



# Table of Contents

<b><u>Copyright and License</u></b> .....	<b>1</b>
<u>Copyright for the PyMOL Manual</u> .....	1
<u>PyMOL License</u> .....	1
<b><u>Introduction</u></b> .....	<b>2</b>
<u>Why PyMOL?</u> .....	2
<u>Strengths</u> .....	2
<u>Weaknesses</u> .....	3
<u>Future Outlook</u> .....	3
<b><u>Installation</u></b> .....	<b>4</b>
<u>Windows</u> .....	4
<u>Unix</u> .....	4
<b><u>Basics</u></b> .....	<b>6</b>
<u>Launching</u> .....	6
<u>By Mouse</u> .....	6
<u>By Command</u> .....	6
<u>PyMOL's Windows</u> .....	7
<u>Viewer Window</u> .....	7
<u>External GUI Windows</u> .....	7
<u>Loading PDB Files</u> .....	9
<u>Manipulating the View</u> .....	10
<u>Basic Mouse Control</u> .....	10
<u>Free Rotation</u> .....	10
<u>Moving Clipping Planes</u> .....	11
<u>Sessions and Scripts</u> .....	12
<u>Atom Selections</u> .....	13
<u>Origins</u> .....	13
<u>Hierarchical Atom Selections</u> .....	13
<u>Algebraic Atom Selections</u> .....	14
<u>Named Selections</u> .....	14
<u>Selection Algebra</u> .....	15
<u>Actions on Objects and Selections</u> .....	18
<u>Using the Internal GUI</u> .....	18
<b><u>Command-Line Shortcuts</u></b> .....	<b>22</b>
<u>Command Completion using TAB</u> .....	22
<u>Filename Completion using TAB</u> .....	22
<u>Automatic Inferences</u> .....	22
<b><u>Cartoons</u></b> .....	<b>24</b>
<u>Background</u> .....	24
<u>Accessibility</u> .....	24
<u>Beautiful and Realistic</u> .....	24
<u>Customization</u> .....	28
<u>Cartoon Types</u> .....	28

# Table of Contents

<a href="#">Fancy Helices</a> .....	30
<a href="#">Secondary Structure Assignment</a> .....	31
<b><a href="#">Ray-Tracing</a></b> .....	<b>32</b>
<a href="#">Important Settings</a> .....	32
<a href="#">Saving Images</a> .....	33
<a href="#">png</a> .....	33
<b><a href="#">Movies</a></b> .....	<b>34</b>
<a href="#">Concepts</a> .....	34
<a href="#">States and Frames</a> .....	34
<a href="#">Important Commands To Know</a> .....	34
<a href="#">load</a> .....	34
<a href="#">mset</a> .....	34
<a href="#">mdo</a> .....	35
<a href="#">mmatrix</a> .....	35
<a href="#">Simple Examples</a> .....	36
<a href="#">Complex Examples</a> .....	36
<a href="#">Previewing Ray-traced Movie Images</a> .....	36
<a href="#">cache_frames</a> .....	36
<a href="#">mclear</a> .....	37
<a href="#">Saving movies</a> .....	37
<a href="#">mpng</a> .....	37
<b><a href="#">Advanced Mouse Controls</a></b> .....	<b>38</b>
<a href="#">Picking Atoms and Bonds</a> .....	38
<a href="#">Example Usage of the "pk" Atom Selections</a> .....	38
<a href="#">The "lb" and "rb" Selections</a> .....	38
<a href="#">Conformational Editing</a> .....	39
<b><a href="#">Crystallography Applications</a></b> .....	<b>40</b>
<a href="#">Crystal Symmetry</a> .....	40
<a href="#">load</a> .....	40
<a href="#">symexp</a> .....	40
<a href="#">Electron Density Maps</a> .....	41
<a href="#">load</a> .....	41
<a href="#">isomesh and isodot</a> .....	41
<b><a href="#">Settings</a></b> .....	<b>42</b>
<a href="#">Under Renovation</a> .....	42
<a href="#">set</a> .....	42
<a href="#">Reference</a> .....	42
<b><a href="#">Compiled Graphics Objects (CGOs) and Molscript Ribbons</a></b> .....	<b>45</b>
<a href="#">Introduction</a> .....	45
<a href="#">Molscript Ribbons</a> .....	45
<a href="#">load</a> .....	45

# Table of Contents

<a href="#">Using Molscript</a> .....	45
<a href="#">Creating Compiled Graphics Objects</a> .....	46
<a href="#">CGO Reference</a> .....	47
<a href="#">load cgo</a> .....	47
<b><a href="#">Callback Objects and PyOpenGL</a>.....</b>	<b>49</b>
<a href="#">Introduction</a> .....	49
<a href="#">Example</a> .....	49
<a href="#">load callback</a> .....	50
<b><a href="#">Cookbook: Crystallographic Imagery</a>.....</b>	<b>51</b>
<a href="#">Electron density (density.pml)</a> .....	51
<a href="#">Crystal Packing with Ribbons (packing.pml)</a> .....	53
<a href="#">A Crystal Contact Interaction (contact.pml)</a> .....	55
<b><a href="#">Cookbook: Pushing the Limits</a>.....</b>	<b>57</b>
<a href="#">GroEL/ES (groel_es.pml)</a> .....	57
<a href="#">Nucleic Acid in the Ribosome (ribosome.pml)</a> .....	59
<a href="#">Crystal Packing With Surfaces (packsurf.pml)</a> .....	61
<b><a href="#">Reference</a>.....</b>	<b>63</b>
<a href="#">alias</a> .....	63
<a href="#">alter</a> .....	63
<a href="#">alter_state</a> .....	64
<a href="#">api</a> .....	64
<a href="#">at_sign</a> .....	65
<a href="#">attach</a> .....	65
<a href="#">backward</a> .....	65
<a href="#">bg_color</a> .....	66
<a href="#">bond</a> .....	66
<a href="#">button</a> .....	66
<a href="#">cartoon</a> .....	67
<a href="#">cd</a> .....	67
<a href="#">clip</a> .....	68
<a href="#">cls</a> .....	68
<a href="#">color</a> .....	68
<a href="#">commands</a> .....	69
<a href="#">copy</a> .....	69
<a href="#">count_atoms</a> .....	70
<a href="#">count_states</a> .....	70
<a href="#">create</a> .....	70
<a href="#">cycle_valence</a> .....	71
<a href="#">delete</a> .....	72
<a href="#">deprotect</a> .....	72
<a href="#">deselect</a> .....	72
<a href="#">disable</a> .....	73
<a href="#">distance</a> .....	73

# Table of Contents

<a href="#">do</a>	74
<a href="#">edit</a>	74
<a href="#">edit keys</a>	75
<a href="#">editing</a>	75
<a href="#">enable</a>	76
<a href="#">ending</a>	76
<a href="#">examples</a>	77
<a href="#">extend</a>	77
<a href="#">faster</a>	77
<a href="#">feedback</a>	78
<a href="#">find pairs</a>	79
<a href="#">finish object</a>	79
<a href="#">fit</a>	79
<a href="#">flag</a>	79
<a href="#">forward</a>	80
<a href="#">fragment</a>	81
<a href="#">frame</a>	81
<a href="#">full screen</a>	81
<a href="#">fuse</a>	82
<a href="#">get area</a>	82
<a href="#">get extent</a>	82
<a href="#">get frame</a>	83
<a href="#">get model</a>	83
<a href="#">get names</a>	83
<a href="#">get povray</a>	84
<a href="#">get state</a>	84
<a href="#">get type</a>	84
<a href="#">get view</a>	85
<a href="#">h_add</a>	85
<a href="#">h_fill</a>	86
<a href="#">help</a>	86
<a href="#">hide</a>	86
<a href="#">id atom</a>	87
<a href="#">identify</a>	87
<a href="#">index</a>	87
<a href="#">indicate</a>	88
<a href="#">intra_fit</a>	88
<a href="#">intra_rms</a>	89
<a href="#">intra_rms_cur</a>	89
<a href="#">invert</a>	89
<a href="#">isodot</a>	90
<a href="#">isomesh</a>	90
<a href="#">iterate</a>	91
<a href="#">iterate state</a>	92
<a href="#">keyboard</a>	92
<a href="#">label</a>	93
<a href="#">launching</a>	93

# Table of Contents

<a href="#">load</a> .....	94
<a href="#">load_brick</a> .....	94
<a href="#">load_callback</a> .....	95
<a href="#">load_cgo</a> .....	95
<a href="#">load_map</a> .....	95
<a href="#">load_model</a> .....	95
<a href="#">load_object</a> .....	95
<a href="#">ls</a> .....	96
<a href="#">map_set_border</a> .....	96
<a href="#">mappend</a> .....	97
<a href="#">mask</a> .....	97
<a href="#">math</a> .....	97
<a href="#">mclear</a> .....	98
<a href="#">mdo</a> .....	98
<a href="#">mem</a> .....	99
<a href="#">meter_reset</a> .....	99
<a href="#">middle</a> .....	99
<a href="#">mmatrix</a> .....	99
<a href="#">mouse</a> .....	100
<a href="#">move</a> .....	100
<a href="#">movies</a> .....	101
<a href="#">mplay</a> .....	101
<a href="#">mpng</a> .....	102
<a href="#">mset</a> .....	102
<a href="#">mstop</a> .....	103
<a href="#">operator</a> .....	103
<a href="#">orient</a> .....	103
<a href="#">origin</a> .....	104
<a href="#">pair_fit</a> .....	104
<a href="#">png</a> .....	104
<a href="#">povray</a> .....	105
<a href="#">protect</a> .....	106
<a href="#">push_undo</a> .....	106
<a href="#">pwd</a> .....	106
<a href="#">quit</a> .....	107
<a href="#">ray</a> .....	107
<a href="#">read_mmodstr</a> .....	108
<a href="#">read_molstr</a> .....	108
<a href="#">read_pdbstr</a> .....	108
<a href="#">rebuild</a> .....	109
<a href="#">recolor</a> .....	109
<a href="#">redo</a> .....	109
<a href="#">refresh</a> .....	110
<a href="#">release</a> .....	110
<a href="#">remove</a> .....	111
<a href="#">remove_picked</a> .....	111
<a href="#">rename</a> .....	112

# Table of Contents

<a href="#">replace</a>	112
<a href="#">reset</a>	113
<a href="#">rewind</a>	113
<a href="#">rms</a>	113
<a href="#">rms_cur</a>	114
<a href="#">rock</a>	114
<a href="#">run</a>	114
<a href="#">save</a>	115
<a href="#">select</a>	115
<a href="#">selections</a>	116
<a href="#">set</a>	116
<a href="#">set_color</a>	117
<a href="#">set_geometry</a>	117
<a href="#">set_key</a>	118
<a href="#">set_title</a>	118
<a href="#">set_view</a>	119
<a href="#">show</a>	119
<a href="#">sort</a>	120
<a href="#">spawn</a>	120
<a href="#">splash</a>	121
<a href="#">stereo</a>	121
<a href="#">symexp</a>	121
<a href="#">sync</a>	122
<a href="#">system</a>	122
<a href="#">time</a>	122
<a href="#">torsion</a>	123
<a href="#">transparency</a>	123
<a href="#">turn</a>	124
<a href="#">unbond</a>	124
<a href="#">undo</a>	125
<a href="#">unmask</a>	125
<a href="#">unpick</a>	125
<a href="#">update</a>	126
<a href="#">view</a>	126
<a href="#">viewport</a>	127
<a href="#">wizard</a>	127
<a href="#">zoom</a>	127

# Copyright and License

## Copyright for the PyMOL Manual

The PyMOL Manual is Copyright 1998-2001 by Warren L. DeLano of DeLano Scientific, San Carlos, CA, USA ([www.delanoscientific.com](http://www.delanoscientific.com)). All rights reserved.

The PyMOL Manual is part of the PyMOL Molecular Graphics System and may thus be distributed under the terms of the PyMOL license below.

## PyMOL License

PyMOL Copyright Notice  
=====

The PyMOL source code is copyrighted, but you can freely use and copy it as long as you don't change or remove any of the copyright notices.

-----  
PyMOL is Copyright 1998-2001 by Warren L. DeLano of DeLano Scientific, San Carlos, CA, USA ([www.delanoscientific.com](http://www.delanoscientific.com)).

All Rights Reserved

Permission to use, copy, modify, distribute, and distribute modified versions of this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation, and that the names of Warren L. DeLano or DeLano Scientific not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

WARREN LYFORD DELANO AND DELANO SCIENTIFIC DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL WARREN LYFORD DELANO OR DELANO SCIENTIFIC BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

-----  
Where indicated, portions of the PyMOL system are instead protected under the copyrights of the respective authors. However, all code in the PyMOL system is released as non-restrictive open-source software under the above license or an equivalent license.

--Warren L. DeLano ([warren@delanoscientific.com](mailto:warren@delanoscientific.com))

Copyright © 2001 [DeLano Scientific](http://www.delanoscientific.com). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Introduction

This manual is current as of PyMOL version 0.58 (September 2001), but is in rough form, filled with omissions, errors, obsolete information, and mis-spellings. Use at your own risk, and don't be surprised if the actual program differs from what you read here.

## Why PyMOL?

PyMOL is my answer to the frustration I have encountered as a computational scientist with most visualization and modeling software. Anyone who has studied the remarkable complexity of a macromolecular structure will likely agree that visualization is the most direct route to understanding in structural biology.

However, most researchers who use visualization packages ultimately run up against limitations inherent in the them which make it difficult or impossible to get exactly what you need. Such limitations in a closed-source commercial software package can not be easily surmounted, and the same is still true for free programs which aren't available in source form. **It is my belief that only unrestricted open-source software can provide the essential flexibility needed for research visualization.**

In my case, I needed a software package capable of doing several things very well: (1) visualizing multiple conformations of a single structure (trajectories), (2) interfacing with external programs, (3) providing professional strength graphics under both Windows and Unix, (4) preparing publication quality images, and (5) fitting into a zero dollar budget. That is why I wrote PyMOL, and it does all of these things very well.

Although PyMOL is by no means perfect, it now has a plethora of useful features and I find it to be an essential component in my research toolkit. I hope that others will find it to be as valuable in their own work as well.

## Strengths

- **Cross-Platform.** A single code base supports both Unix and Windows, using OpenGL and Python.
- **Atom Selections.** Arbitrary logical expressions facilitate focused visualization and editing.
- **Molecular Splits/Joins.** Structures can be sliced, diced, and reassembled on the fly and written out to standard files (i.e. PDB).
- **Movies.** Creating movies is as simple as loading multiple PDB files and hitting play.
- **Surfaces.** As good if not better than Grasp, and mesh surfaces are supported too.
- **Cartoon Ribbons.** PyMOL's cartoons are almost as good as Molscript but are much easier to access.
- **Scripting.** The best way to control PyMOL is through reusable scripts.
- **Rendering.** A built-in ray tracer gives you shadows and depth on any scene.
- **Output.** PNG files output from PyMOL can be directly imported into PowerPoint.
- **Conformational Editing.** Click and drag interface allows you to edit conformations naturally.
- **Expandability.** The PyMOL Python API provides a solid way to extend and interface.

## Weaknesses

- **User Interface.** Development has been focused on capabilities, not on easy-of-use for new users.
- **Documentation.** Only recently has any documentation become available.
- **Object–Orientation.** There is a single monolithic, functional API.
- **Electrostatics.** PyMOL is no replacement for Delphi/Grasp.
- **No Mechanics Engine** Although PyMOL sports potent molecular editing features, you can't yet perform any "clean-up".

## Future Outlook

PyMOL is now the principal visualization tool I use in my research, and I continue to add features to the program as I encounter a need for them. Though nothing is guaranteed, I plan to continue supporting PyMOL for the foreseeable future through enhancements, bug-fixes, and release coordination. Hopefully, a 1.0 version release will be attainable at the end of 2001, complete with documentation and examples. However, there is an obvious practical limit to what I will be able to accomplish given that my primary occupation is research, not software development.

Recognizing both the incredible potential of the program and my own time constraints for development, I have placed PyMOL under an **unrestricted open–source license**. Any scientist in the world can now adopt PyMOL as powerful free tool and then build upon it in ways which specifically advance their own research. Furthermore, developers (both commercial and academic) can elect to include or utilize PyMOL with their own programs at no cost. My hope is that PyMOL will develop enough of a user and developer base to become a self-sustaining package. Given that PyMOL has certain capabilities which far exceed those of any other free and open source molecular graphics system, I hope that you will find learning the system to be well worth the effort.

If you do decide to enhance PyMOL, please send your improvements back to me so that they can be integrated into the main version.

*This chapter last updated 8/26/2001 by Warren L. DeLano, Ph.D.*

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Installation

## Windows

### System Requirements

- Windows 95, 98, NT, 2000k, or ME. NOTE: stability problems have been observed under NT up through version 0.52.
- 3D OpenGL compatible graphics accelerator card (ideally, a GeForce2 GTS from nVidia or better).
- Python installed (version 1.5.2 or 2.1)
- At least 128 MB RAM (the more the better).

### Example Installation

1. Download the Python 1.5.2 installer "py152.exe".
2. Run the above and perform a full installation, including Tcl/Tk.
3. Download the latest PyMOL installer "pymol-x\_xx-bin-win32-py152.zip"
4. Extract the zip file into a directory.
5. Run "Setup.exe" from that directory to perform the installation.

You're done! You can now launch PyMOL from the "Start" menu.

Recently, a precompiled binary version which works with Python 2.1 has also become available. Be sure to obtain the proper binary for the version of Python you wish to use. For example:

```
pymol-0_58-bin-win32-py152.zip
```

All other things equal, I recommend using the Python 2.1 version.

## Unix

**NOTE:** The windows installation of PyMOL is much easier than the Unix installation, and so I recommend that people try the program out using Windows before proceeding with Unix in order to determine for themselves whether PyMOL is worth the trouble.

### System Requirements

- C Compiler
- Accelerated OpenGL graphics.

### Software Requirements

(NOTE: all of these can be found in the external source distribution "ext-src..." from the PyMOL download page.)

- Python, Python Megawidgets (Pmw), and Numeric Python

- GLUT 3.7
- libz and libPNG
- Tcl8.x and Tk8.x
- Optional: wxPython/wxWindows/wxGTK/gtk+/glib

## Installation

Because the installation process is often subject to change, please see the INSTALL file from the current distribution for detailed instructions. In summary,

1. Download, extract, configure, and compile the external dependencies.
2. Download and extract the current PyMOL source distribution.
3. Create a symbolic link from the external dependencies to "ext" in the PyMOL directory.
4. Configure PyMOL by copying and modifying a "Rules.make" specific to your system.
5. Run "make" to build pymol.
6. Create a pymol.com specific to your installation location.

You should be able to launch PyMOL by running pymol.com. I usually symbolic link this file into my "bin" directory as "pymol".

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Basics

## Launching

### By Mouse

**Microsoft Windows:** Click on the **Start** menu, follow it to **Programs**, and then release the mouse on **PyMOL**.

### By Command

#### Unix:

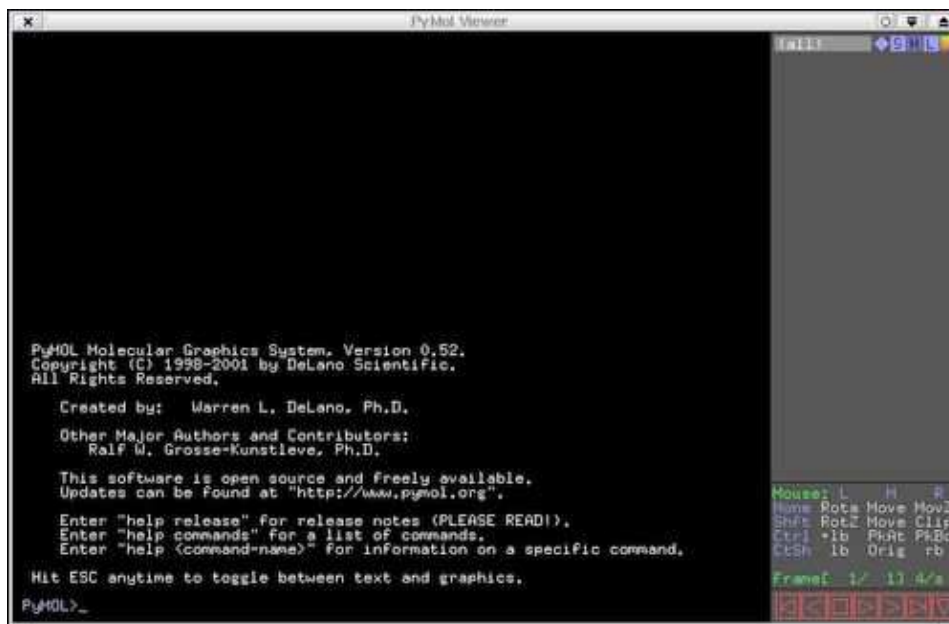
Edit **pymol.com** in the PyMOL distribution and make sure **PYMOL\_PATH** points to the actual location of the distribution. Enter **./pymol.com** to start pymol. You will probably want to create a link "**pymol**" from this file in to a "bin" directory in your path so that you can launch the program anywhere by simply entering **pymol**.

**Microsoft Windows:** Run "c:\program files\delano scientific\pymol\pymolwin.exe"

**NOTE:** Command line options can be included under both Windows and Unix to automatically open files and launch scripts. See "launching" in the reference section for more information on these options.

# PyMOL's Windows

## Viewer Window

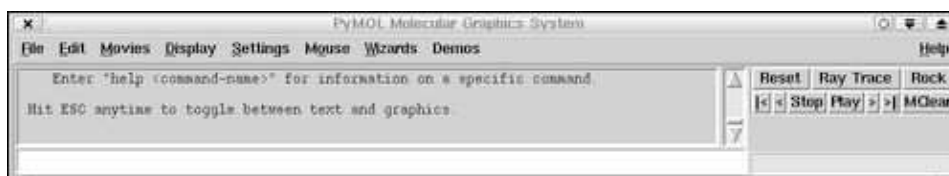


PyMOL Viewer window with Internal GUI enabled (Default).

The PyMOL Viewer represents the heart of the PyMOL system. This is a single OpenGL window where all 3D graphics are displayed and where all direct user interaction with 3D models takes place. This window also contains a simple graphical interface known as the "internal" GUI. The internal GUI has a primitive text display and command input capability as well as an object list, buttons, and a series of pop-up menus. When using the PyMOL Viewer, you can **hit ESC anytime to toggle between text and graphics mode**.

The PyMOL Viewer can be run all by itself, and it provides the complete capability of the PyMOL core system. However, many tasks can be made easier and more efficient through use of external menus and controls. Such objects are found in External GUI windows.

## External GUI Windows



The default external GUI included with PyMOL.

By default, PyMOL comes with a single external GUI window which provides a standard menu bar, an output region, a command input field, and a series of buttons. One important advantage of the

external GUI window is that **standard "cut and paste" functions for text will only work within external GUIs**, and not within in the PyMOL Viewer. Furthermore, **you must use Ctrl-X, Ctrl-C, and Ctrl-V to cut, copy, and paste** because a standard Edit menu has not yet been implemented.

**Notes For Developers:** External GUIs are the foundation for modularity and customizability in the PyMOL system. These windows constitute independent processes (or threads) which can control the behavior of PyMOL, and potentially interact with other programs. They are completely customizable at the Python scripting level, and mutiple external GUIs can exist at once (within the restrictions of Tkinter and wxPython).

External GUIs communicate with PyMOL through the Python API (Application Programming Interface). Those of you who want to link up you own programs with PyMOL should generally use a separate external GUI window to control the interaction, rather than changing internal PyMOL code. That way the programs will continue to work together even while development on each program proceeds independently. The internal GUI and all external GUI windows can be enabled and disabled using simple command line options (see reference for "launching").

# Loading PDB Files

## Using the External GUI:

The default external GUI provides the standard **File:Open...** menu option and dialog box which you can use to select the file you wish to open.

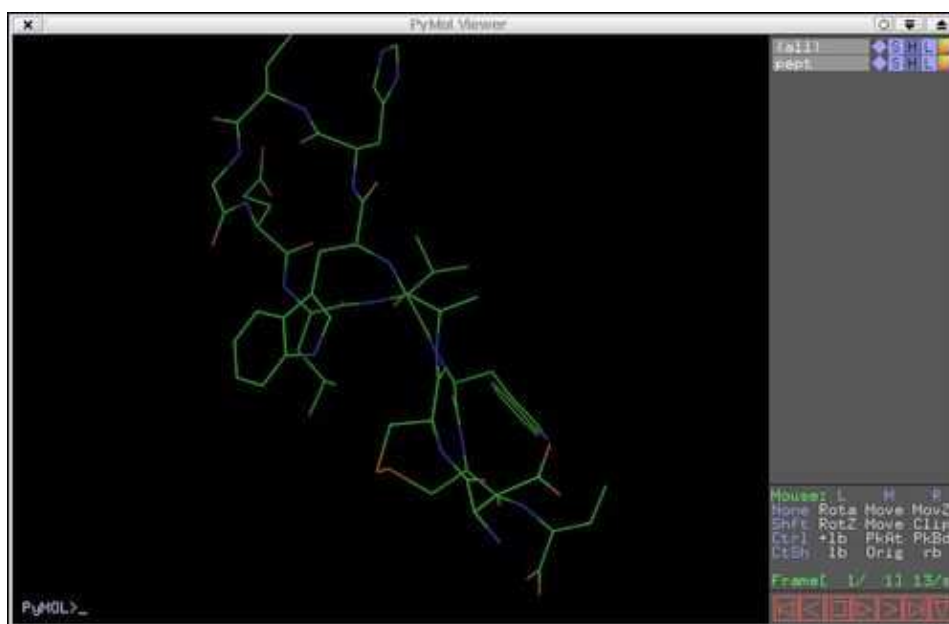
## Using Commands:

SYNTAX

```
load <filename>
```

EXAMPLE

```
load test/dat/pept.pdb
```



PyMOL after loading a PDB file.

By default, the name of the object will be the same as the prefix on the file. If you want a different name, then you must specify it on the command line using a comma after the filename:

SYNTAX

```
load <filename>, <object-name>
```

EXAMPLE

```
load test/dat/pept.pdb, test
```

## Manipulating the View

Mouse control is the primary control device, and keyboard modifier keys (SHIFT, CTRL, SHIFT+CTRL) are used in order to modulate mouse behavior. **A three button mouse is required for PyMOL**, but common mice such as the Microsoft Intellimouse and Microsoft Wheel Mouse will work just fine under Windows.

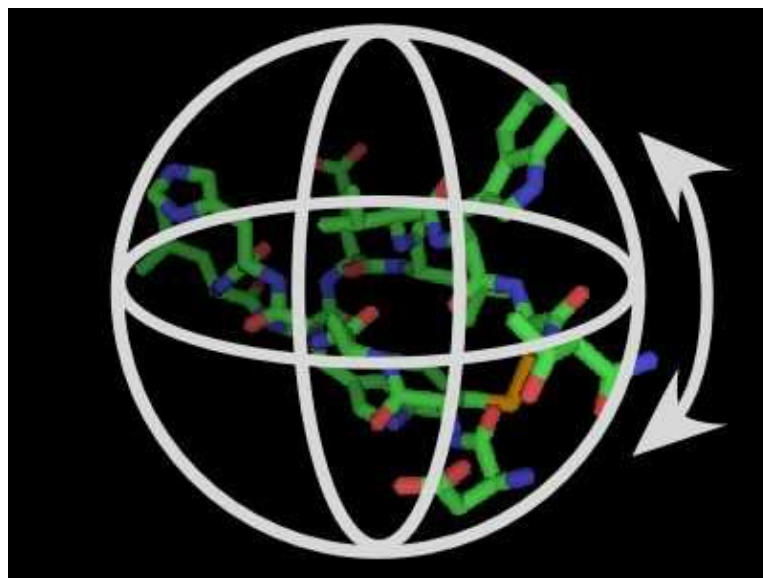
### Basic Mouse Control

Here is a table of the basic mouse button/keyboard combinations for view manipulation:

Keyboard Modifier	Left Button	Middle Button	Right Button
(none)	Rotate (Virtual Trackball)	Move in XY plane	Move in Z (Scale)
Shift Key	Rotate about Z-axis	Move in XY plane	Move Clipping Planes

When using PyMOL on a laptop, it may be necessary to attach an external mouse or reassign the particular mouse controls you plan to use onto the reduced set of buttons that you have available internally (see reference on the "button" command).

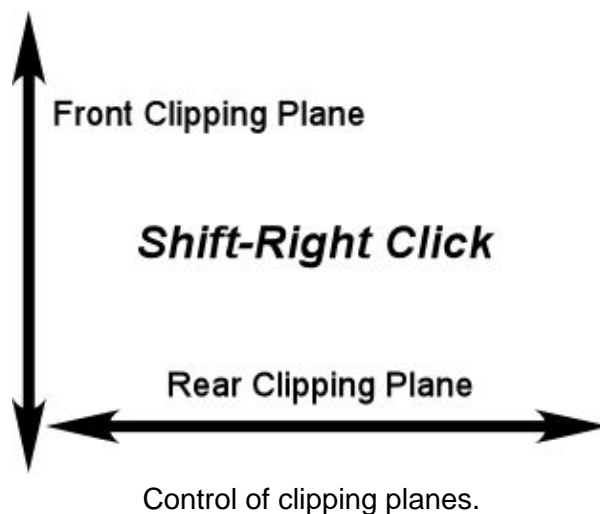
### Free Rotation



Virtual Sphere.

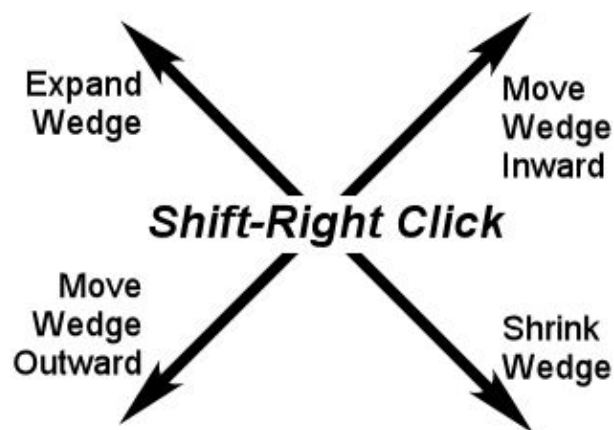
Free rotation works as if there is an invisible sphere in the center of the scene. When you click and drag on this sphere, it is as if you put your finger onto it and moved it in approximately the same manner. If you click outside the sphere, then you get rotation about the Z-axis alone. Generally, the view will be easiest to control by either clicking in the center of the scene and moving outwards (mostly XY-rotation), or by clicking and dragging around the edge of the screen and moving in a circular fashion (Z-rotation).

## Moving Clipping Planes



PyMOL's clipping plane control is somewhat unusual and may take a few minutes to get used to. Instead of having separate controls for the front and back clipping planes, controls are combined into a single mode where **up-down mouse motion moves the front** (near) clipping plane and **left-right mouse motion controls the back** (far) clipping plane.

The advantage of the PyMOL clipping plane control is that tedious tandem manipulations of the clipping planes now becomes quite trivial through the diagonal motions shown below.



You can also use the "clip" command to control the clipping planes.

## Sessions and Scripts

Unlike other common graphics packages, **PyMOL does not provide a means of saving and restoring sessions**. Thus, in order to use PyMOL effectively, you must get into the habit of creating command scripts which you can then call upon as needed.

While some would consider lack of save/restore functionality to be a big drawback, it was not incorporated into PyMOL's basic design because it would have significantly complicated the program code and because **most serious computational scientists prefer to use scripts**. Scripts allow you to apply the same basic solution to a given problem over and over again in new situations. Even though there is more work involved in creating scripts, the future reuse makes up for it in the long run.

Currently, scripts need to be created separately using a text editor and then sourced (using "@" from within PyMOL. It makes sense to have a program like **emacs**, **jot**, or **notepad** available in a separate window while using PyMOL. Commands can then be cut and paste between the two programs. Development of an automatic script-recording capability is planned which would achieve something similar to save/restore and make script writing much more efficient.

**PyMOL expects commands scripts to have a ".pml" extension** (although this is not strictly required in all cases, it is good practice). Once created, scripts can be executed in several ways. Under Windows, scripts can be run in a new PyMOL session by double clicking on the script's icon. Alternatively, you can run a script using the **File** menu's **Run** option. You can also:  
(1) use the "@" symbol from within PyMOL

SYNTAX

```
@<script-file-name>
```

EXAMPLE

```
PyMOL> @my_script.pml
```

or (2) include it on the command-line when launching the program:

SYNTAX

```
pymol <script file>
```

EXAMPLE (Windows)

```
C:\> pymol.bat my_script.pml
```

EXAMPLE (Unix)

```
unix> ./pymol.com my_script.pml
```

NOTE: PyMOL is also fully scriptable using Python, but that is an advanced topic.

# Atom Selections

## Origins

Atom selections are central to how PyMOL works and are functionally interchangeable with objects in most cases. For example, you can use the **color** command to change an object's color, and you can use that same command to change the color of a selection of atoms.

### EXAMPLES

```
color blue,pept                # colors pept blue

color red,(name ca)            # color alpha-carbons red
```

**To use PyMOL beyond the most basic level, you must learn at least one of PyMOL's atom selection languages.**

## Hierarchical Atom Selections

PyMOL provides an ultra-concise hierarchical atom selection syntax which utilizes slashes to denote the hierarchy. The hierarchy is as follows:

```
model/segment/chain/residue/atom
```

Note that **all hierarchical selections must contain at least one slash (/) character** and **hierarchical selections which contain a comma (,) must have surrounding parentheses** in order to indicate that they are not separate arguments to a command.

Hierarchical selections are interpreted left-to-right if they have a preceding slash, and right-to-left if they do not.

### EXAMPLE FORMS

```
                                residue/atom
                                chain/residue/atom
                                chain/residue/
                                /segment/chain//
/model/segment/chain
/model//chain
/model//chain/residue
model///residue/
model////atom

... etc.
```

### EXAMPLE USAGE

```
select bb,*/ca

color red, e/4:8/                # color residues 4 through 8 red in chain E

zoom ( 42/c,n,o )                # zoom in on atom C,N,O in residue 42

zoom ( /prot/e )                 # zoom in on segment E of model "prot"
```

## Algebraic Atom Selections

The design of PyMOL's algebraic atom selection language is inspired by Axel T. Brunger's CNS and X-PLOR programs, which share a very powerful but unreasonably verbose languages for selections. PyMOL's selection language reproduces much of the XPLOR selection language but extends it by including many short-cuts which substantially reduce the amount of typing required and permit more concise statements.

### Algebraic atom selections are always surrounded by parentheses "(...)"

#### EXAMPLE SELECTIONS

```
(name ca or name c or name n or name o or name h) # X-PLOR or PyMOL
(name ca,c,n,o,h) # PyMOL
(n;ca,c,n,o,h) # Equivalent PyMOL
```

### Object names are valid symbols within atom selections

```
load 1fc2.pdb, fc_comp
color yellow,(fc_comp and elem C) # color carbons yellow in fc_comp object
```

### Hierarchical atom selections can be included in algebraic selections. For example:

```
color green,( 10:100/ca and his/ ) # these are all equivalent...
color green,( 10:100/ca and resn his )
color green,( resi 10:100 and name ca and his/ )
color green,( resi 10:100 and name ca and resn his )
```

## Named Selections

PyMOL goes beyond the X-PLOR selection syntax by introducing the concept of "named" selections. These are atom selections whose names, once defined, appear in the Internal GUI names list and can be treated like independent objects in many cases. Named selections can span multiple objects and will survive any changes made to the molecular structure. They can also be referred to by name in subsequent atom selections.

#### EXAMPLES

```
load fc.pdb
load pept.pdb

select backbone = (name ca,c,n,o,h) # Create named selection "backbone"
# which covers atoms in both objects.

color red, backbone # Colors the selection red.
```

Named selections are distinguished from objects in the Internal GUI object list by a surrounding set of parentheses, and the options available available differ slightly between objects and selections.

**NOTE:** Named selections are static. Only atoms which exist at the time the selection is defined are considered part of a selection, even if atoms which are loaded subsequently would fall within the given criterion (NOTE: A mechanism for "refreshing" named selections is planned but does not yet exist).

## Selection Algebra

Selections can be arbitrarily complex and contain complicated algebraic constructs.

Note that in the following tables, **semicolons ";" in abbreviations are significant and are required (where shown)** in order to delimit the abbreviation.

### Operators:

Selection operators and modifiers are listed below in decreasing order to precedence.

Operator	Abbreviation	Priority	Meaning
not ...	! ...	6	Negation
... and ...	... & ...	4	Logical AND
... or ...	...   ...	3	Logical OR
... around X	... a; X	2	Select atoms around a selection within X Angstroms.
... expand X	... e; X	2	Expand selection by all atoms within X Angstroms.
byres ...	b;...	1	Expand selection to complete residues

The order of precedence rules for the above operators mean that the following atoms selections are equivalent:

```
(byres chain a or chain b and not resi 125 around 5)
```

```
(byres ((chain a or (chain b and (not resi 125))) around 5))
```

Some additional operations are shown below, but you can probably ignore them for now.

Operator	Abbreviation	Priority	Meaning
... gap X	N/A	6	Select all atoms whose van der Waals radii are separated from the van der Waals radii of the selection by a minimum of X Angstroms.
... in ...	N/A	3	Select atoms in the first selection with matched identifiers for atoms in the second.
... like ...	l;	3	Select atoms in the first selection with matched residue numbers and names for atoms in the second.
neighbor ...	N/A	1	Select atoms bonded to the selection.

### Simple Keyword Selectors:

Selector	Abbreviation
all	*
none	N/A
hydro	h;
visible	v;
hetatm	het

#### EXAMPLE

```
hide (hydro) # hide all hydrogen atoms
```

### Property Selectors:

Where plural arguments are indicated for property selectors, multiple values can be provided so long as they are separated by only commas (and no spaces). In each case, the implied operation is a logical OR with implicit priority.

#### EXAMPLE

```
(name cb, cg1, cg2 and chain A) # is equivalent to
((name cb or name cg1 or name cg2) and chain A)
```

Ranges are separated by a colon ":", and currently only one range can be given to those selectors which accept ranges.

```
(resi 100:120)
```

Currently, the only valid comparison operators are: "<", ">" , and "=". More will be added.

```
(b > 10)
```

The basic property selectors are:

<b>Selector</b>	<b>Abbreviation</b>
elem <element-names>	e;<element-names>
name <atom-names>	n;<atom-names>
resn <residue-names>	r;<residue-names>
resi <residue-identifiers> resi <residue-range>	i;<residue-identifiers> i;<residue-range>
alt <alternate-conformation-identifier>	N/A
chain <chain-names>	c;<chain-names>
segi <segment-names>	s;<segment-names>
b <comparison-operator> <value>	N/A
q <comparison-operator> <value>	N/A
formal_charge <comparison-operator> <value>	fc; <comparison-operator> <value>
partial_charge <comparison-operator> <value>	pc; <comparison-operator> <value>

Shown below are some advanced property selectors which can probably be ignored for now:

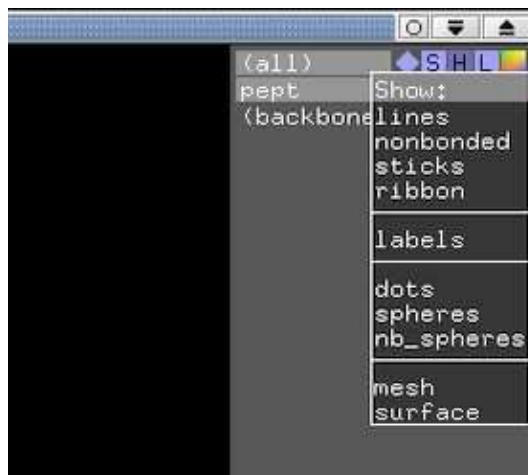
<b>Selector</b>	<b>Abbreviation</b>
flag <flag-number>	f;<flag-number>
numeric_type <type-number>	nt;<type-number>
text_type <type-string>	tt;<type-string>
id <external-index-number>	N/A
index <internal-index-number>	N/A

# Actions on Objects and Selections

## Using the Internal GUI:

The internal GUI allows you to perform a number of actions directly on objects or selections through a set of pop-up menus which appear to the right of every object or selection name. The four menus are:

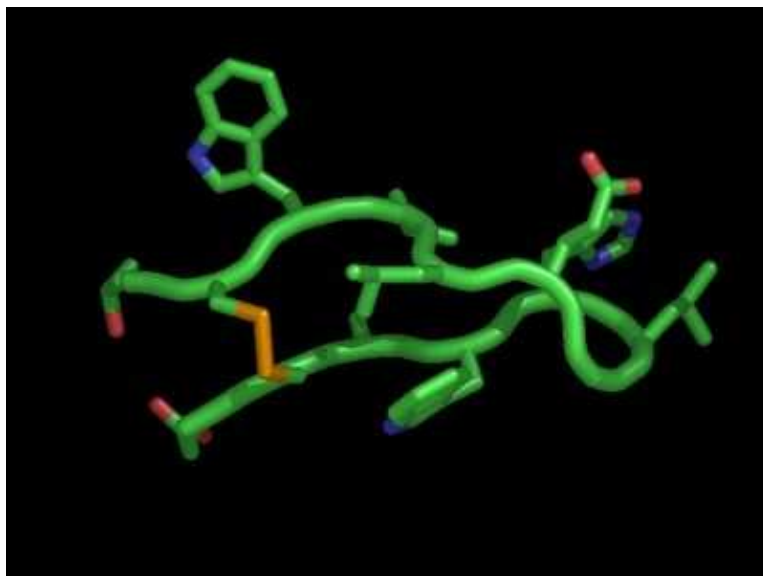
- **Action (Diamond):** Common actions you might perform, such as zooming or deleting.
- **Show [S]:** Show various representations, such as spheres or surfaces.
- **Hide [H]:** Hide various representations.
- **Label [L]:** Label atoms according to various properties.
- **Color (Rainbow):** Color atoms, by atom type or uniformly.



Example pop-up "Show" menu.

You can experiment with most of the actions in the pop-up menus in the internal GUI to learn how they work. However, certain actions, such as showing surfaces and meshes, can be time consuming to compute on large structures. Thus, you may want to experiment using the PDB file of a small peptide.

Named atom selections work well with the internal GUI because it is often easiest to identify a specific set of atoms using the selection language and then use the internal GUI menus to change the representation and appearance of those atoms. For example, the following figure can be prepared using the script below or by just creating a named atom selection on the command-line and performing the rest of the actions using the GUI.



Generating figures using the GUI with named atom selections.

```
load test/dat/pept.pdb # or load this file using the external GUI's File menu.
orient pept           # or select "orient" from pept's Action menu.
select sc = (!n;c,n,o) # this selection includes all sidechain bonds.
show sticks, sc       # or select "sticks" from sc's Show menu.
hide lines, pept      # or select "lines" from pept's Hide menu.
show ribbon, pept     # or select "ribbon" from pept's Show menu.
ray                   # or click "ray trace" on the external GUI.
```

## Using Commands on Objects and Selections

All actions performed by the internal GUI have command-line equivalents.

In general, PyMOL commands which take an object as an argument can also take a named selection or even an inline selection. For example, all three of the color commands below will have the same result:

```
load test/dat/pept.pdb # create object "pept"
select vals = (pept and r;val) # creates named selection "vals"

color yellow,(pept and r;vals) # color inline selection which uses an object name
color yellow,vals              # color named selection
color yellow,(vals)           # color inline selection which uses a named selection
```

However, there are some exceptions, such as with object and selection deletion, where it is necessary to specify the object or selection name explicitly without surrounding parenthesis.

```
load test/dat/pept.pdb # creates object "pept"
select vals = (pept and r;val) # creates named selection "vals"
```

```

delete (vals)      # invalid: implies deletion of an inline selection.
delete (pept)     # invalid: same problem; deleting "(vals)" not "vals"

delete vals       # valid: destroys the selection, but the object is unchanged.
delete pept       # valid: deletes the object and all associated atoms.

```

## Essential Commands

Below is a list of the most frequently used commands. Note that when a object or selection is not specified, the default selections of (all) is used.

Command Syntax	Example
show <representation>,<object-or-selection>	show mesh,(resi 11)
hide <representation>,<object-or-selection>	hide mesh,(n; ca,c,n,o)
zoom <object-or-selection>	zoom (pept or fc)
color <color-name>,<object-or-selection>	color red, pept
origin <selection-or-name>	origin (resi 8)
delete <selection-or-name>	delete pept
remove <selection-or-name>	remove (hydro)
cartoon <cartoon-type>,<selection-or-name>	cartoon tube,(i;10:30)

The meaning of show, hide, zoom, color are probably obvious. Delete is used to remove entire objects. In contrast, remove is used to remove only a subset of atoms within objects. The origin command redefines the center of rotation in space about the indicated object or selection without changing the current view.

For more information on the individual commands, use "help command-name" or consult the reference section of the manual.

### EXAMPLES

```

# How do I zoom in on residue 100?

    zoom (resi 100)

# How do I color the molecule white?

    color white

# How do I color residues 50, 54, and 58 green?

    color green, (resi 50,54,58)

# How do I color residues 60 to 90 yellow?

    color yellow, (resi 60:90)

# How do I show a surface on model "lig"?

```

```
show surface, lig

# How do I show a cartoon representation on model "prot"?

show cartoon, prot

# How do I show a helix-like representation on residues 14:30?

cartoon oval, (resi 14:30)

# How do I show a CPK-style space filling representation?

show spheres

# How do I hide hydrogens?

hide (hydro)

# How do I get rid of hydrogens permanently?

remove (hydro)

# How do I delete an object named "prot"?

delete prot
```

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Command-Line Shortcuts

Since almost nobody likes to type, PyMOL's internal command-line interface includes several "shortcut" features designed to ease the process and reduce typing. If you are a unix user, you will recognize the similarity with features found in tcsh or bash.

## Command Completion using TAB

If you type the first few characters of a command and then hit TAB, PyMOL will either complete the command or print out a list of which commands match the command.

```
sel      # hitting TAB will produce
select
```

If you hit the TAB key with a blank command line, PyMOL will output a list of all known commands.

## Filename Completion using TAB

When using the "load" command, it can be necessary to type some long paths and filenames. PyMOL makes this process easier by automatically completing unambiguous paths and filenames when you hit the TAB key. For instance,

```
load cry          # hitting TAB will generate
load crystal.pdb # if "crystal.pdb" exists in the
load crystal.pdb # current working directory
```

If there is some ambiguity in the filename, PyMOL will complete the name up to the point of ambiguity and then print out the matching files in the directory.

## Automatic Inferences

There are a number of fixed string arguments to PyMOL commands. For example, in

```
show sticks
```

"sticks" is a fixed string argument to show. Since there is only a small set of such arguments you can pass to show, PyMOL will infer your meaning even if you only provide it with a few letters. For example

```
show st
```

works just as well. Note that commands are also inferred in this manner, so

```
sh st
```

will currently work just as well, since "show" is the only command starting with "sh".

NOTE: because PyMOL's command language continues to grow and develop, it is important to use full-length commands and string arguments when writing scripts. Otherwise, you can not be sure

that a later command or argument will not cause your abbreviation to become ambiguous. For example, "sh st" would be no longer work if a "shutoff" command were added to the PyMOL language.

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

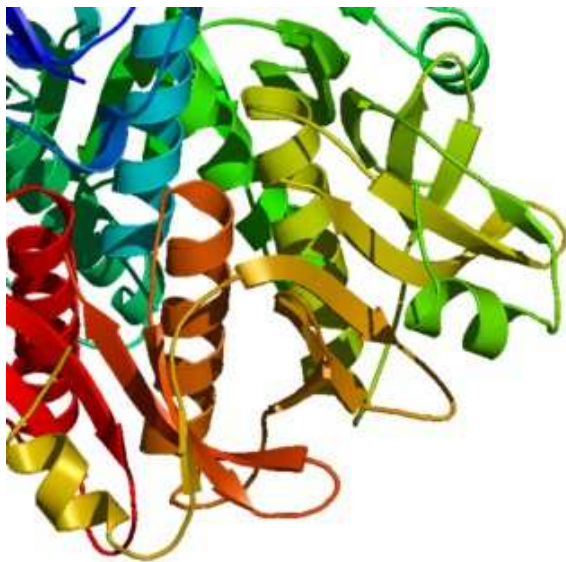
Hosted at: \_\_\_\_\_

# Cartoons

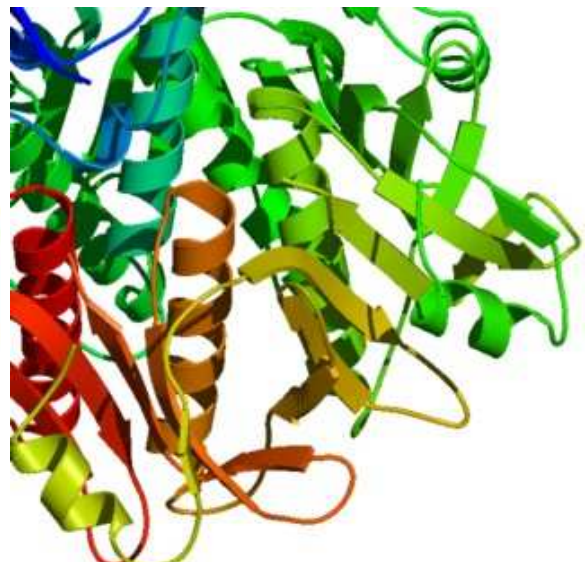
## Background

### Accessibility

Cartoon ribbons in PyMOL rival those of the popular Molscript/Raster3D packages, but PyMOL makes creating high quality images much easier. While PyMOL can read Molscript output directly (see the chapter on Molscript), this is no longer necessary or as convenient as utilizing PyMOL's new built-in cartoon ribbon capability:



PyMOL built-in ribbons



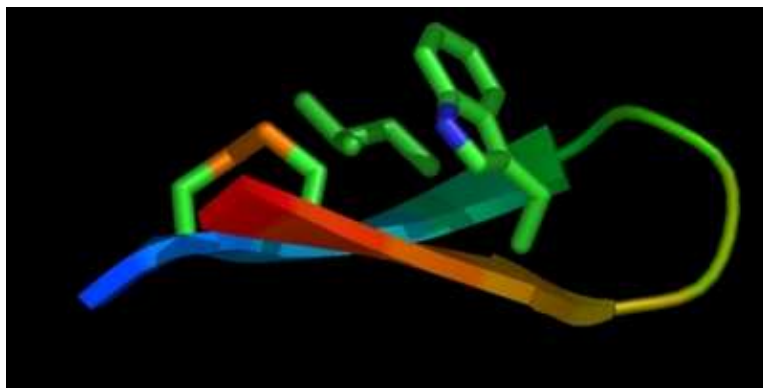
"molauto -nice ... | molscript -r > ..."

Can you tell the difference? Molscript's cartoons are slightly more geometrically perfect, but PyMOL comes pretty darn close!

Note that all of the images in this section were colored using the rainbow feature (color pop-up menu) and ray-traced with antialiasing enabled.

### Beautiful *and* Realistic

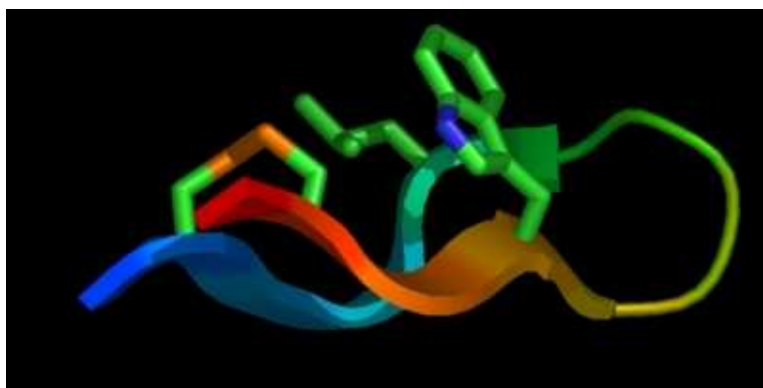
One of the advantages of PyMOL's cartoon ribbon facility is that it is easy to switch between "smoothed" versions of protein secondary structure, and "realistic" renditions which portray actual main chain coordinates. Although cartoons are often used solely to represent protein structures in a schematic sense, sometimes it is desirable to combine a schematic overall picture with atomic resolution in one particular location. However, unless the cartoon track properly with alpha-carbon positions, the resulting figures will look a little silly:



In the above image, the side chains are floating off into space. Disabling "flat sheets" from the Cartoons Menu or issuing the command

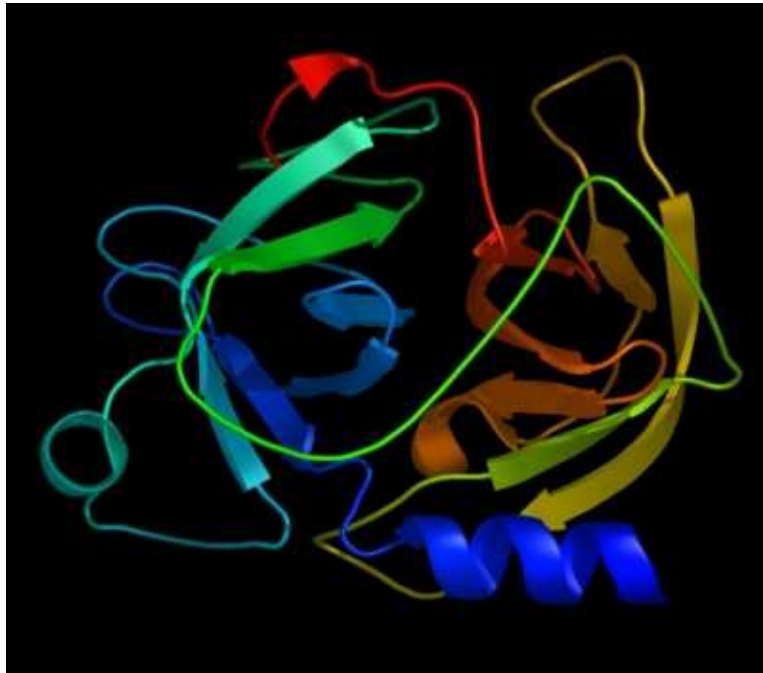
```
set cartoon_flat_sheets = 0
```

will make the beta strands follow the true path of the backbone through space and give a more accurate rendition of the structure.



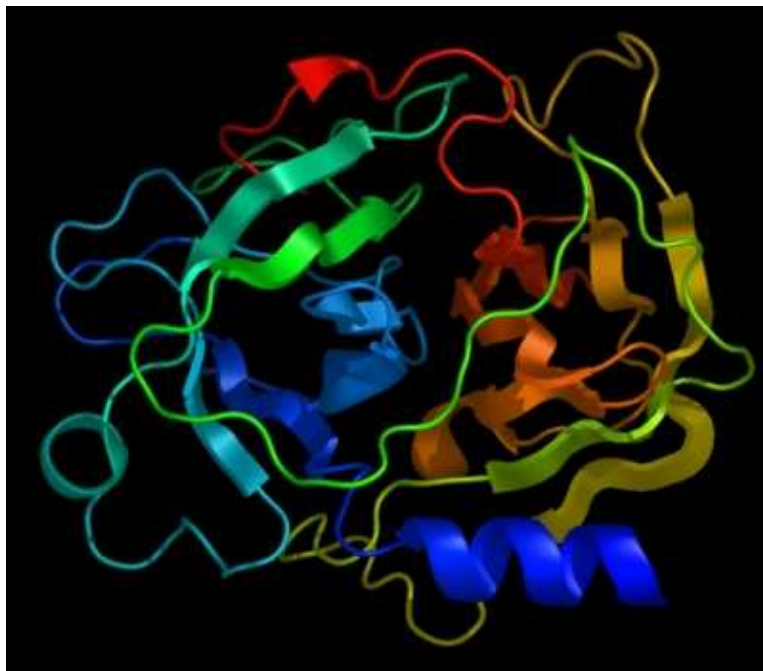
The appearance of a cartoon over the entire molecule will be substantially different when all smoothing features are turned off. For instance, with smoothing enabled:

```
set cartoon_flat_sheets = 1  
set cartoon_smooth_loops = 1
```

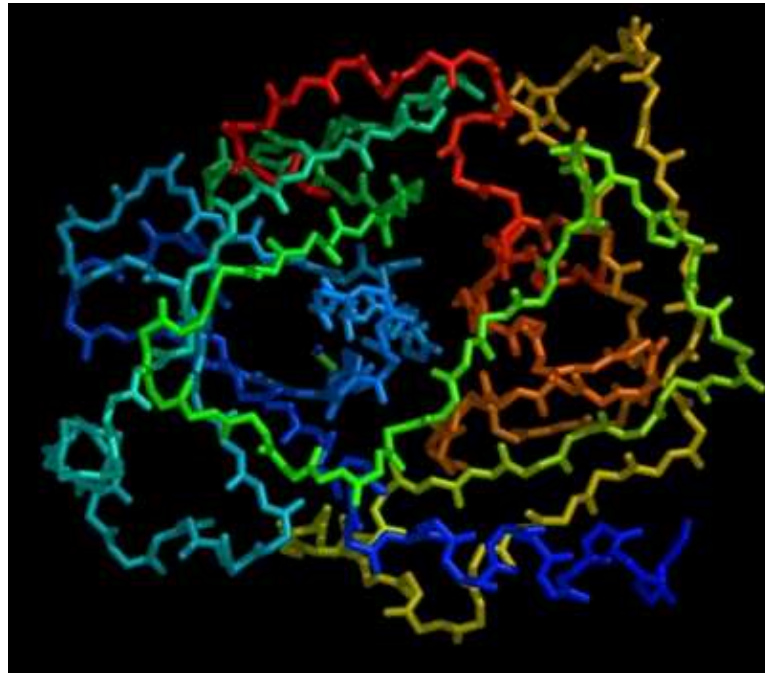


the image differs substantially from:

```
set cartoon_flat_sheets = 0  
set cartoon_smooth_loops = 0
```



which more accurately reflects the true path of the peptide backbone:



To facilitate beautiful imagery, smoothing is enabled by default (just like Molscript). Just be sure to turn it off when you want to study structures at atomic resolution (*always remember, real life is much more complicated than what you seen in cartoons*).

# Customization

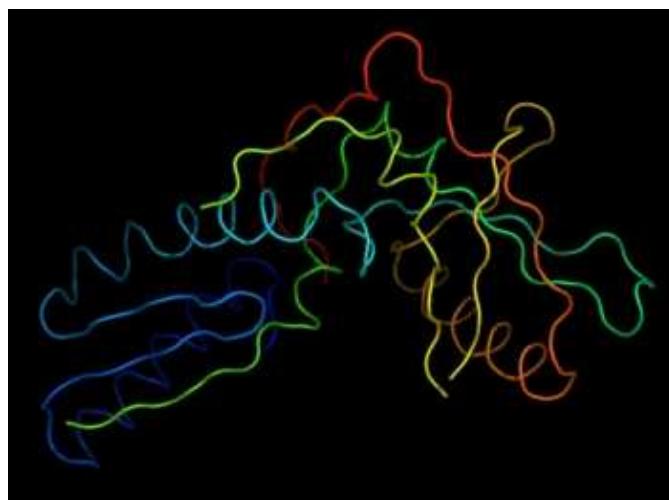
## Cartoon Types

When secondary structure information is present, multiple cartoon types can be automatically combined to compose a conventional figure. You can also ignore such information, and manually control which cartoons are used when and where.

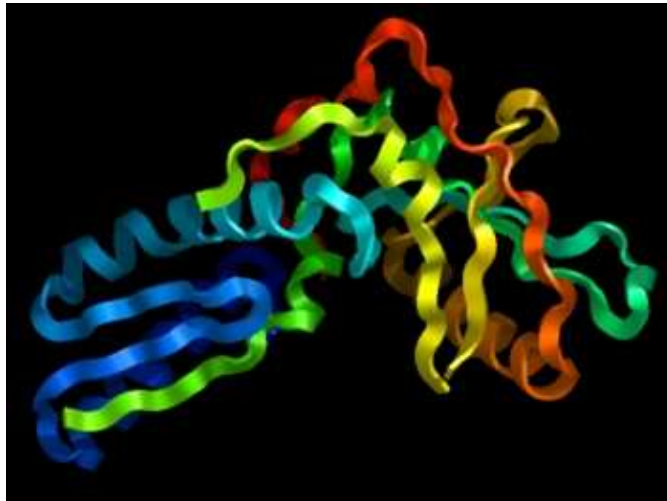
```
show cartoon
cartoon automatic # default
```



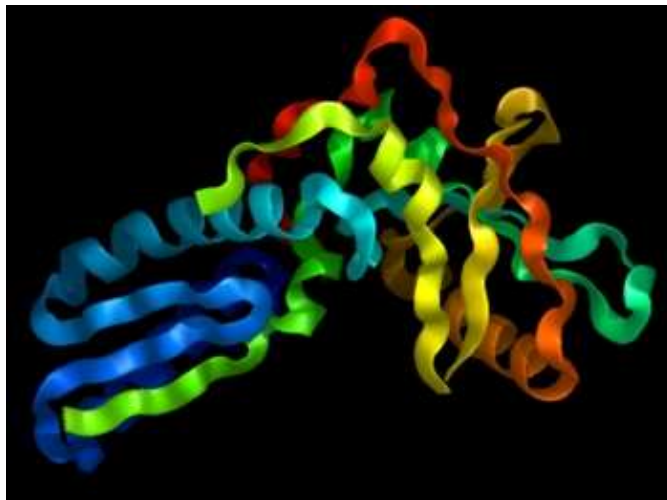
```
cartoon loop
```



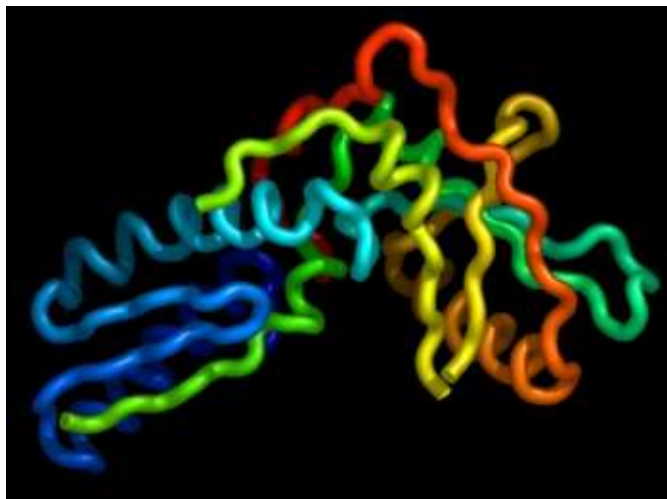
```
cartoon rect
```



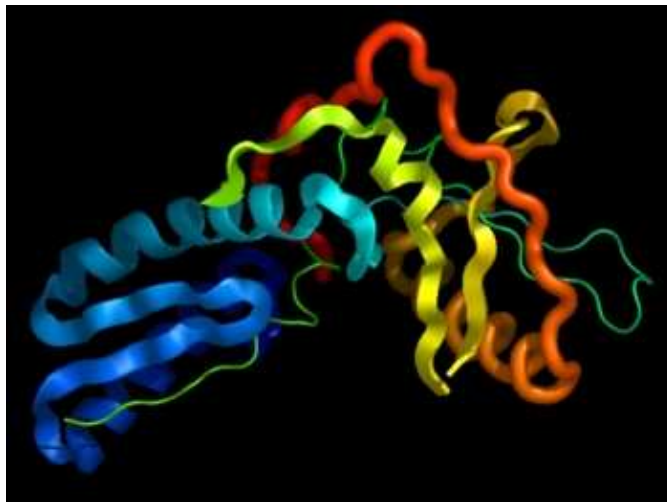
cartoon oval



cartoon tube



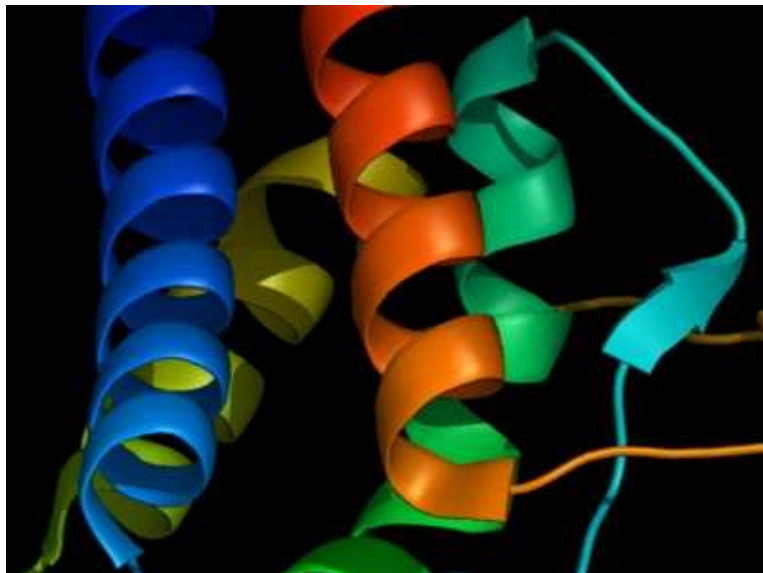
```
cartoon tube, 1:49/  
cartoon arrow, 50:99/  
cartoon loop, 100:149/  
cartoon oval, 150:199/  
cartoon rect, 200:250/
```



All cartoon ribbons have associated parameters accessible from the "set" command which allow you to change their appearance. See the chapter on Settings for more information.

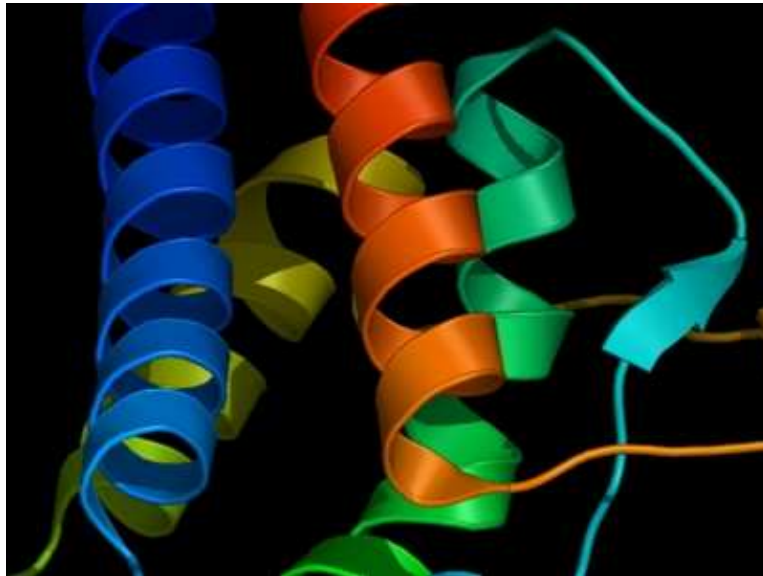
## Fancy Helices

```
set cartoon_fancy_helices=0
```



Molscript addicts who simply must have those ribbon helices with tubular edges will not be disappointed with "fancy helices":

```
set cartoon_fancy_helices=1
```



Not bad for free software, eh?

## Secondary Structure Assignment

It is strongly recommended that you read in PDB files which already have correct secondary structure assignments from a program like DSSP. PyMOL includes a slow and very coarse secondary structure assignment function

```
util.ss <selection>
```

which should only be used as a last resort. WARNING: it will make mistakes, so don't publish anything based on this algorithm! It can be accessed from the GUI under the action (diamond) menu for objects.

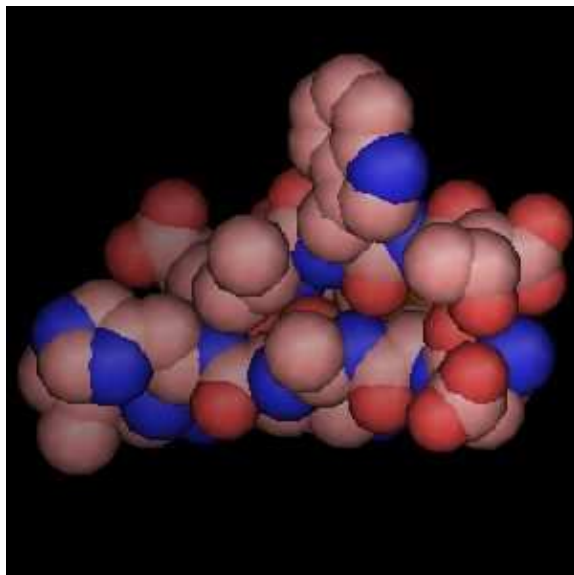
Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

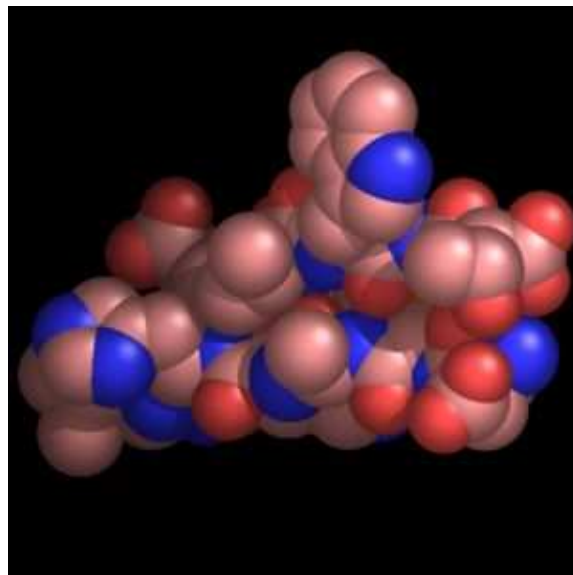
Hosted at: \_\_\_\_\_

# Ray-Tracing

Ray-tracing produces the highest quality molecular graphics images. **PyMOL is the first full-featured molecular graphics program to include a high-speed ray-tracer** which works with its native internal geometries (except text).



OpenGL Rendering (real-time manipulation)



Ray-traced Rendering (seconds or minutes per frame)

You can ray-trace any Scene in PyMOL by clicking the "Ray Trace" button in the external GUI or using the "ray" command. The built-in raytracer also makes it possible to easily assemble very high-quality movies in a snap.

## Important Settings

These can be changed using the "set" command. Unless otherwise specified, the settings apply only to the ray-tracing engine and not the OpenGL renderer. Some reconciliation between the two renderers is much needed, so be warned that these settings may change in the future.

Normally, the only settings you will need to change are **orthoscopic**, **antialias**, and **gamma**. If you are down in an enzyme active site which is heavily shadowed, you may want to increase **direct** to 0.5–0.7 in order to improve brightness and contrast.

- **orthoscopic** (0 or 1) controls whether the OpenGL renderer uses the same orthoscopic transformation as the renderer. You'll want to set this to 1 when preparing figures so that OpenGL and raytracing match pixel-for-pixel.
- **ambient** (0.0–1.0) controls the ambient light intensity for both OpenGL and the ray-tracer.
- **ambient\_scale** (float) controls the relative ambient intensity between OpenGL and the ray-tracer.
- **antialias** (0 or 1) generate a "smooth" image (best quality, but takes 4X as long).
- **direct** (0.0–1.0) the planer light intensity originating from the camera.
- **gamma** (0.1–2.0) gamma transformation applied after rendering is complete.

- **light** (vector) the position of the light.
- **reflect** (0.0–1.0) the planer light intensity originating from the light source.
- **spec\_reflect** (0.0–1.0) intensity of the specular reflection from the light.
- **spec\_power** (1–100) how localized is the specular reflection (higher=smaller).

## Saving Images

### png

All images (ray-traced or not) can be saved in PNG format using the "png" command. This format is directly readable by PowerPoint, and can be easily converted into other formats using a package like ImageMagick. You can also save images using the "Save Image" option in the "File" menu. Images are always saved at the same resolution as the viewer window.

```
ray
png my_image.png
```

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Movies

## Concepts

### States and Frames

PyMOL has a powerful and unique molecular movie-making capability. In order to use it, you first need to understand a few key concepts:

- **States:** States most directly correspond to particular arrangements of atoms at a point in time. For example, they could consist of steps in a molecular dynamics simulation or individual points of a coordinate interpolation. If you are making a movie of a static coordinate set (such as a single crystal structure) then you have only one state. All objects in PyMOL can potentially consist of multiple states.
- **Frames:** Frames are like the individual images you'd find on a movie reel, except that in PyMOL, frames are composed of states instead of images, and frames can have additional actions associated with them (such as rotation of the camera).

The user can fully interact with models while movies are playing.

**NOTE:** State and frame indexes begin with 1, and not 0 as most C and Python programmers would expect. If you load states into an object with a state index of 0, you are indicating that you want the state to be appended after the last existing state in the object.

## Important Commands To Know

### load

The "load" command is used to populate states of an object. By default, each new file loaded will be appended onto the object's states. However, the optional third argument to the load command is the frame index into which the file should be loaded. See "help load" or consult the reference section for more information.

```
load foo1.pdb,mov      # loads foo1.pdb into state 1 of "mov".
load foo2.pdb,mov      # loads foo2.pdb into state 2 of "mov".
load foo3.pdb,mov,3    # loads foo3.pdb into state 3 of "mov".
load foo4.pdb,mov,4    # loads foo4.pdb into state 4 of "mov".
```

### mset

The "mset" command is used to specify which states get included as frames of a movie. If the mset command is not used, PyMOL will by default play through all states in sequential order. However, if you wish to use the other movie commands (such as mdo), it is necessary to explicitly use the mset

command to create a movie definition inside of PyMOL.

The `mset` command is followed by an arbitrarily list of statements which defines the entire movie. Each statement takes on one of three forms:

1. `#` A number simply indicates a state is to be played next.
2. `x#` A lowercase "x" immediately followed by a number (no space) indicates that the previous state should be repeated that many times total.
3. `-#` A hyphen immediately followed by a number (no space) indicates that a numeric sequence of states are to be appended onto the movie starting with the previously played state going to indicated state.

Once a movie has been defined, the red "VCR" controls in the lower-right-hand corner of the viewer can be used to step or play through the movie.

## Examples

```
mset 1 x30      # creates a 30 frame movie consisting of state 1 played 30 times.
mset 1 -30     # creates a 30 frame movie: states 1, 2, ... up through 30.
mset 1 -30 -2  # 58 frames: states 1, 2, ... , 29, 30, then 29, 28, ... , 4, 3, down to 2
mset 1 6 5 2 3 # 5 frames: states 1, 6, 5, 2, 3 in that order.
```

See "help mset" or the reference section for more information.

## mdo

The "mdo" command allows you to bind a particular series of PyMOL commands to a frame in the movie. For instance, you can perform a rotation about the axis at each frame of the movie in order to sweep the camera about the object. See "help mdo" or the reference section for more information.

**NOTE:** The "util" module includes two python commands for generating mdo commands, "util.mrock" and "util.mdo". These functions have not been documented, but the source code can be found in the file `modules/pymol/util.py`. Since they are actual python functions, explicit parenthesis are required to invoke them.

```
util.mrock(start, finish, angle, phase, loop-flag)
util.mroll(start, finish, loop-flag)
```

## mmatrix

The "mmatrix" command allows you to store and recall a particular viewing matrix to be used to set up frame 1 of the movie. This can be particularly helpful when you're trying to preserve a movie's orientation while performing other actions within PyMOL during the same session. See "help mmatrix" or the reference section for more information.

## Simple Examples

Here a static structure is subject to a gentle rock. The following statements create a sixty frame movie which simply rocks the protein by 10 degrees.

```
load test/dat/pept.pdb      # load a structure
mset 1 x60                  # define the movie
util.mrock(1,60,10,1,1)    # issues mdo commands to create +/- 10 deg. rock over 60 frames
```

In this next example, the protein is rotated through a full 360 sweep about the Y-axis over 120 frames

```
load test/dat/pept.pdb      # load a structure
mset 1 x120                  # define the movie
util.mroll(1,120,1)         # issues mdo commands to create full rotation over 120 frames
```

## Complex Examples

The following is a Python program (with a .py or .pym extension) which uses a Python loop to load a large number of numbered PDB files, and then configures PyMOL to show them both forwards and backwards.

```
from glob import glob
from pymol import cmd

file_list = glob("mov*.pdb"):

for file in file_list
    cmd.load(file,"mov")

cmd.mset("1 -%d -2"%len(file_list))
```

## Previewing Ray-traced Movie Images

PyMOL has the ability to cache a series of images in RAM and to play them back at a much higher rate than they could be rendered originally. This is most-useful for ray-traced images, but it can also be used with OpenGL images.

### cache\_frames

The **cache\_frames** option controls whether or not PyMOL saves frames in memory. Its usage is demonstrated in the following script. NOTE: caching images takes a tremendous amount of memory, so you should use the "viewport" command to shrink the window before utilizing this option.

```
viewport 320,240
load test/dat/pept.pdb
orient
hide
show sph
mset 1 x30
```

```
util.mrock 1,30,3,1,1
set ray_trace_frames=1
set cache_frames=1
mplay
```

## **mclear**

Once you have loaded a set of frames into RAM, the frames will remain there until you run the "mclear" command, even if you manipulate that model. You can also press the mclear button on the external GUI window.

```
mclear # flushes the frame cache
```

## **Saving movies**

### **mpng**

You can save movie images to numbered PNG format files with a common prefix. If you want each frame to be ray-traced, you should turn on raytracing of frames, turn off caching, and clear the cache (see the Movie Menu or use the following commands).

```
set ray_trace_frames=1
set cache_frames=0
mclear
```

You can save the movie using the "mpng" command, or you can save it from the "File" menu. Either way, you must provide a prefix which will be used to create numbered PNG files.

```
mpng mov # will create mov0001.png, mov0002.png, etc.
```

If you are compressing movies using Adobe Premiere (recommended for best quality), you will probably want to convert the files using ImageMagick or a similar package into a format that Premiere is capable of reading (such as ".tga" – targa format).

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: [\\_\\_\\_\\_\\_](#)

# Advanced Mouse Controls

## Picking Atoms and Bonds

The current mouse configuration is visible on the lower right-hand corner of the screen as a matrix. Under the default mouse configuration:

- Atoms are "picked" using the "PkAt" function which is CTRL/middle-click.
- Bonds are "picked" using the "PkBd" function which is CTRL/right-click.

Whenever an atom or bond has been picked, a number of atom selections are automatically defined as described in the following table:

- **(pk1)** The selected atom (or the first selected atom in a bond selection).
- **(pk2)** The second selected atom in a bond selection.
- **(pkfrag#)** A fragment of the molecule with its base adjacent to the selected bond or atom.
- **(pkchain)** The contiguous chain of atoms which contains the selected atom or bond.
- **(pkresi)** All atoms in the residue you picked.

You can click on a selection name to see visually which atoms are included in the selection. All of these selections can be used and manipulated as if they were created manually using the select command. Note however, that these selections are quite fragile, and will be automatically deleted in response to a number of common occurrences, such as loading a new object.

## Example Usage of the "pk" Atom Selections

Assuming that you picked an atom or bond...

```
show sticks,(pkresi)      # show sticks on the residue you picked
color read,(pkchain)     # color the chain you picked
remove (byres pk1)       # removes all atoms in the residue you picked
```

## The "lb" and "rb" Selections

Most of the time, the "pk1" atom selection will suffice. However, there are times when you need to specify two or more sets of atom selections. This is where "lb" and "rb" come in.

In addition to the "pk" atom selection set, there are two more atom selections which can be manipulated with the mouse. These are: (1) the left-button or "lb" selection and (2) the right-button or "rb" selection. Under the default mouse configuration:

- The "lb" selection can be redefined using the "lb" function which is CTRL-SHIFT/left-click.
- The "lb" selection can be expanded using the "+lb" function which is CTRL/left-click.
- The "rb" selection can be redefined using the "rb" function which is CTRL-SHIFT/right-click.

Certain commands are designed to use "(lb)" and "(rb)" as default arguments. For instance, the "distance" command, if called without any arguments, will attempt to create a distance object between the (lb) and (rb) selections if they exist

```
# define (lb) by CTRL-SHIFT/left-clicking one atom
# define (rb) by CTRL-SHIFT/right-clicking another

dist # will create a distance object between the two atoms.
```

## Conformational Editing

Sorry, no documentation yet -- these features won't be too useful until PyMOL is coupled up with an energy minimiation engine.

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Crystallography Applications

## Crystal Symmetry

Ralf Grosse-Kunstleve has provided his SgLite module to enable PyMOL to deduce symmetry relationships from standard space group and unit cell information. Currently that information can only be provided to PyMOL as a CRYST1 record in the PDB file, which includes the correct space group identifier. However, it would be only a minor development task to add a means of assigning unit-cell and symmetry to any molecule object directly from the API.

The format of the CRYST1 record is as follows.

1 - 6	Record name	"CRYST1"	
7 - 15	Real(9.3)	a	a (Angstroms).
16 - 24	Real(9.3)	b	b (Angstroms).
25 - 33	Real(9.3)	c	c (Angstroms).
34 - 40	Real(7.2)	alpha	alpha (degrees).
41 - 47	Real(7.2)	beta	beta (degrees).
48 - 54	Real(7.2)	gamma	gamma (degrees).
56 - 66	LString	sGroup	Space group.
67 - 70	Integer	z	Z value. # ignored by PyMOL

## load

When you use the "load" command to read in a PDB file with symmetry information, matrix information should be output. Verify that this information is produced before attempting to display symmetry related molecules.

## symexp

The "symexp" command is used to display symmetry related molecules in the crystal lattice about an atom selection. This command creates a set of new objects with a common prefix. Each object in the series corresponds to one symmetry-related object, which can be treated independently. See "help symexp" or the reference section for usage information.

In order to visualize only symmetry-related atoms within a given distance, you need to break the process down into two steps. First, you use the symexp command to create complete symmetry-related objects. Then you use "hide" commands to restrict what is visible to only those areas which you are interested.

```
load foo.pdb # load PDB file with CRYST record

symexp sym=foo,(foo),5.0 # Create symmetry related "foo" objects
                        # which pass within 5 angstroms of foo
                        # using the prefix "sym"

hide (not (foo expand 5)) # hide atoms greater than 5 A from foo
```

NOTE: The symexp command can potentially create large numbers of objects. You will want to use

the "delete" command with a wildcard "\*" to remove all objects that share a common prefix.

```
delete sym*           # deletes objects starting with "sym"
```

## Electron Density Maps

The only map file format currently supported is the CNS and XPLOR ASCII format map file. PyMOL can read large maps of this format and then display arbitrary "bricks" of density within these maps about atom selections.

### load

PyMOL expects XPLOR/CNS map files to have a ".xplor" extension. This requirement can be avoided by supplying an explicit type of "xplor" to the "load" command.

```
load 2fofc.xplor,map1      # type inferred from the extension
load 2fofc.map,map1,1,xplor # type explicitly provided
```

See "help load" or the reference section for additional information.

### isomesh and isodot

Map objects are used to store the data and are represented by a wire-frame brick in space indicating the extent of the map. An arbitrary number of mesh or dots objects can be created from a given map using the "isomesh" and "isodot" commands.

```
isomesh msh1,map1,1.0 # display an isosurface-mesh at level 1.0 over
                      # the entire map object "map1"

isomesh msh2,map1,1.0,(chain A),3.0 # display isosurface-mesh at 1.0
                                     # in a brick about chain A with a
                                     # border of 3.0 Angstroms
```

See "help isomesh" or the reference section for additional information.

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: [\\_\\_\\_\\_\\_](#)

# Settings

## Under Renovation

PyMOL's settings infrastructure is about halfway through a major overhaul. Previously, there was only one settings database for the entire program and all settings were floating point numbers. Now settings can be of several different types, and every state and every object can have an optional set of distinct settings. However, much of the internal code has yet to be modified to take advantage of these new features, so many object and state specific settings won't yet have the desired effect. This work should be completed by version 0.60 at the latest.

In addition, there are some settings names which should be revised because they are misleading or confusing. Therefore, expect some setting names and functions to change up through release 0.70.

### set

The "set" command is used to change settings. See "help set" or the reference section for more information.

```
set stick_radius,0.1          # global
set stick_radius,0.2,prot     # object specific
set stick_radius,0.3,prot,3   # object and state specific
```

## Reference

Because the names and functions of settings are still somewhat in flux, they are simply listed here without description. Your best bet is to avoid relying on these until we approach the 1.0 release. However, an attempt will be made to support old settings names in the new releases.

You can always view the current settings database using the "Edit All..." option of the "Settings" menu

```
all_states
ambient
ambient_scale
antialias
auto_zoom
autohide_selections
autoshow_lines
autoshow_nonbonded
autoshow_selections
backface_cull
bg_rgb
bonding_vdw_cutoff
button_mode
cache_display
cache_frames
cartoon_debug
cartoon_dumbbell_length
```

cartoon\_dumbbell\_radius  
cartoon\_dumbbell\_width  
cartoon\_fancy\_helices  
cartoon\_fancy\_sheets  
cartoon\_flat\_sheets  
cartoon\_loop\_quality  
cartoon\_loop\_radius  
cartoon\_oval\_length  
cartoon\_oval\_quality  
cartoon\_oval\_width  
cartoon\_power  
cartoon\_power\_b  
cartoon\_rect\_length  
cartoon\_rect\_width  
cartoon\_refine\_normals  
cartoon\_round\_helices  
cartoon\_sampling  
cartoon\_smooth\_loops  
cartoon\_tube\_quality  
cartoon\_tube\_radius  
cavity\_cull  
cgo\_ray\_width\_scale  
cull\_spheres  
dash\_gap  
dash\_length  
dash\_radius  
dash\_width  
depth\_cue  
direct  
dist\_counter  
dot\_density  
dot\_hydrogens  
dot\_mode  
dot\_size  
dot\_width  
fast\_idle  
fog  
gamma  
half\_bonds  
hash\_max  
idle\_delay  
internal\_gui  
internal\_gui\_width  
isomesh\_auto\_state  
label\_color  
light  
line\_radius  
line\_smooth  
line\_width  
max\_triangles  
mesh\_radius  
mesh\_width  
min\_mesh\_spacing  
movie\_delay  
no\_idle  
nonbonded\_size  
normal\_workaround  
orthoscopic  
overlay  
pickable  
power  
ray\_trace\_fog

ray\_trace\_fog\_start  
ray\_trace\_frames  
reflect  
ribbon\_power  
ribbon\_power\_b  
ribbon\_radius  
ribbon\_sampling  
ribbon\_width  
rock\_delay  
sel\_counter  
selection\_overlay  
selection\_width  
shininess  
single\_image  
slow\_idle  
solvent\_radius  
spec\_power  
spec\_reflect  
specular  
sphere\_quality  
spheroid\_fill  
spheroid\_scale  
spheroid\_smooth  
static\_singletons  
stereo\_angle  
stereo\_shift  
stick\_nub  
stick\_overlap  
stick\_quality  
stick\_radius  
surface\_best  
surface\_normal  
surface\_proximity  
surface\_quality  
sweep\_angle  
sweep\_speed  
test1  
test2  
text  
trim\_dots  
valence

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Compiled Graphics Objects (CGOs) and Molscript Ribbons

## Introduction

Although PyMOL uses OpenGL for all real-time rendering, the simple ray-tracing engine inside of PyMOL is incapable of understanding arbitrary OpenGL calls. Thus, any graphics scene must be translated into a set of primitives (spheres, cylinders, and triangles) that can be provided to the ray-tracer in order to generate high quality images with "ideal" geometries, lighting, and shadows.

Compiled graphics objects are a PyMOL-specific format which enables any Python programmer to create three-dimensional geometries and animations which can be displayed at high-speed via OpenGL and also rendered into maximum-quality images via the raytracer without any additional work.

## Molscript Ribbons

NOTE: Molscript is a commercial software (free to academics) available at <http://www.avatar.se/molscript/> and must be obtained separately. It is our intention to eventually implement our own Molscript-quality ribbons directly from within PyMOL, but that day has not yet come.

PyMOL can automatically translate Raster3D format input files output by Molscript (with "-r" option) into Compiled Graphics Objects for display and rendering inside of PyMOL. PyMOL expects these files to have the file extension ".r3d". NOTE: the Raster3D-to-CGO interpreter is a bare-minimum Python implementation, and doesn't include anything beyond what is required to read what is output by Molscript.

## load

```
load test/dat/pept.r3d # loads one of the example raster3d files
```

## Using Molscript

### molauto

When using molauto to preparing input files for PyMOL, it is important to use the "-nocentre" option to prevent any transformation of the protein. That way the PDB file and the Molscript ribbons will be in the same frame of reference.

```
Unix> molauto -nocentre 3all.pdb | molscript -r > test1.r3d
Unix> molauto -nocentre -nice 3all.pdb | molscript -r > test2.r3d
```

You can load both PDB and ribbon files directly into PyMOL as separate objects.

```
load 3all.pdb # loads the coordinates
```

```
load test1.r3d # loads molscript ribbon
```

## Molscript Input Files

Unfortunately, PyMOL does not have the ability to write molscript input files which reflect the current atom colorings and visibilities. Therefore, you will need to get in the habit of manually editing Molscript input files in order color and customize ribbons appropriately. Here are some tips:

1. Remove any line starting with "transform atom" from existing Molscript input files in order to preserve the frame of reference. For example:

```
transform atom * by centre position atom *;
```

2. For performance reasons, you may want to set the segments to a small number while working with Molscript ribbons in real-time. Later on you can increase this number, recreate, and reload the ".r3d" files.

```
set segments 2; # good for real-time graphics
```

```
set segments 8; # good for rendering
```

The easiest way to create new ribbons using PyMOL is to use the "save" command to write out a PDB file containing the atom selection of interest. You can then apply the "system" command to run molauto and molscript, and then load the Raster3D file back into PyMOL.

```
save tmp.pdb,(chain C)
system molauto -nocentre tmp.pdb | molscript -r > tmp.r3d
load tmp.r3d
```

## Creating Compiled Graphics Objects

Compiled graphics objects contain equivalents to the normal line and triangle primitives found in OpenGL but also include primitives for spheres and cylinders.

At the Python level, compiled graphics objects are constructed as a simple linear list of Python floating point numbers, which is conceptually equivalent to an OpenGL stream.

```
from pymol.cgo import * # get constants
from pymol import cmd

obj = [
    BEGIN, LINES,
    COLOR, 1.0, 1.0, 1.0,

    VERTEX, 0.0, 0.0, 0.0,
    VERTEX, 1.0, 0.0, 0.0,

    VERTEX, 0.0, 0.0, 0.0,
    VERTEX, 0.0, 2.0, 0.0,

    VERTEX, 0.0, 0.0, 0.0,
    VERTEX, 0.0, 0.0, 3.0,
```

```
END  
]
```

```
cmd.load_cgo(obj, 'cgo01')
```

CGOs support the standard OpenGL BEGIN/END formalism as well as a few stand-alone primitives, SPHERE, CYLINDER, and TRIANGLE, which should NOT appear within a BEGIN/END block.

## CGO Reference

A CGO is simply a Python list of floating point numbers, which are compiled by PyMOL into a CGO object and associated with a given state.

Lowercase names below are should be replaced with floating-point numbers. Generally, the TRIANGLE primitive should only be used only as a last restore since it is much less effective to render than using a series of vertices with a BEGIN/END group.

```
BEGIN, { POINTS | LINES | LINE_LOOP | LINE_STRIP | TRIANGLES | TRIANGLE_STRIP | TRIANGLE_FAN },  
VERTEX, x, y, z,  
COLOR, red, green, blue,  
NORMAL, normal-x, normal-y, normal-z,  
END,  
LINEWIDTH, line-width,  
WIDTHSCALE, width-scale, # for ray-tracing  
SPHERE, x, y, z, radius # uses the current color  
CYLINDER, x1, y1, z1, x2, y2, z2, radius,  
          red1, green1, blue1, red2, green2, blue2,  
TRIANGLE, x1, y1, z1,  
          x2, y2, z2,  
          x3, y3, z3,  
          normal-x1, normal-y1, normal-z1,  
          normal-x2, normal-y2, normal-z2,  
          normal-x3, normal-y3, normal-z3,  
          red1, green1, blue1,  
          red2, green2, blue2,  
          red3, green3, blue3,
```

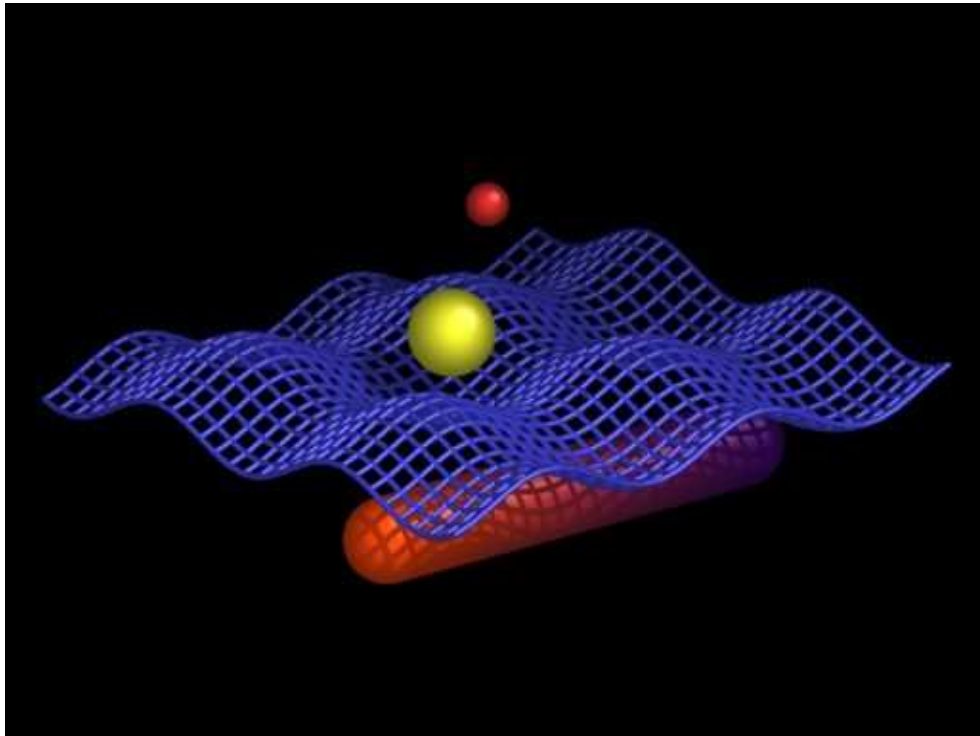
## load\_cgo

CGO lists are loaded into PyMOL using the "load\_cgo" function.

```
cmd.load_cgo(list, name, state)
```

Arbitrary 3D animations can be created by loading CGOs into consecutive states of a given object.

The example below is a static image from the "examples/devel/cgo03.py" cgo animation demonstration program.



Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Callback Objects and PyOpenGL

This is mainly a developer's function, so most users can skip this section. You will need some Python knowledge to understand the example.

## Introduction

Although all OpenGL rendering in PyMOL is performed at the C level, PyOpenGL provides an alternative binding of the OpenGL API from Python. Unfortunately, it is impossible for PyMOL to produce ray-traced images of objects rendered using PyOpenGL. Nevertheless, PyOpenGL can be used within PyMOL via Callback objects for pure OpenGL-based rendering purposes. If you need your graphics to be ray-traceable, then you should use Compiled Graphics Objects (see previous section).

Callback objects are trivial Python objects which have a "`__call__`" method for rendering and a "`get_extent`" method which tells PyMOL where in space the object is located. Once a callback object has been loaded into PyMOL, Python will automatically call this object when needed to update the screen.

PyMOL includes a copy of PyOpenGL under the pymol module hierarchy (`pymol.opengl`), but usage of this copy is of course optional. You can instead bind to the latest version without problems, provided that you install it yourself into the Python library that PyMOL is using by default.

NOTE: The current Windows version of PyMOL does not include Numeric, which makes heavy usage of PyOpenGL from within PyMOL impractical under Windows at present.

## Example

The following Python program shows how you can use a Callback object within PyMOL to perform rendering using OpenGL. For more examples, see the directory "`$PYMOL_PATH/examples/dev`".

```
from pymol.opengl.gl import *
from pymol.callback import Callback
from pymol import cmd

class myCallback(Callback):

    def __call__(self):

        glBegin(GL_LINES)

        glColor3f(1.0,1.0,1.0)

        glVertex3f(0.0,0.0,0.0)
        glVertex3f(1.0,0.0,0.0)

        glVertex3f(0.0,0.0,0.0)
        glVertex3f(0.0,2.0,0.0)

        glVertex3f(0.0,0.0,0.0)
        glVertex3f(0.0,0.0,3.0)
```

```
glEnd()  
  
def get_extent(self):  
    return [[0.0,0.0,0.0],[1.0,2.0,3.0]]  
  
cmd.load_callback(myCallback(), 'gl01')
```

## load\_callback

Callback objects are loaded into PyMOL using the "load\_callback" function.

```
cmd.load_callback(object,name,state)
```

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: [\\_\\_\\_\\_\\_](#)

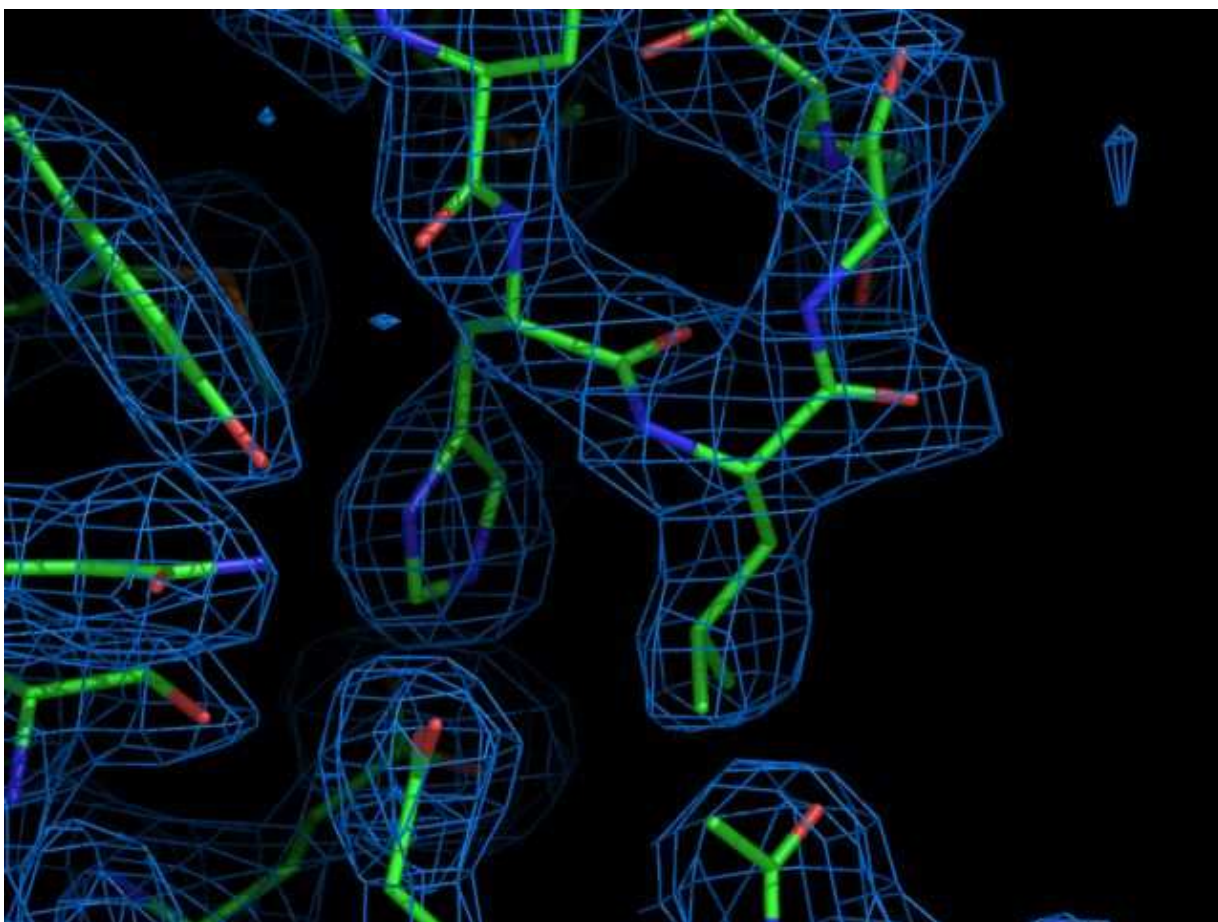
# Cookbook: Crystallographic Imagery

In this section and those which follow, you will find complete recipes for generating figures directly from the PyMOL command language. Note that when you are using PyMOL interactively, you can manually perform the same actions using the mouse and menus instead of typing commands, and thus your scripts can be quite simple and shorter than what you see here.

**WARNING:** Some of these examples require significant memory and processor speed in order to be practical. I recommend a 500+ mhz machine with at least 256 MB of RAM for most of these. Less RAM is okay, but you'll end up using virtual memory (slow!).

---

## Electron density (density.pml)



```
# alter settings for publication quality

set mesh_radius = 0.02
set antialias = 1
set stick_radius = 0.1

# load pdb and map file
```

```
load 1DN2.pdb,1dn2
load 2fofc.xplor

# display region of interest

hide
show sticks,(byres ((c:f & i:5,6) x:4))

# show e-density nearby

isomesh den, 2fofc, 1.0, (c:F & i:6), 8.0
color marine,den

# zoom in, turn, and clip

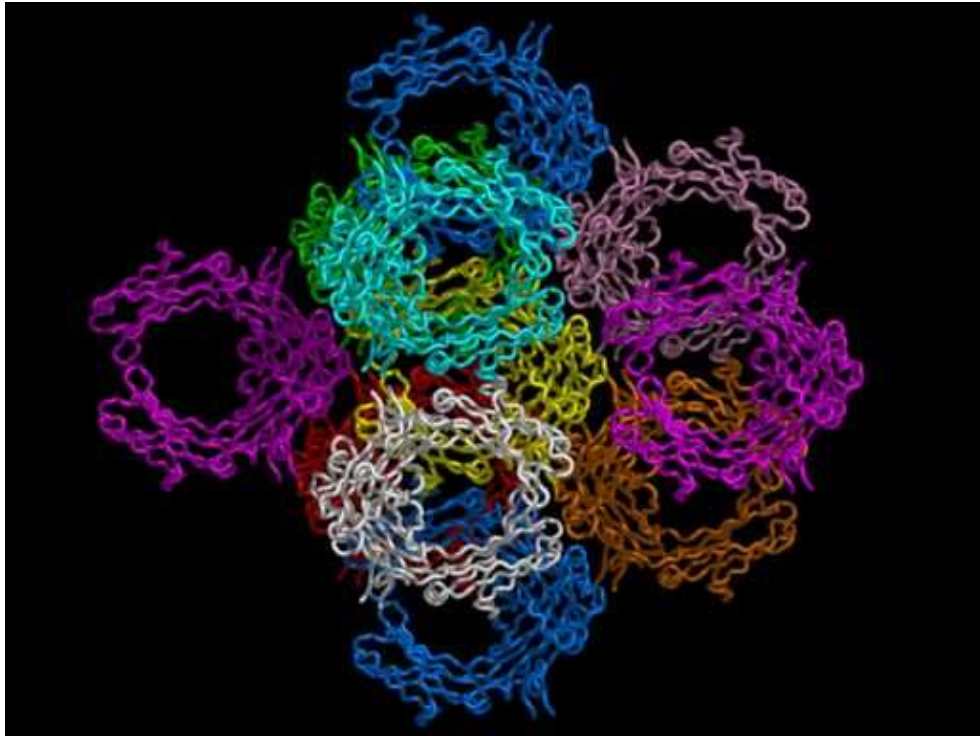
zoom (c:f & i:5,6),2
turn x,10
turn y,-10
clip slab, 6.5

# ray trace and write image file

ray
png density.png
```

---

## Crystal Packing with Ribbons (packing.pml)



```
# load pdb file which has a CRYST1 record
load 1DN2.pdb,1dn2

# create neighbors with contacts
symexp s,1dn2,(all),4.0

# display ribbon only, and make it a little bit "fatter"
hide
set ribbon_radius = 0.75
show ribbon

# rotate and zoom
turn y,90
move z,50
zoom

# color symmetry-related copies distinctly

color white
color yellow, (1dn2)
color cyan, (s01-10100)
color marine, (s0000-100 or s00000100)
color red, (s01000000)
color violet, (s00000001 or s000000-1)
```

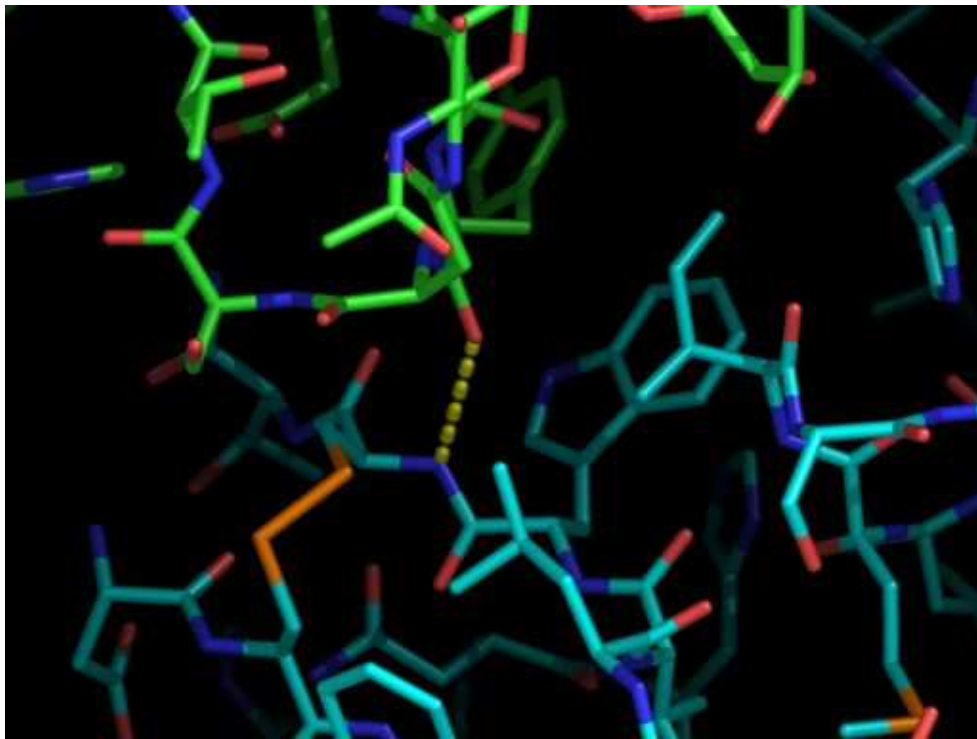
```
color orange, (s01000001)
color pink,   (s01000101)
color green,  (s01000100)

# now render (about 1 minute)

set antialias=1
ray
png packing.png
```

---

## A Crystal Contact Interaction (contact.pml)



```
# load pdb file which has a CRYST1 record
load 1DN2.pdb,1dn2

# create neighbor with contact near a certain atom
symexp s,1dn2,(c; F and i;12 and n;N),4.0

# hide everything outside region of interest
hide (!(byres ((1dn2 and c; F and i;12 and n;N) x;12)))

# color-by-atom-cyan on 1dn2 to distinguish it from symmetry-related copy
util.cbac 1dn2

# restore view saved using from get_view in a previous session

set_view (\
  -0.042664494,  -0.003409438,   0.999083281,\
  -0.617746830,   0.786019623,  -0.023697896,\
  -0.785219014,  -0.618191123,  -0.035642438,\
  -1.056621075,  -1.729980350,  -39.743129730,\
  29.859001160,  38.402999878,   19.087999344,\
  27.443143845,  46.443149567,   1.000000000 )

# show the single hydrogen bond in the interface
```

```
dist ( s01000000 segi '' chain B resn TYR resi 296 name O ),\  
      ( ldn2 segi '' chain F resn CYS resi 12 name N )
```

```
# image
```

```
set antialias=1  
ray  
png contact.png
```

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

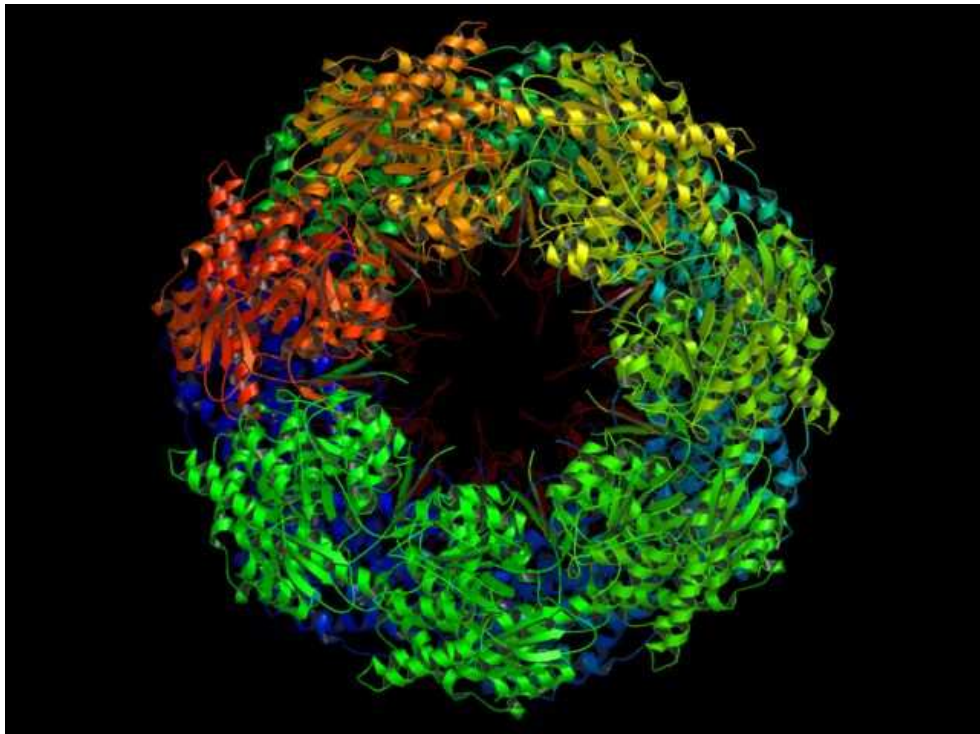
# Cookbook: Pushing the Limits

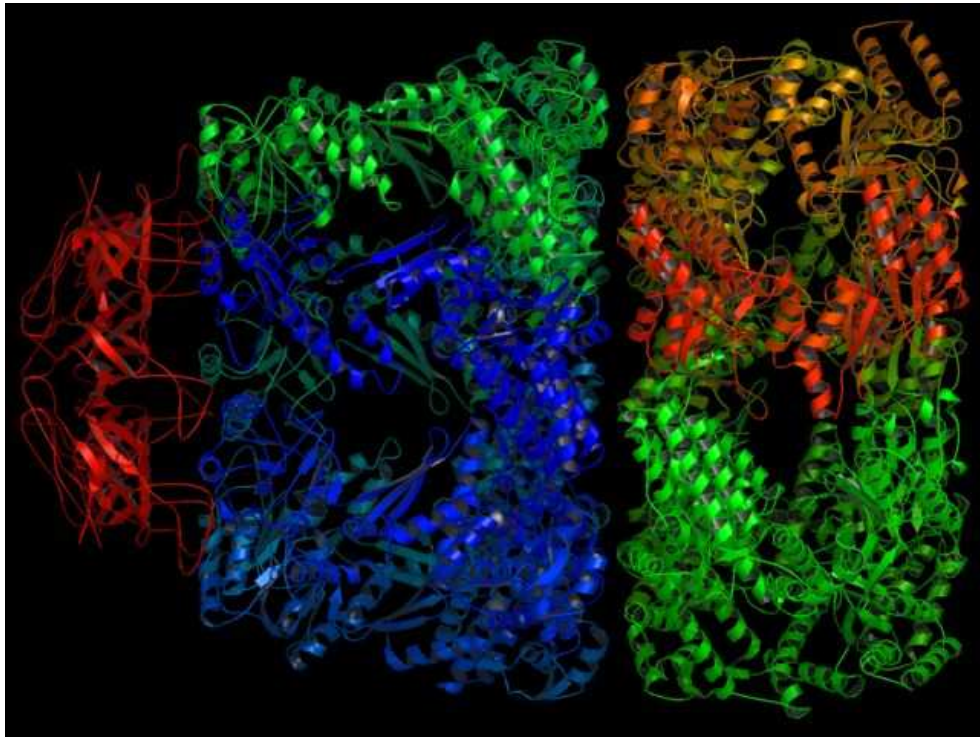
*Here are some examples of things that PyMOL was never designed to do, but that it manages to accomplish anyhow!*

---

## GroEL/ES (groel\_es.pml)

Requires Molscrip. CPU Time: about 4.5 minutes on a 1 Ghz Athlon (256 MB/Linux) for two images. Used over 300MB RAM/Swap





```
# run molscript on the PDB file
system molauto -nocentre 1AON.pdb | molscript -r > 1AON.r3d

# load raster3D input file
load 1AON.r3d

# zoom in a little

reset
move z,100

# render

set antialias=1
ray
png groel_es1.png

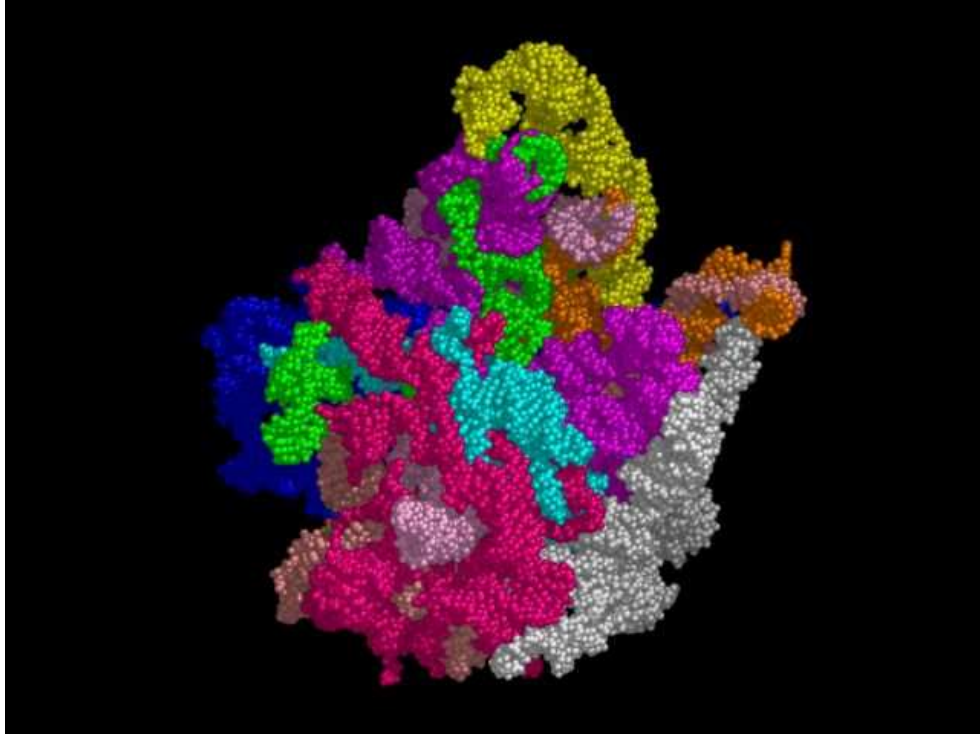
# render

turn y,90
ray
png groel_es2.png
```

---

## Nucleic Acid in the Ribosome (ribosome.pml)

CPU Time: 66 seconds on a 1 Ghz Athlon (256 MB/Linux). Used over 350MB RAM/swap.



```
load 1FFK.pdb,1ffk

# create object with only nucleic acid

create nuc = (1ffk and not n;ca,cd)
del 1ffk

# color contiguous chains

edit (c;0 & n;P & i;713)
color blue,(pkchain)

edit (c;0 & n;P & i;2250)
color green,(pkchain)

edit (c;0 & n;P & i;1876)
color magenta,(pkchain)

edit (c;0 & n;P & i;2596)
fcolor violet,(pkchain)

edit (c;0 & n;P & i;1993)
color cyan,(pkchain)

edit (c;0 & n;P & i;2764)
color white,(pkchain)
```

*Nucleic Acid in the Ribosome (ribosome.pml)*

```
edit (c;0 & n;P & i;1552)
color salmon,(pkchain)

edit (c;0 & n;P & i;857)
color pink,(pkchain)

edit (c;0 & n;P & i;1149)
color orange,(pkchain)

edit (c;9 & n;P & i;21)
color yellow,(pkchain)

# orient the "hand"

reset
turn y,-60
turn z,180
move y,-15

# show spheres

show spheres

# ray trace

set antialias=1
ray

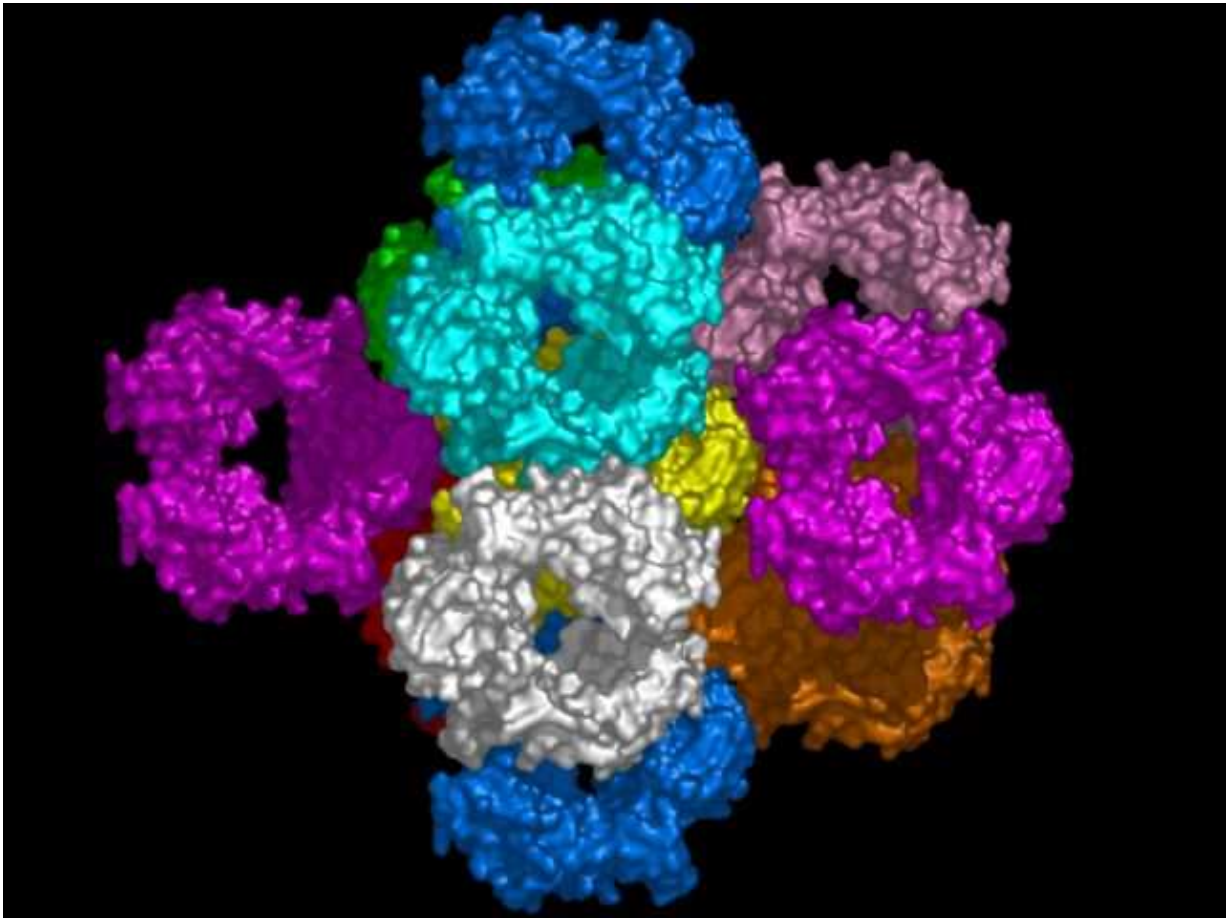
# write output

png ribosome.png
```

---

## Crystal Packing With Surfaces (packsurf.pml)

CPU Time: 25 minutes on a 1 Ghz Athlon (256 MB/Linux). Used over 840 MB (!) of RAM/swap and involved raytracing over a half-million triangles. Can you imagine working with 500 Kd worth of protein surfaces in any other program?



```
# load pdb file which has a CRYST1 record
load 1DN2.pdb,1dn2

# remove waters
remove (resn hoh)

# create neighbors with contacts
symexp s,1dn2,(all),4.0

# display surfaces

hide
show surface
```

*Crystal Packing With Surfaces (packsurf.pml)*

```
# rotate and zoom

reset
turn y,90
move z,30
turn x,5
turn y,5

# color symmetry-related copies distinctly

color white
color yellow, (ldn2)
color cyan, (s01-10100)
color marine, (s0000-100 or s00000100)
color red, (s01000000)
color violet, (s00000001 or s000000-1)
color orange, (s01000001)
color pink, (s01000101)
color green, (s01000100)

# now render

set antialias=1
ray
png packsurf.png
```

Copyright © 2001 [DeLano Scientific](#). All rights reserved.

---

Hosted at: \_\_\_\_\_

# Reference

---

## alias

### DESCRIPTION

"alias" allows you to bind a commonly used command to a single word

### USAGE

```
alias name, command-sequence
```

### PYMOL API

```
cmd.alias(string name,string command)
```

### EXAMPLES

```
alias go,load "test.pdb"; zoom (i;500); show sticks,(i;500 a;4)
go
```

### SEE ALSO

```
extend
```

---

## alter

### DESCRIPTION

"alter" changes one or more atomic properties over a selection using the python evaluator with a separate name space for each atom. The symbols defined in the name space are:

```
name, resn, resi, chain, alt, elem, q, b, segi,
type (ATOM,HETATM), partial_charge, formal_charge,
text_type, numeric_type, ID
```

All strings must be explicitly quoted. This operation typically takes several seconds per thousand atoms altered.

WARNING: You should always issue a "sort" command on an object after modifying any property which might affect canonical atom ordering (names, chains, etc.). Failure to do so will confound subsequent "create" and "byres" operations.

### USAGE

```
alter (selection),expression
```

### EXAMPLES

```
alter (chain A),chain='B'
alter (all),resi=str(int(resi)+100)
sort
```

SEE ALSO

`alter_state`, `iterate`, `iterate_state`, `sort`

---

## **alter\_state**

DESCRIPTION

"alter\_state" changes the atomic coordinates of a particular state using the python evaluator with a separate name space for each atom. The symbols defined in the name space are:

`x,y,z`

USAGE

`alter_state state,(selection),expression`

EXAMPLES

`alter 1,(all),x=x+5`

SEE ALSO

`iterate_state`, `alter`, `iterate`

---

## **api**

DESCRIPTION

The PyMOL Python Application Programming Interface (API) should be accessed exclusively through the "cmd" module (never "\_cmd!"). Nearly all command-line functions have a corresponding API method.

USAGE

```
from pymol import cmd
result = cmd.<command-name>( argument , ... )
```

NOTES

Although the PyMOL core is not multi-threaded, the API is thread-safe and can be called asynchronously by external python programs. PyMOL handles the necessary locking to insure that internal states do not get corrupted. This makes it very easy to build complicated systems which involve direct realtime visualization.

---

## at\_sign

### DESCRIPTION

"@" sources a PyMOL command script as if all of the commands in the file were typed into the PyMOL command line.

### USAGE

```
@ <script-file>
```

### PYMOL API

Not directly available. Instead, use `cmd.do("@...")`.

---

## attach

### DESCRIPTION

"attach" adds a single atom onto the picked atom.

### USAGE

```
attach name, geometry, valence
```

### PYMOL API

```
cmd.attach( name, geometry, valence )
```

### NOTES

Immature functionality. See code for details.

---

## backward

### DESCRIPTION

"backward" moves the movie back one frame.

### USAGE

```
backward
```

### PYMOL API

```
cmd.backward()
```

### SEE ALSO

`mset`, `forward`, `rewind`

---

## bg\_color

### DESCRIPTION

"bg\_color" sets the background color

### USAGE

```
bg_color [color]
```

### PYMOL API

```
cmd.color(string color="black")
```

---

## bond

### DESCRIPTION

"bond" creates a new bond between two selections, each of which should contain one atom.

### USAGE

```
bond [atom1,atom2 [,order]]
```

### PYMOL API

```
cmd.bond(string atom1, string atom2)
```

### NOTES

The atoms must both be within the same object.

The default behavior is to create a bond between the (lb) and (rb) selections.

### SEE ALSO

unbond, fuse, attach, replace, remove\_picked

---

## button

### DESCRIPTION

"button" can be used to redefine what the mouse buttons do.

### USAGE

```
button <button>,<modifier>,<action>
```

### PYMOL API

```
cmd.button( string button, string modifier, string action )
```

## NOTES

```
button:      L, M, R
modifiers:   None, Shft, Ctrl, CtSh
actions:     Rota, Move, MovZ, Clip, RotZ, ClpN, ClpF
             lb, mb, rb, +lb, +lbX, -lbX, +mb, +rb,
             PkAt, PkBd, RotF, TorF, MovF, Orig
```

Switching from visualization to editing mode will redefine the buttons, so do not use the built-in switch if you want to preserve your custom configuration.

---

# cartoon

## DESCRIPTION

"cartoon" changes the default cartoon for a set of atoms.

## USAGE

```
cartoon type, (selection)
```

```
type = skip | automatic | loop | rectangle | oval | tube | arrow | dumbbell
```

## PYMOL API

```
cmd.cartoon(string type, string selection )
```

## EXAMPLES

```
cartoon rectangle,(chain A)
cartoon skip,(resi 145:156)
```

## NOTES

the "automatic" mode utilizes ribbons according to the information in the PDB HELIX and SHEET records.

---

# cd

## DESCRIPTION

"cd" changes the current working directory.

## USAGE

```
cd <path>
```

## SEE ALSO

```
pwd, ls, system
```

---

# clip

## DESCRIPTION

"clip" alters the near and far clipping planes according to

## USAGE

```
clip {near|far|move|slab}, distance
```

## EXAMPLES

```
clip near, -5 # moves near plane away from you by 5 A
clip far, 10 # moves far plane towards you by 10 A
clip slab, 20 # sets slab thickness to 20 A
clip move, -5 # moves the slab away from you by 5 A
```

## PYMOL API

```
cmd.clip( string mode, float distance )
```

## SEE ALSO

zoom, reset

---

# cls

## DESCRIPTION

"cls" clears the output buffer.

## USAGE

```
cls
```

---

# color

## DESCRIPTION

"color" changes the color of an object or an atom selection.

## USAGE

```
color color-name
color color-name, object-name
color color-name, (selection)
```

## PYMOL API

```
cmd.color( string color, string color-name )
```

## EXAMPLES

color yellow, (name C\*)

---

## commands

### COMMANDS

INPUT/OUTPUT	load	save	delete	quit		
VIEW	turn	move	clip	rock		
	show	hide	enable	disable		
	reset	refresh	rebuild			
	zoom	origin	orient			
	view	get_view	set_view			
MOVIES	mplay	mstop	mset	mdo		
	mpng	mmatrix	frame			
	rewind	middle	ending			
	forward	backward				
IMAGING	png	mpng				
RAY TRACING	ray					
MAPS	isomesh	isodot				
DISPLAY	cls	viewport	splash			
SELECTIONS	select	mask				
SETTINGS	set	button				
ATOMS	alter	alter_state				
EDITING	create	replace	remove	h_fill	remove_picked	
	edit	bond	unbond	h_add	fuse	
	undo	redo	protect	cycle_valence	attach	
FITTING	fit	rms	rms_cur	pair_fit		
	intra_fit	intra_rms	intra_rms_cur			
COLORS	color	set_color				
HELP	help	commands				
DISTANCES	dist					
STEREO	stereo					
SYMMETRY	symexp					
SCRIPTS	@	run				
LANGUAGE	alias	extend				

Try "help <command-name>". Also see the following extra topics:

"movies", "keyboard", "mouse", "selections",  
"examples", "launching", "editing", and "api".

---

## copy

### DESCRIPTION

"copy" creates a new object that is an identical copy of an existing object

### USAGE

copy target, source

copy target = source # (DEPRECATED)

PYMOL API

```
cmd.copy(string target,string source)
```

SEE ALSO

create

---

## count\_atoms

DESCRIPTION

"count\_atoms" returns a count of atoms in a selection.

USAGE

```
count (selection)
```

PYMOL API

```
cmd.count(string selection)
```

---

## count\_states

DESCRIPTION

"count\_states" is an API-only function which returns the number of states in the selection.

PYMOL API

```
cmd.count_states(string selection="(all)")
```

SEE ALSO

frame

---

## create

DESCRIPTION

"create" creates a new molecule object from a selection. It can also be used to create states in an existing object.

NOTE: this command has not yet been thoroughly tested.

USAGE

```
create name, (selection) [,source_state [,target_state ] ]
```

```
create name = (selection) [,source_state [,target_state ] ]
# (DEPRECATED)

name = object to create (or modify)
selection = atoms to include in the new object
source_state (default: 0 - copy all states)
target_state (default: 0)
```

#### PYMOL API

```
cmd.create(string name, string selection, int state, int target_state)
```

#### NOTES

If the source and target states are zero (default), all states will be copied. Otherwise, only the indicated states will be copied.

#### SEE ALSO

load, copy

---

## cycle\_valence

#### DESCRIPTION

"cycle\_valence" cycles the valence on the currently selected bond.

#### USAGE

```
cycle_valence [ h_fill ]
```

#### PYMOL API

```
cmd.cycle_valence(int h_fill)
```

#### EXAMPLES

```
cycle_valence
cycle_valence 0
```

#### NOTES

If the h\_fill flag is true, hydrogens will be added or removed to satisfy valence requirements.

This function is usually connected to the DELETE key and "CTRL-W".

#### SEE ALSO

remove\_picked, attach, replace, fuse, h\_fill

---

## delete

### DESCRIPTION

"delete" removes an object or a selection.

### USAGE

```
delete name  
delete all # deletes all objects
```

name = name of object or selection

### PYMOL API

```
cmd.delete (string name = object-or-selection-name )
```

### SEE ALSO

remove

---

## deprotect

### DESCRIPTION

"deprotect" reverses the effect of the "protect" command.

### USAGE

```
deprotect (selection)
```

### PYMOL API

```
cmd.deprotect(string selection="(all)")
```

### SEE ALSO

protect, mask, unmask, mouse, editing

---

## deselect

### DESCRIPTION

"deselect" disables any and all visible selections

### USAGE

```
deselect
```

### PYMOL API

```
cmd.deselect()
```

---

## disable

### DESCRIPTION

"disable" disables display of an object and all currently visible representations.

### USAGE

```
disable name  
disable all
```

"name" is the name of an object or a named selection

### PYMOL API

```
cmd.disable( string name )
```

### EXAMPLE

```
disable my_object
```

### SEE ALSO

```
show, hide, enable
```

---

## distance

### DESCRIPTION

"distance" creates a new distance object between two selections. It will display all distances within a cutoff.

### USAGE

```
distance  
distance (selection1), (selection2)  
distance name = (selection1), (selection1) [,cutoff [,mode] ]
```

```
name = name of distance object  
selection1,selection2 = atom selections  
cutoff = maximum distance to display  
mode = 0 (default)
```

### PYMOL API

```
cmd.distance( string name, string selection1, string selection2,  
             string cutoff, string mode )  
returns the average distance between all atoms/frames
```

### NOTES

The distance wizard makes measuring distances easier than using the "dist" command for real-time operations.

"dist" alone will show distances between selections (lb) and (rb) created by left and right button atom picks. CTRL-SHIFT/left-click on the first atom, CTRL-SHIFT/right-click on the second, then run "dist".

---

## do

### DESCRIPTION

"do" makes it possible for python programs to issue simple PyMOL commands as if they were entered on the command line.

### PYMOL API

```
cmd.do( commands )
```

### USAGE (PYTHON)

```
from pymol import cmd
cmd.do("load file.pdb")
```

---

## edit

### DESCRIPTION

"edit" picks an atom or bond for editing.

### USAGE

```
edit (selection) [ ,(selection) ]
```

### PYMOL API

```
cmd.edit( string selection [ ,string selection ] )
```

### NOTES

If only one selection is provided, an atom is picked.  
If two selections are provided, the bond between them is picked (if one exists).

### SEE ALSO

```
unpick, remove_picked, cycle_valence, torsion
```

---

# edit\_keys

## EDITING KEYS

These are defaults, which can be redefined. Note that while entering text on the command line, some of these control keys take on text editing functions instead (CTRL - A, E, and K, and DELETE), so you should clear the command line before trying to edit atoms.

## ATOM REPLACEMENT

CTRL-C	Replace picked atom with carbon	(C)
CTRL-N	Replace picked atom with nitrogen	(N)
CTRL-O	Replace picked atom with oxygen	(O)
CTRL-S	Replace picked atom with sulpher	(S)
CTRL-G	Replace picked atom with hydrogen	(H)
CTRL-F	Replace picked atom with fluorene	(F)
CTRL-L	Replace picked atom with chlorine	(Cl)
CTRL-B	Replace picked atom with bromine	(Br)
CTRL-I	Replace picked atom with iodine	(I)

## ATOM MODIFICATION

CTRL-J	Set charge on picked atom to -1
CTRL-K	Set charge on picked atom to +1
CTRL-D	Remove atom or bond (DELETE works too).
CTRL-Y	Add a hydrogen to the current atom
CTRL-R	Adjust hydrogens on atom/bond to match valence.
CTRL-E	Inverts the picked stereo center, but you must first indicate the constant portions with the (lb) and (rb) selections.
CTRL-T	Connect atoms in the (lb) and (rb) selections.
CTRL-W	Cycle the bond valence on the picked bond.

UNDO and REDO of conformational changes (not atom changes!)

CTRL-Z	undo the previous conformational change. (you can not currently undo atom modifications).
CTRL-A	redo the previous conformational change.

---

# editing

## SUMMARY

PyMOL has a rudimentary, but quite functional molecular structure editing capability. However, you will need to use an external mimizer to "clean-up" your structures after editing. Furthermore, if you are going to modify molecules other than proteins, then you will also need a way of assigning atom types on the fly.

To edit a conformation or structure, you first need to enter editing mode (see Mouse Menu). Then you can pick an atom (CTRL-Middle click) or a bond (CTRL-Right click). Next, you can use the other CTRL-key/click combinations listed on the right hand side of the screen to adjust the attached fragments. For example, CTRL-left click will move fragments about the selected torsion.

Editing structures is done through a series of CTRL key actions applied to the currently selected atom or bonds. See "help edit\_keys" for the exact combinations. To build structures, you usually just replace hydrogens with methyl groups, etc., and then repeat. They are no short-cuts currently available for building common groups, but that is planned for later versions.

#### NOTE

Only "lines" and "sticks" representations can be picked using the mouse, however other representations will not interfere with picking so long as one of these representation is present underneath.

---

## enable

#### DESCRIPTION

"enable" enable display of an object and all currently visible representations.

#### USAGE

```
enable name
enable all
```

name = object or selection name

#### PYMOL API

```
cmd.enable( string object-name )
```

#### EXAMPLE

```
enable my_object
```

#### SEE ALSO

show, hide, disable

---

## ending

#### DESCRIPTION

"ending" goes to the end of the movie.

#### USAGE

```
ending
```

#### PYMOL API

```
cmd.ending()
```

---

## examples

### EXAMPLE ATOM SELECTIONS

```
select bk = ( name ca or name c or name n )
    * can be abbreviated as *
sel bk = (n;ca,c,n)

select hev = ( not hydro )
    * can be abbreviated as *
sel hev = (!h;)

select site = ( byres ( resi 45:52 expand 5 ))
    * can be abbreviated as *
sel site = (b;(i;45:52 x;5))

select combi = ( hev and not site )
    * can be abbreviated as *
sel combi = (hevsite)
```

---

## extend

### DESCRIPTION

"extend" is an API-only function which binds a new external function as a command into the PyMOL scripting language.

### PYMOL API

```
cmd.extend(string name,function function)
```

### PYTHON EXAMPLE

```
def foo(moo=2): print moo
cmd.extend('foo',foo)
```

The following would now be valid within PyMOL:

```
foo
foo 3
foo moo=5
```

### SEE ALSO

alias, api

---

## faster

### RAY TRACING OPTIMIZATION

1. Reduce object complexity to a minimum acceptable level.  
For example, try lowering:  
"cartoon\_sampling"

"ribbon\_sampling", and  
"surface\_quality", as appropriate.

2. Increase "hash\_max" so as to obtain a voxel dimensions of 0.3-0.6. Proper tuning of "hash\_max" can speed up rendering by a factor of 2-5X for non-trivial scenes.

WARNING: memory usage depends on  $\text{hash\_max}^3$ , so avoid pushing into virtual memory. Roughly speaking:

```
hash_max = 80 --> ~9 MB hash + data
hash_max = 160 --> ~72 MB hash + data
hash_max = 240 --> ~243 MB hash + data
```

Avoid utilizing virtual memory for the voxel hash, it will slow things way down.

3. Recompiling with optimizations on usually gives a 25-33% performance boost for ray tracing.

---

## feedback

### DESCRIPTION

"feedback" allows you to control what and how much text is output from PyMOL.

### USAGE

```
feedback action,module,mask
```

action is one of ['set','enable','disable']

module is a space-separated list of strings or simply "all"

mask is a space-separated list of strings or simply "everything"

### NOTES:

"feedback" alone will print a list of the available choices

### PYMOL API

```
cmd.feedback(string action,string module,string mask)
```

### EXAMPLES

```
feedback enable, all , debugging
```

```
feedback disable, selector, warnings actions
```

```
feedback enable, main, blather
```

## find\_pairs

### DESCRIPTION

"find\_pairs" is currently undocumented.

---

## finish\_object

### DESCRIPTION

"finish\_object" is used in cases where many individual states are being loaded and it is advantageous to avoid processing them until all states have been loaded into RAM. This function should always be called after loading an object with the finish flag set to zero.

### PYMOL API

```
cmd.finish(string name)
```

"name" should be the name of the object

---

## fit

### DESCRIPTION

"fit" superimposes the model in the first selection on to the model in the second selection. Only matching atoms in both selections will be used for the fit.

### USAGE

```
fit (selection), (target-selection)
```

### EXAMPLES

```
fit ( mutant and name ca ), ( wildtype and name ca )
```

### SEE ALSO

```
rms, rms_cur, intra_fit, intra_rms, intra_rms_cur
```

---

## flag

### DESCRIPTION

"flag" sets the indicated flag for atoms in the selection and clears the indicated flag for atoms not in the selection. This is primarily useful for passing selection information into Chempy models, which have a 32 bit attribute "flag" which holds

this state information.

#### USAGE

```
flag flag, selection [ ,action ]
```

```
flag flag = selection [ ,action ]      # (DEPRECATED)
```

action can be:

"reset" (default) sets flag on selection, clear it on other atoms

"set" sets the flag for selected atoms, leaves other atoms unchanged

"clear" clear the flag for selected atoms, leaves other atoms unchanged

#### PYMOL API

```
cmd.flag( int number, string selection, string action="reset" )
```

```
cmd.flag( string flag_name, string selection, string action="reset" )
```

#### EXAMPLES

```
flag free, (resi 45 x; 6)
```

#### RESERVATIONS

Flags 0-7 are reserved for molecular modeling \*/

focus 0 = Atoms of Interest (i.e. a ligand in an active site)

free 1 = Free Atoms (free to move subject to a force-field)

restrain 2 = Restrained Atoms (typically harmonically constrained)

fix 3 = Fixed Atoms (no movement allowed)

ignore 4 = Atoms which should not be part of any simulation

Flags 8-15 are free for end users to manipulate

Flags 16-23 are reserved for external GUIs and linked applications

Flags 24-31 are reserved for PyMOL internal usage

exfoliate 24 = Remove surface from atoms when surfacing

ignore 25 = Ignore atoms altogether when surfacing

---

## forward

#### DESCRIPTION

"forward" moves the movie one frame forward.

#### USAGE

```
forward
```

#### PYMOL API

```
cmd.forward()
```

#### SEE ALSO

```
mset, backward, rewind
```

---

## fragment

### DESCRIPTION

"fragment" retrieves a 3D structure from the fragment library, which is currently pretty meager (just amino acids).

### USAGE

```
fragment name
```

---

## frame

### DESCRIPTION

"frame" sets the viewer to the indicated movie frame.

### USAGE

```
frame frame-number
```

### PYMOL API

```
cmd.frame( int frame_number )
```

### NOTES

Frame numbers are 1-based

### SEE ALSO

```
count_states
```

---

## full\_screen

### DESCRIPTION

"full\_screen" enables or disables PyMOL's full\_screen mode. This is only functions well on PC's.

### USAGE

```
full_screen on  
full_screen off
```

---

# **fuse**

## DESCRIPTION

"fuse" joins two objects into one by forming a bond. A copy of the object containing the first atom is moved so as to form an approximately reasonable bond with the second, and is then merged with the first object.

## USAGE

```
fuse (selection1), (selection2)
```

## PYMOL API

```
cmd.fuse( string selection1="(lb)", string selection2="(lb)" )
```

## NOTES

Each selection must include a single atom in each object. The atoms can both be hydrogens, in which case they are eliminated, or they can both be non-hydrogens, in which case a bond is formed between the two atoms.

## SEE ALSO

```
bond, unbond, attach, replace, fuse, remove_picked
```

---

# **get\_area**

PRE-RELEASE functionality - API will change

---

# **get\_extent**

## DESCRIPTION

"get\_extent" returns the minimum and maximum XYZ coordinates of a selection as an array:

```
[ [ min-X , min-Y , min-Z ], [ max-X, max-Y , max-Z ] ]
```

## PYMOL API

```
cmd.get_extent(string selection="(all)", state=0 )
```

---

## get\_frame

### DESCRIPTION

"get\_frame" returns the current frame index (1-based)

### PYMOL API

Frames refers to sequences of images in a movie. Sequential frames may contain identical molecular states, they may have one-to-one correspondance to molecular states (default), or they may have an arbitrary relationship, specific using the "mset" command.

### SEE ALSO

get\_state

---

## get\_model

### DESCRIPTION

"get\_model" returns a Chempy "Indexed" format model from a selection.

### PYMOL API

```
cmd.get_model(string selection [,int state] )
```

---

## get\_names

### DESCRIPTION

"get\_names" returns a list of object and/or selection names.

### PYMOL API

```
cmd.get_names( [string: "objects"|"selections"|"all"] )
```

### NOTES

The default behavior is to return only object names.

### SEE ALSO

get\_type, count\_atoms, count\_states

---

## get\_povray

### DESCRIPTION

"get\_povray" returns a tuple corresponding to strings for a PovRay input file.

### PYMOL API

```
cmd.get_povray()
```

---

## get\_state

### DESCRIPTION

"get\_state" returns the current state index (1-based)

### PYMOL API

```
cmd.get_state()
```

### NOTES

States refer to different geometric configurations which an object can have. By default, states and movie frames have a one-to-one relationship. States can be visited in an arbitrary order to create frames. The "mset" command allows you to build a relationship between states and frames.

### SEE ALSO

```
get_frame
```

---

## get\_type

### DESCRIPTION

"get\_type" returns a string describing the named object or selection or the string "nonexistent" if the name is unknown.

### PYMOL API

```
cmd.get_type(string object-name)
```

### NOTES

Possible return values are

```
"object:molecule"  
"object:map"  
"object:mesh"  
"object:distance"  
"selection"
```

SEE ALSO

`get_names`

---

## **get\_view**

DESCRIPTION

"get\_view" returns and optionally prints out the current view information in a format which can be embedded into a command script and used in subsequent calls to "set\_view"

USAGE

`get_view`

PYMOL API

`cmd.get_view(output=1)`

API USAGE

`cmd.get_view(0) # zero option suppresses output`

---

## **h\_add**

DESCRIPTION

"h\_add" uses a primitive algorithm to add hydrogens onto a molecule.

USAGE

`h_add (selection)`

PYMOL API

`cmd.h_add( string selection="(all)" )`

SEE ALSO

`h_fill`

---

## h\_fill

### DESCRIPTION

"h\_fill" removes and replaces hydrogens on the atom or bond picked for editing.

### USAGE

```
h_fill
```

### PYMOL API

```
cmd.h_fill()
```

### NOTES

This is useful for fixing hydrogens after changing bond valences.

### SEE ALSO

edit, cycle\_valences, h\_add

---

## help

### DESCRIPTION

"help" prints out the online help for a given command.

### USAGE

```
help command
```

---

## hide

### DESCRIPTION

"hide" turns of atom and bond representations.

The available representations are:

```
lines      spheres  mesh      ribbon    cartoon
sticks     dots      surface   labels
nonbonded  nb_spheres
```

### USAGE

```
hide representation [,object]
hide representation [,(selection)]
hide (selection)
```

### PYMOL API

*h\_fill*

```
cmd.hide( string representation="", string selection="")
```

#### EXAMPLES

```
hide lines,all  
hide ribbon
```

#### SEE ALSO

```
show, enable, disable
```

---

## id\_atom

#### DESCRIPTION

"id\_atom" returns the original source id of a single atom, or raises an exception if the atom does not exist or if the selection corresponds to multiple atoms.

#### PYMOL API

```
list = cmd.id_atom(string selection)
```

---

## identify

#### DESCRIPTION

"identify" returns a list of atom IDs corresponding to the ID code of atoms in the selection.

#### PYMOL API

```
list = cmd.identify(string selection="(all)",int mode=0)
```

#### NOTES

mode 0 (default): only return a list of identifier  
mode 1: return a list of tuples of the object name and the identifier

---

## index

#### DESCRIPTION

"index" returns a list of tuples corresponding to the object name and index of the atoms in the selection.

#### PYMOL API

```
list = cmd.index(string selection="(all)")
```

## NOTE

Atom indices are fragile and will change as atoms are added or deleted. Whenever possible, use integral atom identifiers instead of indices.

---

# indicate

## DESCRIPTION

"indicate" shows a visual representation of an atom selection.

## USAGE

```
indicate (selection)
```

## PYMOL API

```
cmd.count(string selection)
```

---

# intra\_fit

## DESCRIPTION

"intra\_fit" fits all states of an object to an atom selection in the specified state. It returns the rms values to python as an array.

## USAGE

```
intra_fit (selection),state
```

## PYMOL API

```
cmd.intra_fit( string selection, int state )
```

## EXAMPLES

```
intra_fit ( name ca )
```

## PYTHON EXAMPLE

```
from pymol import cmd  
rms = cmd.intra_fit("(name ca)",1)
```

## SEE ALSO

```
fit, rms, rms_cur, intra_rms, intra_rms_cur, pair_fit
```

---

## intra\_rms

### DESCRIPTION

"intra\_rms" calculates rms fit values for all states of an object over an atom selection relative to the indicated state. Coordinates are left unchanged. The rms values are returned as a python array.

### PYMOL API

```
cmd.intra_rms( string selection, int state)
```

### PYTHON EXAMPLE

```
from pymol import cmd
rms = cmd.intra_rms("(name ca)",1)
```

### SEE ALSO

```
fit, rms, rms_cur, intra_fit, intra_rms_cur, pair_fit
```

---

## intra\_rms\_cur

### DESCRIPTION

"intra\_rms\_cur" calculates rms values for all states of an object over an atom selection relative to the indicated state without performing any fitting. The rms values are returned as a python array.

### PYMOL API

```
cmd.intra_rms_cur( string selection, int state)
```

### PYTHON EXAMPLE

```
from pymol import cmd
rms = cmd.intra_rms_cur("(name ca)",1)
```

### SEE ALSO

```
fit, rms, rms_cur, intra_fit, intra_rms, pair_fit
```

---

## invert

### DESCRIPTION

"invert" inverts the stereo-chemistry of the atom currently picked for editing (pk1). Two additional atom selections must be provided in order to indicate which atoms remain stationary during the inversion process.

## USAGE

```
invert (selection1),(selection2)
```

## PYMOL API

```
cmd.api( string selection1="(lb)", string selection2="(lb)" )
```

## NOTE

The invert function is usually bound to CTRL-E in editing mode.

The default selections are (lb) and (rb), meaning that you can pick the atom to invert with CTRL-middle click and then pick the stationary atoms with CTRL-SHIFT/left-click and CTRL-SHIFT/right-click, then hit CTRL-E to invert the atom.

---

# isodot

## DESCRIPTION

"isodot" creates a dot isosurface object from a map object.

## USAGE

```
isodot name = map, level [,(selection) [,buffer [, state ] ] ]
```

"map" is the name of the map object to use.

"level" is the contour level.

"selection" is an atom selection about which to display the mesh with an additional "buffer" (if provided).

## NOTES

If the dot isosurface object already exists, then the new dots will be appended onto the object as a new state.

## SEE ALSO

load, isomesh

---

# isomesh

## DESCRIPTION

"isomesh" creates a mesh isosurface object from a map object.

## USAGE

```
isomesh name, map, level [,(selection) [,buffer [,state [,carve ]]]]
```

"name" is the name for the new mesh isosurface object.

"map" is the name of the map object to use for computing the mesh.

"level" is the contour level.

"selection" is an atom selection about which to display the mesh with an additional "buffer" (if provided).

"state" is the state into which the object should be loaded.

"carve" is a radius about each atom in the selection for which to include density. If "carve" is not provided, then the whole brick is displayed.

#### NOTES

If the mesh object already exists, then the new mesh will be appended onto the object as a new state (unless you indicate a state).

#### SEE ALSO

isodot, load

---

## iterate

#### DESCRIPTION

"iterate" iterates over an expression with a separate name space for each atom. However, unlike the "alter" command, atomic properties can not be altered. Thus, "iterate" is more efficient than "alter".

It can be used to perform operations and aggregations using atomic selections, and store the results in any global object, such as the predefined "stored" object.

The local namespace for "iterate" contains the following names

```
name, resn, resi, chain, alt, elem,  
q, b, segi, and type (ATOM,HETATM),  
partial_charge, formal_charge,  
text_type, numeric_type, ID
```

All strings in the expression must be explicitly quoted. This operation typically takes a second per thousand atoms.

#### USAGE

```
iterate (selection),expression
```

#### EXAMPLES

```
stored.net_charge = 0  
iterate (all),stored.net_charge = stored.net_charge + partial_charge
```

```
stored.names = []  
iterate (all),stored.names.append(name)
```

SEE ALSO

`iterate_state`, `atler`, `alter_state`

---

## iterate\_state

DESCRIPTION

"`iterate_state`" is to "`alter_state`" as "`iterate`" is to "`alter`"

USAGE

```
iterate_state state,(selection),expression
```

EXAMPLES

```
stored.sum_x = 0.0
iterate 1,(all),stored.sum_x = stored.sum_x + x
```

SEE ALSO

`iterate`, `alter`, `alter_state`

---

## keyboard

KEYBOARD COMMANDS and MODIFIERS

ESC            Toggle onscreen text.  
INSERT        Toggle rocking.

LEFT ARROW, RIGHT ARROW    Go backward or forward one frame, or when editing, go forward or back one character.  
HOME, END        Go to the beginning or end of a movie.

Command Entry Field in the Internal GUI (black window)

TAB            Complete command or filename (like in tcsh or bash).  
CTRL-A        Go to the beginning of the line.  
CTRL-E        Go to the end of the line.  
CTRL-K        Delete through to the end of the line.

Command Entry Field on the External GUI (gray window).

CTRL-C        These operating system-provided cut and paste functions  
CTRL-V        will only work in the external GUI command line.

EDITING

type "`help edit_keys`" for keyboard shortcuts used in editing.

---

# label

## DESCRIPTION

"label" labels one or more atoms properties over a selection using the python evaluator with a separate name space for each atom. The symbols defined in the name space are:

```
name, resn, resi, chain, q, b, segi, type (ATOM,HETATM)
formal_charge, partial_charge, numeric_type, text_type
```

All strings in the expression must be explicitly quoted. This operation typically takes several seconds per thousand atoms altered.

To clear labels, simply omit the expression or set it to ''.

## USAGE

```
label (selection),expression
```

## EXAMPLES

```
label (chain A),chain
label (n:ca,"%s-%s" % (resn,resi))
label (resi 200,"%1.3f" % partial_charge)
```

---

# launching

## PyMOL COMMAND LINE OPTIONS

```
pymol.com [-ciqstwx] <file.xxx> [-p <file.py> ] ...
```

```
-c  Command line mode, no GUI. For batch operations.
-i  Disable the internal OpenGL GUI (object list, menus, etc.)
-x  Disable the external GUI module.
-t  Use Tcl/Tk based external GUI module (pmg_tk).
-w  Use wxPython based external GUI module (pmg_wx).
-q  Quiet launch. Suppress splash screen.
-p  Listen for commands on standard input.
-e  Start in full-screen mode
```

```
-f <# line> Controls display of commands and feedback in OpenGL (0=off).
-r <file.py>[,global|local|module] Run a python program on startup.
-l <file.py>[,global|local|module] Spawn a python program in new thread.
-d <string> Run pymol command string upon startup.
-u <script> Load and append to this PyMOL script or program file.
-s <script> Save commands to this PyMOL script or program file.
```

<file> can have one of the following extensions, and all files provided will be loaded or run after PyMOL starts.

```
.pml          PyMOL command script to be run on startup
.py, .pym, .pyc Python program to be run on startup
.pdb          Protein Data Bank format file to be loaded on startup
.mmod        Macromodel format to be loaded on startup
.mol         MDL MOL file to be loaded on startup
```

.xplor	X-PLOR Map file (ASCII) to be loaded on startup
.ccp4	CCP4 map file (BINARY) to be loaded on startup
.pkl	Pickled ChemPy Model (class "chempy.model.Indexed")
.r3d	Raster3D Object
.ccl, .cc2	ChemDraw 3D cartesian coordinate file

---

## load

### DESCRIPTION

"load" reads several file formats. The file extension is used to determine the format. PDB files must end in ".pdb", MOL files must end in ".mol", Macromodel files must end in ".mmod", XPLOR maps must end in ".xplor", CCP4 maps must end in ".ccp4", Raster3D input (Molscrip output) must end in ".r3d".

Pickled ChemPy models with a ".pkl" can also be directly read.

If an object is specified, then the file is load into that object. Otherwise, an object is created with the same name as the file prefix.

### USAGE

```
load filename [,object [,state [,format [,finish [,discrete ]]]]]
```

### PYMOL API

```
cmd.load( filename [,object [,state [,format [,finish [,discrete ]]]]]
```

### NOTES

You can override the file extension by giving a format string:

```
'pdb' : PDB, 'mmod' : Macromodel, 'xyz' : Tinker, 'ccl' : ChemDraw3D  
'mol' : MDL MOL-file, 'sdf' : MDL SD-file  
'xplor' : X-PLOR/CNS map, 'ccp4' : CCP4 map,  
'callback' : PyMOL Callback object (PyOpenGL)  
'cgo' : compressed graphics object (list of floats)
```

### SEE ALSO

```
save
```

---

## load\_brick

Temporary routine for GAMESS-UK project.

---

## load\_callback

### DESCRIPTION

"load\_callback" is used to load a generic Python callback object. These objects are called every time the screen is updated and can be used to trigger OpenGL rendering calls (such as with PyOpenGL).

### PYMOL API

```
cmd.load_callback(object,name,state,finish,discrete)
```

---

## load\_cgo

### DESCRIPTION

"load\_cgo" is used to load a compiled graphics object, which is actually a list of floating point numbers built using the constants in the \$PYMOL\_PATH/modules/pymol/cgo.py file.

### PYMOL API

```
cmd.load_cgo(object,name,state,finish,discrete)
```

---

## load\_map

Temporary routine for the Phenix project.

---

## load\_model

### DESCRIPTION

"load\_model" reads a ChemPy model into an object

### PYMOL API

```
cmd.load_model(model, object [,state [,finish [,discrete ]]])
```

---

## load\_object

### DESCRIPTION

"load\_object" is a general developer function for loading Python objects into PyMOL.

## PYMOL API

```
cmd.load_object(type,object,name,state=0,finish=1,discrete=0)
```

NOTE type is one one of the numeric cmd.loadable types

---

# ls

## DESCRIPTION

List contents of the current working directory.

## USAGE

```
ls [pattern]
dir [pattern]
```

## EXAMPLES

```
ls
ls *.pml
```

## SEE ALSO

cd, pwd, system

---

# map\_set\_border

## DESCRIPTION

"map\_set\_border" is a function (reqd by PDA) which allows you to set the level on the edge points of a map

## USAGE

```
map_set_border <name>,<level>
```

## NOTES

unsupported.

## SEE ALSO

load

---

## mappend

### DESCRIPTION

### USAGE

```
mappend frame : command
```

### PYMOL API

### EXAMPLE

### NOTES

### SEE ALSO

```
mset, mplay, mstop
```

---

## mask

### DESCRIPTION

"mask" makes it impossible to select the indicated atoms using the mouse. This is useful when you are working with one molecule in front of another and wish to avoid accidentally selecting atoms in the background.

### USAGE

```
mask (selection)
```

### PYMOL API

```
cmd.mask( string selection="(all)" )
```

### SEE ALSO

```
unmask, protect, deprotect, mouse
```

---

## math

This module is always available. It provides access to the mathematical functions defined by the C standard.

---

# mclear

## DESCRIPTION

"mclear" clears the movie frame image cache.

## USAGE

```
mclear
```

## PYMOL API

```
cmd.mclear()
```

---

# mdo

## DESCRIPTION

"mdo" sets up a command to be executed upon entry into the specified frame of the movie. These commands are usually created by a PyMOL utility program (such as util.mrock). Command can actually contain several commands separated by semicolons ';'.

## USAGE

```
mdo frame : command
```

## PYMOL API

```
cmd.mdo( int frame, string command )
```

## EXAMPLE

```
// Creates a single frame movie involving a rotation about X and Y

load test.pdb
mset 1
mdo 1, turn x,5; turn y,5;
mplay
```

## NOTES

The "mset" command must first be used to define the movie before "mdo" statements will have any effect. Redefinition of the movie clears any existing mdo statements.

## SEE ALSO

```
mset, mplay, mstop
```

---

## mem

### DESCRIPTION

"mem" Dumps current memory state to standard output. This is a debugging feature, not an official part of the API.

---

## meter\_reset

### DESCRIPTION

"meter\_reset" resets the frames per second counter

### USAGE

```
meter_reset
```

---

## middle

### DESCRIPTION

"middle" goes to the middle of the movie.

### USAGE

```
middle
```

### PYMOL API

```
cmd.middle()
```

---

## mmatrix

### DESCRIPTION

"mmatrix" sets up a matrix to be used for the first frame of the movie.

### USAGE

```
mmatrix {clear|store|recall}
```

### PYMOL API

```
cmd.mmatrix( string action )
```

### EXAMPLES

```
mmatrix store
```

---

## mouse

### MOUSE CONTROLS

The configuration can be changed using the "Mouse" menu. The current configuration is described on screen with a small matrix on the lower right hand corner, using the following abbreviations:

#### Buttons (Horizontal Axis)

L = left mouse click  
M = middle mouse click  
R = right mouse click

#### Modifiers (Vertical axis on the matrix)

None = no keys held down while clicking  
Shft = hold SHIFT down while clicking  
Ctrl = hold CTRL down while clicking  
CtSh = hold both SHIFT and CTRL down while clicking

#### Visualization Functions

Rota = Rotates camera about X, Y, and Z axes  
RotZ = Rotates camera about the Z axis  
Move = Translates along the X and Y axes  
MovZ = Translates along Z axis  
Clip = Y motion moves the near clipping plane while  
PkAt = Pick an atom  
PkBd = Pick a bond  
Orig = Move origin to selected atom  
+lb = Add an atom into the (lb) selection  
lb = Define the (lb) selection with the indicated atom.  
rb = Define the (rb) selection with the indicated atom.

#### Editing Functions

RotF = Rotate fragment  
MovF = Move fragment  
TorF = Torsion fragment

---

## move

### DESCRIPTION

"move" translates the world about one of the three primary axes.

### USAGE

move axis,distance

### EXAMPLES

move x,3

```
move y,-1
```

PYMOL API

```
cmd.move( string axis, float distance )
```

SEE ALSO

```
turn
```

---

## movies

MOVIES

To create a movie, simply load multiple coordinate files into the same object. This can be accomplished at the command line, using script files, or by writing PyMOL API-based programs.

The commands:

```
load frame001.pdb,mov
load frame002.pdb,mov
```

will create a two frame movie. So will the following program:

```
from pymol import cmd

for a in ( "frame001.pdb","frame002.pdb" ):
    cmd.load(a,"mov")
```

which can be executed at the command line using the "run" command.

Python built-in glob module can be useful for loading movies.

```
from pymol import cmd
import glob
for a in ( glob.glob("frame*.pdb") ):
    cmd.load(a,"mov")
```

NOTE

Because PyMOL stores all movie frames in memory, there is a practical limit to the number of atoms in all coordinate files. 160 MB free RAM enables 500,000 atoms with line representations. Complex representations require significantly more memory.

---

## mplay

DESCRIPTION

"mplay" starts the movie.

USAGE

mplay

PYMOL API

```
cmd.mplay()
```

SEE ALSO

mstop, mset, mdo, mclear, mmatrix

---

## mpng

DESCRIPTION

"mpng" writes a series of numbered movie frames to png files with the specified prefix. If the "ray\_trace\_frames" variable is non-zero, these frames will be ray-traced. This operation can take several hours for a long movie.

Be sure to disable "cache\_frames" when issuing this operation on a long movie (typically >100 frames to avoid running out of memory).

USAGE

```
mpng prefix
```

PYMOL API

```
cmd.mpng( string prefix )
```

---

## mset

DESCRIPTION

"mset" sets up a relationship between molecular states and movie frames. This makes it possible to control which states are shown in which frame.

USAGE

```
mset specification
```

PYMOL API

```
cmd.mset( string specification )
```

EXAMPLES

```
mset 1          // simplest case, one state -> one frame
mset 1 x10      // ten frames, all corresponding to state 1
mset 1 x30 1 -15 15 x30 15 -1
// more realistic example:
// the first thirty frames are state 1
// the next 15 frames pass through states 1-15
```

```
// the next 30 frames are of state 15
// the next 15 frames iterate back to state 1
```

SEE ALSO

`mdo`, `mplay`, `mclear`

---

## mstop

DESCRIPTION

"mstop" stops the movie.

USAGE

`mstop`

PYMOL API

`cmd.mstop()`

SEE ALSO

`mplay`, `mset`, `mdo`, `mclear`, `mmatrix`

---

## operator

Operator interface.

This module exports a set of functions implemented in C corresponding to the intrinsic operators of Python. For example, `operator.add(x, y)` is equivalent to the expression `x+y`. The function names are those used for special class methods; variants without leading and trailing `'__'` are also provided for convenience.

---

## orient

DESCRIPTION

"orient" aligns the principal components of the atoms in the selection with the XYZ axes. The function is similar to the `orient` command in X-PLOR.

USAGE

`orient object-or-selection`  
`orient (selection)`

PYMOL API

```
cmd.orient( string object-or-selection )
```

SEE ALSO

```
zoom, origin, reset
```

---

## origin

DESCRIPTION

"origin" sets the center of rotation about a selection

USAGE

```
origin object-or-selection  
origin (selection)
```

PYMOL API

```
cmd.origin( string object-or-selection )
```

SEE ALSO

```
zoom, orient, reset
```

---

## pair\_fit

DESCRIPTION

"pair\_fit" fits a set of atom pairs between two models. Each atom in each pair must be specified individually, which can be tedious to enter manually. Script files are recommended when using this command.

USAGE

```
pair_fit (selection), (selection), [ (selection), (selection) [ ... ] ]
```

SEE ALSO

```
fit, rms, rms_cur, intra_fit, intra_rms, intra_rms_cur
```

---

## png

DESCRIPTION

"png" writes a png format image file of the current image to disk.

USAGE

*origin*

png filename

PYMOL API

```
cmd.png( string file )
```

---

## povray

### DESCRIPTION

PovRay: Persistence of Vision Support Information (EXPERIMENTAL)

The built-in ray-tracer (technically a ray-caster) is as fast or faster than PovRay for most figures (provided that hash\_max is tuned for your system). However, PovRay blows PyMOL away when it comes to rendering images without using lots of RAM, and with PovRay you get the ability use perspective, textures, reflections, infinite objects, and a superior lighting model.

PovRay does not include interpolated colors within triangles (!!), so you must patch the source code in order to obtain this basic functionality required for rendering molecular surfaces. Details can be found on the DINO website.

<http://www.biozentrum.unibas.ch/~xray/dino>

To use PovRay, you must be running under Unix and have x-povray in your path. Unfortunately, the developers of PovRay do not share PyMOL's open-source philosophy, so you will need to download, configure, patch (for smooth\_color\_triangles), and install it yourself. Despite being free and open-source, PovRay's license prevents it from being modified or conveniently combined with PyMOL, and thus it serves as a textbook example for why open-source licenses should be wholly non-restrictive.

Assuming that PovRay is built and in your path...

```
ray renderer=1 # will use PovRay instead of the built-in engine

set ray_default_renderer=1 # changes the default renderer to PovRay
ray # will now use PovRay by default

cmd.get_povray() # will give you a tuple of PovRay input strings
# which you can manipulate from Python

set smooth_color_triangles = 1 # is required in order to get decent
surfaces in PovRay. You must patch PovRay first.
```

---

## protect

### DESCRIPTION

"protect" protects a set of atoms from transformations performed using the editing features. This is most useful when you are modifying an internal portion of a chain or cycle and do not wish to affect the rest of the molecule.

### USAGE

```
protect (selection)
```

### PYMOL API

```
cmd.protect(string selection)
```

### SEE ALSO

deprotect, mask, unmask, mouse, editing

---

## push\_undo

### DESCRIPTION

"push\_undo" stores the currently conformations of objects in the selection onto their individual kill rings.

### USAGE

```
push_undo (all)
```

### SEE ALSO

undo, redo

---

## pwd

### DESCRIPTION

Print current working directory.

### USAGE

```
pwd
```

### SEE ALSO

cd, ls, system

---

# quit

## DESCRIPTION

"quit" terminates the program.

## USAGE

```
quit
```

## PYMOL API

```
cmd.quit()
```

---

# ray

## DESCRIPTION

"ray" creates a ray-traced image of the current frame. This can take some time (up to several minutes, depending on image complexity).

## USAGE

```
ray [width,height [,renderer]]
```

## EXAMPLES

```
ray
ray 1024,768
ray renderer=0
```

## PYMOL API

```
cmd.ray(int width,int height,int renderer=-1)
```

## NOTES

```
renderer = -1 is default (use value in ray_default_renderer)
renderer = 0 uses PyMOL's internal renderer
renderer = 1 uses PovRay's renderer. This is Unix-only
and you must have "x-povray" in your path. It utilizes two
two temporary files: "tmp_pymol.pov" and "tmp_pymol.png".
```

## SEE ALSO

```
"help faster" for optimization tips with the builtin renderer.
"help povray" for how to use PovRay instead of PyMOL's built-in
ray-tracing engine.
```

---

## read\_mmodstr

### DESCRIPTION

"read\_mmodstr" reads a macromodel format structure from a Python string.

---

## read\_molstr

### DESCRIPTION

"read\_molstr" reads an MDL MOL format file as a string

### PYMOL API ONLY

```
cmd.read_molstr( string molstr, string name, int state=0,  
                int finish=1, int discrete=1 )
```

### NOTES

"state" is a 1-based state index for the object, or 0 to append.

"finish" is a flag (0 or 1) which can be set to zero to improve performance when loading large numbers of objects, but you must call "finish\_object" when you are done.

"discrete" is a flag (0 or 1) which tells PyMOL that there will be no overlapping atoms in the file being loaded. "discrete" objects save memory but can not be edited.

---

## read\_pdbstr

### DESCRIPTION

"read\_pdbstr" is an API-only function which reads a pdb file from a Python string. This feature can be used to load or update structures into PyMOL without involving any temporary files.

### PYMOL API ONLY

```
cmd.read_pdbstr( string pdb-content, string object name  
                [ ,int state [ ,int finish [ ,int discrete ] ] ] )
```

### NOTES

"state" is a 1-based state index for the object.

"finish" is a flag (0 or 1) which can be set to zero to improve performance when loading large numbers of objects, but you must call "finish\_object" when you are done.

"discrete" is a flag (0 or 1) which tells PyMOL that there will be no overlapping atoms in the PDB files being loaded. "discrete"

objects save memory but can not be edited.

---

## rebuild

### DESCRIPTION

"rebuild" forces PyMOL to recreate geometric objects in case any of them have gone out of sync.

### USAGE

```
rebuild [selection [, representation ]]
```

### PYMOL API

```
cmd.rebuild(string selection = 'all', string representation = 'everything')
```

### SEE ALSO

refresh

---

## recolor

### DESCRIPTION

"rebuild" forces PyMOL to reapply colors to geometric objects in case any of them have gone out of sync.

### USAGE

```
recolor [selection [, representation ]]
```

### PYMOL API

```
cmd.recolor(string selection = 'all', string representation = 'everything')
```

### SEE ALSO

recolor

---

## redo

### DESCRIPTION

"redo" reapplies the conformational change of the object currently being edited.

### USAGE

```
redo
```

*rebuild*

SEE ALSO

undo, push\_undo

---

## refresh

DESCRIPTION

"refresh" causes the scene to be refresh as soon as it is safe to do so.

USAGE

refresh

PYMOL API

cmd.refresh()

SEE ALSO

rebuild

---

## release

RELEASE NOTES

PyMOL is a free, open, and expandable molecular graphics system written by a computational scientist to enable molecular modeling from directly within Python. It will be of most benefit to hybrid scientist/developers in the fields of structural biology, computational chemistry, and informatics who seek an open and unrestricted visualization tool for interfacing with their own programs. PyMOL will also be of benefit to advanced non-developers familiar with similar programs such as Midas, O, Grasp, X-PLOR and CNS.

Due to PyMOL's current "user-unfriendliness", this release is most appropriate for those who prefer to use text commands and scripts, and for developers who want to integrate PyMOL's visualization and molecular editing capabilities with their own work.

PyMOL currently includes a diverse command language, a powerful application programmers interface (API), and a variety of mouse and keyboard driven functionality for viewing, animation, rendering, and molecular editing. A partial manual is now available on the web.

Two external GUI development options are supported for PyMOL: "Tkinter" and "wxPython". Developers can take their pick. I am committed to insuring that PyMOL will work with both of them, but it is unlikely that I will have time to develop a complete external GUI myself any time soon using either toolkit.

Note that only Tkinter is supported under Windows with the default PyMOL and Python distributions, so for maximum ease of installation under Windows, stick with Tkinter (Tcl/Tk). For this reason, the Tkinter-based GUI is going to be the default GUI for standard PyMOL despite its drawbacks.

Warren L. DeLano (5/1/2001), warren@delanoscientific.com

---

## remove

### DESCRIPTION

"remove" eliminates a selection of atoms from models.

### USAGE

```
remove (selection)
```

### PYMOL API

```
cmd.remove( string selection )
```

### EXAMPLES

```
remove ( resi 124 )
```

### SEE ALSO

```
delete
```

---

## remove\_picked

### DESCRIPTION

"remove\_picked" removes the atom or bond currently picked for editing.

### USAGE

```
remove_picked [hydrogens]
```

### PYMOL API

```
cmd.remove_picked(integer hydrogens=1)
```

### NOTES

This function is usually connected to the DELETE key and "CTRL-D".

By default, attached hydrogens will also be deleted unless hydrogen-flag is zero.

### SEE ALSO

*remove*

attach, replace

---

## rename

### DESCRIPTION

"rename" creates new atom names which are unique within residues.

### USAGE

#### CURRENT

```
rename object-name [ ,force ]
```

force = 0 or 1 (default: 0)

#### PROPOSED

```
rename object-or-selection,force
```

### PYMOL API

#### CURRENT

```
cmd.rename( string object-name, int force )
```

#### PROPOSED

```
cmd.rename( string object-or-selection, int force )
```

### NOTES

To regenerate only some atom names in a molecule, first clear them with an "alter (sele),name=" command, then use "rename"

### SEE ALSO

alter

---

## replace

### DESCRIPTION

"replace" replaces the picked atom with a new atom.

### USAGE

```
replace name, geometry, valence
```

### PYMOL API

```
cmd.replace(string name, int geometry,int valence )
```

### NOTES

Immature functionality. See code for details.

SEE ALSO

remove, attach, fuse, bond, unbond

---

## reset

DESCRIPTION

"reset" restores the rotation matrix to identity, sets the origin to the center of mass (approx.) and zooms the window and clipping planes to cover all objects.

USAGE

reset

PYMOL API

cmd.reset ( )

---

## rewind

DESCRIPTION

"rewind" goes to the beginning of the movie.

USAGE

rewind

PYMOL API

cmd.rewind()

---

## rms

DESCRIPTION

"rms" computes a RMS fit between two atom selections, but does not tranform the models after performing the fit.

USAGE

rms (selection), (target-selection)

EXAMPLES

fit ( mutant and name ca ), ( wildtype and name ca )

SEE ALSO

*reset*

`fit, rms_cur, intra_fit, intra_rms, intra_rms_cur, pair_fit`

---

## **rms\_cur**

### DESCRIPTION

"rms\_cur" computes the RMS difference between two atom selections without performing any fitting.

### USAGE

```
rms_cur (selection), (selection)
```

### SEE ALSO

`fit, rms, intra_fit, intra_rms, intra_rms_cur, pair_fit`

---

## **rock**

### DESCRIPTION

"rock" toggles Y axis rocking.

### USAGE

```
rock
```

### PYMOL API

```
cmd.rock()
```

---

## **run**

### DESCRIPTION

"run" executes an external Python script in a local name space, the global namespace, or in its own namespace (as a module).

### USAGE

```
run python-script [, (local | global | module) ]
```

### PYMOL API

Not directly available. Instead, use `cmd.do("run ...")`.

### NOTES

The default mode for run is "global".

Due to an idiosyncrasy in Pickle, you can not pickle objects directly created at the main level in a script run as "module", (because the pickled object becomes dependent on that module).  
Workaround: delegate construction to an imported module.

---

## save

### DESCRIPTION

"save" writes selected atoms to a file. The file format is autodetected if the extension is ".pdb" or ".pkl"

### USAGE

```
save file [, (selection) [, state [, format]] ]
```

### PYMOL API

```
cmd.save(file, selection, state, format)
```

### SEE ALSO

```
load, get_model
```

---

## select

### DESCRIPTION

"select" creates a named selection from an atom selection.

### USAGE

```
select (selection)
select name, (selection)
select name = (selection)          # (DEPRECATED)
```

### PYMOL API

```
cmd.select(string name, string selection)
```

### EXAMPLES

```
select near , (ll expand 8)
select near , (ll expand 8)
select bb, (name ca,n,c,o )
```

### NOTES

'help selections' for more information about selections.

---

# selections

## DESCRIPTION

Selections are enclosed in parentheses and contain predicates, logical operations, object names, selection names and nested parenthesis: ( [... [(...) ... ] ] )

name <atom names>	n;<atom names>
resn <residue names>	r;<residue names>
resi <residue identifiers>	i;<residue identifiers>
chain <chain ID>	c;<chain identifiers>
segi <segment identifiers>	s;<segment identifiers>
elem <element symbol>	e;<element symbols>
flag <number>	f;<number>
alt <code>	
numeric_type <numeric type>	nt;<numeric type>
text_type <text type>	tt;<text type>
b <operator> <value>	
q <operator> <value>	
formal_charge <op> <value>	fc;<operator> <value>
partial_charge <op> <value>	pc;<operator> <value>
id <original-index>	
hydrogen	h;
all	*
visible	v;
hetatm	
<selection> and <selection>	<selection>lt;<selection>
<selection> or <selection>	<selection> <selection>
not <selection>	!<selection>
byres <selection>	br;<selection>
byobj <selection>	bo;<selection>
around <distance>	a;<distance>
expand <distance>	e;<distance>
gap <distance>	
in <selection>	
like <selection>	l;<selection>

---

# set

## DESCRIPTION

"set" changes one of the PyMOL state variables,

## USAGE

```
set name, value [,object-or-selection [,state ]]
```

```
set name = value      # (DEPRECATED)
```

WARNING: object and state specific settings are not yet fully implemented -- look for them in version 0.51.

## PYMOL API

```
cmd.set ( string name, string value,  
         string selection='', int state=0,
```

```
int quiet=0, int updates=1 )
```

#### NOTES

The default behavior (with a blank selection) changes the global settings database. If the selection is 'all', then the settings database in all individual objects will be changed. Likewise, for a given object, if state is zero, then the object database will be modified. Otherwise, the settings database for the indicated state within the object will be modified.

If a selection is provided, then all objects in the selection will be affected.

---

## set\_color

#### DESCRIPTION

"set\_color" defines a new color with color indices (0.0-1.0)

#### USAGE

```
set_color name, [ red-float, green-float, blue-float ]
```

```
set_color name = [ red-float, green-float, blue-float ]  
# (DEPRECATED)
```

#### PYMOL API

```
cmd.set_color( string name, float-list rgb )
```

#### EXAMPLES

```
set_color red = [ 1.0, 0.0, 0.0 ]
```

---

## set\_geometry

#### DESCRIPTION

"set\_geometry" changes PyMOL's assumptions about the proper valence and geometry of the picked atom.

#### USAGE

```
set_geometry geometry, valence
```

#### PYMOL API

```
cmd.set_geometry(int geometry,int valence )
```

#### NOTES

Immature functionality. See code for details.

SEE ALSO

remove, attach, fuse, bond, unbond

---

## set\_key

DESCRIPTION

"set\_key" binds a specific python function to a key press.

PYMOL API

```
cmd.set_key( string key, function fn, tuple arg=(), dict kw={})
```

PYTHON EXAMPLE

```
from pymol import cmd

def color_blue(object):
    cmd.color("blue",object)

cmd.set_key( 'F1' , make_it_blue, ( "object1" ) )
cmd.set_key( 'F2' , make_it_blue, ( "object2" ) )

// would turn object1 blue when the F1 key is pressed and
// would turn object2 blue when the F2 key is pressed.
```

SEE ALSO

button

---

## set\_title

DESCRIPTION

"set\_title" attaches a text string to the state of a particular object which can be displayed when the state is active. This is useful for display the energies of a set of conformers.

USAGE

```
set_title object,state,text
```

PYMOL API

```
cmd.set_title(string object,int state,string text)
```

---

## set\_view

### DESCRIPTION

"set\_view" sets viewing information for the current scene, including the rotation matrix, position, origin of rotation, clipping planes, and the orthoscopic flag.

### USAGE

```
set_view (...) where ... is 18 floating point numbers
```

### PYMOL API

```
cmd.set_view(string-or-sequence view)
```

---

## show

### DESCRIPTION

"show" turns on atom and bond representations.

The available representations are:

```
lines      spheres  mesh      ribbon    cartoon
sticks     dots      surface   labels
nonbonded  nb_spheres
```

### USAGE

```
show
show representation [,object]
show representation [, (selection)]
show (selection)
```

### PYMOL API

```
cmd.show( string representation="", string selection="" )
```

### EXAMPLES

```
show lines, (name ca or name c or name n)
show ribbon
```

### NOTES

"selection" can be an object name  
"show" alone will turn on lines for all bonds.

### SEE ALSO

```
hide, enable, disable
```

---

## sort

### DESCRIPTION

"sort" reorders atoms in the structure. It usually only necessary to run this routine after an "alter" command which has modified the names of atom properties. Without an argument, sort will resort all atoms in all objects.

### USAGE

```
sort [object]
```

### PYMOL API

```
cmd.sort(string object)
```

### SEE ALSO

```
alter
```

---

## spawn

### DESCRIPTION

"spawn" launches a Python script in a new thread which will run concurrently with the PyMOL interpreter. It can be run in its own namespace (like a Python module, default), a local name space, or in the global namespace.

### USAGE

```
run python-script [, (local | global | module )]
```

### PYMOL API

Not directly available. Instead, use `cmd.do("spawn ...")`.

### NOTES

The default mode for spawn is "module".

Due to an idiosyncrasy in Pickle, you can not pickle objects directly created at the main level in a script run as "module", (because the pickled object becomes dependent on that module).  
Workaround: delegate construction to an imported module.

The best way to spawn processes at startup is to use the `-l` option (see "help launching").

---

## splash

### DESCRIPTION

"splash" shows the splash screen information.

### USAGE

```
splash
```

---

## stereo

### DESCRIPTION

"stereo" activates or deactivates stereo mode. Currently only high-end stereo graphics are supported on the SGI (stereo in a window).

### USAGE

```
stereo on  
stereo off
```

### PYMOL API

```
cmd.stereo(string state="on")
```

---

## symexp

### DESCRIPTION

"symexp" creates all symmetry related objects for the specified object that occurs within a cutoff about an atom selection. The new objects are labeled using the prefix provided along with their crystallographic symmetry operation and translation.

### USAGE

```
symexp prefix = object, (selection), cutoff
```

### PYMOL API

```
cmd.symexp( string prefix, string object, string selection, float cutoff)
```

### SEE ALSO

```
load
```

---

# sync

## DESCRIPTION

"sync" is an API-only function which waits until all current commands have been executed before returning. A timeout can be used to insure that this command eventually returns.

## PYMOL API

```
cmd.sync(float timeout=1.0,float poll=0.05)
```

## SEE ALSO

frame

---

# system

## DESCRIPTION

"system" executes a command in a subshell under Unix or Windows.

## USAGE

```
system command
```

## PYMOL API

```
cmd.system(string command)
```

## SEE ALSO

ls, cd, pwd

---

# time

This module provides various functions to manipulate time values.

There are two standard representations of time. One is the number of seconds since the Epoch, in UTC (a.k.a. GMT). It may be an integer or a floating point number (to represent fractions of seconds). The Epoch is system-defined; on Unix, it is generally January 1st, 1970. The actual value can be retrieved by calling `gmtime(0)`.

The other representation is a tuple of 9 integers giving local time.

The tuple items are:

```
year (four digits, e.g. 1998)
month (1-12)
day (1-31)
hours (0-23)
minutes (0-59)
seconds (0-59)
weekday (0-6, Monday is 0)
```

Julian day (day in the year, 1-366)  
DST (Daylight Savings Time) flag (-1, 0 or 1)  
If the DST flag is 0, the time is given in the regular time zone;  
if it is 1, the time is given in the DST time zone;  
if it is -1, mktime() should guess based on the date and time.

Variables:

timezone -- difference in seconds between UTC and local standard time  
altzone -- difference in seconds between UTC and local DST time  
daylight -- whether local time should reflect DST  
tzname -- tuple of (standard time zone name, DST time zone name)

Functions:

time() -- return current time in seconds since the Epoch as a float  
clock() -- return CPU time since process start as a float  
sleep() -- delay for a number of seconds given as a float  
gmtime() -- convert seconds since Epoch to UTC tuple  
localtime() -- convert seconds since Epoch to local time tuple  
asctime() -- convert time tuple to string  
ctime() -- convert time in seconds to string  
mktime() -- convert local time tuple to seconds since Epoch  
strftime() -- convert time tuple to string according to format specification  
strptime() -- parse string to time tuple according to format specification

---

## torsion

DESCRIPTION

"torsion" rotates the torsion on the bond currently picked for editing. The rotated fragment will correspond to the first atom specified when picking the bond (or the nearest atom, if picked using the mouse).

USAGE

torsion angle

PYMOL API

cmd.torsion( float angle )

SEE ALSO

edit, unpick, remove\_picked, cycle\_valence

---

## transparency

TRANSPARENCY

As of version 0.68, transparent surfaces are supported in both realtime (OpenGL) rendering mode as well as with ray-traced images.

Transparency is currently managed by setting either the global transparency variable or one attached to an individual molecule object.

It isn't yet possible to control transparency on a per-atom basis.

#### EXAMPLES

```
set transparency=0.5      # makes all surfaces 50% transparent
set transparency=0.5, mol3 # makes only mol3's surface transparent
```

---

## turn

#### DESCRIPTION

"turn" rotates the world about one of the three primary axes

#### USAGE

```
turn axis, angle
```

#### EXAMPLES

```
turn x,90
turn y,45
```

#### PYMOL API

```
cmd.turn( string axis, float angle )
```

#### SEE ALSO

move

---

## unbond

#### DESCRIPTION

"unbond" removes all bonds between two selections.

#### USAGE

```
unbond atom1,atom2
```

#### PYMOL API

```
cmd.unbond(selection atom1="(lb)",selection atom2="(rb)")
```

#### SEE ALSO

bond, fuse, remove\_picked, attach, detach, replace

---

## undo

### DESCRIPTION

"undo" restores the previous conformation of the object currently being edited.

### USAGE

```
undo
```

### SEE ALSO

```
redo, push_undo
```

---

## unmask

### DESCRIPTION

"unmask" reverses the effect of "mask" on the indicated atoms.

### PYMOL API

```
cmd.unmask( string selection="(all)" )
```

### USAGE

```
unmask (selection)
```

### SEE ALSO

```
mask, protect, deprotect, mouse
```

---

## unpick

### DESCRIPTION

"unpick" deletes the special "pk" atom selections (pk1, pk2, etc.) used in atom picking and molecular editing.

### USAGE

```
unpick
```

### PYMOL API

```
cmd.unpick()
```

### SEE ALSO

```
edit
```

---

# update

## DESCRIPTION

"update" transfers coordinates from one selection to another.  
USAGE

```
update (target-selection),(source-selection)
```

## EXAMPLES

```
update target,(variant)
```

## NOTES

Currently, this applies across all pairs of states. Fine control will be added later.

## SEE ALSO

```
load
```

---

# view

## DESCRIPTION

"view" makes it possible to save and restore viewpoints on a given scene within a single session.

## USAGE

```
view key[,action]  
view ?
```

key can be any string  
action should be 'store' or 'recall' (default: 'recall')

## PYMOL API

```
cmd.view(string key,string action)
```

## EXAMPLES

```
view 0,store  
view 0
```

## SEE ALSO

```
get_view
```

---

# viewport

## DESCRIPTION

"viewport" changes the size of the viewing port (and thus the size of all png files subsequently output)

## USAGE

viewport width, height

## PYMOL API

```
cmd.viewport(int width, int height)
```

---

# wizard

## DESCRIPTION

"wizard" launches one of the built-in wizards. There are special Python scripts which work with PyMOL in order to obtain direct user interaction and easily perform complicated tasks.

## USAGE

wizard name

## PYMOL API

```
cmd.wizard(string name)
```

## EXAMPLE

```
wizard distance # launches the distance measurement wizard
```

---

# zoom

## DESCRIPTION

"zoom" scales and translates the window and the origin to cover the atom selection.

## USAGE

```
zoom object-or-selection [,buffer]  
zoom (selection) [,buffer]
```

## PYMOL API

```
cmd.zoom( string object-or-selection [,float buffer] )
```

## SEE ALSO

*viewport*

origin, orient