Connect button to establish the connection. See the text console for possible error messages and status updates. Below the connection display is a browser used to set some connection parameters. 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, thereby decreasing the amount of network processing and rendering that needs to be done.
- **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 timesteps. When this is 0, then when VMD receives a new timestep it *replaces* the last timestep in the animation list with the new timestep, instead of *appending* it. When it is set to some number N larger than 0, then every Nth timestep received from the remote simulation will be appended to the molecule.

Parameters may be changed by entering text into the appropriate entry field and pressing `<return>`. When a new value is entered, a command is sent to the remote simulation to change it. There may be some delay between when the simulation gets commands, acts on them, and the results propagate back to VMD. Connection state is shown in the center of the form. The simulation status text area displays energy values for the system being simulated (kinetic, electrostatic, etc.), as well as the current timestep and the temperature. It is updated each time a new coordinate set (timestep) is received by VMD. The Stop Sim button will terminate the remote simulation, but will not delete the molecule in VMD. The Detach Sim button will sever the connection between VMD and NAMD, but will allow the simulation to continue running.

4.4.14 Sequence Form

This form is used to list the residue sequences of proteins and the base sequences of nucleic acids, and to select residues/bases from the sequence list for highlighting in the 3-D structure in the main VMD window. When residues are selected in the main VMD window, the corresponding residue is highlighted in the sequence list in this form. Color-coded protein structure information is displayed for amino-acid residues, and B-factor information is displayed for all residues. In this section, "residues" refers both to amino acid residues in proteins, and to nucleotide bases with associated backbone in DNA and RNA molecules.

Sequence information

The Sequence form contains a vertical listing of the residue sequence of a loaded molecule. The Molecule pop-up menu control chooses which molecule to display the sequence of, the current 'top'

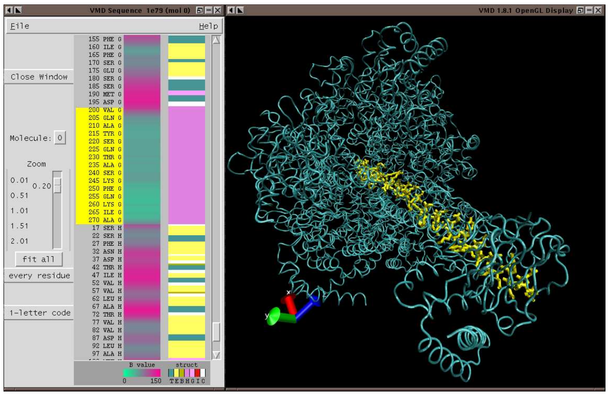

Figure 4.13: The Sequence form

molecule is displayed the first time the Sequence form is opened. The name and molecule number of the sequence displayed is shown in the title frame of the Sequence form.

For each residue displayed, the form lists: residue number, residue name/code, and chain letter. If no chain is specified, chain letter is set to “X”. To the right of this are two color coded columns, “B value” and “struct”. “B-value” shows the contents of the B-value (temperature factor) field. The “struct” field shows protein secondary structure; select Help:Structure Codes from the form menu, or see Table 4.5, for an explanation of the single letter codes in the color key.

Code	Description
T	Turn
E	Extended conformation
B	Isolated bridge
H	Alpha helix
G	3-10 helix
I	Pi-helix
C	Coil

Table 4.5: Description of secondary structure codes in the Sequence form.

Selecting residues from the Sequence form listing

Click anywhere in the vertical listing with the left mouse button to highlight one residue. Click and drag with the left mouse button to highlight multiple residues, shift-click to add a single residue to the current selection, shift-click and drag to add multiple residues to your selection, right-click to de-select a residue. Highlights appear as thick yellow “Bonds” representations, these can be changed or turned off[§ 4.4.14].

Selecting residues by clicking on the 3-D structure

Use the Mouse menu to enter “Pick Atom” mode (or press “1”, the standard keyboard shortcut). Click on any protein atom or nucleic acid atom, and its residue will highlight, and the sequence list will scroll to display this residue. Shift-click works the same way, but adds to the current selection.

Note that if the zoom factor is smaller than 1.0, the single-residue sequence highlight will be shorter in height than a full line of text. Once the Sequence window has been opened, any "Pick" will create or add to selections, until highlighting is turned off[§ 4.4.14].

Sequence Zooming

Larger molecules contain thousands of residues, too many to display in a linear text list all at once. The sequence form can only list about 40 text lines; to work with a molecule of more than 40 residues use the scroll bars to scroll through the long list, or use the Zoom controls to fit the data from a long list into a small space.

The Zoom slider, and the Fit all, Every Residue buttons, zoom in and out of a long sequence list to allow viewing and selecting from the entire list all at once. To represent more than 40 residues on the form, the text list seems to "skip" residues, but selections, highlights and color-coded data are still active for all residues.

By setting the Zoom slider to a value smaller than 1.0, or by pressing the Fit all button, more or all of the sequence information for a large molecule can be seen at once. To show a text line for every residue in the sequence (zoom factor = 1.0), click on the Every Residue button. The Zoom slider can be dragged with the left mouse button (to re-scale sequence smoothly) or it can jump to a given value by clicking along the slider track with the middle button (this is useful to work more quickly with very long sequences).

For a multi-thousand residue protein with Fit all selected, hundreds of residues can be selected at once, and trends in B-value and structure across the entire protein sequence can be detected. In the screen-shot above, a section of 70 residues with lower B-values than surrounding sequence is selected, by dragging a rectangle around the green stretch in the B-value column.

Other controls include:

- **Toggle display of 3-letter and 1-letter codes** – Click on 1-letter code to switch from 3-letter to 1-letter amino acid codes. The same button then reads 3-letter code, click it to switch back from 1-letter to 3-letter codes.
- **Print contents of sequence window** – Select File:Print to File to create a postscript file containing the current sequence listing and highlighting.

Turn off highlighting / Change highlight style

To clear all highlights, reselect the current molecule from the Molecule pop-up menu. To turn the highlight representation off completely for a given molecule, find the representation in the Graphics form which the Sequence form has created (appears with "Bonds ColorID 4") and set the style to "none". To change highlighting style, set this same representation to your preferred style and coloring. The selection for this representation will still change whenever the sequence form selection changes. Example application: specify Multiple Frames in the Trajectory tab of the highlight representation. This will display the trajectory motions of the residues clicked on in the main VMD window, or in the Sequence window.

Caveats

- Pause on first use: Since the sequence form displays secondary structure of loaded molecules, there may be a pause for structure calculation the first time the sequence for a protein is displayed.

- Selections by chain: When there are multiple segments in a chain, it is possible for several residues to have the same residue number and chain name. These residues will be highlighted/selected/deselected together.
- B-values can be user assigned: To use the B-value column to view arbitrary data, use the selection `set beta` commands to change B-values. To refresh the displayed B-value data, re-select the currently displayed molecule from the Molecule pop-up menu.

4.4.15 RamaPlot

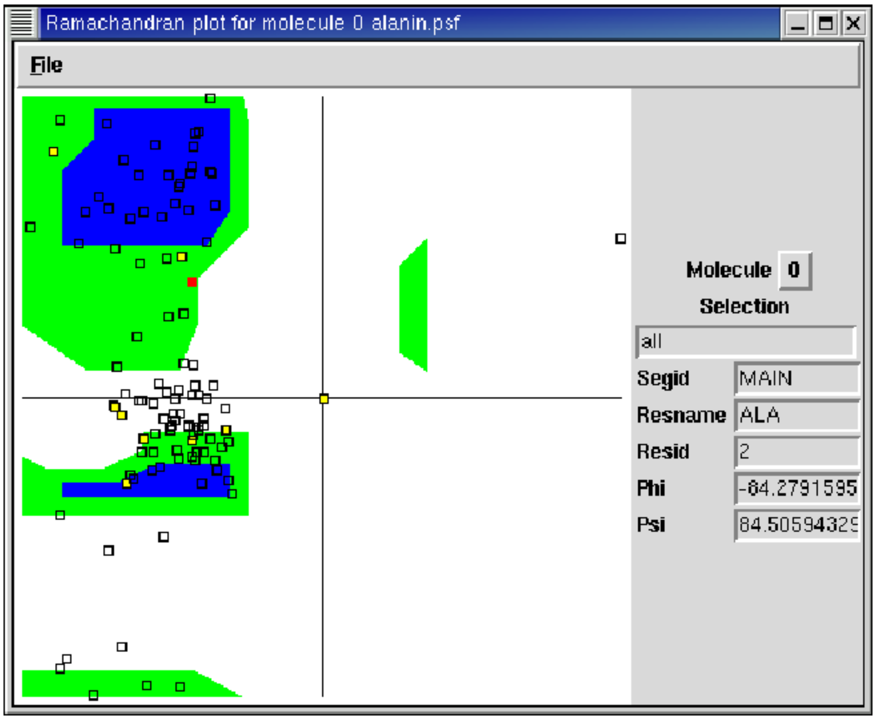

Figure 4.14: The RamaPlot Window

The RamaPlot window displays a Ramachandran plot for a selected molecule. If you animate your molecule over a range of frames , RamaPlot will update the Ramachandran plot automatically. You can select a range of residues to be displayed in the plot. Clicking on a point in the Ramachandran plot will show the trajectory of the selected residue in Ramachandran space over all frames . Fields on the right of the window show the computed value of phi and psi for the most recently selected residue. Finally, you can create a PostScript image of the current Ramachandran plot. RamaPlot functionality is summarized in Fig. 4.14.

Using RamaPlot

Start RamaPlot by typing "`ramaplot`" in the VMD text console, or by selecting the ramaplot menu item in the Extensions menu. The main window contains a Ramachandran graph, with phi and psi running along the horizontal and vertical axis, respectively, from -180 to 180 degrees. The most allowed region of Ramachandran space is colored blue; partially allowed regions are colored green.

After loading a molecule, using the pulldown menu in the upper right part of the window to choose a molecule. Protein residues in the current molecule are mapped to the Ramachandran diagram with yellow squares. Clicking on a square causes the square to turn red, displays residue information in the fields on the right side of the window, and, if trajectory data is present, draws the location of the selected residue in Ramachandran space for all frames in the trajectory as empty black squares. Clicking one of the empty squares causes VMD to redraw the graphics display window with coordinates from the timestep corresponding to that square. Clicking a second time on a red highlighted residue switches off the trajectory information in the RamaPlot window.

When a protein contains many residues, it may be inconvenient to display all residues at once. Enter an atom selection in the Selection input to choose which residues to display. Note that the selection must contain the alpha carbons (name CA) of the residues you want to show. Note also that, just like the Graphics form, the selection will not be recomputed if you change the animation frame.

To print the contents of the white Ramachandran plot, select "Print to file..." from the RamaPlot File pulldown menu. Enter a filename to save the contents of the window.

Chapter 5

Molecular Drawing Methods

Each molecule in VMD is drawn as a collection of several *representations*, of the molecule. A representation is just one particular way of drawing the molecule, and consists of several characteristics:

- 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. Atom selections don't apply when drawing volumetric data such as an electron density map, electrostatic potential map, etc. Section 5.3 describes the syntax used to select atoms.
- A *drawing method* (representation *style*), which determines what shape to draw the atoms, bonds, and other components of the molecule. Section 5.1 describes the 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. Section 5.2 describes VMD's coloring methods.
- A *material*, which determines the shininess, opacity, and other lighting and shading characteristics used when rendering the molecule.

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.

5.1 Rendering methods

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 4.4.7). Table 5.1 lists the available rendering methods, and the following sections describe these methods and the parameters which modify their appearance.

Representation styles	Description
Lines	simple lines for bonds, points for atoms
Bonds	lighted cylinders for bonds
DynamicBonds	dynamically calculated distance-based bonds
HBonds	display hydrogen bonds
Points	just points for atoms, no bonds
VDW	solid van der Waal spheres for atoms, no bonds
CPK	scaled VDW spheres, with cylinders for bonds
Licorice	spheres for atoms, cylinders for bonds, same radius
Trace	connected cylindrical segments through C_α atoms
Tube	smooth cylindrical tube through the C_α atoms
Ribbons	flat ribbon through the C_α atoms
NewRibbons	smooth ribbon through the C_α atoms
Cartoon	cartoon diagram (cylinders and ribbons) based on secondary structure
NewCartoon	smooth cartoon diagram (smooth ribbons) based on secondary structure
MSMS	molecular surface as determined by the program MSMS
Surf	molecular surface as determined by SURF
Beads	per-residue approximate bounding spheres
VolumeSlice	display a texture mapped slice from a volumetric data set
Isosurface	display an isovalue surface from a volumetric data set
Dotted	dotted van der Wall spheres for atoms, no bonds
Solvent	dotted representation of the solvent accessible surface

Table 5.1: Molecular view representation styles.

5.1.1 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 the lines representation is the line Thickness.

5.1.2 Bonds

Nearly everything about this option is the same as 'Lines' [§ 5.1.1] 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 [§4.4.7] by the Bond Resolution control and the radius is given by the value of Bond Radius, 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.

5.1.3 DynamicBonds

The 'DynamicBonds' representation will automatically perform a distanced-based bond search for the active atom selection and active trajectory frame. This representation does not perform the endpoint fixup procedure described above for the regular 'Bonds' [§ 5.1.2] representation. Instead, it is intended to be used in concert with the 'VDW' [§ 5.1.6] representation to show bonds that are being created and destroyed during the course of a trajectory. A bond is drawn if the atoms are within Distance Cutoff of eachother.

5.1.4 HBonds

The 'HBonds' 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 Cutoff and Distance Cutoff 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 Line Thickness 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.

5.1.5 Points

'Points' draws each atom as a point, and does not draw any of the bonds. This option is useful when rendering very large molecules containing hundreds of thousands of atoms, particularly for rendering water or other structures for which geometric detail may not be necessary. The Size of the points can be changed.

5.1.6 VDW

'VDW' draws the atoms as spheres. The Sphere Scale 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 many polygons.

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
```

5.1.7 CPK

'CPK' is a combination of both 'Bonds' [§ 5.1.2] and 'VDW' [§ 5.1.6] in that it draws the atoms as spheres and the bonds as cylinders. The resolution and radius can be modified independently. The size of the sphere drawn in CPK mode is by default the scaled-down VDW radius, but this scaling-factor can be changed by adjusting the Sphere Scale parameter. 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.

5.1.8 Licorice

'Licorice' draws the atoms as spheres and the bonds as cylinders. The difference between this and 'CPK' [§ 5.1.7] 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.

5.1.9 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_α atoms in a protein fragment. Five evenly spaced interpolation points are found along the curve to break the curve connecting the two C_α atoms 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' [§ 5.1.2] control. However, if the bond's Radius becomes 0 or 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.

5.1.10 Trace

This representation applies much of the procedure used to construct the 'Tube' [§ 5.1.9]. 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.

5.1.11 Ribbons

The 'Ribbons' representation is similar to 'Tube' [§ 5.1.9] 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 Width 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.

5.1.12 NewRibbons

The 'NewRibbons' representation is similar to 'Tube' [§ 5.1.9] in that it follows the a 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 an 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.)

The NewRibbons representation uses the alpha Carbons as control points for a spline which defines the ribbon backbone. The ribbon is drawn by extruding a two-dimensional cross section along the length of the spline orienting it by referencing the positions of the Oxygens on the protein backbone. 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.

The Aspect Ratio parameter controls the width of the ribbon relative to the thickness value, as a multiplicative factor. An aspect ratio of 1.0 yields a Tube-like representation. The Resolution parameter controls the degree to which the ribbon surface is tesselated with triangles. Higher settings yield nicer looking images at the expense of interactive rendering performance. Points can be interpolated with either a Catmull Rom or B-Spline by changing the value of Spline Style. Note that the B-Spline does not always pass through the C_α positions, as it is a smoother spline.

5.1.13 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 radius 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 thickness 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 'Ribbons' [§ 5.1.11] 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.

5.1.14 NewCartoon

The 'NewCartoon' representation is a variation of the original 'Cartoon' combined with the 'NewRibbons' representation look and features. The main difference between the original 'Cartoon' representation and 'NewCartoon' is that helices are left in a ribbon representation which follows curved structures much more accurately than the straight cylinders used in the original 'Cartoon' did.

The Aspect Ratio parameter controls the width of the ribbon relative to the thickness value, as a multiplicative factor. An aspect ratio of 1.0 yields a Tube-like representation. The Resolution parameter controls the degree to which the ribbon surface is tesselated with triangles. Higher settings yield nicer looking images at the expense of interactive rendering performance. Points can be interpolated with either a Catmull Rom or B-Spline by changing the value of Spline Style. Note that the B-Spline does not always pass through the C_α positions, as it is a smoother spline.

5.1.15 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.

- Probe Radius – Probe radius used to construct the molecular surface

- Representation Method – The surface can optionally be drawn using lines rather than solid triangles

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 an interprocess communications medium takes up about half of the run time.

There is an environment variable [§ 12.2] 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)

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 much faster surface rendering method, see the description of 'MSMS' [§ 5.1.16].

5.1.16 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 Surf, and can be a better choice depending on how it is used. See the web page `http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html` for more details. Available options include

- Which Atoms – should the surface be of the selection (0) or of the contribution of this selection to the surface of all the atoms? (1)
- Sample 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 Radius – Probe radius used to construct the molecular surface
- Representation Method – The surface can optionally be drawn using lines rather than solid triangles

There is an environment variable [§ 12.2] which can affect the MSMS display option:

- `MSMSSERVER` – location of the MSMS binary (defaults to `msms` which is assumed to be in the user's path) On Windows machines, sets this as a systemwide environment variable in the environment variables window found in the system properties control panel.

5.1.17 VolumeSlice

The 'VolumeSlice' representation draws a texture mapped two-dimensional slice from a volumetric data set already loaded into VMD using the `mol volume` [§ 8.3.19] text command, or by other means. The colors span the scalar value range of the data set, with red indicating low values and blue indicating high values in the data. The slice is drawn as a plane perpendicular to the X, Y, or Z axis, and can be positioned anywhere within the coordinate system of the volumetric data set. This feature is currently only available on machines that have full support for hardware 3-D texture mapping. On machines lacking 3-D texturing, nothing will be displayed. Future versions of VMD will greatly enhance the user interfaces and capabilities of this feature.

The following selectors control the VolumeSlice representation:

- Data Set – This controls which volume data set is referenced in the representation, since multiple volumetric data sets can be loaded for a single molecule.
- Slice Offset – The slice setting indicates the position of the volume slice along the chosen axis, in the coordinate system of the volumetric data, range 0 to 1.
- Slice Axis – The orthogonal axis along which the slice plane moves, can be X, Y, or Z.
- Render Quality – The quality can be set to either Low, or Medium. The Low setting causes the slice texture map to be rendered using the color nearest the sample point. A quality level of Medium indicates that the slice texture map will be rendered using bilinear interpolation.

5.1.18 Isosurface

The 'Isosurface' representation computes and draws a surface within a volumetric data field, on a 3-D surface corresponding to points with a single scalar value.

There are several settings which control how the isosurface is displayed.

- Data Set – This control selects which volume dataset is used for the isosurface calculation, since a given molecule can contain multiple volumetric data sets.
- Isovalue – The Isovalue control selects the value for which the isosurface will be computed. In the GUI, when dragging the isovalue slider, the drawn isosurfaces are temporarily calculated at a lower resolution to improve interactivity; to prevent this behavior, you can use the middle or right button (or the control/shift/alt modifier keys) while dragging the slider.
- Draw – This can be set to Points, Shaded Points, Wireframe, or Solid Surface. The default drawing mode is Points. When viewing very dense isosurfaces of huge volumetric maps, the Shaded Points drawing method can be an excellent compromise between the speed of the Points method and the quality of the Solid Surface method.
- Boundary – Setting the boundary to Box causes the volume data bounding box and coordinate axes to be drawn rather than the isosurface for the data. This is often useful when first working with volumetric data, and checking that the coordinate systems of the volume data and the molecule match.
- Step – This setting can be used to greatly reduce the resolution of the generated isosurface, by skipping voxels.
- Size – This sets the thickness of the point and line based isosurface representations.

This and other volumetric display features will be greatly expanded in forthcoming releases of VMD.

5.1.19 Beads

A bounding sphere is drawn in place of each residue in the atom selection. This representation can be used as a crude means of drawing very large structures in a space filling representation and can be particularly useful for animating trajectories.

5.1.20 Dotted

Same as 'VDW' [§ 5.1.6] 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.

5.1.21 Solvent

This method is similar in spirit to the 'Dotted' [§ 5.1.20] 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 Level, 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 Representation 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.

5.2 Coloring Methods

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 - its RGB value;
- 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 1041 colors available in VMD, with color ids ranging from 0 to 1040. The first 17 are, in order: blue, red, gray, orange, yellow, tan, silver, green, white, pink, cyan, purple, lime, mauve, ochre, iceblue, and black.

The next group of 1024 colors (from 17 to 1040) are colors used in the color map, 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.

5.2.1 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 [§4.4.9]. To color items in a gradation manner, there are additional 1024 colors used in the color scale [§5.2.4].

The different color categories in VMD are listed in table 5.2. 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.

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)
Surface	The surface types
Labels	The different labels (atoms, bonds, etc.)

Table 5.2: Color categories used in VMD.

5.2.2 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 5.3 lists the different coloring methods available. The description for each method explains the source of the information used to determine the color.

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
Chain	The one-character chain identifier, using the Chain category
SegName	Segment name, using the Segname category
Molecule	Molecule all one color, using the Molecule category
Structure	Helix, sheet, and coils are colored differently
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.
User	Provides a single data value for each atom for each timestep
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
Throb	Animates colors through the active color scale based on wall clock time
Timestep	Color scale based on the current trajectory timestep frame

Table 5.3: Molecular coloring methods.

5.2.3 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 [§4.4.9].

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.

5.2.4 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 1024 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.

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 5.4: Available Color Scale Gradations.

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.

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

5.2.5 Materials

VMD allows users to apply a materials property to the molecular models they create. The material determines such things as how transparent an object is, or how shiny, or how large the specular reflections are. 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.

VMD maintains a database of materials which can be applied to any representation in the system, much like the database for colors. There are two default materials, "Opaque" and "Transparent", which cannot be modified. Each material is defined by five settings, as follows: material is

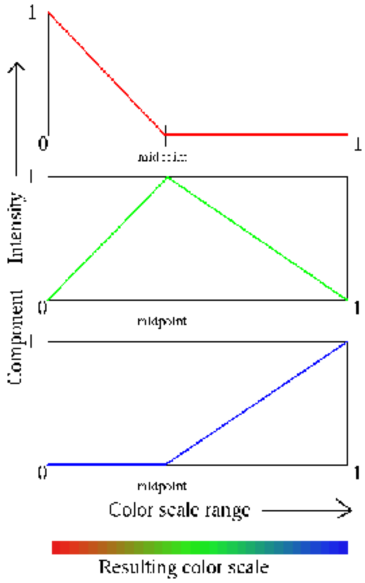

Figure 5.1: RGB color scale: the three plots shows the contributions of each color, and the resulting colors are on the bottom.

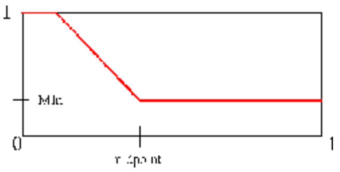

Figure 5.2: The shift to the red component of the RGB scale caused by the value of "min".

- Opacity: a number (0.0 to 1.0) describing the transparency; 1 is solid, 0 is transparent. By default, transparent objects are drawn with Opacity set to .3
- Ambient: a value describing how strongly the material reflects ambient light. Ambient light provides a uniform illumination of objects with a background lighting of the object color.
- Diffuse: 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 number describing the intensity of specular reflections. Produces highlights, the higher the value, the smaller and the brighter the highlight.
- Shininess: a number 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 corresponds to a Phong exponent of 40.

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

5.3 Selection Methods

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
exwithin 3 of protein
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 §5.3.6

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.

5.3.1 Definition of Keywords and Functions

The keywords available for selecting atoms in VMD are listed in tables 5.5 and 5.6 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 5.7 lists the built-in functions which may be used in atom selection expressions with keywords which take on a numeric value.

Table 5.8 lists the built-in atom selection keywords which may be used in atom selection expressions to query the values of an underlying volumetric map in the same molecule. These are read-only.

5.3.2 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
```

5.3.3 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.

5.3.4 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 \*"`.

5.3.5 Double Quotes and Regular Expressions

Double quotes around a string are used to specify a regular expression search (compatible with Perl 5.005, using the Perl-compatible regular expressions library written by Philip Hazel). If you don't know how to use them, try consulting the man pages for `ed`, `egrep`, `vi`, or `regex`. If not, read the Perl docs, or get any one of a number of books including the O'Reilly and Associates Sed and Awk book. The following examples show just a few ways that regular expressions can be used within VMD.
Selection of all atoms with a name starting with `C`:

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

Segment names containing a number:

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

Multiple terms can be provided on the list of matching keywords. This example selects residues starting with an `A`, the glycine residues, and residues ending with a `T`. As with a string, a regular expression in a numeric context gets converted to an integer, which will always be zero:

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

Selections containing special characters such as $+$, $-$, or $*$, must be escaped with the \ character. In order to select atoms named `Na+`, one would use the selection:

```
name "Na\+"
```

In brief, a regular selection allows matching to multiple possibilities, instead of just one character. Table 5.9 shows some of the methods that can be used.

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 5.10 is a list of conversions from X-PLOR style wildcards to the matching regular expression.

5.3.6 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 5.7). 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)
```

5.3.7 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.

5.3.8 Other selections

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`.

`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. Some selections may be sped up by short circuiting [§5.3.3].

A related atom selection construct is `exwithin`, short for 'exclusive within'. The atom selection `(within 3 of protein) and not protein` is equivalent to `exwithin 3 of protein`.

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 ....`

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)
```

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
serial	*num*	the atom number, starting at 1
atomicnumber	*num*	atomic number (0 if undefined)
element	*str*	atomic element symbol string ('X' if undefined)
altloc	*str*	alternate location/conformation identifier
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	*float*	x, y, or z coordinates
radius	*float*	atomic radius
mass	*float*	atomic mass
charge	*float*	atomic charge
beta	*float*	temperature factor
occupancy	*float*	occupancy
user	*float*	time-varying user-specified value
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 5.5: 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
rasmol	*str*	translates Rasmol selection string 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
phi, psi	*float*	backbone conformational angles
within	*str*	selects atoms within a specified distance of a selection (i.e `within 5 of name FE`).
exwithin	*str*	exclusive within, equivalent to `(within 3 of X) and not X`.
same	*str*	selects atoms which have the same keyword as the atoms in a given selection (i.e. `same segname as resid 35`)
ufx, ufy, ufz	*num*	force to apply in the x, y, or z coordinates

Table 5.6: 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 less than 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

Table 5.7: Atom selection functions.

Function	Arg	Description
vol*N*	*float*	value of the voxel of the volumetric data of ID *N* nearest to the atom
interpvol*N*	*float*	interpolated value of the voxels of the volumetric data of ID *N* around the atom

Table 5.8: Read-only atom selection keywords which may be used to query the values of an underlying volumetric map in the same molecule. The value of *N*, which can be 0 to 7 inclusively, refers to the volID of the underlying volumetric data (*e.g.*, you could type `interpvol2`).
Read-only atom selection keywords for querying volumetric data

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

Table 5.9: 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 5.10: Regular expression conversions.

Chapter 6

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 `stereo` entry in the Display form or the text command `display stereo` *`mode`*.

6.1 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.

6.2 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. For stereo, 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. The left and right monoscopic modes are most useful when exporting scenes to external ray tracers.

6.3 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. Anaglyph stereo (requires stereo-capable monitor, quad-buffered stereo video board, and red-blue horror-movie-style stereo glasses).

4. DTI Side-by-side stereo for DTI stereo flat panel displays

5. Scanline-interleaved, also known as line blanking stereo, works with compatible shutter glasses or panel displays.

6. CrystalEyes stereo (requires stereo-capable monitor, quad-buffered stereo video board, stereo sync emitters and special stereo glasses equipped with liquid crystal or polarized lenses).

6.3.1 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.

6.3.2 DTI Side-by-side Stereo

This stereo mode is the same as the regular side-by-side stereo mode except that the aspect ratio of the displayed image is adjusted to work correctly on DTI stereo flat panel displays.

6.3.3 Scanline Interleaved Stereo

Scanline interleaved stereo, also referred to as line blanking stereo works by interleaving the left and right eye views every other scanline in the display window. The stereo hardware then decodes the interlaced signal and either separates them into two separate displays or blanks the even or odd scanlines to display only the left or right eye image at the same time that shutter glasses are polarized in the appropriate way. This mode works well with the Eye3D glasses and similar stereo hardware. This stereo mode is nice because the only special requirements for the graphics accelerator are that it provide a stencil buffer which is used to generate the alternating scanlines in the final image.

6.3.4 Anaglyph Stereo

Anaglyph stereo refers to the use of colors to separate the left and right eye views from each other. The user must wear glasses with colored lenses, such as the red-blue glasses one finds at some sci-fi and horror movie showings. The computer hardware requirements for Anaglyph stereo are currently the same as for CrystalEyes stereo except that LCD shutter glasses and emitters are not required. The red-blue glasses take the place of the LCD shutter glasses used for the CrystalEyes stereo modes. Anaglyph stereo has one major disadvantage when compared with CrystalEyes stereo, which is that its color rendition is obviously severely constrained. This is just a fact of life with Anaglyph stereo, and it is up to the user to use color schemes for their molecules that still look visually pleasing when displayed in Anaglyph stereo mode.

6.3.5 CrystalEyes Stereo

CrystalEyes stereo is the name used within VMD for the quad-buffered frame-sequential stereo display mode found on appropriately equipped graphics workstations. Typically this mode is used to drive LCD shutter glasses with a CRT display or projection system.

Quad-buffered frame-sequential stereo is available on workstations such as SGI InfiniteReality, Sun XVR-1000, XVR-4000, HP SV6, NVidia Quadro FX, and others. This mode provides separate left and right eye frame buffers. It allows the a window display in stereo, while all other windows appear as normal. Quad-buffered stereo modes usually display at lower-resolution since they require twice as much video memory as non-stereo display modes. The display must be set in a stereo-capable mode before starting VMD. On SGI systems, one must use either the **setmon** or **xscreen** utilities to configure the display mode. On Sun systems this is done using the **fbconfig** utility. HP systems use the **sam** utility for display configuration. Windows machines use the **Display Properties** tab to control video board display mode settings.

Once set in the proper display mode, start VMD as normal, and select 'CrystalEyes stereo' from the Display form. The image should switch to two images nearly superimposed, but slightly offset.

6.3.6 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 [§4.4.6].

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.

Chapter 7

Scene Export and Rendering

One of the most common tasks performed by users of VMD is producing images which can be loaded into other programs or used in printed documents, posters, slides, and transparencies. The Render form provides a simple mechanism for generating image files from snapshots of the VMD graphics window and through the use of external rendering and ray tracing programs.

7.1 Screen Capture Using Snapshot

The simplest way to produce raster image files in VMD is to use the "Snapshot" feature. The snapshot feature captures the contents of the VMD graphics window, and saves them to a raster image file. On Unix systems, the captured image is written to a 24-bit color SGI "RGB" file. On Windows systems, the captured image is written to a 24-bit color Windows Bitmap, or "BMP" file. To use the `snapshot` feature, simply open the Render[§ 4.4.11] form and choose the `snapshot` option. VMD will capture the contents of the graphics window, and attempt to save the resulting image to the filename given in the Render form. You may find that it is important not to have other windows or cursors in front of the VMD graphics display when using snapshot, since the resulting images may include obscuring windows or cursors. This is a platform-dependent behavior, so you will need to determine if your system does this or not.

7.2 Higher Quality Rendering

Sometimes images produced by screen capture aren't good enough; you may want a very large, high quality picture, or a picture with shadows, reflections, or high quality rendering of transparent surfaces. While VMD generally produces nice looking images in its graphics window, it was designed to generate its images very rapidly to maximize interactivity, which precludes the use of photorealistic rendering techniques that would slow down the operation of whole program. Instead of producing high quality images directly, VMD writes scene description files which can be used as input to several popular scanline rendering and ray tracing programs. Tables 7.1 and 7.2 list the currently supported output formats, and where appropriate rendering software may be obtained.

Making the raster image is a two step process. First you must make a scene description file suitable for the chosen rendering program, and then execute the program using the new file as input to produce the raster image output. The problem is that each of the external rendering programs support different output file formats, which may need to be converted to something more appropriate for you. It is impossible to predict what that might be, so we'll describe how to convert

Name	Description	Default Render Command
Raster3D[1]	Fast raster file generator	`render < %s -sgi %s.rgb; ipaste %s.rgb`
PostScript	Direct PostScript Output	`ghostview %s &`
Renderman	Renderman RIB Format	`rgl` - Can be rendered with BMRT, PRMan, Maya, etc
STL	Stereolithography Format	`true` - Renders Triangles Only
VRML-1	Virtual Reality Markup Language	`true` - view this with VRML viewers like webspace

[1]See `http://www.bmsc.washington.edu/raster3d/` for more info.

Table 7.1: Miscellaneous Rendering Options

Name	Description	Default Render Command
Tachyon[1]	Very fast multiprocessor ray tracer	`tachyon -aasamples 2 -mediumshade %s -format TARGA -o %s.tga`
POV3[2]	POV-Ray 3.x ray tracer	`povray +H500 +W400 -I%s -O%s.tga +D +X +A +FT`
Rayshade[3]	Rayshade ray tracer	`rayshade < %s > %s.rle`
Radiance[4]	Radiosity ray tracer	`oconv %s > %s.oct; rview -pe 100 -vp -3.5 0 0 -vd 1 0 0 %s.oct`
ART[5]	VORT ray tracer	`art %s 500 650`

[1]See `http://www.megapixel.com/` for more info.
[2]See `http://www.povray.org/` for more info.
[3]See `http://www-graphics.stanford.edu/ cek/rayshade/rayshade.html` for more info.
[4]See `http://radsite.lbl.gov/radiance/HOME.html` for Radiance
[5]Available from `ftp://gondwana.ecr.mu.oz.au/pub` along with the rest of VORT package

Table 7.2: Supported ray tracing formats.

the different file types to RGB and let you use the tools listed in Table 7.1 to get what you need. Tachyon and Raster3D can produce SGI RGB files, so you don't need to do anything but specify this output format. POV-Ray produces 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.imagemagick.org/` – should be able to read and convert between all of these formats.

We suggest using either Tachyon or Raster3D as they are generally the fastest programs. Both programs are easy to understand, and are fast even when rendering very complex molecules.

The generated scene files are plain text so they are very 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 renderer's 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 [§ 4.4.11], 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 7.1, 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.

7.3 Caveats

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 can be some problems. The colors in the final raster image can sometimes look different from what is seen in the VMD graphics window. This is because the external rendering programs use different shading equations and algorithms from what VMD uses. Potential rendering discrepencies include:

- Geometry may look slightly different; in VMD curved surfaces are polygonalized and drawn using a number of polygonal facets, curved surfaces may be rendered entirely smoothly in the final output (which is generally looked upon as an improvement!)
- The rendered object colors or intensities may be slightly different due to different colormaps, gamma values, or lighting models; This is particularly true with the material properties used for performing complex shading. VMD's real-time rendering of these material properties is often simplistic or limited compared to full-fledged photorealistic renderers, so there can potentially be big differences between implementations of transparency, specular highlights, etc.
- Many of the external renderers do not support true orthographic rendering. This can bed "faked" by translating the camera very far away from the molecule, followed by zooming the camera so that the image size is acceptable again. This will significantly decrease the perspective effect, but is not a true orthographic projection.
- 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; Rendering in stereo is accomplished by setting the display mode to "left", then rendering an image, followed by "right", and rendering again. This will yield a stereo pair to the best of VMD's ability with the external rendering program.
- The near and far clipping planes are ignored by all external renderers;
- Text is generally not available as a graphics primitive in the renderer scene languages, 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;

7.4 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, a few are listed below:

- 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[§ 4.4.11] form and type `render < %s -sgi %s.rgb; convert %s.rgb eps:%s.ps; lpr %s.ps` in the render command box.
- Choose the PostScript output option in the Render form and type `lpr %s` in the render command box.

7.5 Making Stereo Images

Stereoscopic images can be rendered with a simple sequence of text commands, cycling between the left and right monoscopic stereo modes and exporting one scene for each eye:

```
display stereo left
render Tachyon left.dat
display stereo right
render Tachyon right.dat
```

External renderers don't always 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. The text commands for this are something like:

```
display stereo off
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.

7.6 Making a Movie

It is possible to make movies with VMD, through the use of Tcl or Python scripts, or with the "vmdmovie" extension included with VMD. Several movie making scripts are provided in the VMD script library on the VMD home page. These scripts can be used as-is, or they can be customized to perform complex animation tasks beyond the scope of this user guide. In general, movies are created by driving render commands with a script, producing a sequence of individual image files. When the script has completed rendering all of the individual frames, the images are ready for import into an animation package, or can be converted to one of several popular compressed movie formats by further processing. The “vmdmovie” extension provided with VMD completely automates the movie creation process, though it requires a number of software packages be installed in order to do the job. Please see the separate documentation on the movie scripts and “vmdmovie” in the VMD script library.

Chapter 8

Tcl Text Interface

The Tcl text interface provides complete access to all the VMD commands. Anything that can be done from the menus can be done with VMD text commands.

8.1 Using text commands

Text commands can be entered into VMD in several ways:

- Commands can be 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.

- 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. One is the `play` 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, so that VMD stays interactive during execution of the script. The second way is the Tcl command `source`. This reads the whole file before allowing the mouse and forms to respond to new input. This is often more efficient when your script contains many lines.

- On Unix/Linux platforms, if the file `.vmdrc` (see section 12.3.3) exists in your home directory, it is played at VMD startup. If you don't have a `.vmdrc` file, VMD uses a default script in the VMD installation directory. Similarly, at startup the `-e` command line flag can be used to specify an input file to be played after reading the `.vmdrc` file. The Windows version of VMD works similarly, though the startup file is named `vmd.rc`.

  A good use of the `.vmdrc file` is to specify which VMD menus you would like to have open when you start VMD and where they should be placed; see section 8.3.18) for information on usage.

8.2 Tcl/Tk

The standard distribution is compiled with Tcl, 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, for creating menus with buttons bound to one's favorite actions.

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.

VMD uses Tcl and Tk version 8.4.1. We refer you to `http://www.tcl.tk/` for more information about Tcl.

8.3 Tcl Text Commands

All Tcl commands in VMD are composed of one or more words or phrases separated by white space, and terminated by a newline. In Tcl, a "phrase" is text surrounded by double quotes or by a matching set of open and close braces. 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 8.1 summarizes the text commands in VMD by listing the first words, and describing the general purpose for commands starting with those words.

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.

8.3.1 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** *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*] [**waitfor** *nw*] [*molecule_number*]:

  Read data for *molecule_number* from *filename* of type *file_type*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*. Return the number of frames read from this file; if the file contains more than this number, the remaining frames will be loaded during subsequent VMD display updates. By default, one frame will be loaded before the command returns. The `waitfor` option allows you to specify how many frames to load before returning. The `waitfor` parameter *nw* can be any integer, or `all`; choosing *nw* less than zero is the same as choosing `all`. If frames from other files are still being loaded when the animate command is issued, these frames will be loaded first.
- **write** *file_type filename* [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [**waitfor** *nw*] [**sel** *selection*] [*molecule_number*]: Write data from *molecule_number* to *filename* of type *file_type*, beginning with frame *nb*, ending with frame *ne*, with a stride of *ns*. Return the number of frames written to this file; if more frames have been specified than this number, the remaining frames will be written during subsequent VMD display updates. By default, one frame will be written before the command returns. The `waitfor` option allows you to specify how many frames to write before returning. The `waitfor` parameter *nw* can be any integer, or `all`; choosing *nw* less than zero is the same as choosing `all`. Pass the name of an atom selection as *selection* to write only the selected atoms to the file.
- **delete** [**beg** *nb*] [**end** *ne*] [**skip** *ns*] [*molecule_number*]: Delete data for *molecule_number*, beginning with frame *nb*, ending with frame *ne*, and keep frames with a stride of *ns* (a stride of -1 implies to keep all frames).

8.3.2 atomselect

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.

- **list**: Return a list of all undeleted atom selections.

- **keywords**: Return a list of all recognized keywords in an atom selection text.
- **macro** *name selection*: Create a new singleword atom selection out of existing atom selections. *name* must be a single word starting with a non-numeric character and contain no spaces or special characters. *selection* can be any valid atom selection, and can even contain other macros. You should ensure that your macros do not contain themselves, either directly or through a chain of other macros. If VMD detects this situation, it will abort the evaluation of the atom selection.

  If no selection is given, the macro for the given name is returned.

  If no name is given, a list of all macro names is returned.

  If a macro already exists for the given name, the old selection will be replaced with the new selection. Singlewords that are not defined as macros, like `protein` and `water`, cannot be redefined with the macro command.
- **delmacro** *name*: Delete the macro corresponding to *name*. Singlewords that are not defined as macros cannot be deleted.
- *molecule_id selection_text* [**frame** *frame_number*] Creates a new atom selection and returns its name. The returned name can be used as a Tcl proc in order to access the atom selection. The *selection text* is the same language used in the Graphics form [§ 4.4.7] and described in Chapter 5.3. 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).

  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 newly created atom selection is a Tcl proc, which takes the following options:

  - **num**: Return the number of atoms in the selection.
  - **list**: Return a list of the atom indices 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 selection.
  - **frame**: Returns the animation frame associated with this selection. The result will be either `now`, `last`, or an integer corresponding to the frame. When the frame is `now`, the atom selection will use atomic coordinates from the current frame for its associated molecule. If the frame is `last`, the atom selection will always use coordinates from the

last frame. If the frame is a specific integer, the selection will always use coordinates from that frame, even if the current animation frame changes. Note that if a nonexistent frame is specified, the atomic coordinates will all be taken to be 0.

- **frame** *frame*: Set the frame for the selection. *frame* should be either `now`, `last`, or an integer.
- **delete**: Delete this object (removes the function).
- **global**: Moves the object into the global namespace. Atom selections created within a Tcl proc that are not made global are deleted when the proc exits.
- **uplevel** *level*: Moves the object to a new level in the namespace stack. Works the same as the Tcl function `uplevel`.
- **get** *attribute_list*: Given an attribute or a list of attributes, returns the attribute values. If only a single attribute is given, a list of corresponding attributes values will be returned. If a list of attributes is given, then a list of sublists will be returned; each sublist will contain the values for the corresponding attributes. See Tables 5.5, 5.6, and 5.8 for the recognized attribute keywords.
- **set** *attribute_list values_lists*: Set the attributes in the attribute list with the values gven in the values lists. If there is only one attribute, then *values_lists* can be either a single value or a list of values, one for each selected atom. If there is more than one attribute, then *values_lists* must be a list of sublists; the number of sublists must equal the number of selected atoms, and the number of items in each sublist must equal the number of attributes.

  Example:

  ```
  set sel [atomselect top all]
  set mass [$sel get mass]
  set xyz [$sel get {x y z}]
  $sel set beta 0      # all values are set to zero
  $sel set beta $mass  # copy mass to beta
  # set occupancy to x, mass to y, beta to z
  $sel set {occupancy mass beta} $xyz
  ```

  It is an error to set integer or floating point keywords using non-numeric values. If floating point values are passed to integer keywords, they will be converted to integers, and vice versa.

  The set command immediately updates all representations of the selected molecule. If speed is an issue, delete all representations of the molecule before setting the values.
- **getbonds**: returns a list of bondlists; each bondlist contains the id's of the atoms bonded to the corresponding atom in the selection.
- **setbonds**: Set the bonds for the atoms in the selection; the second argument should be a list of bondlists, one bondlist for each selected atom.
- **move** *4x4 matrix*: Applies the given transformation matrix to the coordinates of each atom in the selection.
- **moveby** *offset*: move all the atoms by a given offset.
- **lmoveby** *offset_list*: move each atom by an offset given in the list.

– **moveto** *position*: move all the atoms to a given location.
– **lmoveto** *position_list*: move each atom to a point given by the appropriate list element.
– **writeXXX** *filename*: write the selected atoms to a file of type XXX; e.g., pdb, dcd. **New in VMD 1.8**: writepdb requires a filename; omitting the filename no longers returns the PDB data as a string. To get the PDB data as a string, first write to a file, then enter the following commands: `set fd [open` *filename* `r]; set s [read $fd]; close $fd`. The text will be contained in the variable `s`.
– **update**: Update the atom selection based on the frame for the selection (the frame can be specified using the **frame** option as described above).

See section 11.2 for more on using atom selections for fun and profit, as well as issues relating to speed of analysis scripts.

8.3.3 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.

8.3.4 color

Change the color assigned to molecules, or edit the color scale. All color values are in the range $0 \ldots 1$. Please see the section on coloring [§ 5.2] 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 rgb** *color*: Reset rgb of *color* to default value.
- **change rgb** *color r g b*: Set rgb of *color* to *r g b*.
- **restype** *resname* `[` *restype* `]`: Set the residue type for *resname* to *restype*. If the *restype* parameter is ommitted, the current residue type is returned.
- **add item** *category name colorname*: Adds colors for the named color category, item name, using the colorname color.

See the graphics § 8.3.9 command for how to change color of a user-defined graphics object.

8.3.5 colorinfo

(Tcl) This command provides access to the color definitions. For information on the color properties see the chapter on Coloring [§5.2].

- **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 (1041)
- **colorinfo colors**: returns a list of the named solid colors
- **colorinfo** `[` **index** `|` **rgb** `]` `<` **name** `|` **value** `>` : returns the index or rgb of 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
}
```

8.3.6 display

Change various aspects of the graphical display window. For information about the options, see the section describing the Display form [§4.4.6].

- **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 the VMD script library 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.
- **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*.
- **resize** *valueX valueY*: Set the size of the display window to *valueX* $\times$ *valueY*. Note that this option currently works only with X11 versions of VMD.
- **reposition** *valueX valueY*: Set the position of the upper-left corner of the display window to *valueX* $\times$ *valueY* pixels from the lower-left corner of the screen. Note that this option currently works only with X11 versions of VMD.

- **antialias < on | off >**: Turn antialiasing on or off.
- **culling < on | off >**: Turn backface culling on or off.
- **depthcue < on | off >**: Turn depth cueing on or off.
- **projection < perspective | orthographic >**: Set the projection mode to *mode*.
- **rendermode < Normal | AlphaBlend | GLSL >**: Set the rendering mode to *mode*. This parameter allows the use of various OpenGL extensions to implement alpha-blended transparency, or programmable shading for higher quality molecular graphics. The default rendering mode does not enable these features since they significantly alter the rendering and performance characteristics of VMD when they are enabled.
- **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 | culling | rendermode | size | stereo | projection | nearclip | farclip >**: Return the current value of the requested option.
- **get < rendermodes | stereomodes | projections | details >**: Return a list of the availalble values for the given options. (See section 4.4.6 and chapter 6 for more information.)
- **fps < on | off >**: Turn frames-per-second indicator on or off.

8.3.7 draw

VMD offers a way to display user-defined objects built from graphics primitives such as points, lines, cylinders, cones, spheres, triangles, and text. Since these are displayed in the scene just like all other graphics, they can also be exported to the various ray tracing formats, 3-D printers, etc. User-defined graphics can be used to draw a box around a molecule, draw an arrow between two atoms, place a text label somewhere in space, or to test a new method for visualizing a molecule.

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*]

The draw command is equivalent (in most cases) to `graphics top` *command* [*arguments*], in that it simply adds graphics primitives to the top molecule, saving you the trouble of typing an extra argument. However, `draw` extends graphics in two ways. First, if no molecule exists, `draw` creates one for you automatically. Second, `draw` can be extended with user-defined drawing commands. This is done by defining for a function of the form `vmd_draw_$command`. If the function exists, it is called with the first parameter as the molecule index and the rest as 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
}
```

After entering this command into VMD, you can use a command such as `draw arrow {0 0 0} {1 1 1}` to draw an arrow. In addition to defining new commands, user-defined drawing commands can also be used to override existing commands. For example, if you define `vmd_draw_sphere`, then `draw sphere {0 0 0}` will call your sphere routine, not the one from `graphics`.

Here's a quick way to add your own label to an atom selection [§5.3]. 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
}
```

8.3.8 exit

Quit VMD.

8.3.9 graphics

The `graphics` command draws low-level graphics primitives. These primitives can be used to draw a box around a molecule, or an arrow between two atoms, or place a text label somewhere in space. The command syntax is `graphics <molid> <cmd>`, where `<molid>` is a valid molecule id and `<cmd>` is one of the commands listed below. To create a "blank" molecule, use the Tcl command `mol new`. See the draw [§ 8.3.7] command for a possibly more convenient interface. Also refer to the VMD script library[1] for some examples of user-defined graphics scripts.

As graphical primitives are added to the list they are assigned a unique, increasing `id`. The first object added is assigned 0, the second is assigned 1, etc. The commands which add an item return its value.

- **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.

[1] http://www.ks.uiuc.edu/Research/vmd/script_library

- **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. The normals must be normalized to unit-length for proper display.

- **tricolor** {*x1 y1 z1*} {*x2 y2 z3*} {*x3 y3 z3*} {*nx1 y1 z1*} {*nx2 ny2 nz3*} {*nx3 ny3 nz3*} *c1 c2 c3*:

  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. The last three integers indicate the colors to apply to each vertex. This is used for making a smooth shading across the triangle. The normals must be normalized to unit-length for proper display.

- **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*" [**size** *s*]:

  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 5.2 and sections on color [§ 8.3.4] and colorinfo [§ 8.3.5] 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.

- **material** <**name**>: Sets the material to use for the corresponding graphics molecule. `name` must be a valid material name, as displayed in the Materials menu.
- **delete** *id*: Deletes the graphics primitive with the given id.
- **delete all**: Deletes all graphics primitives.
- **replace** *id*: Causes the next graphics primitive to replace the one with the given id. Subsequent graphics primitives will be added to the end of the list as usual.
- **exists** *id*: Returns whether the primitive with the given id exists.
- **list**: Returns a list of valid graphics id's.
- **info** *id*: Returns the text of a Tcl command which will recreate the graphics primitive with the given *id*.

8.3.10 gettimestep

Retrieve the specified molecle's timestep as a Tcl byte array which can be used for high-efficiency analysis calculations by compiled Tcl plugins.

- `<` *molid*`>` `<` *timestep* `>`: retrieve timestep as a Tcl byte array for use in compiled analysis plugins.

8.3.11 help

Display the on-line help file with an HTML viewer. See Chapter 12 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.

8.3.12 imd

Controls the connection to a remote simulation.

- **connect** *host port*: connect to an MD simulation running on the machine named *host* and listening on port *port*. This command will fail if a previously-established connection has not yet been disconnected.
- **detach**: Disconnect from the simulation; the simulation will continue to run.
- **kill**: Disconnect from the simulation and also cause it to halt.
- **pause** `<` **on** `|` **off** `|` **toggle** `>`: Pause, unpause or toggle the paused state of the remote simulation.

- **transfer** *rate*: Set the rate at which new coordinates are sent by the remote simulation to VMD to the specified value.
- **keep** *rate*: Set the keep rate, i.e. the frequency at which VMD saves simulation frames to memory, to the specified value.

8.3.13 label

Turn on or off labels for the four categories: atoms, bonds, angles, or dihedral angles; create and destroy simulated springs. 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 `xmgrace`. 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* `[`*filename*`]`: Retrieve the values of the labels for all timesteps. If the optional `filename` is given, the data will be written to that file; otherwise it will be returned as a list. You can use the Tcl `exec` command to launch an external graphing program to plot the data if you wish.
- **addspring** *molID1 atomID1 atomID2* `k`: Add a spring connecting the atom(s) *atomID* and *atomID2* of the molecule *molID*. The spring will have spring constant *k*.
- **textsize** [*newsize*]: Get/set the text size for all labels, which is 1.0 by default. *newsize* should be a decimal value greater than zero.

8.3.14 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 setting their positions places them along a ray from the origin through the given point.

- **num**: Return the number of lights available.
- *light_number* **on**: Turn a light on.
- *light_number* **off**: Turn a light off.

- *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.
- *light _numer* **pos**: Return current position.
- *light _numer* **pos default**: Return default position.
- *light _numer* **pos {** `x y z`**}**: Set light position.

8.3.15 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*.
- **console**: Turn on logging and direct it to the VMD text console window.
- **off**: Turn off logging.

To write log information to the file 'off', use the file name './off'.

8.3.16 material

This set of commands is used to create new material definitions and modify existing ones.

- **list**: Return a list of the available materials
- **settings** *name*: Return a list of the five material settings for the material of the given name. These settings take on floating point values between 0 and 1. The values are returned in the following order: ambient, specular, diffuse, shininess, opacity. If the specified material has not been defined, nothing is returned.
- **add** *name*: create a new material with the given name. The new material will start with the settings for Opaque. If the name already exists, no new material is created.
- **add copy** *name*: create a new material copied from the selected material name.
- **rename** *oldname newname*: rename the given material. The command will fail if the name is already used.
- **change** *property name value*: Change a material property of the material named *name* to the value *value*. *property* must be one of the following:
  - ambient
  - specular
  - diffuse

  - shininess
  - opacity

- **delete** *name*: Delete the given material.

8.3.17 measure

The meaure command supplies several algorithms for analyzing molecular structures. In the following options, *selection* refers to an atom selection, as returned by the `atomselect` command described in section 8.3.2. The optional *weight* must be either `none`, an atom selection keyword such as `mass`, or a list of values, one for each atom in the selection, to be used as weights. If *weight* is missing or is `none`, then all weights are taken to be 1. When an atom selection keyword is used, the weights are taken from *selection1*.

- **avpos** *selection* [**first** *first*] [**last** *last*] [**step** *step*]: Returns the average position of each of the selected atoms, for the selected frames. If no first, last, or step values are provided the calculation will be done for all frames.

- **center** *selection* [**weight** *weight*]: Returns the geometric center of atoms in *selection* using the given `weight`.

- **contacts** *cutoff selection1* [*selection2*]: Find all atoms in *selection1* that are within *cutoff* of any atom in *selection2* and not bonded to it. If *selection2* is omitted, it is taken to be the same as *selection1*. *selection2* and *selection1* can either be from the same of from different molecules. Returns two lists of atom indices, the first containing the first index of each pair (taken from *selection1*) and the second containing the second index (taken from *selection2*). Note that the index is the global index of the atom with respect to its parent molecule, as opposed to the index within the given atom selection that contains it.

- **fit** *selection1 selection2* [**weight** *weight*]: Returns a 4x4 transformation matrix which, when applied to the atoms in *selection1*, minimizes the weighted RMSD between *selection1* and *selection2*. See section 11.4.2 for more on RMSD alignment.

- **hbonds** *cutoff angle selection1* [*selection2*]: Find all hydrogen bonds in the given selection(s), using simple geometric criteria. Donor and acceptor must be within the *cutoff* distance, and the angle formed by the donor, hydrogen, and acceptor must be less than *angle* from 180 degrees. Only non-hydrogen atoms are considered in either selection. If both *selection1* and *selection2* are given, the *selection1* is considered the donor and *selection2* is considered the acceptor. If only one selection is given, all non-hydrogen atoms in the selection are considered as both donors and acceptors. The two selections must be from the same molecule. The function returns three lists; each element in each list corresponds to one hydrogen bond. The first list contains the indices of the donors, the second contains the indices of the acceptors, and the third contains the index of the hydrogen atom in the hydrogen bond.

  Known Issue: The output of hbonds cannot be considered 100% accurate if the donor and acceptor selection share a common set of atoms.

- **inverse** *matrix*: Returns the inverse of the given 4x4 matrix.

- **minmax** *selection*: Returns two vectors, the first containing the minimum x, y, and z coordinates of all atoms in *selection*, and the second containing the corresponding maxima.
- **rgyr** *selection* [**weight** *weight*]: Returns the radius of gyration of atoms in *selection* using the given `weight`. The radius of gyration is computed as

$$r_{gyr}^2 = \left(\sum_{i=1}^{n} w(i)(r(i) - \bar{r})^2\right) / \left(\sum_{i=1}^{n} w(i)\right) \tag{8.1}$$

  where $r(i)$ is the position of the ith atom and $\bar{r}$ is the weighted center as computer by `measure center`.
- **rmsf** *selection* [**first** *first*] [**last** *last*] [**step** *step*]: Returns the root mean square position fluctuation for each selected atom in the selected frames. If no first, last, or step values are provided the calculation will be done for all frames.
- **rmsd** *selection1 selection2* [**weight** *weight*]: Returns the root mean square distance between corresponding atoms in the two selections, weighted by the given *weight*. *selection1* and *selection2* must contain the same number of atoms (the selections may be from different molecules that have different numbers of atoms).
- **sasa** *srad selection* [**-points** *varname*] [**-restrict** *restrictedsel*] [**-samples** *numsamples*]: Returns the solvent-accessible surface area of atoms in the *selection* using the assigned radius for each atom, extending each radius by `srad` to find the points on a sphere that are exposed to solvent. If the *restrictedsel* selection is used, only solvent-accessible points near that selection will be considered. The restrict option can be used to prevent internal protein voids or pockets from affecting the surface area results. The points option can be used to see where the area contributions are coming from, and then the restrict flag can be used to eliminate any unwanted contributions after visualizing them. The *varname* parameter can be used to collect the points which are determined to be solvent-accessible.
- **sumweights** *selection* **weight** *weights*: Returns the sum of the list of weights (or data field to use for the weights) for all of the atoms in the selection.

8.3.18 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

The parameter *menu_name* is one of the following menu names: color, display, files, graphics, labels, main, material, ramaplot. render, save, sequence, simulation, or tool.

8.3.19 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 [§8.3.21]. 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 [§4.4.7]. The representations can be added, deleted or changed with the mol command. See also sections on molinfo command [§ 8.3.21] for more ways of retrieving information about the representations.

- **new** [*filename*] [*options*]:
- **addfile** < *filename* > [*options*]:

  mol new is used to create a new molecule from a file; if the optional *filename* parameter is omitted, a plain, "blank" molecule is created with no atoms (this can be used to create a canvas for drawing user-defined geometry). mol addfile is like mol new except that the structure and coordinate data are loaded into the top molecule (whichever molecule was loaded last) instead of creating a new one. Both mol new and mol addfile accept the following set of options:

  - type <type>: Specifies the file type (psf, pdb, etc.) If this option is omitted, the filename extension is used to guess the filetype; otherwise, it overrides what would be guessed from the filename.
  - first <frame>:
  - last <frame>:
  - step <frame>: For files containing coordinate frames , specifies which frames to load. Frames are indexed starting at 0. A step of 1 means all frames in the range will be loaded; a step of 2 means load every other frame.
  - waitfor <frames>: For files containing coordinate frames , specifies how many frames to load before returning; the default is 1. If frames is less than the number of frames in the file, the rest of the frames will be loaded in the background on subsequent VMD display updates. If frames is -1 or all, then all frames in all files still in progress will be loaded at once before the command returns. Frames loaded this way will load faster than if they are loaded in the background. If files are still being loaded in the background when the addfile command is issued, frames from the files in progress will be loaded first.
  - volsets <set ids>: For files containing volumetric data, specifies which data sets to load. <set ids> should be a list of zero-based indices.

- `autobonds <on|off>`: Turn automatic bond calculation on/off. This can be useful for loading unusual non-molecular coordinates for which VMD's bond-finding algorithm is too slow (e.g., if the point density is very high). Default is on.
- `molid`: For addfile only. The molecule id of the molecule into which the file should be loaded may be specified. It must be the last option specified. If omitted, the default is the top molecule.

- **load** *structure_file_type structure_file* [*coordinate_file_type coordinate_file*] : Load a new molecule from *filename(s)* using the given *format*. If an additional coordinate file is specified, load this file as well. **New in VMD 1.8:** All frames from the coordinate file will be loaded before the command returns. If this is not desirable, use the `animate read` command for more fine-grained control over how coordinate files are loaded. Previous version of VMD loaded only one frame before returning. The function will return the id of the newly created molecule, or return an error if unsuccessful.
- **urlload <file_type> <URL>**: Load a molecule of *file_type* from a given URL address. Return the id of the newly created molecule, or an error if unsuccessful.
- **pdbload <four_letter_accession_id>**: Retrieve the PDB file with the specified accession code from the RCSB web site. Returns the id of the newly created molecule, or an error if unsuccessful.
- **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.
- **clipplane center** *clipplane_id rep_number molecule_number* [*vector*]
- **clipplane color** *clipplane_id rep_number molecule_number* [*vector*]
- **clipplane normal** *clipplane_id rep_number molecule_number* [*vector*]
- **clipplane status** *clipplane_id rep_number molecule_number* [*boolean*]
- **modcolor** *rep_number molecule_number coloring_method*: Change the current coloring method for the given representation in the specified molecule.
- **modmaterial** *rep_number molecule_number material_name*: Change the current material 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** *molecule_number*: Using the current default settings for the atom selection, coloring, and rendering methods, add a new representation to the specified molecule.
- **default** *category value*: Set the default settings for color, style, selection, or material to the supplied value.
- **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.
- **cancel** *molecule_number*: Cancel loading trajectories.
- **ssrecalc** *molecule_number*: Recalculate secondary
- **rename** *molecule_number newname*: Rename the specified molecule.
- **repname** *molecule_number rep_number*: Returns the name of the given rep. This name is guaranteed to be unique for all reps in the molecule, and will stay with the rep even if the rep_number changes.
- **repindex** *molecule_number name*: Return the *rep_number* for the rep with the given name, or -1 if no rep with that name exists in that molecule.
- **selupdate** *rep_number molecule_number [onoff]*: Update the selection for the specified rep each time the molecule's timestep changes. If *onoff* is not specified, returns the current update state.
- **colupdate** *rep_number molecule_number [onoff]*: Update the calculated color for the specified rep each time the molecule's timestep changes. If *onoff* is not specified, returns the current update state.

- **drawframes** *molecule_number rep_number [frame_specification]*: Draw multiple trajectory frames or coordinate sets simultaneously. This setting allows the user to select one or more ranges of frames to display simultaneously. The frame specification takes one of the following forms **now**, *frame_number*, *start:end*, or *start:step:end*. If the *frame_specification* is not specified, the command returns the currently active frame selection text.

- **smoothrep** *molecule_number rep_number [n]*: Get/set the window size for on-the-fly smoothing of trajectories. Instead of drawing the specified rep from the current coordinates, VMD will calculate the average of the coordinates from the n previous and subsequent timesteps. If n is zero then no smoothing is performed. Note that this smoothing does not affect any label measurements, and does not change the values of the coordinates returned by atom selections or written to files; it only affects how the rep is drawn. Smoothing can be especially useful in visualizing rapidly fluctuating molecules or making movies.

- **scaleminmax** *molecule_number rep_number [min max | auto]*: Get/set the color scale range for this rep. Normally the color scale is automatically scaled to the minimum and maximum of the corresponding range of data. This command overrides the autoscaled values with the values you specify. Omit the *min* and *max* arguments to get the current values. Use "auto" instead of a min and max to rescale the color scale to the maximum range again.

- **showrep** *molecule_number rep_number [on | off]*: Get/set whether the given rep is shown or hidden. Hidden reps cannot be picked and do not show any graphics.

- **volume** *molecule_number* <*volumeset_name*> <*Origin*> <*a*> <*b*> <*c*> *#a #b #c* <*Data*> Add a volumetric data set to the current molecule. Origin, a, b, and c are vectors setting the origin and the three cell vectors. #a, #b, and #c are the number of grid points in the respective cell vector directions and finally the data has to be provided as one list with the data following the grid points along the c-axis fastest, then the b-axis and finally the a-axis.

8.3.20 molecule

Same as mol.

8.3.21 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.

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 [§4.4.2], which is used to determine some parameters, such as the center of view, the data in the animation form, etc.

- **list**: Returns a list of all current molecule id's.

- **num**: Returns the number of loaded molecules.

- **top**: Returns the id of the top molecule.

- **index** *n*: Returns the id of the *n*'th molecule.
- *molecule_id* **get** {*list of keywords*}
- *molecule_id* **set** {*list of keywords*} {*list of values*} Access and, in some cases, modify information about a given molecule. The list of recognized keywords is given in Table 8.3.

```
Examples:
vmd > molinfo top get numatoms
568
molinfo 0 get {filetype filename}
pdb /home/dalke/pdb/bpti.pdb
vmd > molinfo 0 get { {rep 0} {color 0} {rep 1} {color 1} }
{VDW 1.000000 8.000000} {ColorID 5} Lines 1.0000 SegName
```

8.3.22 mouse

Change the current state (mode) of the mouse, optionally active TCL callbacks.

- **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
- **callback on/off**: Turn the callbacks on or off. To use the callbacks, trace the variable `vmd_pick_atom_silent`. See below for information on tracing.

- **rocking on/off**: Enable/disable persistent rotation of the scene with the mouse.
- **stoprotation**: Stop any mouse-initiated scene rotation as well as any rocking initiated with the "rock" command.

8.3.23 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*.

8.3.24 quit

Same as exit.

8.3.25 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:
  - ART
  - Tachyon
  - TachyonInternal
  - snapshot
  - POV3
  - PostScript
  - Radiance
  - Raster3D
  - Rayshade
  - Renderman
  - STL
  - VRML-1
  - VRML-2
- *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.

8.3.26 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.

8.3.27 rotate

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

- **stop**: Stop all rotation, similar to rock off, but it also stops mouse rotations as well.
- **< 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.

8.3.28 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*.

8.3.29 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

8.3.30 tool

Initialize and control the tools that are controlled by external tracking devices.

- **create**: Create a new tool
- **change** *type* `[` *toolid* `]`: Change the type of a tool.
- **scale** *scale* `[` *toolid* `]`: Change the scale of the coordinates reported by a tool.
- **scaleforce** *scale* `[` *toolid* `]`: Increase or decrease the force on a force-feedback device.
- **offset** *x y z* `[` *toolid* `]`: Add a vector to a tool's position.
- **delete** `[` *toolid* `]`: Remove a tool.
- **rep** *molid repid*: Choose only a single representation for tugging or SMD.
- **adddevice** *name* `[` *toolid* `]` : Add a device to a tool, using a name found in the sensor configuration file.
- **removedevice** *name* `[` *toolid* `]` : Remove a device from a tool, using a name found in the sensor configuration file.
- **callback** `on/off` : Enable callbacks for the tools.

8.3.31 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.

8.3.32 user

Add user-customized commands.

- **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 keys** : Print out the current definition of the hot keys.

See section 4.1.3 for examples of the use of the `user` command.

8.3.33 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.

8.3.34 volmap

The `volmap` command creates volumetric maps (3D grids containing a value at each grid point) based on the molecular data, which can then be visualized in VMD using the Isosurface and VolumeSlice representations or using the Volume coloring mode. Also note that the VolMap plugin, accessible from the VMD Extension menu, provides a graphical front-end to many of the `volmap` command's capabilities.

To create a volumetric map, the `volmap` command is run in the following way, where the atom selection specifies the atoms and molecule to include in the calculation, and where the maptype specifies the type of volumetric data to create:

```
volmap <atom selection> <maptype> [optional arguments]
```

For example, to create a mass density map with a cell side of 0.5 Å, averaged over all frames of the top molecule, and add the volumetric data to the top molecule, on would use:

```
volmap [atomselect top "all"] density -res 0.5 -weight mass -allframes \
                                               -combine avg -mol top
```

The various volumetric data map types currently supported by `volmap` are listed as follows. Please note that when a map type refer's to an atoms radius or beta field, *etc.*, that these values will be read directly from VMD's associated fields for that atom. In certain cases, you may want to adjust the atom selections fields (such as radius, beta, *etc.*) before performing the volmap analysis.

- **density**: creates a map of the weighted atomic density at each gridpoint. This is done by replacing each atom in the selection with a normalized gaussian distribution of width (standard deviation) equal to its atomic radius. The gaussian distribution for each atom is then weighted using an optional weight (see the `-weight` argument), and defaults to a weight of one (*i.e*, the number density). The various gaussians are then additively distributed on a grid.

- **distance**: creates a map for which each gridpoint contains the distance between that point and the edge of the nearest atom. In other words, each gridpoint specifies the maximum radius of a sphere cnetered at that point which does not intersect with the spheres of any other atoms. All atoms are treated as spheres using the atoms' VMD radii.

- **ligand**: creates a map of the estimated potential of mean force (in units of $k_B T$ at 300 K) of placing a weakly-interacting gas monoatomic or diatomic ligand at every gridpoint. These results will only be valid when averaging over a large set of frames using the `-combine pmf` option. Like `slowligand` but uses an optimized algorithm (for example, it calculates far-away interactions less often). Unlike slow ligand, it can also computes many samples per gridpoint, which makes the pmf map more accurate. Please refer to and cite: Cohen, J., A. Arkhipov, R. Braun and K. Schulten, "Imaging the migration pathways for O_2, CO, NO, and Xe inside myoglobin", 2006 (Submitted). See additional information about the ligand map type below.

- **slowligand**: Like `ligand`, but uses a slower and more rigourous algorithm. It is used to test the validity of **ligand** under unusual conditions. See additional information about the ligand map type below.

- **mask**: creates a map which is set to 0 or 1 depending on whether they are within a specified cutoff distance (use the `-cutoff` argument) of any atoms in the selection. The mask map is typically used in combination with other maps in order to hide/mask data that is far from a region of interest.

- **occupancy**: Each grid point is set to either 0 or 1, depending on whether it contains onbe or more atoms or not. When averaged over many frames, this will provide the fractional occupancy of that grid point. By default, atoms are treated as spheres using the atomic radii and a gridpoint is considered to be "occupied" if it lies inside that sphere. Use the `-points` argument to treat atoms as points (a grid point is "occupied" if its grid cube contains an atom's center).

The following optional arguments are understood by most volmap map types. Some arguments may only apply to certain map types or may have different meaning for different map types.

- **-allframes**: Use every frame in the molecule instead of just the current one to compute the volumetric map. The method used to combine the various trajectory frame maps can be specified using the `-combine` argument. By default, volmap only uses the current frame.

- **-combine** < **avg** | **max** | **min** | **stdev** | **pmf** >: Specifies the rule to use to combine frames when using the `-allframes` argument. These correspond to keeping the average, maximum or minimum values from the range of calculated frames. `stdev` will return the standard deviation for each point over the range of frames, and `pmf` uses a thermal average $-\ln \sum_i^N e^{-value_i}/N$ for each point. The default is `avg`.

- **-cutoff** *cutoff*: Specifies a cutoff distance. For the distance maps, specifies the largest distance that will be considered (large number is better but slower). For the mask maps, specifies the diatnce from each atom which will be considered part of the mask.

- **-mol** < *molid* | **top** >: Exports the final volumetric data into the VMD molecule specified by *molid*. By default, all maps are exported to a file or name *maptype*_out.dx; using the `-mol` option overrides this.

- **-o** *filename*: Exports the final volumetric data into a DX file (.dx extension is added if missing). By default, all maps are exported to a file or name *maptype*_out.dx.
- **-points**: For the occupancy map type. Treat atoms as point particles intead of as spheres.
- **-radscale** *factor*: For the density map type. Sets a multiplication factor that multiplies all the VMD atomic radii for the purpose of the calculation.
- **-res** *resolution*: Sets the resolution of the map. This means that the volume will be subdivided into many small cubes whose side have a length of *resolution*.
- **-weight** < *field name* | *value list* >: For the density map type. Sets a per-atom weight to be used when computing the density. This can be the name of any VMD numerical atomic field (such as mass, charge, beta, occupancy, user, radius, *etc.*) or else a Tcl list of numbers of the same length as the number of atoms.

For the ligand and slowligand map types, the various frames should be averaged using the "pmf" combination rule, over very many frames. Before starting the computation, the atomic radii of each atom should be set to the corresponding CHARMM Lennard-Jones Rmin/2 parameter, and the beta value of each atom should be set to the CHARMM Lennard-Jones epsilon (energy well depth) parameter. This can be done using VMD's VolMap plugin. Simply call in succession the following commands within the VMD console environment to use default CHARMM values for the various atoms of a molecule:

```
package require volmapgui
VolMapGUI::readcharmmparams [optional list of CHARMM parameter files]
VolMapGUI::assigncharmmparams <molid>
```

The following optional arguments are understood only by the ligand and/or slowligand map types.

- **-dimono**: Use a diatomic molecule probe with two identical atoms, instead of using a single atom as a probe.
- **-dihetero**: Use a diatomic molecule probe with two different atoms, instead of using a single atom as a probe.
- **-conf** *numconf*: Sample a number $numconf$ of different random rotamers of a diatomic molecule at each grid point.
- **-probe1** *epsilon Rmin/2*: Set the CHARMM van der Waals/Lennard-Jones parameters for the monoatomic probe, or for the first atom of a diatomic molecule. Units of *epsilon* are kcal/mol, and of *Rmin/2* are Å.
- **-probe2** *epsilon Rmin/2*: Set the CHARMM van der Waals/Lennard-Jones parameters for the second atom of a diatomic molecule.
- **-bond** *length*: Set the bondlenth of a diatomic molecule. Units of Å.
- **-cutoff** *cutoff*: Set the CHARMM van der Waals cutoff beyong which the interaction between the probe and protein atoms is set to zero.

8.3.35 wait

Specify a number of seconds to wait before reading another command. Animation *continues* during this time. The wait command will not behave as expected if called within a complex Tcl proc or loop structures. The wait command doesn't actually run until the next complete Tcl code block due to the way VMD processes its commands.

- *time*: wait *time* seconds.

8.3.36 sleep

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

- *time*: sleep *time* seconds.

8.4 Tcl callbacks

When certain events occur, VMD notifies the Tcl interpreter by setting certain Tcl variables to new values. You can use this feature to customize VMD, for instance, by causing new graphics to appear when the user picks an atom, or recalculating secondary structure on the fly.

To make these new feature happen at the right time, you need to write a script that takes a certain set of arguments, and register this script with the variable you are interested. Registering scripts is done with the built-in Tcl command `trace`; see `http://www.tcl.tk/man/tcl8.4/TclCmd/trace.htm` for documentation on how to use this command. The idea is that after you register your callback, when VMD changes the value of the variable, your script will immediately be called with the new value of the variable as arguments. Table 8.4 summarizes the callback variables available in VMD.

In the VMD script library at `http://www.ks.uiuc.edu/Research/vmd/script_library/`, you can find a number of scripts that take advantage of Tcl variable tracing. Below, we give a very simple 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
```

Since VMD sets the vmd_pick_atom and vmd_pick_mol variables, they can be traced.

```
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
```

First Word	Description
animate	Play/Pause/Rewind a molecular trajectory.
atomselect	Create atom selection objects for analysis.
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
display	Change various aspects of the graphical display window.
exit, quit	Quit VMD.
gettimestep	Retrieve a timestep as a binary Tcl array (use for plugins)
help	Display an on-line help file with an HTML viewer.
imd	Control the connection to a remote simulation.
label	Turn on/off labels for atoms, bonds, angles, dihedral angles, or springs.
light	Control the light sources used to illuminate graphical objects.
logfile	Turn on/off logging a VMD session to a file or the console.
material	Create new material definitions and modify their settings.
measure	Measure properties of moleculear structures.
menu	Control or query the on-screen GUI menu forms.
molecule or mol	Load, modify, or delete a molecule.
molinfo	Get information about a molecule or loaded file.
mouse	Change the current state (mode) of the mouse.
play	Start executing text commands from a specified file.
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.
stage	Position a checkerboard stage on the screen.
tool	Initialize and control external spatial tracking devices.
translate	Translate the objects in the current scene.
user	Add new keyboard commands.
vmdinfo	(Tcl) Get information about this version of VMD
volmap	Create volumetric data based on molecular information
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 8.1: Summary of core text commands in VMD.

Source	Associated URL
raster3d	http://www.bmsc.washington.edu/raster3d/
msms	http://www.scripps.edu/pub/olson-web/people/sanner/html/msms_home.html
faq	http://www.ks.uiuc.edu/Research/vmd/allversions/vmd_faq.html
biocore	http://www.ks.uiuc.edu/Research/biocore/
tachyon	http://jedi.ks.uiuc.edu/~johns/raytracer/
babel	http://www.eyesopen.com/babel/
homepage	http://www.ks.uiuc.edu/Research/vmd/
quickhelp	http://www.ks.uiuc.edu/Research/vmd/vmd_help.html
radiance	http://radsite.lbl.gov/radiance/HOME.html
maillist	http://www.ks.uiuc.edu/Research/vmd/mailing_list/
scripts	http://www.ks.uiuc.edu/Research/vmd/script_library/
namd	http://www.ks.uiuc.edu/Research/namd/
vrml	http://www.web3d.org/
rayshade	http://www-graphics.stanford.edu/~cek/rayshade/rayshade.html
povray	http://www.povray.org/
plugins	http://www.ks.uiuc.edu/Research/vmd/plugins/
python	http://www.python.org/
software	http://www.ks.uiuc.edu/Research/vmd/allversions/related_programs.html
tcl	http://www.tcl.tk/
userguide	http://www.ks.uiuc.edu/Research/vmd/vmd-1.8.1/ug/ug.html

Table 8.2: On-line Help Sources

Keyword	Aliases	Arg	Set	Description
id		*int*	N	molecular id
index		*int*	N	index on the molecule list
numatoms		*int*	N	number of atoms
name		*str*	N	the name of the molecule (usually the name of the file)
filename		*str*	N	list of filenames for all files loaded for this molecule
filetype		*str*	N	list of file types for this molecule
database		*str*	N	list of databases for this molecule
accession		*str*	N	list of database accession codes for this molecule
remarks		*str*	N	list of freeform remarks for this molecule
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
timesteps		*int*	Y	number of elapsed timesteps in an interactive simulation
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
pressure		*float*	Y	the simulation pressure
volume		*float*	Y	the simulation volume
efield		*float*	Y	efield
alpha		*float*	Y	unit cell angle alpha in degrees (for the current frame)
beta		*float*	Y	unit cell angle beta in degrees (for the current frame)
gamma		*float*	Y	unit cell angle gamma in degrees (for the current frame)
a		*float*	Y	unit cell length a in Angstroms (for the current frame)
b		*float*	Y	unit cell length b in Angstroms (for the current frame)
c		*float*	Y	unit cell length c in Angstroms (for the current frame)

Table 8.3: `molinfo set/get` keywords

Table 8.4: Description of Tcl callback variables in VMD.

When called	Name	New value
Molecule `molid` was deleted	`vmd_molecule(molid)`	0
Molecule `molid` was created (data may not have been loaded yet)	`vmd_molecule(molid)`	1
Molecule `molid` was renamed	`vmd_molecule(molid)`	2
Structure file loaded	`vmd_initialize_structure(molid)`	1
Coordinate file loaded	`vmd_trajectory_read(molid)`	*name of coordinate file*
Molecule `molid` changed animation frames	`vmd_frame(molid)`	*new animation frame*
Any VMD command executed	`vmd_logfile`	Tcl text equivalent of command
Bond, angle or dihedral label added	`vmd_pick_value`	Value of geometry label
Atom picked	`vmd_pick_mol`	id of picked molecule
Atom picked	`vmd_pick_atom`	id of picked atom
Pointer moved.	`vmd_pick_client`	name of pointer
Pointer moved.	`vmd_pick_mol_silent`	id of nearby mol
Pointer moved.	`vmd_pick_atom_silent`	id of nearby atom
Atom picked	`vmd_pick_shift_state`	1 if shift key down during pick, 0 otherwise
IMD coordinate set received	`vmd_timestep(molid)`	frame containing new coordinates
Set of labels to be graphed	`vmd_graph_label`	{labeltype labelid} {labeltype labelid} ...
Tcl interpreter is shutting down	`vmd_quit`	1

Chapter 9

Python Text Interface

VMD 1.6 and later contain an embedded, fully-functional Python interpreter. The interpreter acts just like the Python command line: you can import your own modules and run them from with the text console of VMD. In addition, VMD provides a number of modules for loading molecules and controlling their display.

VMD currently uses Python version 2.2.

9.1 Using the Python interpreter within VMD

When you start VMD, the VMD text console normally uses the Tcl command interpreter to process what you type. In order to use the Python interpreter, you have to tell VMD to switch to 'Python mode'. There are three ways to do this: (1) Type `gopython` in the console window; (2) pass `-python` as a command line option; or (3) put `gopython` on the last line of your `.vmdrc` file. If VMD prints an error message reporting that the Python interpreter is not available, your version of VMD was not compiled with Python support; contact the VMD developers for help. If all goes well, you should see Python command prompt '>>> ' in the console window. To switch back to the Tcl interpreter, press Ctrl-D as though you were exiting Python. Switching back and forth between Python and Tcl does not destroy any of your work; all variables and modules will still be defined until you exit VMD.

Typing 'gopython <filename>', where <filename> is the name of a file containing Python code will cause VMD to switch to Python mode, process the file, then switch back to Tcl. In this way, you can embed Python functions inside your Tcl scripts!

You can also type 'gopython -command "*your code here*" to run an arbitrary line of python code.

9.2 Python modules within VMD

Once you enter the VMD Python environment, you will find a module called "VMD" already loaded. This module contains all the other built-in modules for writing VMD Python scripts.

VMD is not distributed with an entire Python environment. In order to use the set of libraries that normally come with a Python distribution, you must tell Python where to find the libraries. There are two primary means of doing this. The `PYTHONHOME` environment variable points to the location where Python is installed; the version installed at this point must match VMD's version (2.2). Thus if you have Python libraries in /usr/local/lib/python2.2, adding the line `set env(PYTHONHOME) /usr/local` to your startup script or .vmdrc file will do the trick.

If you have additional modules that you want to use within VMD, use the `PYTHONPATH` environment variable to tell Python where to find them. See any Python book for more information.

if the Tkinter module is found in the Python installation, VMD will load it at Python startup in order to make Tkinter windows work in harmony with windows created from within Tk. In addition, if you have Numeric Python installed in your system, a submodule called vmdnumpy will become available within the VMD module; see below for details.

9.3 Atom selections in Python

VMD provides an atom selection class for use in the Python interpreter. Instances of this class correspond to a set of atom indices in a particular molecule for a particular coordinate set. Once an atom selection is made, you can query the properties of the selected atoms, such as their names, residue ids, or coordinates. In a similar fashion, you can set the values of these properties. You can also perform logical operations on atom selections, including finding the intersection or union of two atom selections or finding the inverse of the set. Finally, you can perform tuple operations on the atom selection object to query the indices of the atoms in the selection.

Atom selection macros can be defined using the `macro` method of the AtomSel module. The syntax is just as in the corresponding `atomselect macro` and `atomselect delmacro` Tcl commands; see section 8.3.2 for details.

Below we summarize the methods available from the AtomSel class.

- `AtomSel(selection = 'all', molid = 0, frame = 0)`: Creates a new atom selection object.

  >>> sel = AtomSel('name CA', 1) # Selects the alpha carbons of molecule 1

- `select(selection)`: Change the selected atoms.

  >>> sel.select('resid 5')

- `list()`: Return a copy of the selected atom indices.

- `frame(value = -1)`: Set/get the coordinate frame for the selection. Nonpositive values will return the current value of the frame without changing it.

  >>> sel.frame(5)

  >>> sel.frame()

  5

- `get(attr1, attr2, ...)`: Takes any number of string arguments, which should correspond to a valid atom property such as "name", "x", or "water". Returns a list of the value of the property for each atom in the selection. For boolean properties such as "water", the returned value will be 1 if true and 0 if false.

  >>> x, y, z = sel.get('x', 'y', 'z')

- `set(attr, val)`: Set the atom property corresponding to *attr* using the values in *val*. The number of elements in *val* should be either 1 or the number in the atom selection.

>>> len(sel)

12

>>> sel.set('beta',3)

>>> sel.set('beta',(1,2,3,4,5,6,7,8,9,10,11,12))

- `write(filename, filetype=None)`: Write the atoms in the selection to filename. Filetype is guessed from the filename, or can be specified with filetype.
- `sel1 & sel2`: Create a new atom selection using the atoms found in both *sel1* and *sel2*.
- `sel1 | sel2`: Create a new atom selection using the atoms found in either *sel1* or *sel2*.
- `-sel`: Create a new atom selection using the atoms not found in *sel*.
- `len(sel)`: Returns the number of atoms in the selection.
- `sel[0], sel[0:3]`: Index and slice operations return the corresponding atoms in the selection.
- `center(weight=None)`: Return the center of the selected atoms, possibly weighted by `weight`, which must be a sequence.
- `sasa(srad, samples=-1, points=None, restrict=None)`: Returns the solvent-accessible surface area (SASA) of atoms in the selection using the assigned radius for each atom, extending each radius by `srad` to find the points on a sphere that are exposed to solvent. If a *restrict* selection is given, only solvent-accessible points near that selection bill be considered. The `points` parameter can be used to collect the points which are determined to be solvent-accessible; this must be a list variable.
- `getbonds()`: Returns a list of the atoms bonded to each atom in the selection.
- `setbonds(bonds)`: Set the bonds for the atoms in the selection. `bonds` must be a list of the same length as the selection; each element in the list must be a sequence containing the indices of the atoms to which the atom has a bond.
- `minmax()`: Returns the minimum and maximum coordinates of the atoms in the selection as a tuple of the form `(xmin, ymin, zmin), (xmax, ymax, zmax)`.
- `rmsd(sel, frame=None, weight=None)`: Returns the root-mean-square distance of the atoms in `sel` from the selection. If `frame` is given, the coordinates from the corresponding frame will be used (see the example). If `weight` is given, the computed RMSD will be weighted using the values in `weight`.
- `align(ref=None, move=None, frame=None, weight=None)`: Finds the transformation that aligns the atoms in the selection with the atoms in `ref`, with optional weights `weight`, and applies this transformation to the atoms in `move`. The following default values for all arguments are provided:
  - `ref`: Same molecule and atoms as selection, but always using the

- align(ref=None, move=None, frame=None, weight=None):
  Finds the transformation that aligns the atoms in the selection with the atoms in ref, with optional weights weight, and applies this transformation to the atoms in move. The following default values for all arguments are provided:
    * ref: Same molecule and atoms as selection, but always using the first timestep in the molecule.
    * move: All atoms in the selection molecule.
    * frame: overrides both the selection's frame and the move frame, but does not affect the ref frame.
    * weight: Defaults to uniform weights on all atoms in selection.

  first timestep in the molecule.
- move: All atoms in the selection molecule.
- frame: overrides both the selection's frame and the move frame, but does not affect the ref frame.
- weight: Defaults to uniform weights on all atoms in selection.

• contacts(cutoff, sel=None): Returns pairs of of atoms within cutoff of each other. If sel is None, atoms in the pairs must be in the selection; otherwise, the first atom in each pair will be from the selection, and the second will be from sel.

9.3.1 An atom selection example

In the first example, we load the molecule alanin.pdb, and create an atom selection consisting of the alpha carbons. Note that AtomSel is the name of the class which generates atom selection instances. We show the string representation of the object by entering its name and pressing return; this shows the text used in the selection.

Next we demenstrate how atom selections act like tuples: we can get their length using the built-in len() command,and return a copy of the selected atoms in a tuple by using the slice operator [:].

Finally, we demonstrate the get and set operations. The get() operation takes any number of string arguments; for each argument, it returns a Python list of values corresponding to that string. The set() operation allows only one property to be changed at a time. However, you can pass in either a single value, which will be applied to all atoms in the selection, or a tuple or list of the same size as the atom selection, in which case values in the list will be assigned to the corresponding atom in the selection. We take advantage of this behavior in the following example by first saving the current value of beta for the selection, then setting the value of beta to 5 for all selected atoms, and finally resetting the original values using the results of the get().

```
>>> from molecule import *
>>> from AtomSel import AtomSel
>>> load('alanin.pdb')

>>> CA = AtomSel('name CA')
>>> CA
name CA
>>> len(CA)
```

```
12
>>> CA[:]
(0, 5, 11, 17, 23, 29, 35, 41, 47, 53, 59, 65)
>>> resname, resid = CA.get('resname', 'resid')
>>> resname
['ACE', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'ALA', 'C
BX']
>>> resid
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>> x,y,z=CA.get('x','y','z')
>>> x
[-2.1840000152587891, 1.4500000476837158, 1.9809999465942383, 0.54100000858306885,
2.8090000152587891, 5.9079999923706055, 5.0440001487731934, 4.5659999847412109,
7.9340000152587891, 9.7329998016357422, 8.1689996719360352, 9.2229995727539062]
>>> y
[0.5910000205039978, 0.0, 3.6429998874664307, 4.8410000801086426, 2.5559999942779541,
3.7860000133514404, 7.4190001487731934, 6.7989997863769531, 5.0819997787475586,
7.9559998512268066, 10.515999794006348, 8.5710000991821289]
>>> z
[0.9100000262260437, 0.0, -0.9089999794960022, 2.3880000114440918, 4.3920001983642578,
2.5859999656677246, 3.244999885559082, 6.9559998512268066, 7.2639999389648438,
5.5669999122619629, 7.8870000839233398, 11.013999938964844]

>>> beta = CA.get('beta')
>>> CA.set('beta',5)
>>> CA.set('beta',beta)
>>>
```

9.3.2 Changing the selection and the frame

When molecule in VMD contains multiple coordinate sets (frames), atom selections must know which frame they are referring to, especially when you make distance-based atom selections or request time-varying properties like the x, y, or z coordinates. By default, atom selections in Python use frame 0, i.e. the first coordinate set. You can specify the frame either when you create the atom selection, or by using the frame() method. Passing no arguments to frame() returns the current value of the frame.

```
>>> load('psf','alanin.psf','dcd','alanin.dcd')

>>> resid5 = AtomSel('resid 5', frame=50)
>>> resid5.frame()
50
>>> resid5.frame(22)
>>> resid5.frame()
22
```

In a similar way, you can change the selected atoms of an atom selection object using the select() operation. Continuing with the previous example:

```
>>> resid5
resid 5
>>> resid5.select('resid 7')
>>> resid5
resid 7
>>>
```

9.3.3 Combining atom selections

Once you've created one or more atom selections, you can combine them to create new ones.

```
>>> CA = AtomSel('name CA')
>>> resid5 = AtomSel('resid 5')
>>> CA
name CA
>>> resid5
resid 5
>>> ANDsel = CA & resid5
>>> ORsel = CA | resid5
>>> NOTsel = -CA
>>> ANDsel
(name CA) and (resid 5)
>>> ORsel
(name CA) or (resid 5)
>>> NOTsel
not (name CA)
>>>
```

When the combined atom selections are from different molecules or have different frame numbers, the molecule and frame from the first atom selection are used.

9.3.4 RMS example

Example: find the mass-weighted RMS distance from the initial frame. This assumes a molecule and its timesteps have already been loaded (see the description of the Molecule class).

```
from AtomSel import AtomSel
from Molecule import *

# Get a reference to the first molecule.
m=moleculeList()[0]

# Select all atoms.
sel=AtomSel()
```

```
# We are comparing to the first frame.
sel.frame(0)

# Get the mass weights.
mass = sel.get('mass')

Here's another RMSD example that uses the {\tt align} method:

\begin{verbatim}
from AtomSel import AtomSel
from Molecule import Molecule

mol1=Molecule()
mol1.load('proteins/alanin.psf')
mol1.load('proteins/alanin.dcd')
n = mol1.numFrames()

sel=AtomSel('backbone')

# align all frames with the first frame, using the backbone atoms
for i in range(1,n):
  sel.align(frame=i)

# align all frames with frame 10.
for i in range(1,n):
  sel.align(ref=sel.frame(10), frame=i)

# Align residues 1-3 from frame 10 with frame 20, but use all backbone atoms
# to perform the fit.
resid123=AtomSel('resid 1 to 3')
sel.align(ref=sel.frame(20), frame=10, move=resid123)
# GOTCHA ALERT: sel.frame(10).align(ref=sel.frame(20)) does not work!!
# That's because sel.frame(10) overrides frame 20 in this case since they
# are applied to the same AtomSel instance.  Either use the frame argument,
# as illustrated here, or create a new AtomSel instance for the reference.

# Perform a mass-weighted RMSD alignment and compute the mass-weighted
# RMS distance from the first frame.
w=sel.get('mass')
rms=[]
ref=AtomSel('backbone',frame=0)
for i in range(n):
  rms.append(sel.frame(i).align(ref=ref, weight=w).rmsd(ref, weight=w))

# Initialize result list.
rms=[]
```

```
# Go!
for i in range(m.numFrames()):
  rms.append(sel.rmsd(sel, frame=i, weight=mass))
```

Here's another RMSD example that uses the `align` method:

```
from AtomSel import AtomSel
from Molecule import Molecule

mol1=Molecule()
mol1.load('proteins/alanin.psf')
mol1.load('proteins/alanin.dcd')
n = mol1.numFrames()

sel=AtomSel('backbone')

# align all frames with the first frame, using the backbone atoms
for i in range(1,n):
  sel.align(frame=i)

# align all frames with frame 10.
for i in range(1,n):
  sel.align(ref=sel.frame(10), frame=i)

# Align residues 1-3 from frame 10 with frame 20, but use all backbone atoms
# to perform the fit.
resid123=AtomSel('resid 1 to 3')
sel.align(ref=sel.frame(20), frame=10, move=resid123)
# GOTCHA ALERT: sel.frame(10).align(ref=sel.frame(20)) does not work!!
# That's because sel.frame(10) overrides frame 20 in this case since they
# are applied to the same AtomSel instance.  Either use the frame argument,
# as illustrated here, or create a new AtomSel instance for the reference.

# Perform a mass-weighted RMSD alignment and compute the mass-weighted
# RMS distance from the first frame.
w=sel.get('mass')
rms=[]
ref=AtomSel('backbone',frame=0)
for i in range(n):
  rms.append(sel.frame(i).align(ref=ref, weight=w).rmsd(ref, weight=w))
```

9.4 Python callbacks

Some of your Python scripts may wish to be informed when various events in VMD occur. The mechanism for expressing this interest is to register a callback function with a special module supplied by VMD. When the event of interest occurs, all registered will functions will be called; VMD will pass the functions information specific to the event. The set of callbacks events is listed in Table 9.4.

Table 9.1: Description of callbacks available to scripts running in the embedded Python interpreter.

Name	When called	Function arguments
display_update	Screen redraw	none
frame	Molecule changes coordinate frame	(molid, frame)
help	User pushes help button on main form	(name of topic)
initialize_structure	Molecule created or deleted	(molid, 1 or 0)
pick_atom	Atom picked in graphics window	(molid, atomid, key_shift_state (1 if shift pressed, 0 otherwise))
pick_value	Bond, angle, or dihedral label created	(value)
timestep	New IMD coordinate frame received	(molid, frame)
trajectory	Completion of coordinate file read/write	(molid, filename)

All callback functions must take two arguments. The first argument will be an object given at the time the function is registered; VMD makes no use of this object, it simply saves it and passes it to the callback when the callback function is called. The second argument to the callback function will be a tuple containing 0 or more elements, depending on the type of callback. The type of information for each callback is listed in the third column of Table 9.4.

Callbacks are registered/deregistered using the add_callback/del_callback methods of the VMD.vmdcallbacks module. The syntax for these methods is:

```
def add_callback(name, func, userdata = None):

def del_callback(name, func, userdata = None):
```

`name` should be one of the callback names in Table 9.4. `func` is the function object. `userdata` is any object; if no object is supplied, None will be passed as the first argument to the callback function. To unregister a callback, use the same name, func, and userdata as were used when the callback was registered. The same function may be registered multiple times with different userdata objects.

9.4.1 Using Tkinter menus in VMD

The object-oriented interface to Tk known as Tkinter is included with the embedded Python interpreter. You can create Tkinter GUI's in the usual way, with one caveat: the Tkinter.mainloop() method should never be called, as it will interfere with VMD's own event loop. VMD will take care of updating your GUI windows for you.

9.5 Controlling VMD from Python

Commands for controlling VMD from Python are organized into modules which roughly correspond to Tcl commands. Importing all the commands in a module is not recommended as some of the functions (e.g., listall()) overlap. All commands are listed below, with the name of the module given by the section heading.

9.5.1 animate

Python operations available from the *animate* module, used to control which coordinate frames are displayed.

- `forward()`:
- `reverse()`: `forward()` and `reverse()` causes VMD to start animating frames automatically in order of increasing or decreasing frame number, respectively.
- `once()`:
- `rock()`:
- `loop()`: `once()`, `rock()`, and `loop()` control how frames are cycled when VMD is animating a series of frames. `once()` causes VMD to stop when it reaches the first or last frame. `rock()` causes VMD to reverse direction each time it gets to the beginning or end. `loop()` causes VMD to continue from the beginning when reaches the last frame, or from the last frame if it gets to the beginning.
- `style()`: Returns either `'Once'`, `'Rock'`, or `'Loop'`, corresponding to the animation mode VMD is currently in.
- `goto(frame)`: Set the animation to the given frame, and pause the animation.
- `prev()`: Step to the next-lowest frame, then pause.
- `next()`: Step to the next-highest frame, then pause.
- `pause()`: Stop animating frames.
- `speed(value)`: Get/set the relative rate at which VMD animates frames. `value` should lie between 0 and 1. If a value less than 0 is given, then the speed will not be changed. The new value of the speed is always returned.
- `skip(value)`: Get/set the number of frames to skip when animating. A value of 1 means every frame is shown; 2 means every other frame is shown; etc. If `value` is 0 or less, no change is made. The new value of the speed is always returned.
- `is_active(molid)`: Returns whether the molecule with the given id is active; that is; whether it responds to animation or not.
- `activate(molid, trueorfalse)`: Make the molecule with the given id active or not. Active molecules update their coordinate frames during animation; inactive molecules do not.

9.5.2 axes

Python operations available from the *axes* module, used to change where the axes are displayed in the graphics window.

- `OFF, ORIGIN, LOWERLEFT, LOWERRIGHT, UPPERLEFT, UPPERRIGHT`: String constants defined in the *axes* module for setting the location of the axes.

- `get_location()`: Returns a string object corresonding the current location of the axes.
- `set_location(location)`: Set the location of the axes, using one of constants defined in the module.

9.5.3 color

Python operations available from the *color* module, used to change the color definitions, color maps, or edit the color scale. All rgb and color scale values should be in the range 0 to 1.

- `categories()`: Returns a list of available color categories.
- `get_colormap(name)`: Returns a dictionary of name/color pairs for the given color category.
- `set_colormap(name, dict)`: Change the color definitions for the colors in the given color category. The keys in `dict` must come from the keys listed by `get_colormap` for that color category, though not all keys need be listed. The values must be legal color names.
- `get_colors()`: Returns a dictionary whose keys are all the legal color names and whose corresponding values are the RGB values of the color, represented as a 3-tuple.
- `set_colors(dict)`: Changes RGB values for colors in VMD. Keys must be chosen from the keys returned by `get_colors()`. Values must be 3-tuples of floats.
- `scale_method()`: Returns the current color scale method.
- `scale_methods()`: Returns a list of all available color scale methods.
- `scale_midpoint()`: Returns the current color scale midpoint.
- `scale_min()`: Returns the current color scale minimum.
- `scale_max()`: Returns the current color scale maximum.
- `set_scale(method, midpoint, min, max)`: Change the color scale method, midpoint, minimum, or maximum. All properties may be set using keyword arguments.

9.5.4 display

Python operations available from the *display* module, used to control the VMD camera as well as screen updates.

- `update()`: Force a display update, without checking the VMD FLTK menus
- `update_ui()`: Update the display as well as any other user interfaces.
- `update_on()`: Tell VMD to regularly update the display and the FLTK menus
- `update_off()`: Tell VMD not to regularly update the display. The display will be updated only when display.update() is called.
- `stereomodes()`: Returns a list of the available stereo modes.

- `PROJ_PERSP, PROJ_ORTHO`: String constants defined in the `display` module for setting the `projection` keyword in the `set` method.
- `set(**keywordlist)`:
- `get(key)`: set() and get() control various display properties. The following keywords accept/return floating-point values: `eyesep, focallength, height, distance, nearclip, farclip`. The following keywords accept boolean values for on or off, respectively: `antialias, depthcue, culling`. `stereo` should be one of the values returned by `stereomodes()`. `projection` should be one of the PROJ constants defined in this module. `size` should be a list of two integers corresponding to the width and height of the display in pixels.

9.5.5 graphics

Python operations available from the *graphics* module, used to create custom 3-D objects from graphics primitives. The first argument to all operations is the id of a Graphics molecule. Graphics molecules are created using the `load()` command in the *molecule* module: `load('graphics', 'test')` creates a Graphics molecule named 'test'. For vertices and normals, a tuple with three float items is required.

- `triangle(id, v1, v2, v3)`: Draw a triangle with the given vertices.
- `trinorm(id, v1, v2, v3, n1, n2, n3)`: Draw a triangle with the given vertices and vertex normals.
- `cylinder(id, v1, v2, radius=1.0, resolution=6, filled=0)`: Draw a cylinder with endpoints specified by the given points. Radius, resolution, and filled (whether the ends should be capped or not) may be optionally specified with keyword arguments.
- `point(id, v)`: Draw a point at the given coordinates.
- `line(id, v1, v2, style='solid', width=1)`: Draw a line between the given vertices. Optionally, the line style may be specified as either 'solid' or 'dashed', and width may be any positive integer.
- `materials(id, onoff)`: Turns materials on/off for subsequent graphics primitives. Primitives lying earlier in the stack are not affected. `onoff` should be either 0 (off) or 1 (on).
- `material(id, name)`: Sets the material for all graphics primitives in this molecule. `name` should be one of the material names returned by `material.listall()`.
- `color(id, color)`: Set the color for subsequent graphics primitives. `color` may be (1) a tuple containing three floats specifying the RGB value, (2) an integer corresponding to a color index, or (3) a string corresponding to a color name.
- `cone(id, v1, v2, radius=1.0, resolution=6)`: Draw a cone with base at `v1` and point at `v2`. radius and resolution may optionally be specified with keyword arguments.
- `sphere(id, center=(0.0, 0.0, 0.0), radius=1.0, resolution=6)`: Draw a sphere. The sphere center, radius, and resolution may optionally be specied with keyword arguments.

- `text(id, pos, text, size=1.0)`: Draw text at the given position (specified by a tuple of three floats, using the string `text`. Size may optionally be specified using keyword arguments.
- `delete(id, index)`: Deletes the graphics primitive with the given index. Alternatively, if the string 'all' is passed as the index, all graphics primitives will be deleted.
- `replace(id, index)`: Deletes the graphics primitive with the given index. The next graphics primitive added will go in the newly vacated position. Subsequent graphics primitives will resume at the end of the list.
- `info(ind, index)`: Returns a string describing the graphics primitive with the given index. If the index is invalid, an IndexError exception is raised.
- `listall(ind)`: Returns the indices of the valid graphics primitives in a list.

9.5.6 imd

Python operations available from the *imd* module, used to display and interact with a molecule in a molecular dynamics simulation.

- `connect(host, port)`: Connect to a simulation running on host `host` and listening for incoming connections on port `port`.
- `pause()`: If connected, cause the simulation to pause.
- `detach()`: If connected, detach from the simulation. The simulation will continue to run, but no more frames will be received until a connection is re-established.
- `kill()`: If connected, terminate the simulation. The connection will also be abolished.
- `transfer(rate)`: Set/get how often the remote simulation sends coordinate frames to VMD. If *rate* is omitted or is negative, no action is taken and the current value is returned. A value of 1 corresponds to every frame being sent; a value of 2 corresponds to every other frame, etc.
- `keep(rate)`: Set/get how often received coordinates frames are kept by VMD as part of an animation. If *rate* is omitted or is negative, no action is taken and the current value is returned. A value of 0 means no frames are saved. A value of 1 corresponds to every frame being saved; a value of 2 corresponds to every other frame, etc.

9.5.7 label

Python operations available from the *label* module, used to create, show/hide, and delete labels for atoms, bonds, angles, or dihedrals.

- `ATOM, BOND, ANGLE, DIHEDRAL`: Label types defined by the label module, for use as the first argument to the add, listall, show, hide, and delete methods.

- `add(type, molids, atomids)`: Create a label of the given type. `molids` and `atomids` must be tuples containing 1, 2, 3, or 4 integers for ATOM, BOND, ANGLE, or DIHEDRAL labels, respectively. If the label already exists, no action is performed. Returns a dict corresponding to the referenced label that can be used in the show(), hide(), delete(), and getvalues() methods.

- `listall(type)`: Returns a list of labels of the given type. The elements of the list are python dictionary objects, with the following keys: `molid`, `atomid`, `value`, `on`. The values for `molid` and `atomid` are tuples containing the molecule id and atom id for the label. `value` is the numerical value of the geometry label, or zero for ATOM labels. `on` is 1 if the label is shown, and 0 if the label is hidden.

- `show(type, label)`: Turn the given label on. `label` must be a dictionary containing `molid` and `atomid` keys whose values are tuples. If the tuples match the molecule ids and atom ids of the atoms in an existing label, the label will be turned on. Raises ValueError if the label does not exist.

- `hide(type, label)`: Turn the given label off. `label` must be a dictionary containing `molid` and `atomid` keys whose values are tuples. If the tuples match the molecule ids and atom ids of the atoms in an existing label, the label will be turned off. Raises ValueError if the label does not exist.

- `delete(type, label)`: Delete the given label. `label` must be a dictionary containing `molid` and `atomid` keys whose values are tuples. If the tuples match the molecule ids and atom ids of the atoms in an existing label, the label will be deleted. Raises ValueError if the label does not exist.

- `getvalues(type, label)`: Returns a list of values of the given label for each coordinate frame in the label. If the atoms in the label belong to different molecules, only the coordinates of the first molecule will be cycled. If the labels don't have values (like atom labels), None is returned.

9.5.8 material

Python operations available from the *material* module, used to create and modify material properties of molecular representations.

- `listall()`: Returns a Python list of the names of all available materials.

- `settings(name)`: Returns a Python dictionary of the material settings for material with the given `name`.

- `add(name=None, copy=None)`: Create a new material with the given name. Optionally, copy the properties from material `copy` into the new material. If no name is given, a new one will be provided.

- `delete(name)`: Delete the material with the given name.

- `rename(oldname, newname)`: Rename the material with the given name. The new name must not yet be used.

- `change(name, ambient, specular, diffuse, shininess, opacity)`: Change one or more of the material settings for the material with the given name. Keyword arguments may be used to specify each property.

9.5.9 molecule

Python operations available from the *molecule* module, used to load molecules and change their representations.

- `num()`: Returns the number of loaded molecules.
- `listall()`: Returns the molid's of the all the loaded molecules.
- `exists(molid)`: Returns true if the molid corresponds to an existing molecule.
- `new(name)`: Creates a new empty molecule with the given name and returns its id.
- `load(structure, sfname, coor, cfname)`: Load a molecule with structure type *structure* and filename *sfname*. Additionally, a separate coordinate file may be provided, of type *coor* and name *cfname*. **New in VMD 1.8:** All frames from *cfname* will be processed before the function returns. If successful, the function will return the id of the new molecule.

  >>> load('pdb','alanin.'pdb')

  >>> load('psf','alanin.psf','dcd','alanin.dcd')
- `cancel(molid)`: Cancel loading of coordinates file for the given molecule.
- `delete(molid)`: Delete the specified molecule.
- `read(molid, type, filename, beg = 0, end = -1, skip = 1, waitfor = 1, volsets = [1])`:
- `write(molid, type, filename, beg = 0, end = -1, skip = 1, waitfor = 1)`: Read or write a file to/from the specified molecule. For reading, if `molid` is -1, a new molecule will be created. Optional arguments `beg`, `end`, and `skip` may be specified with keywords; the default is to load/save all coordinate frames. **New in VMD 1.8:** The `waitfor` option will cause VMD to process the specified number of frames before returning. If `waitfor` is negative, all frames from the file will be processed before the function returns. For reading files containing volumetric datasets, set the volsets parameter to a list of set id's, starting from 0, to specify which datasets to load.
- `add_volumetric(molid, name, origin, xaxis, yaxis, zaxis, xsize, ysize, zsize, data)`: Add a volumetric data set to the given molecule. `origin`, `xaxis`, `yaxis`, and `zaxis` must be 3-tuples specifying the center and scale of the data. `xsize`, `ysize` and `zsize` give the number of elements along each dimension. `data` must be a Python list of the correct size as indicated by the three sizes.
- `get_filenames(molid)`: Returns a list of filenames that have been loaded into this molecule.
- `get_filetypes(molid)`: Returns a list of filetypes corresponding to `get_filenames`.

- `get_databases(molid)`: Returns a list of databases corresponding to `get_filenames`.
- `get_accessions(molid)`: Returns a list of accessiosn corresponding to `get_filenames`.
- `get_remarks(molid)`: Returns a list of remarks corresponding to `get_filenames`.
- `delframe(molid, beg=0, end=-1, skip=1)`: Delete frames from the specified molecule. Optional arguments `beg`, `end`, and `skip` may be specified with keywords; the default is to delete all coordinate frames.
- `dupframe(molid, frame)`: Copy the coordinates from the given frame and append them as a new frame.
- `numframes(molid)`: Return the number of coordinate frames in the specified molecule.
- `get_frame(molid)`: Return the current coordinate frame for the specified molecule.
- `set_frame(molid, frame)`: Set the current coordinate frames in the specified molecule.
- `numatoms(molid)`: Returns the number of atoms in the specified molecule.
- `ssrecalc(molid)`: Recalculate the secondary structure for the given molecule, using the current set of coordinates.
- `name(molid)`: Returns the name of the given molecule.
- `rename(molid,newname)`: Rename the given molecule.
- `get_top(molid)`:
- `set_top(molid)`: Get/set the molid of the top molecule.
- `get_periodic(molid, frame=-1)`:
- `set_periodic(molid, frame=-1, a, b, c, alpha, beta, gamma)`: Get/set periodic image settings for the given molecule and timestep (frame). `get_periodic` returns a dictionary whose keys are `a, b, c, alpha, beta, gamma`. `set_periodic` sets the corresponding values; negative values will be ignored.

9.5.10 molrep

Python operations available from the *molrep* module, used to add and modify representation of molecules.

- `num(molid)`: Returns the number of representations in the given molecule.
- `addrep(molid, style=None, color=None, selection=None, material=None)`: Add a representation to the specified molecule. If any of the optional keywords are specified as well, the new rep will have the specified properties. Note that these properties become the default for future calls to addrep, so that `addrep(0, style='VDW'); addrep(0, color='Name')` will create two reps, each with a style of 'VDW'.
- `delrep(molid, rep)`: Delete the specified rep from the given molecule.

- `modrep(molid, rep, style, sel, color, material)`: Modify the style, atom selection, color, and/or material for the specified molecule and representation. Any combination of the last four arguments may be specified, using positional or keyword arguments. Returns success.

  >>> modrep(0,0,color='name') # Color the first rep of molecule 0 by name.

  >>> modrep(0,2, selection='name CA', material='Transparent') # For the third representation of molecule 0, change the atom selection to "name CA" and the material to "Transparent"

- `get_style(molid, rep)`:

- `get_selection(molid, rep)`:

- `get_color(molid, rep)`:

- `get_material(molid, rep)`: Returns the representation style, selection, color, or material, respectively, for the given representation of the given molecule.

- `get_repname(molid, rep)`:

- `repindex(molid, name)`: These two commands let you assign names to reps and access them by that name. The name returned by `get_repname` is guaranteed to be unique for all reps in the molecule, and will stay with the rep it was assigned to even when the order of the reps changes. Use `repindex` to find the repid of the rep with the given name; -1 is returned if no rep with that name exists.

- `get_autoupdate(molid, rep)`:

- `set_autoupdate(molid, rep, onoff)`: These two commands let you turn on/off automatic updating of the atom selection for a given rep. Automatic updating means the atom selection for the rep will be recalculated every time the coordinate frame of the molecule changes.

- `get_colorupdate(molid, rep)`:

- `set_colorupdate(molid, rep, onoff)`: These two commands let you turn on/off automatic updating of the color for a given rep. Automatic updating means the color for the rep will be recalculated every time the coordinate frame of the molecule changes; this is useful for coloring by Position or User.

- `get_smoothing(molid, rep)`:

- `set_smoothing(molid, rep, n)`: These two commands let you get/set on-the-fly smoothing of molecular representations. Atom coordinates used to draw the given rep will be smoothed with a moving average window size of $2n - 1$.

- `get_scaleminmax(molid, rep)`:

- `set_scaleminmax(molid, rep, min, max)`:

- `reset_scaleminmax(molid, rep)`: Get/set the color scale range for this rep. Normally the color scale is automatically scaled to the minimum and maximum of the corresponding range of data. This command overrides the autoscaled values with the values you specify. Omit the *min* and *max* arguments to get the current values. Use `reset_scaleminax` to rescale the color scale to the maximum range again.

- `get_visible(molid, rep)`:

- `set_visible(molid, rep, onoff)`: These two commands let you show or hide a selected molecular representation, and retrieve the visibility status of a given rep.

9.5.11 render

Python operations available from the *render* module, used to export the scene to a file that can be read by external rendering programs.

- `listall()`: Return a Python list of the names of all supported rendering methods. One of these should be the first argument to the `render()` operation below.

- `render(method, filename)`: Using the the given rendering method, export the current scene to the file `filename`. `method` should be one of the values returned by `listall()`.

9.5.12 trans

Python operations available from the *trans* module, used to change the view of the rendered scene.

- `rotate(axis, angle)`: Rotate the scene about the specified axis by the given angle. `axis` should be 'x', 'y', or 'z'; `angle` is measured in degrees.

- `translate(x, y, z)`: Translate the scene by the given x, y, and z values.

- `scale(factor)`: Scale (zoom) the scene by the given factor.

- `resetview(molid)`: Sets the center, scale, rotation for all molecules so that the viewpoint is centered on the molecule with the given id.

- `get_center(molid)`:

- `set_center(molid, vector)`: Get/set the coordinates of the center of the given molecule as a Python list.

- `get_scale(molid)`:

- `set_scale(molid,scale)`: Get/set the scale factor used to display the given molecule.

- `get_rotation(molid)`:

- `set_rotation(molid, matrix)`: Get/set the rotation matrix for the given molecule as a 16-element Python list in row-major order.

- `get_trans(molid)`:

- `set_trans(molid, vector)`: Get/set the global translation applied to the given molecule as a Python list.

- `is_fixed(molid)`: Returns whether the molecule with the given id is fixed; that is, whether it is affected by translation, rotation, or scaling. Fixed molecules may still be animated (see `is_active` in the *animate* section).

- `fix(molid, trueorfalse)`: Make the molecule with the given id fixed or not.

- `is_shown(molid)`: Returns whether the molecule with the given id is shown or not.

- `show(molid, trueorfalse)`: Make the molecule with the given id shown or not.

9.5.13 vmdnumpy

This optional module is made available from within the toplevel VMD module if VMD detects a Numeric Python installation in the Python search path. When present, the following methods are provided:

- `timestep(molid, frame)`: Returns a single-precision Numeric array containing a **direct** reference to the given set of atom coordinates. Atom coordinates are arranged *xyzxyzxyz...* for each atom in the molecule. No copy of VMD's internal coordinates is made; therefore, modifications to this array will directly affect atom coordinates in VMD. Using the array after the timestep has been deleted will likely cause VMD to crash. The advantage is maximum efficiency and the ability to easily modify atom coordinates without going through the atom selection interface.

- `atomselect(molid, frame, selection)`: Returns an array of int's representing flags for on/off atoms in the given atom selection. The syntax for the selection is the same as for the AtomSel class. An array of this form can be used in conjunction with the Numeric `take` function to get selected coordinates from a timestep. Creating the array in this way can be 50-100 times faster than converting from an AtomSel object.

9.6 High-level Python Interface

VMD provides three modules for accessing and manipulating VMD state with objects that represent important entities. These objects can be thought of as references for the actual object within VMD: you can create as many references as you want and delete them, but modifying the reference changes the actual state of VMD. This is different from the AtomSel class, where each AtomSel instance is independent of the molecules and reps in VMD. These *proxy classes* are written in pure Python and use the lower level built-in interfaces to communicate with VMD.

9.6.1 Molecule

The Molecule class is a proxy for molecules loaded into VMD. Most operations raise ValueError if the proxy no longer refers to a valid molecule (i.e. if the molecule has been deleted).

Molecule instances provide the following methods:

- `__init__(id=None)`: Creating a new Molecule instance with no arguments will create a new empty molecule in VMD. Passing a valid molecule id will make the Molecule instance mirror the state of the corresponding molecule in VMD.
- `__int__()`: Casting a Molecule to an int returns the molecule ID.
- `rename(self, newname)`: Changes the name of the molecule.
- `name()`: Returns the name of the molecule.
- `delete()`: Deletes the molecule corresponding to this Molecule instance. The object can no longer be used.
- `load(filename, filetype=None, first=0, last=-1, step=1, waitfor=-1, volsets=[0])`: Load molecule data from the given file. The filetype will be guessed from the filename extension; this can be overrideen by setting the filetype option. `first`, `last`, and `step` control which coordinates frames to load, if any. `volsets` indicates which volumetric data sets to load from the file. Raises IOError if the file cannot be loaded.
- `save(filename, filetype=None, first=0, last=-1, step=1, waitfor=-1, sel=None)`: Save timesteps to the given file. The filetype will be guessed from the filename extension; this can be overridden by setting the filetype option. first, last, and step control which timesteps to save. Returns the number of frames written before the command completes. Pass an AtomSel instance as `sel` to write only a selection of atoms to the file. Note that this differs from the `AtomSel.write()` method in that `Molecule.save()` writes a range of timesteps, while `AtomSel.write()` writes only the coordinates corresponding to the selection's currently selected frame.
- `files()`:
- `types()`: Returns a list of filenames and file types, respectively, for the files that have been loaded into this molecule.
- `numAtoms()`: Returns the number of atoms in the molecule.
- `numFrames()`: Returns the number of coordinate frames in the molecule.
- `setFrame(frame)`: Set the coordinate frame to the given value. Must be in the range [0, numFrames())
- `curFrame()`: Returns the current coordinate frame for the molecule.
- `delFrame(first=0, last=-1, step=1)`: Deletes the given range of frames.
- `dupFrame(frame = None)`: Duplicate the given frame, apending it to the end. if `frame` is `None` then the current frame is used.
- `numReps()`: Returns the number of molecular representations (reps) in the given molecule.
- `reps()`: Returns a list of MoleculeRep objects, one for each rep in the molecule.

- `addRep(rep)`: Add the given MoleculeRep instance to the Molecule. Modifications to the rep will affect all Molecules to which the rep has been added. Raises ValueError if the rep has already been added to this molecule.
- `delRep(rep)`: Removes the given MoleculeRep from the Molecule. The rep is not affected and can be added to other molecules, but changes to it will no longer affect this Molecule.
- `clearReps()`: Removes all reps from this molecule.
- `autoUpdate(rep, onoff = None)`: If `onoff` is not `None`, sets the auto-update status for this rep and molecule (note that a rep's auto-update status may be different for different molecules). Returns the reps auto-update status.
- `ssRecalc()`: Recalculate the secondary structure for this molecule.

Examples:

```
>>> from VMD import *
>>> from Molecule import *
>>> bR=Molecule()
>>> bR.load('../proteins/brH.pdb')
    <snip>
<Molecule.Molecule instance at 0x406d878c>
>>> bR.name()
'molecule'
>>> bR.rename('bR')
<Molecule.Molecule instance at 0x406d878c>
>>> bR.name()
'bR'
>>> bR.numAtoms()
3762
>>> bR.dupFrame()
<Molecule.Molecule instance at 0x406d878c>
>>> bR.numFrames()
2
```

9.6.2 MoleculeRep

The MoleculeRep class, defined in the Molecule module, is designed to make it easy to keep track of the reps in a molecule and to update reps in many molecules simultaneously. The way it works is to create a MoleculeRep instance, then add it to as many molecules as you want using the Molecule.addRep() method. The only operations on MoleculeRep objects is to change their properties; when this occurs, all molecules to which the rep has been added will be updated. Deleting a MoleculeRep instance has no effect. A list of MoleculeRep instances for a given molecule can be gotten from the Molecule.reps() method.

The MoleculeRep class provides the following methods:

- `__init__(style=defStyle, color=defColor, selection=defSelection, material=defMaterial)`: Initialize the Rep object with optional style, color, selection and material properties. MoleculeRep objects also have attributes with the same names as the above keywords; these can be used to query the state of the rep. Don't set these attributes directly; use the `change*` methods below instead.

- `changeStyle(style)`:
- `changeColor(color)`:
- `changeSelection(selection)`:
- `changeMaterial(material)`: Set the draw style, color, atom selection and material for this rep. If the rep is assigned to any molecules, the molecule rep will be updated accordingly. `style` must be a valid draw style; see the `*Style` functions below.

In the following example, we load a molecule, add a new transparent VDW rep to the molecule, then change the atom selection for the rep to "name CA":

```
>>> from VMD import *
>>> from Molecule import *
>>> bR=Molecule()
>>> bR.load('../proteins/brH.pdb')
     <snip>
>>> reps=bR.reps()
>>> reps[0].style
'Lines'
>>> vdw=MoleculeRep(style='VDW', material='Transparent')
>>> bR.addRep(vdw)
>>> vdw.changeSelection('name CA')
```

9.6.3 Draw Style Methods

The syntax for changing the draw style in the MoleculeRep.changeStyle() method is fairly simple and easy to remember as long as the default values for each style are used; however, remembering that `rep.changeStyle("CPK 0.5 0.5 8")` is the way to set the bond radius, sphere scale, and sphere resolution for CPK is a little more difficult. The Molecule class defines a function for each draw style to make it easier to generate the required strings to pass to the changeStyle methods. Each function accepts keyword arguments for specifying the draw style parameters and returns a string suitable for `changeStyle()`.

9.6.4 Saving and Restoring Molecule State

Molecule and MoleculeRep instances can be saved used the `pickle` module from the Python standard library. Molecules will be saved with information about their name, files, and reps. The files themselves are not saved with the molecule; they will be reloaded when the molecule instance is recreated using `pickle.load()`. MoleculeRep instances can also be pickled; when restored they will be unassigned to any Molecules.

Chapter 10

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.

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`.

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++).

10.1 Vectors

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

```
Example:
vmd > veczero
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}
1 4 7
vmd > vecadd {0.1 0.2 0.4 0.8} {1 1 2 3} {3 1 4 1}
4.1 2.2 6.4 4.8
vmd > vecadd 4 5
9
```

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

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

- (C++) **vecsum** *v* – Returns the sum of the elements in v

```
Examples:
vmd > vecsum { 1 2 3 }
6.0
```

- (C++) **vecmean** *v* – Returns the mean of the elements in v

```
Examples:
vmd > vecmean { 1 2 3 }
2.0
```

- (C++) **vecstddev** *v* – Returns the standard deviation of the elements in v

```
Examples:
vmd > vecstddev { 1 2 3 4 5 6 7 8 9 10 }
2.87228131294
```

- (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}
0.2 0.4 0.6
vmd > vecscale {-5 4 -3 2} -2
10 -8 6 -4
vmd > vecscale -2 3
-6
```

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

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

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

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

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

```
Examples:
vmd> veclength 5
5.0
vmd > veclength {5 12}
13.0
vmd > veclength {3 4 12}
13.0
vmd > veclength {1 -2 3 -4}
5.47723
```

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

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

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

```
Examples:
vmd > vecnorm -10
-1.0
vmd > vecnorm {1 1 }
0.707109 0.707109
vmd > vecnorm {2 -3 1}
0.534522 -0.801783 0.267261
vmd > vecnorm {2 2 -2 2 -2 -2}
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
7.0
vmd > vecdist {0 0 0} {3 4 0}
5.0
vmd > vecdist {0 1 2 3 4 5 6} {-6 -5 -4 -3 -2 -1 0}
15.8745
```

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

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

10.2 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
{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}}
{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]
{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
{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
  {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
  {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
  {-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}
  {-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}
  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}
  {-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}
  13.0 -1.8e-05 5.8e-05
  vmd > transmult [transvec {6 -5 7}] [transvecinv {6 -5 7}]
  {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}
  {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}
  {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
{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
{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
{-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).

10.3 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}
-3.67321e-06 1.0 0.0
vmd > vectrans [transvecinv {-3 4 -12}] {-3 4 -12}
13.0 -1.8e-05 5.8e-05
```

10.4 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
3.14159265358979323846
vmd > expr 90 * ($M_PI / 180)
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"
180 deg north
vmd > find_rotation_value a
3.14159
vmd > set a
north
vmd > set a "1 pi to eat"
1 pi to eat
vmd > find_rotation_value a
3.14159
vmd > set a
to eat
vmd > set a 45
45
vmd > find_rotation_value a
0.785398
vmd > expr $M_PI * 3.0 / 2.0
4.71239
vmd > set a "4.71238 radians"
4.71238 radians
vmd > find_rotation_value a
4.71238
```

Chapter 11

Molecular Analysis

11.1 Using the molinfo command

This section covers how to extract information about molecules and atoms using the VMD text command `molinfo`.

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 8.3.6 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`.

11.2 Using the `atomselect` command

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', ...).

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, as described in section 8.3.2,

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}
```

Note that the format of the data you get back from the `get` command depends on how many attributes you requested. If you request only one attribute, as in the `get name` example above, you will get back a simple list of elements. On the other hand, if you request two or more attributes, you will get back a list of sublists. 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.

Your scripts will run faster if you retrieve only one attribute at a time, because then VMD does not have to construct the sublists for each attribute. Remember that in Tcl you can loop over several lists at once using the `foreach` command:

```
foreach resid [$sel get resid] resname [$sel get resname] {
  # process each resid and resname here
}
```

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 [§10], 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". Many 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. You can also change the resname, segid, and so forth, which may be easier to do within VMD than, for example, editing a PDB file by hand.

```
  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} {
  foreach coord [$sel get {x y z}] {
    lappend newcoords [vecadd $coord $offset]
  }
  $sel set {x y z} $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]
```

11.3 Analysis scripts

Following are some more examples of routines that could be used for analysing molecules. These are not the best routines to use since many of these can be implemented with the `measure` command, which calls a much faster built-in function.

Finding waters near a protein This example finds the waters near the protein for each frame of a trajectory and writes out a PDB file containing those waters:

```
set sel [atomselect top "water and same residue as (within 2 of protein)"]
set n [molinfo top get numframes]
for { set i 0 } { $i < $n } { incr i } {
  $sel frame $i
  $sel update
  $sel writepdb water_$i.pdb
}
```

The `frame` option sets the frame of the selection, `update` tells the atom selection to recompute which waters are near the protein, and `writepdb` writes the selected waters to a file.

Total mass of a selection

```
proc total_mass {selection} {
  set sum 0
  foreach mass [$selection get mass] {
    set sum [expr {$sum + $mass}]
  }
  return $sum
}
```

Note the curly braces after the `expr` command in the above example. Omitting those braces causes this script to run about three times slower! The moral of the story is: always put curly braces around the expression that you pass to `expr`.

Here's another (slightly 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"
}
```

Coordinate min and max 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 top all]
  set sx [$sel get x]
  set sy [$sel get y]
  set sz [$sel get z]
  set minx [lindex $sx 0]
  set miny [lindex $sy 0]
  set minz [lindex $sz 0]
  set maxx $minx
  set maxy $miny
  set maxz $minz
  foreach x $sx y $sy z $sz {
    if {$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]]
}
```

Radius of gyration 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`. Note that the `measure rgyr` command does the same thing as this script, only much much faster.

```
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 new alanin.pdb
vmd > set sel [atomselect top all]
vmd > gyr_radius $sel
Info) 5.45443
```

Root mean square deviation 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 new alanin.psf
vmd > mol addfile 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} {
  # 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]]
    lappend 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
vmd > mol new 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'.

11.4 RMS Fit and Alignment

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. VMD allows you to fit and align based on any valid atom selection, as long as the atom selection specifies the same number of atoms in each molecule being compared.

11.4.1 RMS Fit and Alignment Extension

To get started with RMS fitting and alignment, open the RMSD item from the **Extensions** menu. You should now have a new window titled RMSD Tool We'll describe the RMSD calculator function first.

RMSD calculation

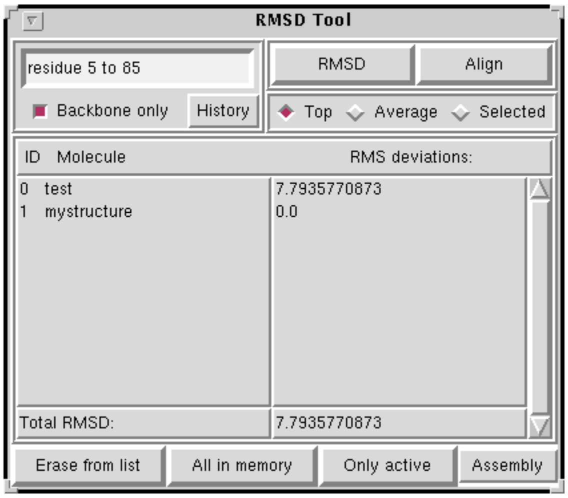

Figure 11.1: RMS calculation and alignment extension

The RMSD calculator button is used to calculate RMS distances between molecules. The upper left corner of the menu is where you specify which atoms are to be used in the calculation. In the input field, type the atom selection text just as you would in the Graphics form. The checkbox below the input field entitled *Backbone only* restricts whatever atom selection you typed to just the backbone atoms of the selection; in effect, it adds "and backbone" to the atom selection text.

The upper right corner of the menu has a button labeled **RMSD**. Its effect depends on which of the **Top**, **Average**, or **Selected** radio buttons are selected. If **Top** is selected, VMD calculates the RMS distance between the top molecule (which is usually the last molecule loaded) and every other molecule. If **Average** is selected, VMD first computes the average x, y, z coordinates of the selected atoms in each molecule, then computes the RMS distance of each molecule from that average structure.

Results of the RMS calculations for each molecule are shown in the browser in the bottom half of the menu. Note that this list is not updated until you presse the RMSD button, so the effects of loading/deleting molecules will not be immediately reflected. The **Total RMSD** label at the bottom of the menu shows the average RMSD for all molecules listed.

RMS Alignment

The RMS Alignment button fits molecules based on selected groups of atoms. Whereas the RMSD calculator button finds the RMS distance between molecules without disturbing their coordinates, the RMS Alignment button actually moves molecules to new positions.

This button is quite simple: Enter an atom selection in the input field, and press Align to align the molecules based on the atoms in that selection. If you recompute the RMSD between molecules with the RMSD calculator button, you will probably find that the values are different; this is because the calculation is made based on the current positions of the atoms.

11.4.2 RMS and scripting

The same actions can be taken on the scripting level. The Text interface also gives you more flexibility through the atom selection mechanism allowing to choose the atoms to fit/compare.

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.

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, say you want to align all of molecule 1 with molecule 9 using only the backbone atoms of residues 4 to 10 in both systems. 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
```

A simulation example script

Here's a longer script which you might find useful. The problem is to compute the RMSD between each frame 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
 [...]
```

11.5 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 8, 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` [§8.3.4] and `colorinfo` [§8.3.5] commands.

11.5.1 Changing the color scale definitions

Suppose that of the 1024 colors, the first 511 should be red, then 2 whites, and finally 511 blues. You can use the 'color' command to modify the color scale values accordingly.

```
proc tricolor_scale {} {
  set color_start [colorinfo num]
```

```
  display update off
  for {set i 0} {$i < 1024} {incr i} {
    if {$i == 0} {
      set r 1;  set g 0;  set b 0
    }
    if {$i == 511} {
      set r 1;  set g 1;  set b 1
    }
    if {$i == 513} {
      set r 0;  set g 0;  set b 1
    }
    color change rgb [expr $i + $color_start     ] $r $g $b
  }
  display update on
}

tricolor_scale
```

11.5.2 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.

11.5.3 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
}
```

11.5.4 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.

Chapter 12

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 (On Unix this is usually `/usr/local/lib/vmd`, on Windows this defaults to `C:\Program Files\University of Illinois\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. 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.

12.1 VMD 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 filename, and its extension is matched to the extension registered by the proper plugin. Thus, the Unix command

```
vmd molecule.pdb
```

will start VMD and load a molecule from the file `molecule.pdb`, while the command

```
vmd molecule.psf molecule.dcd
```

will load the corresponding structure and coordinate files into the same molecule. On the Windows platform, one must preface the VMD invocation with the Windows `start` command

```
start vmd molecule.pdb
```

- `-h | --help` : 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.
- `-python` : After initialization, switch to the Python interpreter before executing commands in the file specified by `-e` (if any), and leave the text interpreter in Python mode.
- `filename` : Load the specified file at startup. The file type will be determined from the filename extension; if there is no filename extension, and the filename contains 4 letters, it is assumed to be a PDB accession code and will be loaded accordingly; otherwise the format is assumed to be PDB.
- `-<filetype> filename` : Load the specified file using the given filetype.
- `-f` : Load all subsequent files into the same molecule. This is the default. A new molecule is created for each invocation of `-f`; thus, `vmd -f 1.pdb 2.pdb -f 3.pdb` loads `1.pdb and 2.pdb` into the same molecule and `3.pdb` into a different molecule.
- `-m` : Load all subsequent files into separate molecules. The `-f` and `-m` options may be specified multiple times on the command line in order to load multiple molecule containing one or more files.
- `-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 are 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.
- `-dist z` : Specify the distance to the VMD image plane.
- `-height y` : Specify the height of the VMD image plane.
- `-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.

- `-eofexit` : Make VMD exit when EOF on stdin is reached; for example, when a script is redirected to VMD. The combination `vmd -dispdev text -eofexit < input.tcl > output.log` is useful for batch mode scripting.

- `-startup filename` : Use filename as the VMD startup command script, instead of the default **.vmdrc** or **vmd.rc** file.

- `-args` : Pass subsequent command line arguments to the text interpreter. The Tcl interpreter will store these arguments in the list variable `argv`. By default, no arguments are stored in this variable.

- `-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.

12.2 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:

- `DISPLAY` : **(Unix-only)** The X-Windows display that VMD should use for displaying the VMD forms and menus, as well as the graphics window. If this environment variable is not overridden by VMDGDISPLAY all VMD windows will be directed to this display.

- `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 on Unix systems, and `C:\Program Files\University of Illinois\VMD` on Windows sytems.

- `VMDTMPDIR` : The directory which VMD should use for temporary data files. By default, this is /tmp, or /usr/tmp on Unix systems, and `C:\` on Windows.

- `VMDCUSTOMIZESTARTUP` : **(Unix-only)** The name of a C-shell script to source prior to running the actual VMD process. This shell script can contain any commands necessary for performing machine-specific spaceball, graphics, and other customizations necessary to run VMD. This can be anything from a simple script that sets the right serial port for a Spaceball based on the hostname, or it can be a complex script for turning on a projection system, logging demos, configuring multi-display stereo-framelock features, etc.

- `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.

- `VMDFILECHOOSER` : Specifies which file chooser to use for loading and saving files from the GUI. At present, this should be either FLTK, which uses Fltk's platform- independent file chooser, or TK, which uses Tk's file chooser. The Tk file chooser is the default and uses a native Windows interface on Windows platforms. The Fltk file chooser looks the same on all platforms, supports tab completion but not drive letters, and is probably most appropriate for Unix environments. The file chooser can be overridden at any time by changing the environment variable (e.g., in Tcl, `set env(VMDFILECHOOSER) FLTK`).

- `VMDCAVEMEM` : **(Unix-only)** This overrides the default size of the shared memory arena which is allocated by VMD when the CAVE starts up. The variable must be an integer number of megabytes. Since this is the only shared memory pool allocated, and it is done only once, you must choose a value sufficient to account for the largest scene you intend to render in VMD in that CAVE session. The default value unless otherwise specified is 80 Megabytes. Values of 200MB to 512MB are commonly needed for large molecular systems containing several hundred thousand atoms.
- `VMDFREEVRMEM` : **(Unix-only)** This overrides the default size of the shared memory arena which is allocated by VMD when the FreeVR starts up. The variable must be an integer number of megabytes. Since this is the only shared memory pool allocated, and it is done only once, you must choose a value sufficient to account for the largest scene you intend to render in VMD in that FreeVR session. The default value unless otherwise specified is 80 Megabytes. Values of 200MB to 512MB are commonly needed for large molecular systems containing several hundred thousand atoms.
- `VMDFORCECONSOLETTY` : **(Unix-only, intended only for clusters** This environment variable forces VMD to treat the text console as an interactive terminal, despite what the operating system says. This is only useful for running an interactive VMD session on a Clustermatic or Scyld Linux cluster node.
- `VMDGDISPLAY` : **(Unix-only)** The name of an X-Windows display that VMD will use to display the graphics window. This environment variable is only used on Unix systems. Through the use of the the DISPLAY and VMDGDISPLAY envrironment variables, the VMD graphics window can be placed on a separate screen from the forms and menus. This is particularly useful when giving 3-D demonstrations using a projector. The forms and menus can be kept on a different screen from the graphics so that they do not distract the audience.
- `VMDHTMLVIEWER` : The name of a command that will run a web browser in the background (Netscape, Mozilla, Firefox or whatever you prefer) that VMD should use to display HTML documents (such as this help file). By default, on UNIX, this is mozilla. (usage examples in Tcl: `set env(VMDHTMLVIEWER) ‘‘mozilla -remote openURL(%s)"`, `set env(VMDHTMLVIEWER) ‘‘mozilla %s &"`)
- `VMDIMAGEVIEWER` : The name of the external program to use for displaying VMD snapshots (or other images), in various formats.
- `VMDMSECDELAYHACK` : Add in a user-specified delay which causes VMD to sleep for specified number of milliseconds each time it renders the molecular scene on the display. This feature is meant as a workaround to poor performing display drivers which make the windowing system unresponsive if VMD is allowed to run unrestricted at maximum drawing rate.
- `VMDMSMSUSEFILE` : Force VMD to communicate with MSMS through the filesystem rather than with the socket-based network interface. This option can be used when the socket

interface isn't working properly for some reason. This is the default behavior when using VMD on Windows.

- VMDPREFERSTEREO : **(Unix-only)** This environment variable allows NVidia Quadro users to override the normal X11 visual search order, skipping multisample capable visuals in favor of stereo visuals. VMD still attempts to get the more complex visuals first, but if it comes down to a choice between stereo and multisample as mutually exclusive options, this variable provides the ability to force the use of stereo if available.
- VMDSCRDIST : Distance to the VMD image plane.
- VMDSCRHEIGHT : Height of the VMD image plane.
- VMDSCRPOS : Position of the VMD graphics window (x,y).
- VMDSCRSIZE : Size of the VMD graphics window (x,y).
- VMD_EXCL_GL_EXTENSIONS : Disable the use of named OpenGL extensions according to their official OpenGL extension names. This is intended to be used only when one encounters severe stability problems caused by buggy display drivers.
- VMDSIMPLEGRAPHICS : Forces VMD to use absolutely minimalistic graphics features with no use of OpenGL extensions. Essentially, nothing but bread-and-butter vertex arrays and immediate mode rendering will be used. This mode is intended to be used only when one encounters severe stability problems caused by buggy display drivers.
- VMDWIREGL : This environment variable disables several graphics features which are unsupported (or poorly supported) by WireGL and Chromium. This variable will be superceded with a more general implementation in a future release.

12.3 Startup Files

12.3.1 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. VMD read Tcl scripts at initialization, which are contained in VMD distribution. The locations of the scripts is determined by the `TCL_LIBRARY` environment variable, which is set in the vmd startup script to `$VMDDIR/scripts/tcl`. In addition, VMD has its own directory of core Tcl routines.

The most important of these is `$VMDDIR/scripts/vmd/vmdinit.tcl`. This file sets up the basic Tcl initialization commands, defines some environment variables, and adds 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 8.3.19 and section 12.4).

12.3.2 User Script Files

A user-written run-time command file, `.vmdrc` on Unix, `vmd.rc` on Windows, can be used with a list of initial VMD text commands to process. This file may be changed to customize individual user's initial screen appearance and to set the proper display characteristics for displaying in stereo. If it does not exist, default values are used.

12.3.3 `.vmdrc` and `vmd.rc` Files

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 on Unix; `./.vmdrc`, `$HOME/.vmdrc` and `$VMDDIR/.vmdrc`. On Windows, VMD searches in `./vmd.rc`, `$HOME/vmd.rc` and `$VMDDIR/vmd.rc`. Only the first file found will be read in and processed.

See chapter 8 for a description of the VMD text commands which may be put in this file. Also, section 4.1.3 discusses how to put commands into the `.vmdrc` file to customize the behavior of the 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 graphics move 5 455
menu files move 5 496

menu main on

# start the scene a-rockin'
rock y by 1
```

12.4 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.

12.4.1 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.

12.4.2 Setting up your `.mailcap`

The Unix versions of VMD have an extra `-webhelper` command line flag which causes VMD not to be spawned in the background, so that it has time to read temporary files downloaded by the web browser. This command line flag is just slightly simpler to use than the `chemical2vmd` script, as it does not depend on having Perl installed, so may be more appropriate for some cases.

In the VMD installation directory ($VMDDIR/scripts/vmd/) there is a perl script called `chemical2vmd` which will create a VMD command file and execute VMD. Since VMD does not block the calling process, Netscape and other web browsers cannot directly call VMD, as they do not know when to delete the temporary file containing the molecule or other data. The `chemical2vmd` script starts VMD with the -e command line option which runs the saved VMD script or molecule file.

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.

12.4.3 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`.

Index