

# VMD Tutorial

CSC, Espoo, Finland

October 29, 2010

## General Information

The goal of this tutorial is to give an overview of the features available in VMD. No previous knowledge of VMD is assumed, but there should be challenging exercises also for those who already know the basics of using VMD. Part of the exercises is formed by official VMD tutorials, while other parts are specifically written to put emphasis on how to use certain features of VMD and how to use VMD to visualize results produced by Gromacs. Basic knowledge of Gromacs is assumed, but feel free to ask during the tutorial if you have not used Gromacs earlier. For the official VMD tutorials, all input data is provided. For other exercises, it is possible to do them using your own simulation results, or using a provided trajectory of a large membrane protein.

There is a lot of material, and if you are new to VMD, it is not likely that you manage to do everything in the allocated time. Choose the parts that you feel would be most useful to you. It is possible to continue working on the basic exercises also in the afternoon session. Feel free to ask questions also on topics not directly covered by the exercises.

As a general tip, on the course machines you want to make sure that the rendering mode is set to GLSL to get the full benefit from the graphics boards. Select “Display → Rendermode → GLSL”.

# VMD Hands-On Exercises: Basics

## Before lunch

In the morning session, the exercises are divided into two groups: 1) working through VMD tutorials to get to know VMD for those that are not very experienced in using the VMD graphical user interface (some of the tutorial material is also useful for users that already have a significant amount of experience on VMD), and 2) Gromacs-specific sections for those who know the basics of using VMD, and want to learn how to use it to visualize simulation output from Gromacs.

## Introduction to VMD

With these exercises, you can familiarize yourself with the graphical user interface of VMD and different kinds of visualizations that you can create. They rely heavily on tutorial material available from the official VMD site.

1. **VMD Tutorial** If you have little or no previous experience of VMD, a great place to start is to work through the official VMD Tutorial:  
<http://www.ks.uiuc.edu/Training/Tutorials/vmd/tutorial-html/index.html>

The files required for the tutorial are also available on the machines in the course folder under `VMD-Tutorial/vmd-tutorial-files/`, and a PDF version of the tutorial in `VMD-Tutorial/vmd-tutorial.pdf`.

The Gromacs exercises assume that you are on some level familiar with the contents of the sections “Working with a Single Molecule”, “Trajectories and Movie Making”, and “Working with Multiple Molecules” of the tutorial. The “Scripting in VMD” section provides a basic introduction to the scripting capabilities of VMD, which we will discuss more in the afternoon.

2. **VMD Images and Movies Tutorial** If you already have some experience on VMD, you may find the VMD Images and Movies Tutorial useful to learn more about advanced visualization options available through the GUI. The tutorial can be found here:  
<http://www.ks.uiuc.edu/Training/Tutorials/vmd-ref/imgmv/tutorial-html/imgmv-tutorial.html>

The files required for the tutorial are also available on the machines in the course folder under `VMD-Images-and-Movies/imgmv-tutorial-files/`, and a PDF version of the tutorial in `VMD-Images-and-Movies/imgmv-tutorial.pdf`. The knowledge of the material in this tutorial is not required to complete (most of) the Gromacs exercises.

Work through the “Working with Still Frames” section, and the “Basic” section of “Working with Trajectories”. In the afternoon, we will discuss scripting, which is used in the advanced sections of “Working with Trajectories” to create custom graphics and complex movies.

## Working with Gromacs Trajectories

These exercises instruct you how to use VMD to visualize output of Gromacs simulations and how to solve common problems that occur in the visualization. As such, it is more of a “Tips & Tricks” nature than an actual tutorial with a clear-cut end result.

It is possible to use output from your own simulations, or use a provided trajectory of a ligand-gated ion channel. If you opt for the latter, the required files are available on the machines in the course folder under `Glycine-Receptor/`. This system is 100 ns simulation of a big membrane protein that functions as an ion channel activated by binding of glycine into a big extracellular domain. The system contains the protein (with 5 identical chains), a membrane (can be selected with “resname DOPC” in VMD), ions (residue names Na and Cl), ethanol (“resname LIG”), and water.

**1. Loading Gromacs Data into VMD** Take a look at the files produced by a simulation. For visualization, files that contain coordinates are particularly interesting. In Gromacs, the files with extensions `.trr` and `.xtc` are the main output formats for storing sets of coordinates; formats other extensions are also used for certain purposes: `.gro` (human-readable, used for the final configuration by default), `.tpr` (used for simulation parameters and initial configuration), and `.cpt` (binary file used for checkpointing). Check which of these formats can be directly read into VMD.

**2. Loading Gromacs Data into VMD** To make things more concrete, load first a `.gro` file into VMD, and then load a corresponding `.xtc` file into the same molecule (“File → New Molecule. . .”). Do you observe some problems? You can also try to load an `.xtc` file directly into a new molecule without a `.gro` file.

To understand the reasons for possible problems, one should understand that VMD stores two kinds of information about the system. The main data are *the coordinates of the atoms*, and these can be read from a wide range of formats. In addition to the coordinates, VMD uses metadata that describes the system: names of the atoms, which atoms are bonded to each other, etc. In Gromacs terminology, this information is referred to as *a*

*topology*. VMD cannot read full topology information from file formats used by Gromacs. It does understand atom names etc. available in, e.g., *.gro* files, but information about bonds it simply guesses when coordinates are first read in. These guesses are based on atom names and distances between the atoms, and work in practice quite well for systems where particles are atoms.

Armed with the above knowledge, can you think of additional problems that can occur, in addition to those you already observed?

**3. Periodic Boundary Conditions** Periodic boundary conditions (PBC) are the main source of problems for many beginning users (and even for more advanced users in some cases), and it can take a significant amount of thinking to get them visualized just the way you want. In VMD, there are two main ways to visualize the periodicity. You can either open “Graphics → Representations... → Periodic” to draw periodic copies, or you can type `pbx box -on` in the console window to show the actual box.

One common problem with PBC are bonds that are drawn through the whole box. They occur when a molecule is broken across the boundaries after VMD has detected a bond between the two atoms (see the next exercise). Proper use of `trjconv` (see below, in particular the `-pbx mol` option) should create a trajectory that does not contain such broken molecules. However, sometimes (and in several examples below) you want to have some broken molecules to keep them not protruding from the box. In these cases, you should select a representation that does not draw any bonds. For this exercise, the “Points” representation does well. Also, this exercise is most useful if your simulation is in a non-rectangular box. If you don’t have such a simulation available, you can use the provided glycine receptor trajectory, which uses a hexagonal membrane.

For working with PBC, Gromacs provides a tool called `trjconv` that allows one to manipulate a trajectory and change the way PBC are treated. Type `trjconv -h` for available options; basic usage is

```
trjconv -s glycinereceptor.tpr -f glycinereceptor.xtc
        -o glycinereceptor-processed.xtc ...
```

To understand different PBC visualization modes, run

```
trjconv -s glycinereceptor.tpr -f glycinereceptor.gro
        -o gr-triclinic.gro -ur tric
trjconv -s glycinereceptor.tpr -f glycinereceptor.gro
        -o gr-compact.gro -ur compact
```

and load the resulting *.gro* files into the same molecule in VMD. Both files contain exactly the same information, but on first look they seem very dif-

ferent. Play around with the periodic images and the `pbx box` command until you understand how they are related. Also try out the `-pbx`, `-center` (center the protein), and `-boxcenter` options of `trjconv`. Try to create a representation where all molecules are whole, the protein is centered in the origin, and the membrane is hexagonal.

VMD also comes with a script that serves the same purpose as `trjconv`. The above example `pbx box -on` is just one example of what this script can do. The commands `pbx wrap`, `pbx unwrap` and `pbx join` perform similar tasks to `trjconv`; you can simply use the console window to type the commands. Try to do (some of) the above `trjconv` exercises with these commands; see <http://www.ks.uiuc.edu/Research/vmd/plugins/pbxtools/> for documentation of these commands. Beware that `pbx join` can be very slow on larger systems!

**4. Gettings Bonds etc. Right** The easiest way of getting bond information right in VMD is to find a coordinate file that allows VMD to guess all the bonds right. The same applies to atom and residue names, but these are relatively easy to edit manually in any human-readable coordinate file format such as `.gro` or `.pdb`. Such a coordinate file should not have molecules that are broken across periodic boundaries, and it also should have bond lengths as close to “standard” as possible. `trjconv` is able to take care of the first requirement, and typically the final structure of a simulation works well for the second one as well, even if the starting structure was too deformed. You can also type `mol bondsrecalc top` in the console window to recalculate the bonds for a new frame for experimenting.

If you are using secondary structure visualization for proteins, such as the “NewCartoon” representation, the `mol ssrecalc top` command (in the console window) can be useful. It recalculates the secondary structure based on the current frame, and can be used to, e.g., force the visualization of the secondary structure as it occurs in the crystal structure.

If it seems difficult to get VMD to guess the bond information correctly, but you finally manage to get it right, it is often useful to save the bonds in a format that can be directly read into VMD later. You can do this by writing

```
[atomselect top all] writepsf glycinereceptor.psf
```

into the console (if you want to write the information for some other molecule than the top molecule, you can give the number of the molecule). Now, you can just create a new molecule from this `.psf` file (it does not contain any coordinates, only the topology information), and then load the coordinates you want.

If despite all efforts, you still can't get the bonds right, you can do the above and manually edit the resulting *.psf* file. There are also scripts available, one on the VMD website that works on certain types of topologies, that generate a *.psf* file from a Gromacs topology file (*.top* and *.itp* files). It is also possible to modify bonds and other atom data directly in VMD using its scripting interface, which will be discussed in the afternoon.

Congratulations! After the above exercises, you should know the basics required for visualizing (atom-scale) simulation data from Gromacs with VMD. If you still have time, you can either look at one of the tutorials mentioned in the beginning if you haven't already done so, or continue experimenting with different visualization options for the glycine receptor or your own data. Feel free to ask any questions you may have!

## VMD Hands-On Exercises: Advanced

### After lunch

In the afternoon session, there are several groups of exercises that are quite independent of each other, once you know how to write scripts for VMD. If you don't know the Tcl scripting language, you can start with the first exercises, which will introduce the basic features. Most of the exercises dealing with volumetric data also don't require knowledge of Tcl.

As in the morning sessions, feel free to choose exercises that you feel would be most suitable to you. It is also possible to continue working on the basic exercises if you don't yet feel comfortable with the basic features of VMD. In most of the exercises in this section, it is possible to use your own data, but the glycine receptor data used in the morning can also be used. For certain exercises, additional data and/or script files are given on the machines in the course folder under `hands-on-advanced-files/`.

## Scripts in VMD

These exercises will introduce writing scripts for controlling VMD. This will allow you to do things that are not possible through the GUI, and also to automate actions. The first exercises will introduce the Tcl (Tool Command Language) programming language, and how it is used in VMD. The exercises after this give a few examples of what can be accomplished with scripts. VMD also supports scripts written in Python through a bit different interface, but that we will not discuss here.

**0. Tcl Command Input** Tcl commands can be input in 1) the console window, 2) the Tk Console accessible in the "Extensions" menu, or 3) in script files that are run from 1) or 2) using `source script.tcl` or directly from the command line using `vmd -e script.tcl`. In addition, many Unix systems come with a separate `tclsh` executable that provides a simple Tcl interpreter that one can use for testing basic Tcl functionality. For the Tk Console, you should beware commands that produce a lot of output because they can make the Console very slow.

**1. Introduction to Tcl** A basic walkthrough of how to use Tcl in VMD can be found in the VMD Tutorial:  
<http://www.ks.uiuc.edu/Training/Tutorials/vmd/tutorial-html/node4.html>

Work through it to get an idea how to use Tcl scripting in VMD.

A more complete walkthrough through features available in Tcl can be found in the Tcl Tutorial at

<http://www.tcl.tk/man/tcl/tutorial/tcltutorial.html>

This tutorial is not VMD-specific, but you are encouraged to read through at least the first sections to get an idea of the general syntax and features of Tcl.

Finally, a reference of VMD-specific Tcl commands can be found in the VMD User Guide:

<http://www.ks.uiuc.edu/Research/vmd/current/ug/node113.html>

**2. Subroutines** As noted above, the simplest way of writing scripts is to put the Tcl commands in a file that is directly sourced. For more flexibility, it is often useful to write the script as a set of subroutines that can then be called from the Tk Console. A simple example:

```
proc increase_atom_radii {mol selection radius} {
    set sel [atomselect $mol $selection]
    set new_radii [list]
    foreach old_radius [$sel get radius] {
        lappend new_radii [expr {$old_radius + $radius}]
    }
    $sel set radius $new_radii
}
```

After sourcing a script with this subroutine, it can then be called from the Tk Console using `set_atom_radii top "protein" 1.4` to increase the radius of all protein atoms by 1.4 Å.

**3. Packages** If you have many scripts that you use in more than one project, it is convenient to create a script library out of them. This works best if you write your scripts as packages; see

<http://www.tcl.tk/man/tcl/tutorial/Tcl31.html> and

<http://www.wjduquette.com/tcl/namespaces.html>.

A concrete example of a package is the `gromacs.tcl` script discussed in the next exercise. Many VMD scripts are also provided as packages; thus, it is useful to understand the basics of how to use them. The idea of packages is that instead of sourcing the script directly using its full path, one can simply use, e.g., `package require gromacs` to load the code. For this to work, a few simple steps are required: 1) you should create a directory to hold all your packages, e.g., `~/scripts/vmd/packages`, and put all package files there, 2) you should tell VMD where to find the packages, and 3) you should

update the list of packages whenever you add or change your packages. 1) is straightforward, and 2) can be accomplished by adding

```
lappend auto_path ~/script/vmd/packages
```

to `~/vmdrc` (if the file does not exist, create it and also add `menu main on`). For 3), you can create a script in the package directory with the following contents:

```
#!/bin/sh

# Run tclsh \
exec tclsh "$0" "$@"
eval pkg_mkIndex . $argv
```

Running this script in the package directory with `*.tcl` as the argument updates the package index, which is kept in `pkgIndex.tcl`.

To test that your setup works, you can use the Gromacs package mentioned below. If, after putting the `gromacs.tcl` into your package directory, making the above steps and restarting VMD, `package require gromacs` works, it is set up properly.

One caveat with packages is that if you modify a package without changing the version number and updating the package list, you will need to manually source the file to get your modifications to work. For this reason, packages are not very well suited for scripts still under active development.

**4. Example: Topology Manipulation** As an example of a more complex script, you can find a `gromacs.tcl` in the `hands-on-advanced-files/` folder. This script reads in and parses a Gromacs `.top` file, and sets atom properties and bonds based on it. You can take a look at how it is implemented to get a taste on more advanced Tcl.

To use the script, copy it to your package directory (created in the exercise above), update the package index, and type `package require gromacs` in the Tk Console. For an example of how it works, change into the directory `hands-on-advanced-files/cg-membrane/`, load the `system.gro` file into VMD, and run

```
::gromacs::load_topology system.top
::gromacs::set_topology top
::gromacs::set_martini_radii top
```

**4. Example: Custom Graphics** As another example, consider creating a visualization that illustrates a coarse-grained representation of a lipid. Let us walk through how to create a nice visualization for this.

First, we need a single lipid. Let us extract one from the glycine receptor system. Load a single frame, and find out a selection that selects a nice-looking lipid. Something like `resname DOPC` and `resid NNN` should work, where NNN is a residue number of one of the lipids. Then type

```
set sel [atomselect top "resname DOPC and resid NNN"]
$sel writepdb lipid.pdb
mol new lipid.pdb
```

to write a *.pdb* file containing the lipid and to load it as a new molecule.

We now want to add spheres to represent the coarse-grained description. Each 3-5 atoms should form one sphere, that is centered at their center of mass and has a radius of 2.5 Å. To do this for one sphere, you can do (the lipid molecule should be the top one)

```
draw material Transparent
draw color cyan
set sel [atomselect top "name C10 C1P C1Q C1R"]
draw sphere [measure center $sel] radius 2.5
$sel delete
```

Partition the lipid into sets of 3-5 atoms (you can use, e.g., the VMD Query mode to find out names of individual atoms in the lipid), and write a script that creates a sphere for each set. Try to do this using a for loop instead of a lot of cut-and-paste. To remove your previous drawings, you can do `draw delete all`.

The above could have been achieved with a cunning use of auxiliary files: if we had created a PDB file with the positions of the coarse-grained spheres, we could have loaded that, set the radius of all the spheres to what we wanted, and we could have had our representation this way. However, the approach we took is more general. Change your script such that instead of several spheres for the headgroup (atoms with names “CN\*”, “N”, “C[AB]”, “P”, “O[A-D]”), you use an arrow that points from the “P” atom to the “N” atom. To easily draw arrows, you can add

```
proc vmd_draw_arrow {mol start end} {
    # an arrow is made of a cylinder and a cone
    set middle [vecadd $start [vecscale 0.9 [vecsub $end $start]]]
    graphics $mol cylinder $start $middle radius 0.15
    graphics $mol cone $middle $end radius 0.25
}
```

to the beginning of your script, and then simply use `draw arrow {x1 y1 z1} {x2 y2 z2}`.

## Advanced Movies

Scripting can be used with VMD to perform custom actions when the trajectory frame changes. This can be used, e.g., to implement custom graphics for use with the animation controls in the main window. A similar mechanism can also be used to set up custom actions for each frame in a movie, making it possible to rotate the camera, change trajectory frames, and/or add custom graphics in movies.

1. **VMD Images and Movies Tutorial** The VMD Images and Movies Tutorial contains very good examples of how to use these techniques. The tutorial can be found here:

<http://www.ks.uiuc.edu/Training/Tutorials/vmd-ref/imgmv/tutorial-html/imgmv-tutorial.html>

The files required for the tutorial are also available on the machines in the course folder under `VMD-Images-and-Movies/imgmv-tutorial-files/`, and a PDF version of the tutorial in `VMD-Images-and-Movies/imgmv-tutorial.pdf`. For this part, work through the “Advanced Trajectory Topic” and “Making Movies” sections of “Working with Trajectories”.

2. **Create Your Own Movie** Experiment with the techniques demonstrated above and create a movie of your own, using either your own data or the provided trajectory of a glycine receptor. Things to try out: 1) moving the camera in different directions, 2) moving the trajectory animation back and forth, and 3) changing graphical representations during the movie. The last is not done in the tutorial examples; it can be accomplished with the TCL command `mol showrep` and/or other `mol` subcommands. See

<http://www.ks.uiuc.edu/Research/vmd/current/ug/node135.html>

for reference on how to use these commands. The web page for the VMD Movie plugin also contains some example scripts:

<http://www.ks.uiuc.edu/Research/vmd/plugins/vmdmovie/>

## Volumetric Data

These exercises show how to work with volumetric data. In VMD, each molecule can, in addition to topology and coordinates, contain one or more sets of volumetric data. In each set, the data consists of values associated with locations in space. Each volumetric data set contains the values on a rectangular grid of points within a cube in the system. These data sets can be directly visualized as isosurfaces or contour lines on a plane, used to color

other representations, or used in selections.

Unfortunately, the support for volumetric data does not seem very complete in VMD. It is, for example, impossible to remove or alter volumetric data that has been once added without deleting the whole molecule. For this reason, if you have already spent a significant amount of time tuning other properties, it is typically best to create a separate molecule that will contain all the volumetric data. This way, if your initial attempt in creating the volumetric data does not work satisfactorily, you can just remove that molecule and start again without losing your other work. Also, note that it is only possible to use the first 8 volumetric data sets in selections, and this only works for volumetric data in the same molecule.

**0. Introduction to Volumetric Data** If you don't know anything about volumetric data in VMD, start by doing the "Working with Still Frames" section of the VMD Images and Movies tutorial. You can find the reference to all material of this tutorial in the previous section. Towards the end of this tutorial, there is a section on "Volumetric Data", which introduces visualizations of volumetric data. However, it does not address how to create such data, which will be discussed in the next exercises.

**1. VolMap plugin** The simplest way to create volumetric data for VMD is to use a plugin that ships with VMD. You can open it from "Extensions → Analysis → VolMap Tool". This plugin can calculate volumetric data either from a single frame, or by combining maps calculated for each frame in the whole trajectory. Everything that the plugin does can also be achieved with the TCL command `volmap` for scripted use.

Use the volumetric map plugin to compute density maps for water and lipids in the glycine receptor system, and visualize the resulting maps as transparent surfaces. This is a nice way to visualize, e.g., water in cases where its detailed structure is not very interesting in itself, but you want to see the regions it occupies. Think about different ways of improving the appearance of surfaces close to periodic boundaries, and try them out.

**2. Manipulating Volumetric Data** It is also possible to load volumetric data from a file, in various formats. Many of the formats used by quantum chemistry software are directly supported, and the OpenDX format is a text-based format that is quite simple to generate using a script. The OpenDX format is also the one the VolMap plugin uses if you request it to write the generated map into a file. Another option is to generate externally a PDB file that contains an atom for each desired point in the volumetric data, and set the data value in the B-factor field. This file can then be used in the

`volmap interp` command to generate the volumetric data object.

As a concrete example of manipulating volumetric data externally, consider the density maps calculated in the previous exercise. We now want to modify these maps such that we cut them using a plane that goes through the center of the protein. One way is to use a selection that only includes atoms within the region of interest, but this results in a relatively rough surface. Instead, we will directly modify the volumetric data.

First, recalculate the data such that you write them into files. Then, write a script that reads in one of these files, sets the volumetric data to zero in the region where you don't want to see it without modifying other data items, and writes a new file. You can use the `cut_volmap.awk` script in the `hands-on-advanced-files/` folder as a starting point. To get a smooth surface, you will want to cut along a coordinate plane (as an extra exercise, you can try to get a smooth surface along a plane not aligned with coordinate axes; it requires careful interpolation and not simply setting the data to zero). (Hint: to find the center of the protein for determining the plane, you can use `measure center` in VMD)