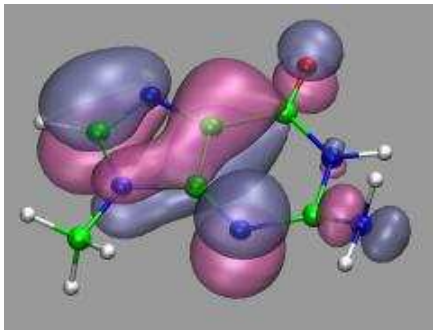
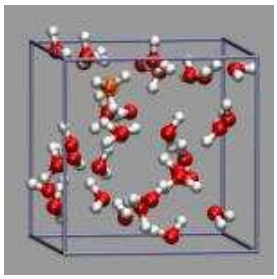


Visualization and Analysis of Quantum Chemical and Molecular Dynamics Data with VMD



1. Introduction

1.1. About this Tutorial

The aim of this tutorial is to help you using the program VMD for visualizing results from (first principles) molecular dynamics and electronic structure calculations, mainly done with the CPMD program, but also data from other codes like ESPRESSO/PWScf, GROMACS or Gaussian that can produce similar outputs are considered. It follows a close *hands−on* approach, so you will be provided with many examples and solutions to common problems. Most of the examples were conceived to satisfy specific needs in the Lehrstuhl für Theoretische Chemie at the Ruhr−Universität Bochum, but we try to not only show the solutions but also to describe the recipes. So we hope that the techniques described can not only be used to solve a particular problem, but are of more general use and can be translated to other, similar problems. Many of the examples should be adaptable with little effort to data created by other electronic structure or simulation programs. To help you getting started, most of the data and input files are available for download, so you in many cases, you will only need to adapt and/or combine the existing solution(s). Please note, however, that the following is assuming some background knowledge about using VMD. If you are new to VMD, you should start instead with the introductory tutorial on the VMD homepage.

1.2. About the Programs

CPMD is an *ab initio* electronic structure and molecular dynamics (MD) program using a plane wave/pseudopotential implementation of density functional theory. It is mainly targeted at Car−Parrinello MD simulations, but also supports geometry optimizations, Born−Oppenheimer MD, path integral MD, response functions, excited states and calculation of some electronic properties. For further information you can take a look at the CPMD consortium homepage at http://www.cpmd.org/.

ESPRESSO is a collection of electronic structure codes for plane wave pseudopotential calculations. Its main components are: PWscf (see below), FPMD and CP, as well as a GUI for creating input files and a code to create pseudopotentials.
PWscf (Plane−Wave Self−Consistent Field) is a set of programs for electronic structure calculations within Density−Functional Theory and Density−Functional Perturbation Theory, using a Plane−Wave basis set and pseudopotentials. PWscf is particularly well suited for the calculation of ground state calculations for solids and

surfaces. It has support ultra−soft pseudopotentials and k−points and can be used to calculation many properties using Density−Functional Perturbation Theory, BOMD, and for determination of energy barriers and reaction paths using Nudged Elastic Band (NEB) and Fourier String Method Dynamics (SMD).

Visual Molecular Dynamics (VMD) is a very powerful and flexible toolkit for visualization and analysis of molecular dynamics simulation data. Although it was initially developed with focus on large biomolecular systems, newer versions contain many features that make it a very attractive choice for visualizing results from CPMD and other electronic structure calculations. Further information can be found on the VMD homepage at http://www.ks.uiuc.edu/Research/vmd/. There also is a VMD Script Library with more examples of VMD scripting.

1.3. Notes

Most of the examples presented here assume, were tested with and may only run with VMD version 1.8.3 or later on a UNIX(TM)/Linux machine.

Contact axel.kohlmeyer@theochem.ruhr−uni−bochum.de if there are compatibility problems with newer versions of VMD or on other platforms. Any other form of feedback, e.g. corrections, enhancements, further examples, or new visualization problems, is also highly welcome.

1.4. Recent Changes

2005/01/03
: Several new examples, tricks, hacks and utilities added. Some examples now also come with directions or inputs for ESPRESSO/PWScf. Reviewed some of html generation macros, added support for computed links between parts.

2004/07/06
: Introduced sections for the start page and added this section, so that people know when and where to look for new stuff. Older changes taken from the CVS changelog.

2004/07/05
: PDF file generation now uses HTMLDOC instead of html2ps. The PDF is looking much nicer now and generation is much faster and needs less memory. Also fixed a bunch of minor markup errors detected by htmldoc.

2004/05/21
: Added example for 'cloning' a representation from one molecule to another (with an optional GUI) and how to hook up a Tk−GUI into the extension menu.

2004/05/16
: Added example for calculating a 2d−Ramachandran distribution and how to draw the result as color−coded 3d−surface using triangle drawing primitives.
Added example for a hardcopy printing command.

2004/05/14
: Added two new examples:
Banging two buckyballs together to illustrate the 'User' colorization method,
and an animated Isosurface (by cycling through the volumetric data sets) of a Hydrogen molecule shot trough Gold atoms.

---

2. Table of Contents

1. Introduction
- **1.1. About this Tutorial**
- **1.2. About the Programs**
- **1.3. Notes**
- **1.4. Recent Changes**

2. Table of Contents

3. Preparation and Installation Issues
- **3.1. Customizing the VMD Setup**
- **3.2. Predefining Additional Items**
- **3.3. Extending the Script and Plugin Search Path**

4. Loading and Displaying Configurations and Trajectories
- **4.1. Loading a Single Geometry**
- **4.2. Creating a Visualization**
- **4.3. Saving and Re−using a Visualization**
- **4.4. Viewing a Trajectory**
- **4.5. How to Show Breaking and Formation of Bonds**
- **4.6. Adding Graphics to a Visualization**

5. Adding Dynamic Graphics to a Trajectory
- **5.1. Adding a Progress Bar for the Elapsed Time**
- **5.2. Display the Total Dipole Moment of the Simulation Cell**
- **5.3. Visualizing Changing Atom Properties with Color**
- **5.4. Modify an Atom Property Dynamically from an External File**

6. Dynamic Atom Selection
- **6.1. Display a Changing Number of Molecules**
- **6.2. Keeping Atoms or a Molecule in the Center and Aligned**
- **6.3. Modify a Selection During a Trajectory**
- **6.4. Using the *User* Field for Computed Selections**
- **6.5. Tracing a Dynamic Property**

7. Visualizing Volumetric Data from Cube−Files
- **7.1. Electron Density and Electrostatic Potential**
- **7.2. Canonical and Localized Orbitals**
- **7.3. Electron Localization Function (ELF)**
- **7.4. Manipulation of Cube Files / Response to an External Potential**
- **7.5. Bulk Systems**
- **7.6. Animations with Isosurfaces**
- **7.7. Volumetric data from Gaussian**

8. Using Data Processing to Tailor Data for VMD
- **8.1. Visualizing Path−Integral Trajectories**
- **8.2. Extracting the Geometry Information from old CPMD Output Files**
- **8.3. Removing Unneeded Parts From a Cube File**
- **8.4. Extract Some Coordinates with Bounding Box Information**
- **8.5. Creating 3d−Ramachandran Histograms**

9. Misc Tips and Tricks
- **9.1. Collected 'draw' Extensions**
- **9.2. Using a Different Default Visualization**
- **9.3. Changing the Default vdW Radii**
- **9.4. Reloading the Current Trajectory**
- **9.5. Visualize a Trajectory With a Changing Number of Atoms or Bonds**

9.6. Set the Unit Cell Information for a Whole Trajectory
9.7. Directly Print the Current Visualization
9.8. Transferring a Visualization from a Molecule to Others
9.9. Adding a TCL−Plugin to the Extensions Menu
9.10. Turn Off Output in Analysis Scripts
9.11. Automatically Turn on TCL mode in (X)Emacs for .vmd Files
10. Credits
11. Script distribution policy

3. Preparation and Installation Issues

Many of the examples presented here utilize the excellent scripting capabilities of VMD and consist in part of reusable subroutines written in the VMD/Tcl dialect. To adapt the VMD scripting examples presented here, you should therefore first have a look at the excellent VMD User's Guide and also get a little bit acquainted with programming in Tcl/Tk. There is a Python scripting interface in VMD as well, but it is not (yet) covered here.

Via the autoloading feature of the Tcl/Tk script engine embedded in VMD users can have a repository of Tcl subroutines which will be loaded on demand. This way you don't have to copy the files into every single VMD script and you can update your scripts by just replacing the files in the repository (of course only as long as the calling sequence stays intact).

3.1. Customizing the VMD Setup

If you want to change some of the default settings of VMD you can put the necessary commands into a file named **.vmdrc** in your home directory. **Note**, that using a personal .vmdrc file turns off loading the default .vmdrc file from the VMD installation directory, so it is usually a good idea to start customizing VMD by copying the default .vmdrc file to your home directory and modifying it. For example I do prefer orthographic projection, a grey background, green carbon atoms and light green flourine atoms. This is achieved by putting the following code into your **~/.vmdrc**:

```
# make sure, that the main menu is active
menu main on

# modify display settings
display projection orthographic
axes location off
color Display Background silver

# redefine some colors
color Name C green
color Name F lime
color Type C green
color Type F lime
```

3.2. Predefining Additional Items

The graphical user interface of VMD has a selection of predefined parameters (materials, color schemes, etc.) that you can use for objects on your screen. You can create additional ones interactively, but you can also create them in your startup script so that they are always available. For example if you want to have a material 'Glass' that is

much more transparent than the default 'Transparent' material, then add the following code to your **~/.vmdrc**:

```
# define a new, very transparent material 'Glass'
material add Glass
material change ambient   Glass 0.00
material change specular  Glass 0.50
material change diffuse   Glass 0.65
material change shininess Glass 0.53
material change opacity   Glass 0.15
```

3.3. Extending the Script and Plugin Search Path

In order to make libraries of additional Tcl scripts (=subroutines) or plugins (e.g. for reading trajectories in a new, yet unsupported format) available to VMD you can also put some code in your **~/.vmdrc** file. The following example assumes that you have created a directory **vmd** in your home directory and therein the following subdirectories **scripts, scripts/tcl, scripts/extensions, and plugins**:

```
# search for new/updated molfile plugins in $HOME/vmd/plugins/$VMDARCH/molfile
# type 'vmdinfo arch' to find your name for $VMDARCH
vmd_plugin_scandirectory [file join $env(HOME) vmd/plugins [vmdinfo arch] molfile] *.so

# add local (autoloaded) scripts to the search path
set auto_path [concat $env(HOME)/vmd/scripts/tcl $auto_path]

# command extensions (e.g. vmd_draw_vector) have to be sourced
foreach ext [glob -nocomplain $env(HOME)/vmd/scripts/extensions/*.tcl ] {
        source $ext
}
unset ext
```

The complete **.vmdrc** file is also available for download. A Collection of several VMD script extensions , custom plugins and so on, can be found in the Tips and Tricks part of this tutorial.

4. Loading and Displaying Configurations and Trajectories

There are several ways of how you may want to look at configurations generated by Single Point and MD calculations. The following sections will give you some examples on how to visualize single geometries, whole trajectories, Wannier centers, and (optionally) copies of the simulation unit cell, which can be very helpful for systems with periodic boundary conditions.

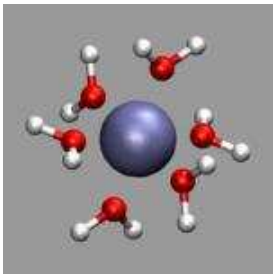
4.1. Loading a Single Geometry

The easiest way to load molecules from CPMD calculations into VMD would be to load the **GEOMETRY.xyz** file that recent CPMD versions (3.5.x and later) generate by default. Beginning with VMD version 1.8.2 CPMD style xyz files are supported. If you have an older version of VMD, you have to first convert the file to a supported format, e.g. PDB, with Babel or Open Babel (VMD has internal Babel support, if installed correctly).

For PWscf calculations you can use the scripts **pwi2xsf.sh** and/or **pwo2xsf.sh** to convert the respective input and output files to the xsf format, which is supported by VMD since version 1.8.3. Alternatively, you can use the simple pwscf2xyz.awk awk script to convert (some) PWscf output files to xyz format (this script may need some adjustments to support older/newer versions of PWscf). Please note, that the xsf format also usually contains the information of the supercell, which is not available for xyz files.

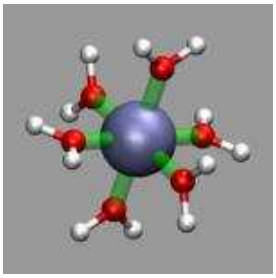
For the configuration on the left and the following examples, use the file al−h2o_6.xyz. You can load this file into VMD by just typing **vmd al−h2o_6.xyz** and you should see something very similar to the picture on the left. In the next sections we are trying to improve it a little bit.

4.2. Creating a Visualization



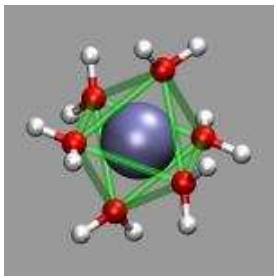
Once you have loaded a geometry into VMD, you can visualize it using a single or multiple representations for all or a subset of the atoms. The standard visualization method 'Lines' (see picture above) is usually best suited to quickly look at a long trajectory or a very large system. Also it does not need a very fast 3d−accelerated graphics hardware, but for presentations you usually want to have something more sophisticated. For example, in the picture on the left the water atoms are selected by *'name H O'* which is the short form of *'name H or name O'* and visualized by a 'CPK' representation while the aluminum is selected with *'name Al'* and visualized by a 'VDW' representation with a sphere size of 0.6. This way you can recognize the structure of your configuration more clearly. More detailed explanations of this procedure and examples of the various representations integrated into VMD can be found in the excellent VMD User's Guide

4.3. Saving and Re−using a Visualization



If you have spent some time on a specific representation of your system, you would usually like to somehow store that effort and reuse or improve it later. For that you can use the **File −> Save State...** dialog. You are asked to provide a filename (using the filename extension **.vmd** is recommended) which will contain the VMD script code to reproduce the current visualization. Since it is not an exact replay of **all** your actions, it will sometimes fail to give the *exact* same state, that you have saved, but usually it is pretty close.

After you have saved a visualization you can reload it from the **File −> Load State...** dialog or just run the script at VMD startup e.g. with **vmd −e saved_state.vmd**. If you want to reuse the visualiztion with a different coordinate or trajectory file, you simply copy the the state file and modify it with a text editor (search for the **mol new** statement(s)).

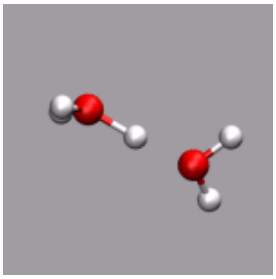


For the images in this section the visualization from the previous section, al−h2o_6.vmd, was loaded and then a 'Dynamic Bonds' representation was added to emphasize the octahedral arrangement of the water molecules (selection: *'name O or name Al'*, cutoff: 2.5, material: Transparent). The final visualization is also available for download: al−h2o_6−b.vmd. Also the selection *'name O'* in combination with a cutoff of 3.0 can be used for a different way to show the octahedron.

4.4. Viewing a Trajectory

For efficient working with VMD it is essential, that you understand how VMD processes the data. Trajectory data is in fact handled in multiple steps. First VMD reads the 'structure', i.e. it wants to know how many atoms of which type, with which bonds you want to display. If the bond information is missing in the file that was read in, it tries to guess the bonds via a distance search. This information will be applied for the **whole** trajectory, so the number or the ordering of the atoms **must not** change (thanks to the scripting capabilities of VMD, this limitation can be worked around to some degree). In the second step VMD wants to know the evolution of coordinates for each atom in the 'structure' during the trajectory. To achieve this with CPMD, you have several options: *1)* You can set the flag **TRAJECTORY XYZ** in the **&CPMD** section of your CPMD input file and then get an additional file named **TRAJEC.xyz** which you can load into VMD similar to GEOMETRY.xyz. *2)* You can use the CPMD trajectory format input plugin to read **TRAJECTORY** file(s) directly into VMD (supported from version 1.8.2 onward). For that you have to first load a **GEOMETRY.xyz** file to get the 'structure' information (**TRAJECTORY** files only contain coordinates, velocities and the time step number). And *3)* you can convert the **TRAJECTORY** file(s) into xyz format, e.g. with the help of an external program, e.g. the perl script traj2xyz.pl

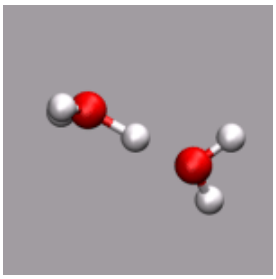
For PWscf the situation is a bit simpler: the xsf format is very flexible and also supports animations, so using **pwo2xsf.sh** converter should be sufficient. If you are using VMD version 1.8.2 or earlier and cannot or do not want to upgrade, you can also try the pwscf2xyz.awk script to convert the a PWscf output to the xyx format.



Note that you can load several trajectory files on top of each other, so you don't have to combine them first into a single file if you want to look at a long trajectory, that has been simulated in parts. For large trajectory files it is recommended that you activate the option **Load all at once** before you click on the load button. Otherwise it can take quite a long time to load the trajectory since VMD tries to run an animation of the trajectory during the load process as soon as enough data is available. If you have already created an elaborate visualization scheme and your graphics hardware is not very fast, this can get pretty bad. If you start from a saved state script file, you should append **waitfor all** to all **mol new** and **mol addfile** commands.

One more tip: Since reading large text mode trajectories takes a lot of time, you could use the **File −> Save Coordinates...** dialog to save the read in coordinates in a binary trajectory format (e.g. DCD). This file will then load much faster (no more parsing required) and also usually takes up less diskspace. You only have to have an .xyz file for the first timestep to provide the structure information.

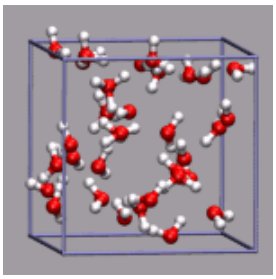
4.5. How to Show Breaking and Formation of Bonds



One big problem for many programs to visualize ab initio MD trajectories is how to handle formation or breaking of bonds. Programs that do this dynamically (e.g. Molden) are usually not able to handle large trajectory files efficiently, while other programs (e.g. Molekel or the default visualization of VMD) stick to the inital bond assignment. Recent versions of VMD now offer a **Dynamic Bonds** representation, which will recalculate the bonds every frame based on a given cutoff distance. In combination with an additional VDW representation, you can easily achieve a nice dynamical view of your simulation (see the small animation on the left an example and compare it to the animation in the previous section where only the CPK representation was used). If you cannot get away with a single distance cutoff for all bonds, just use several dynamic bond representations with different cutoffs on individual groups of atoms.

If your trajectory includes Wannier centers, the distance based bond selection will produce strange results. You could use the selections "not name X" for the atom representations and "name X" for the Wannier centers (looks pretty cool, if you use the Glass material and a VDW representation for the 'Wannier−atoms'. See the next chapter for some examples).

4.6. Adding Graphics to a Visualization



For simulations with periodic boundary conditions it is frequently very helpful to show the actual boundaries of the unit cell employed. For that VMD gives you access to several graphics primitives that can be added to a loaded molecule. This graphics primitives are drawn in the molecule local coordinate system and so they moved, rotated, and resized with all the representations belonging to that molecule. For the following example we need the custom command draw_cubic_unitcell installed. After loading the trajectory from the files 32h2o_h3op.xyz and 32h2o_h3op.dcd we want to mark the center of the simulation cell with a small yello ball and then show the unitcell itself. With an edgelength of 9.956 Angstrom, you have to give the following commands:

```
# put a small yellow ball in the center of the box
draw color yellow
draw sphere {0 0 0} radius 0.25 resolution 20
# add simulation cell object
draw_cubic_unitcell 9.956
```

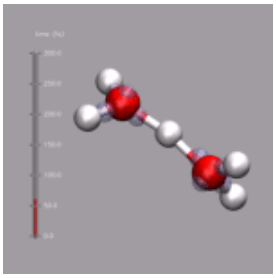
---

5. Adding Dynamic Graphics to a Trajectory

Sometimes you want to draw additional graphics the screen that show how some properties change during a trajectory. VMD offers you access to graphic primitves like lines, spheres, cylinders, cones, trangles and text with which you can add graphical elements to your molecule. To make these change with the evolution of the trajectory you can use the powerful Tcl variable tracing mechanism. In this case you want to trace the variable **vmd_frame()**. The following sections will give a few examples. The strategy is always the same:

1. write a small function that changes the graphics depending on the current frame number and an additional property
2. read a list of values for the property into an array indexed by the corresponding frame number.
3. trace **vmd_frame()** with that function

5.1. Adding a Progress Bar for the Elapsed Time



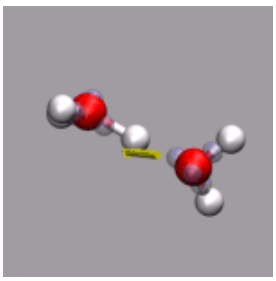
Although VMD shows the current frame number in the main window, it is sometimes nicer to have an indicator of the elapsed time in the visualization window. One big problem is, that the progress indicator should stay fixed, even if the molecule itself is rotated. As described above, usually you 'draw' in the coordinate system of the molecule and so the added graphics would rotate with the visualize molecule. This problem is avoided by creating an (empty) stub molecule (by calling **mol new** without a file name), fixing the position of that 'molecule' in the global coordinate system, and adding the time bar graphics to the fixed 'molecule'. This is done by the script function display_time_bar.

After you have loaded the trajectory from the files zundel.xyz and zundel.dcd and created a nice representation you run the following code:

```
    # function to draw the time bar with the required options
    proc do_time {args} {
    # see source for explanations of the options
       display_time_bar 1.0 50.0 "fs" 0
    }

    # connect do_time to vmd_frame
    trace variable vmd_frame(0) w do_time

    # go back to the start of the trajectory
    animate goto start
```

Now you can animate the trajectory as you like and the red line in the time bar will always the display the current time of the simulation. You can also move the mouse and the time display should stay fixed. The file zundel−time.vmd contains all the commands. You need to have the display_time_bar.tcl file installed properly to make this work. Alternatively you can copy the file to the current directory and add the command **source display_time_bar.tcl** to the .vmd script (or execute it manually) before you define **do_time**.

5.2. Display the Total Dipole Moment of the Simulation Cell



For this example some external data needs to be read in. In this case we use the corresponding data from the **DIPOLE** file created by a CPMD simulation. By executing

```
        awk '{ print $2, $3, $4; }' DIPOLE > zundel.dip
```

we can extract the current total diplol moment of the simulation cell to zundel.dip. We also need to have the following commands installed : center_of_mass and vmd_draw_vector. Now we have to load the trajectory again and then execute the following code:

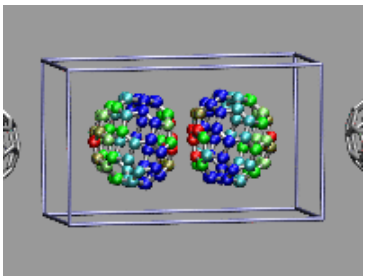
```
# define function to (re−)draw the dipole vector
proc do_dipdraw {args} {
    # dipdata has the center and the direction/lenght of the vector
    # dipgraph has the indices of the vector graphic elements
    global dipdata dipgraph
    set molid 0
    set frame [molinfo $molid get frame]
    if {[info exists dipdata($frame)]} then {
```

```
        if {[info exists dipgraph]} then {
            foreach g $dipgraph {
              graphics $molid delete $g
            }
        }
        graphics $molid color yellow
        lassign $dipdata($frame) cnt vec
        #                                              scaling  res   radius
        set dipgraph [vmd_draw_vector $molid $cnt $vec 100000.0  10   0.08]
    }
}

# read in the dipole data from file
set dip [open "zundel.dip" r]
# define selection to calculate the center of mass
set sel [atomselect 0 "not name X"]
set n [molinfo 0 get numframes]
for {set i 0} {$i < $n} {incr i} {
    # advance selection to the current frame and update.
    $sel frame $n
    $sel update
    set dipdata($i) [list [center_of_mass $sel] [gets $dip]]
}
close $dip

# connect to vmd_frame
trace variable vmd_frame(0) w do_dipdraw
animate goto 0
```

Again the whole vmd script file is also available under the name zundel−dip.vmd.

Of course, you can combine the time bar with the dipole vector display. A vmd script file to do that is available under the name zundel−all.vmd
NOTE: for this simple example to work, the trajectory and the dipole file have to be written with the same frequency.

5.3. Visualizing Changing Atom Properties with Color



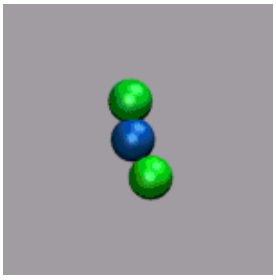
This example shows two C_{60} molecules colliding at high temperature (500K). We utilize the 'user' field of the atom trajectory data to store the distance of each atom from the center of mass of its C_{60} molecule for each frame as a measure of the deformation. This parameter is precomputed for the whole trajectory (so this is extremely useful for expensive calculations) and the coloring scheme 'User' is then used to display the results (red is closer, green is normal, blue is further away). Again we load the trajectory (bucky.xyz and bucky.dcd) in the regular way, visualize the Atoms as VDW spheres and Dynamic Bonds. To compute and store the distances we create a selections for each molecule and use the integrated vector operations of VMD. Note that a selection always returns a list of lists, so that when requesting **`{x y z}`** for the selection we directly get the individual vectors (=coordinate lists) stored into **`$c`** which can be directly used for the vector math. To store the result, we then have to provide a list of values for each atom in the selection. Here is the final code:

```
    set mol 0
    set left  [atomselect $mol {index < 60}]
    set right [atomselect $mol {index>= 60}]
```

```
    set n [ molinfo $mol get numframes ]

    for {set i 0} {$i < $n} {incr i} {
        # compute for first molecule
        $left frame $i
        set com [measure center $left]
        set dlist ""
        # loop over each {x y z} vector in $c
        foreach c [$left get {x y z}] {
            lappend dlist [veclength [vecsub $c $com]]
        }
        # set distance from COM for all atoms in selection.
        $left set user $dlist

        # now repeat for second molecule
        $right frame $i
        set com [measure center $right]
        set dlist ""
        foreach c [$right get {x y z}] {
            lappend dlist [veclength [vecsub $c $com]]
        }
        $right set user $dlist
    }
```

The complete script of this visualization script is also available for download under the name bucky.vmd.

5.4. Modify an Atom Property Dynamically from an External File



In the next example a dissociation trajectory of an Ar_3^+ molecule is shown. The color of the VDW spheres shall indicate how the positive charge distributed between the atoms with values ranging from 0.7 (coded blue) to 0.0 (coded red). Once more we load the trajectory (ar3plus.xyz and ar3plus.dcd) in the regular way, visualize the Atoms as VDW spheres, then read in the atom charges from ar3plus−charge.dat, and store them in an array. The procedure **do_charge** then updates the charge field in of the atoms for the current frame. In contrast to the previous example, the charge file is not stored *per frame* but for the whole trajectory. So by tracing **vmd_frame($molid)** we have this function being called at every change of the animation frame number. The VMD code to achieve this is as follows:

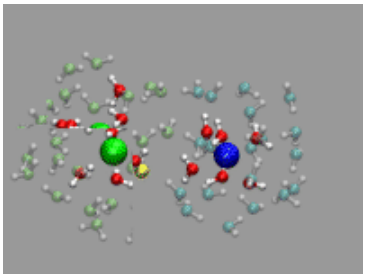
```
set molid 0
# read in charge data
set n [molinfo $molid get numframes]
puts "reading charges"
set fp [open "ar3plus-charge.dat" r]
for {set i 0} {$i < $n} {incr i} {
    set chrg($i) [gets $fp]
}
close $fp

# procedure to change the charge field from the data in $chrg
proc do_charge {args} {
   global chrg molid
   set a [molinfo $molid get numatoms]
   set f [molinfo $molid get frame]
   for {set i 0} {$i < $a} {incr i} {
       set s [atomselect $molid "index $i"]
       $s set charge [lindex $chrg($f) $i]
   }
```

```
}

# turn on update of the charge info and coloring in each frame.
trace variable vmd_frame($molid) w do_charge
mol colupdate   0 $molid on

# set color mapping parameters
mol scaleminmax 0 $molid 0.0 0.7
color scale method RGB
color scale midpoint 0.25
color scale min 0.0
color scale max 0.7
# go back to the beginning and activate the additional feartures.
animate goto start
do_charge
```

The complete script of this visualization is also available for download under the name ar3plus−charge.vmd.

6. Dynamic Atom Selection

6.1. Display a Changing Number of Molecules



Although this is not a problem unique to CPMD simulations, here is one example on how you can program VMD to recalculate the list of atoms or molecules fresh for each displayed timestep. For the example shown on the left, a pair of solvent separated ions (a sodium and a chloride) were selected from a (classical) molecular dynamics simulation with 800 SPC/E waters 16 NaCl ion pairs. The ions are represented by a VDW sphere and the first solvation shell waters with a CPK style representation and 'regular' colors. The second solvation shell ist transparent (cyan for the Na^+ and lime for the Cl^-. Finally oxygens close to the interface between the two solvation shells are highlighted in purple or in yellow. This way you can more easily spot the exchange in the hydration shell of the ions. As you can see from the animation, the chloride has larger solvation shells with more exchange than the sodium and occasionally water molecules in the second solvation shell are shared between both ions.

Now how was this done? First we need to load the trajectory from the files 800h2o−16nacl.xyz and 800h2o−16nacl.dcd and delete the default representation. Next we define a set of atomselect macros that make the selection of the atoms much easier. Here is the block for the sodium hydration shells (the cutoff distances 3.3 and 5.5 were taken from the $g_{NaO}(r)$ function for this simulation).

```
# selection macros. this is the first part of the magic
atomselect macro hyd {name H}
atomselect macro oxy {name O}
# first and second solvation shell of Na+ and oxygens in transition
atomselect macro naox1 {oxy within 3.3 of index 2405}
atomselect macro naox2 {(oxy within 5.5 of index 2405) and not naox1}
atomselect macro naotr {(oxy within 3.35 of index 2405)
    and not (oxy within 3.25 of index 2405)}
atomselect macro nahy1 {hyd within 1.31 of naox1}
atomselect macro nahy2 {hyd within 1.31 of naox2}
```

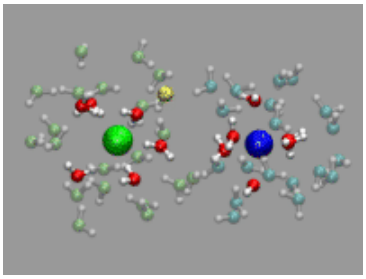
Note how the hydrogens are selected with a distance criterium around the oxygen selections. This way you make sure that you select only complete water molecules for each shell. The selection for the chloride follows the same pattern.

The next step is to create the representations utilizing the selection macros. Finally you have to turn on recalculation of the selection for each representation (and we add a little trajectory smoothing to have a less jumpy animation). This is done with the following code:

```
# let all selections be recalculated for each frame
# and smooth the trajectory a little bit for all representations
# that's part two of the magic.
set molid 0
set n [molinfo $molid get numreps]
for {set i 0} {$i < $n} {incr i} {
    mol selupdate $i $molid on
    mol smoothrep $molid $i 2
}
# go back to the start of the trajectory.
animate goto start
```

That is it. How to display all hydrogen bonds for all visible water molecule is left to you as an exercise. The full VMD script file is also available for download under the name nacl−shell.vmd. There also is a somewhat larger (640x480 pixel, 4.7 MByte) rendered MPEG−1 Movie of the same animation.

6.2. Keeping Atoms or a Molecule in the Center and Aligned

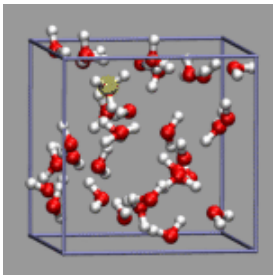


There is some room for improvement in the previous example: the ion pair rotates and moves around quite a bit (in fact it took considerable amount of time to find an ion pair in the trajectory that stays *that* much in place). For the solvated ion pair we want to keep the ion pair centered and aligned along the x−axis. For that we calculate the translation and rotation matrices with the **trans** command by straightforward vector algebra. But using the center and direction of the ion pair directly would give a very jumpy picture. So we average both parameters over several frames (here 20) and used the smoothed realignment instead. See the movie on the left as an example. The code in nacl−follow.vmd differs from the previous example primarily by the following, additional code:

```
proc do_realign {args} {
    global chist dhist hcount hoffs molid

    # this is the axis to align to
    set avec [vecnorm {1.0 0.0 0.0}]
    # this is the sliding window size
    set asize 20

    # initialize the cache counters
    if ([info exists hcount]) { } else {
        set hcount 0
        set hoffs  −1
    }

    # calculate center and direction of the ion pair
    set sel [atomselect $molid "index 2405 or index 2431"]
    lassign [$sel get {x y z}] na cl
    set cent [vecscale [vecadd $na $cl] 0.5]
    set dir [vecsub $na $cl]

    # store data in cache for sliding window averaring
    if {$hcount < $asize} then { incr hcount }
    incr hoffs
```

```
    if {$hoffs>= $hcount} then { set hoffs 0 }
    set chist($hoffs) $cent
    set dhist($hoffs) $dir

    # calculate averages
    set csum [veczero]
    set dsum [veczero]
    for {set i 0} {$i < $hcount} {incr i} {
        set csum [vecadd $csum $chist($i)]
        set dsum [vecadd $dsum $dhist($i)]
    }
    set csum [vecscale [expr 1.0/[expr $hcount * 1.0]] $csum]
    set dsum [vecnorm $dsum]

    # get rotation axis.
    set rvec [vecnorm [veccross $dsum $avec]]

    # set origin and rotation
    molinfo $molid set { center_matrix rotate_matrix} \
        [list [trans origin $csum] \
             [trans axis $rvec [expr acos([vecdot $dsum $avec])] rad]]
    # clean up selections
    $sel delete
}

trace variable vmd_frame($molid) w do_realign

# zoom to fit the display when vmd is started with '-size 800 600'
scale to 0.15
# the smoothing works better with this (no discontinuity)
animate style rock
```

6.3. Modify a Selection During a Trajectory



You will probably have noticed that running the previous two examples in VMD needs somewhat more cpu power to get a smooth animation. This is because all selections have to be recomputed anew in every animation step. In cases where the selection mechanism is even more complicated (and therefore more time consuming) you might want to find an alternative way. Here are two suggestions: a) you compile lists of the indices of the selected atoms with an external program, store them in a file, read them into an array and replace the selection text in each frame; b) you compute the selection directly, but then cache the results in an array so that subsequent viewings of the same frame are faster. In comparison, method a) has the advantage of being faster even at the first viewing than b) but the drawback that you have to keep and read in an additional (lengthy?) file.

In the example on the left, oxygen(s) where the excess proton is within bonding radius (1.32 Angstrom, as taken from the radial pair distribution function) should get a yellow highlight. So we first create the representation, but set the selection to "none". Note the index for the selection; in our case it is 2.

For method a) we compute the selecting by index strings and write them to a file (h3oplus-select.dat). For viewing we read this file into an array and have the selection updated from this array. Here is the code to achieve this (h3oplus-select.vmd):

```
    # function to update the selection from the highlight array
    proc do_highlight {args} {
        global highlight molid selid
```

```
        # get the current frame number
        set frame [molinfo $molid get frame]
        # if we have data for this frame, update the selection
        if {[info exists highlight($frame)]} then {
            mol modselect $selid $molid "$highlight($frame)"
        }
    }

    set molid [molinfo top]
    set selid 2
    set n [molinfo $molid get numframes]
    set highlight(0) none
    # read selections from file
    set fp [open "h3oplus-select.dat" r]
    for {set i 0} {$i < $n} {incr i} {
        set highlight($i) [gets $fp]
    }
    close $fp

    # hook highlight function into animation
    trace variable vmd_frame($molid) w do_highlight
    animate goto start
```

For method b) we put the code to compute the selection within the highlight function, but first check the array whether we have already computed the selection and then use the cached value instead. This example is available under the name h3oplus-cache.vmd:

```
# subroutine to update the selection of
# representation $selid and cache the results
proc do_highlight {args} {
    global highlight molid selid
    # get the current frame number
    set frame [molinfo $molid get frame]
    if {[info exists highlight($frame)]} then {
        mol modselect $selid $molid "$highlight($frame)"
    } else {
        set hl {none}
        # loop over all oxygens to count hydrogens within bondlength
        set osel [atomselect $molid "name O"]
        foreach ox [$osel list] {
            set sel [atomselect $molid "name H and within 1.32 of index $ox"]
            $sel frame $frame
            $sel update
            if {[$sel num] != 2 } {
                set hl "$hl or index $ox"
            }
            $sel delete
        }
        $osel delete
        # apply new selection and cache it
        mol modselect $selid $molid "$hl"
        set highlight($frame) "$hl"
    }
}

set molid [molinfo top]
set selid 2

trace variable vmd_frame($molid) w do_highlight
animate goto start
```

6.4. Using the *User* Field for Computed Selections

With recent versions of VMD the **user** field allows to store properties for each frame for each atom and which can be accessed in selection expressions. Using this method on the previous example one would first need to compute the number of bonded Hydrogens and store it in the **user** field. Using selections for that avoids the time consuming computations in Tcl and thus combines the advantages of both methods presented above.

```
# prep
set num [molinfo $mol get numframes]
set ox  [atomselect $mol {name O}]
set all [atomselect $mol {all}]

# create a selection for each oxygen atom to compute
foreach i [$ox get index] {
    set sel($i) [atomselect $mol "exwithin 1.30 of index $i"]
}

# loop over all frames, and for each frame loop over
# all oxygens and store the number of hydrogens in user
for {set n 0} {$n < $num} {incr n} {
    set bc {}
    foreach i [$ox get index] {
        $sel($i) frame $n
        $sel($i) update
        $all frame $n
        $all set user 0
        lappend bc [$sel($i) num]
    }
    $ox frame $n
    $ox set user $bc
    unset bc
}

# clean up selections
foreach i [$ox get index] {
    $sel($i) delete
}
$ox delete
$all delete
unset ox all sel i n
```

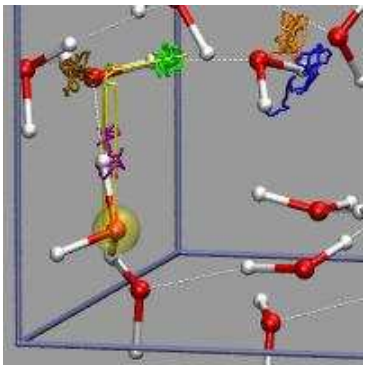
Now to recognize all Oxygen atoms with more than two bonded Hydrogens, we simply need to define a selection with **user > 2** and make sure that the selection is evaluated for every frame during an animation (−> **Trajectory** tab in the **Graphical Representations** menu.

```
        mol representation VDW 0.60000 20.000000
        mol color ColorID 4
        mol selection {name O and user > 2}
        mol material Transparent
        mol addrep top
        mol selupdate 3 $mol on
```

The complete example is available for download: 32h2o_h3op_user.vmd.

6.5. Tracing a Dynamic Property

Sometimes you want to show the dynamics of a process but are not able to use an animation. The trajectory_path procedure from the VMD script library is a nice example for that and can be used for the proton transport example from above as well, after it was adapted to allow updating the selection during the tracing of the path. Compared to the original, trajectory_path.tcl, has two additional, optional arguments to turn on/off the selection update and set the linewidth. The image on the left was created with the VMD script from above plus the following additional code (32h2o_h3op_trace.vmd).



```
# create selection for tracing hydrogens and the h3o-plus
set hyd1 [atomselect $mol {index 85}]
set hyd2 [atomselect $mol {index 99}]
set hyd3 [atomselect $mol {index 86}]
set hyd4 [atomselect $mol {index 97}]
set hyd5 [atomselect $mol {index 98}]
set h3op [atomselect $mol {name O and user > 2}]

# draw trajectory paths
trajectory_path $hyd1 blue 0 4
trajectory_path $hyd2 green 0 4
trajectory_path $hyd3 orange 0 4
trajectory_path $hyd4 purple 0 4
trajectory_path $hyd5 ochre 0 4
trajectory_path $h3op yellow 1 4
```

The image demonstrates nicely and without any animation, that the excess proton 'defect' is much more movable than the individual Hydrogen atoms due to the Grotthuss structural diffusion mechanism.

7. Visualizing Volumetric Data from Cube−Files

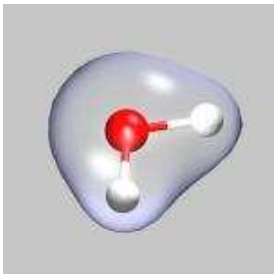
Apart from calculating trajectories various volumetric properties can be calculated with the CPMD program: e.g. electron densities, spin densities, electrostatic potentials, electron localization functions (ELF), localized and canonical (occupied) orbitals. This data can be visualized with VMD via the Gaussian cube file format. Beginning with version 1.8.2 VMD fully supports reading atom coordinates and volumetric data in the cube file format. Please note that some of the examples need a very recent CPMD version (3.9.1 or newer) to work properly, as i found and fixed a few cubefile related bugs while creating this part of the tutorial (the same goes for the cpmd2cube program, a patch relative to the latest released version from cpmd.org is available on request). VMD version 1.8.3 and CPMD version 3.9.2 as well as the cpmd2cube version relased with it contain additional improvements, mainly for handling non−orthogonal supercells.

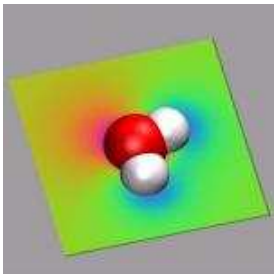
For some of the aforementioned properties cube file can be directly generated from cpmd, the rest is written in a *native* format which can be converted into a cube file with the help of the **cpmd2cube** program (available at the download area of http://www.cpmd.org). Since the CPMD manual is not very detailed about these tasks, the

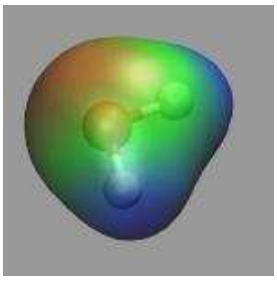
following section will give a few CPMD input file examples alongside the suggestions for visualizations.

7.1. Electron Density and Electrostatic Potential

To proceed with the following examples, we first need a fully converged wavefunction. This example input will generated that for an isolated water molecule in the gas phase:







```
&CPMD
  OPTIMIZE WAVEFUNCTION
  ELECTROSTATIC POTENTIAL
  RHOOUT
&END
&DFT
  FUNCTIONAL PBE
&END
&SYSTEM
  SYMMETRY
   0
  CUTOFF
  120.0
  ANGSTROM
  CELL
   6.0 1.0 1.0 0.0 0.0 0.0
&END
&ATOMS
*O_MT_PBE  KLEINMAN-BYLANDER
 LMAX=P
 1
 2.904516     3.000000     2.926732
*H_MT_PBE  KLEINMAN-BYLANDER
 LMAX=S
 2
 2.900437     3.000000     3.897528
 3.841176     3.000000     2.671532
&END
```

By adding the keywords **RHOOUT** and **ELECTROSTATIC POTENTIAL** the electron density and the electrostatic potential will be written to files named **DENSITY** and **ELPOT**, respectively, at the end of the wavefunction optimization. Note the rather high cutoff of 120ryd which is not really required, but helps to get smooth surfaces. Since we want to re−use this restart file as base for further calculations, please rename it from **RESTART.1** to **RESTART**. With **RESTART WAVEFUNCTION COORDINATES** (Note: *no* LATEST) all further calculation will always read in this 'high quality' restart and it will not be overwritten. Finally we need to convert the two volumetric files to cube files by:

```
cpmd2cube.x -o h2o-dens -dens DENSITY
cpmd2cube.x -o h2o-pot  -dens ELPOT
```

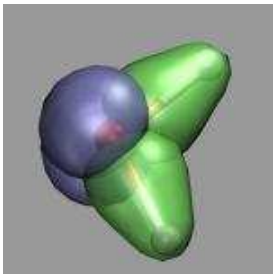
As a result you will get the files **h2o−dens.cube** and **h2o−pot.cube**.

To visualize volumetric data in VMD you currently have two options: Isosurfaces and Volume Slices. The two images in the top left of this section give an example for both styles. The upper image shows in addition to a CPK model of the water molecule an isosurface of the electron density (for ρ = 0.05), the lower image a volume slice through the electrostatic potential (for y = 0.5). The CPMD input (h2o−dens−pot.in), the pseudopotential

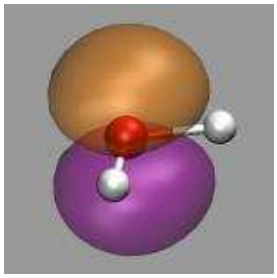
files (O_MT_PBE, H_MT_PBE), the resulting cube files (h2o−dens.cube, h2o−pot.cube) and a combined VMD script file (h2o−dens−pot.vmd) are available for download, so that you can experiment with it.

VMD version 1.8.3 introduces the **Volume** coloring method (for OpenGL implementations that support it), which can be used to colorcode the surface by the value of the electrostatic potential at the position of the surface, i.e. color−map the electrostatic potential to the surface. The value range of the colormap can be adjusted via the *Color Scale Data Range* fields in the **Trajectory** tab of the **Isosurface** representation. Note: that for a 'properly symmetric' behavior the electrostatic potential from CPMD calculations needs to be corrected, e.g. via the trimcube utility. Downloadable example files: h2o−dens.cube, h2o−pot−norm.cube, and h2o−pot−map.vmd.

7.2. Canonical and Localized Orbitals



To calculate the canonical (i.e. projected to an atomic basis set) and localized orbitals for our water molecule we need to do a properties calculation starting from the previously generated wavefunction. Both sets of orbitals can be calculated individually as well as simultaneously (as done here). Note that when calculating orbital cube files, especially for localized orbitals, the default cube files contain a lot of unneeded data, so that using the trimcube utility, or the **−trim** option to **cpmd2cube** is highly recommended to save disk space and reduce the memory requirements of VMD.



For the localized orbitals the keywords **LOCALIZE** and **WANNIER WFNOUT** are required. This generates a series of files with the names WANNIER_*.* which have to be converted to cube format with **cpmd2cube.x −o h2o−local WANNIER_1.1**. Note that you **must** not use the **−dens** option here.

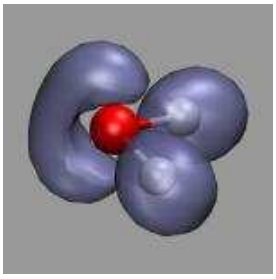
For the projected orbitals the **CUBEFILE ORBITALS** keyword (plus the two additional lines specifying how many and which orbitals shall be written) is required. This will generate a series of cube files with the names **PSI.*.cube**, that can be read into VMD directly.

Alltogether the first part of the CPMD input now contains:

```
&CPMD
  PROPERTIES
  RESTART WAVEFUNCTION COORDINATES
  WANNIER WFNOUT ALL
&END
&PROP
 LOCALIZE
 CUBEFILE ORBITALS HALFMESH
  4
  1 2 3 4
&END
```

The picture on the left demonstrates the localized orbitals. Here all cube files have been read in on top of each other and visualized with an isosurface (blue for the lone pairs, green for the OH−bonds). The picture on the right shows the HOMO of the water molecule. Here the two different phases are visualized by creating two repesentations from the same data set and just using isovalues with opposite sign for each of them. The input and cube files used in this section are: h2o−orbs.in, h2o−local−1.cube, h2o−local−2.cube, h2o−local−3.cube, h2o−local−4.cube, h2o−orbs−local.vmd, h2o−homo.cube, and h2o−homo.vmd.

7.3. Electron Localization Function (ELF)



The electron localization function (ELF) is a tool to describe chemical bonding. It provides an alternative look into chemical bonding and complements the picture gained by looking at the densities and orbitals. The correlation between ELF and chemical bonding is a topological and not an energetical one. So ELF can be said to represent the organization of chemical bonding in direct space. Chemical information can be obtained from ELF attractors taking the other topological elements into account as well. The attractors can be attributed to bonds, lone pairs, atomic shells and other elements of chemical bonding. For more details please consult the ELF homepage at http://www.cpfs.mpg.de/ELF/.

To calculate ELF with CPMD we need to add the keyword **ELF PARAMETER** to the **&CPMD** section of our input file. The resulting file named **ELF** is a density file similar to **DENSITY** and has to be converted to cube format with cpmd2cube. The picture on the left shows the ELF of our water example with an isovalue of 0.85. When increasing this value you can see, that the two lone pairs of the water are only partially localized. The relevant downloadable files are: h2o−elf.in, h2o−elf.cube, and h2o−elf.vmd.

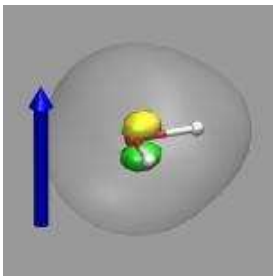
7.4. Manipulation of Cube Files / Response to an External Potential

Sometimes you need to postprocess the cube files, e.g. by calculating the difference between to densities. This can be done with the cubman utility from the Gaussian software suite. In this example we want to visualize the response of the a water molecule to an external potential perpendicular to the H−O−H plane.

For this purpose we first create an electron density cube file from the RESTART file of the previous examples with a simple properties job (cf. h2o−dens−nopot.in). The resulting file RHO_TOT.cube is renamed to h2o−nopot−dens.cube.

Now we need to create an unformatted fortran data file (extpot.unfo.grid) with the external potential on the grid points of the real space mesh of the CPMD job. In the current example this is a 80x80x80 grid so the fortran file has to use the same grid (e.g. in mkextpot.f). We now create a new wavefunction with the keyword EXTERNAL POTENTIAL (h2o−extpot.in), create another density cube file, and rename it to h2o−extpot−dens.cube.

Finally we use the cubman utility (see below for an example) to create die difference of the two densities (h2o−extpot−resp.cube) and visualize it. Here is a snapshot of the interactive dialog.

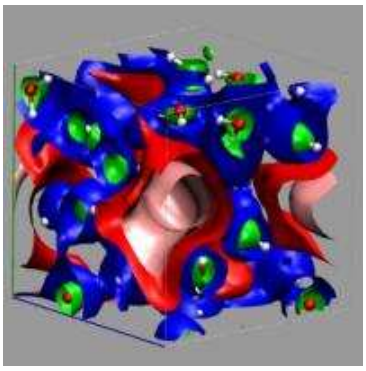


```
# cpmd-vmd/files> cubman
Action [Add, Copy, Difference, Properties, SUbtract, SCale]? su
First input? h2o-nopot-dens.cube
Is it formatted [no,yes,old]?  yes
```

```
Opened special file h2o-nopot-dens.cube.
Second input? h2o-extpot-dens.cube
Is it formatted [no,yes,old]?  yes
Opened special file h2o-extpot-dens.cube.
Output file? h2o-extpot-resp.cube
Should it be formatted [no,yes,old]?  yes
Opened special file h2o-extpot-resp.cube.
```

The image on the left shows the result. The arrow illustrates the direction of the electrostatic field represented by the external potential and the two colored isosurfaced show areas of reduced (green) and increased (yellow) electron density, the transparent isosurface represents the total electron density. The formation of an induced dipole moment in the water molecule is clearly visible. The figure was created with the VMD script h2o-extpot-resp.vmd.

7.5. Bulk Systems



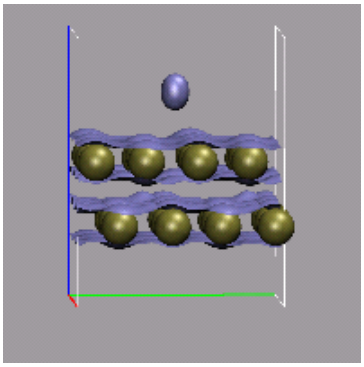
(click here or on the image to view a larger version)

Creating and visualizing volumetric data is not restricted to isolated molecules, but can also be used for bulk systems, for example a bulk water system. Occasionally you have to make sure, that the cube file is properly centered and that all atoms are inside the density/simulation box. The command lines to get the cube files used here were:

```
cpmd2cube.x -o h2o-dens -inbox -center -dens DENSITY
cpmd2cube.x -o h2o-elpot -inbox -center -dens ELPOT
```

After visualizing the water molecules you can add one or more isosurface representations in order to display different isosurfaces from the same data set: blue (= negative electrostatic potential), red (= same value only positive), pink (=even more positive isovalue), and green (= electron density). Again the full saved state and the data file are available for download with the filenames h2o-cube.vmd, h2o-dens.cube, and h2o-elpot.cube.

7.6. Animations with Isosurfaces



(click here or on the image to view a larger version)

With the help of a little bit of VMD scripting we can also do an animation with volumetric data, where the the isosurface is changed in each frame. In the present example, we have a simulation, in which a hydrogen molecule is shot at a double layer of gold atoms with **very** high velocity. We then need a cube file for each frame, that should be animated, therefor we need to instruct the CPMD program to write a (different) restart for each of these frames. This is done by using the **STORE** and **RESTFILE** keywords in the CPMD input file (for this example STORE was set to 20 and RESTFILE to 500 with a TIMESTEP of 2 a.u.). Now during the simulation (which should not exceed 10000 steps), a new restart is written every 20 steps.

The next step is to convert all of these restarts into cube files. This can done by doing one step of wavefunction optimization and using the keyword **RHOOUT** to write a **DENSITY** file, which then can be converted into a cube file with cpmd2cube.x. The first part of the CPMD input thus is:

```
&CPMD
  OPTIMIZE WAVEFUNCTION
  RESTART WAVEFUNCTION COORDINATES
  MAXSTEP
   1
  RHOOUT
&END
```

The conversion can be automated with:

```
for s in `seq 1 25`
do \
  rm -f RESTART
  ln -s RESTART.$s RESTART
  && ./cpmd.x au-dens.in
  && ./cpmd2cube.x -rho -o au-dens-$s DENSITY
done
```

We now load all those cube files in the correct order into VMD and create a visualization including an isosurface. To switch the volumetric data set during the animation, we write a small tcl procedure, that updates the representation and hook it into the animation loop by tracing **vmd_frame**. To make this as transparent as possible, we record the molecule id and the (unique) representation name of the isosurface in question in two global variables.

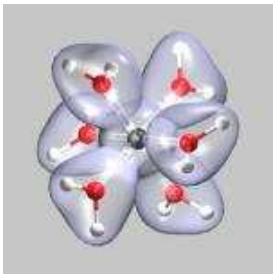
```
set updmol [mol new {au-dens-0.cube} type cube waitfor all]
...
set updrep [mol repname top 3]
...

proc update_iso {args} {
    global updmol
    global updrep

    set repid [mol repindex $updmol $updrep]
    if {$repid < 0} { return }
```

```
    set frame [molinfo $updmol get frame]
    lassign [molinfo $updmol get "{rep $repid}"] rep
    mol representation [lreplace $rep 2 2 $frame]
    mol modrep $repid $updmol
}

trace variable vmd_frame(0) w update_iso
```

There is one drawback: you have to know, which index the isosurface visualization has. But that is easily done (if you can count and remember, that the first representation id is 0), since you start from a script anyways. Any subsequent changes to the representations should be transparent to the script. You can download the full VMD script au−iso.vmd, and an archive with the cubefiles au−dens−cube.tar.gz (25MB).

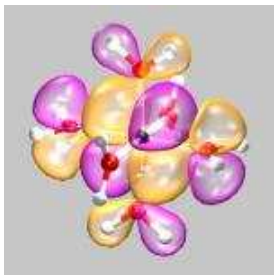
7.7. Volumetric data from Gaussian

In this section we take a small detour and discuss the visualization of volumetric outputs generated with the Gaussian electronic structure program. Since the cube file format was originally used in Gaussian visualizations like shown above are not restricted to CPMD calculations. The following are a few examples for using the Gaussian program, but this should be adaptable to other electron structure programs as well.



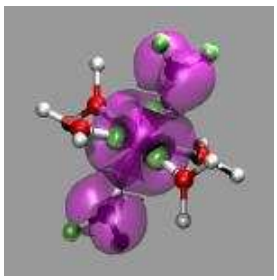
The most convenient way of creating cube files from a gaussian calculation is to use the **cubegen** and **cubman** programs. Prerequisite is that you have saved a checkpoint file from your calculation by adding a **%Chk=filename.chk** statement to the header of your gaussian input file. The (binary) checkpoint file needs to be converted into a formatted file using the **formchk** utility (**formchk filename.chk** will produce the file **filename.fchk**).

After these preparations we can finally start to generate some useful cube files. We start with the electron density of a chromium−(III)−ion surrounded by 6 water molecules (top left). This was created from the formatted checkpoint file using the command:
cubegen 0 Density=scf cr−h2o_6−dubl.fchk cr−h2o_6−dens−dubl.cube 60
For visualizing the density a single isosurface representation (after visualizing the atoms itself) with a small positive isovalue, e.g. 0.05, is sufficient (cr−h2o_6−dens.vmd).



The next example shows orbital number 40 (the third highest occupied orbital) from the same calculation. Here the command to create the cube file was: **cubegen 0 MO=40 cr−h2o_6−dubl.fchk cr−h2o_6−orb−40.cube 60** For this visualization two isosurface representations are needed now, one for positive and one for negative values (here: 0.01 and −0.01, cr−h2o_6−orb−40.vmd). If you want to show multiple different orbitals, you can either load several of these single orbital cube files simultaneously, or create a multi−orbital cube file by running a gaussian cube job (see the gaussian manual on how to do it). VMD can load these multiple orbital files as well as multiple cube file on top of each other and will let you select the individual volumetric data sets in a pop−up menu for the graphical representation menu.

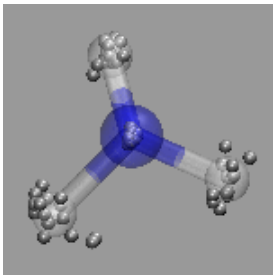
The final example shows the location of unpaired electrons by calculating the difference between *alpha*− and *beta*−electron spin density. For this two open shell Hartree−Fock calculations were performed (one for the dublett state and one for the quartett state), density cube files created and then the density difference calculated by running **cubman** in *subtract* mode on both cube files and thus creating a third cube file (cr−h2o_6−dens−diff.cube). You can see the the single unpaired electron in the dublett state is confined to a Chromium d−orbital, whereas the three unpaired electrons of the quartett state are rather delocalized.



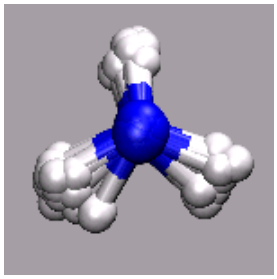
8. Using Data Processing to Tailor Data for VMD

In some cases the existing data is not (well) suited to be read in for a VMD visualization, so it needs to be augmented or converted using external programs. Here are some examples.

8.1. Visualizing Path−Integral Trajectories



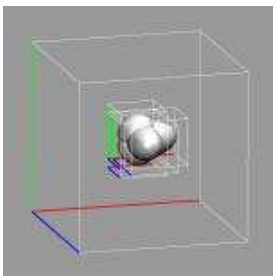
In path integral simulations each atom is represented by several replica, so reading in that data directly would create produce several molecules positioned (almost) on top of each other, which creates a somewhat cluttered visualization (see image on the right). Therefore the atom positions are better taken as the average of the replica and then a different, lucid visualization can be used. The perl script traj2xyz.pl will automatically detect path−integral runs from the fact, that for technical reasons the GEOMETRY.xyz file will only contain the geometry information for a single set of replica atoms, and write out the average replica positions at the beginning of each xyz record. The individual replica coordinates can be recognized, since the script will prepend their name with an X. Any selection with **and not name "X.*"** with ignore them. The processed xyz file (nh3−pimd.xyz) and the visualization (nh3−pimd.vmd) are available for download.



8.2. Extracting the Geometry Information from old CPMD Output Files

When running geometry optimizations, having a look at the 'trajectory' of the optmization is often desirable. Newer CPMD versions support the **XYZ** flag to the **OPTIMIZE GEOMETRY** keyword to create an xyz−file that can be easily visualized, but for outputs from older CPMD versions (or in cases where setting the XYZ keyword was overlook or forgotten), one may want to extract the coordinates from the output file. Doing this with a text editor can be quite tiresome. The perl script out2xyz.pl will try to do this for you. Of course this script also works (or at least it should) for other jobs that contain geometry information.

8.3. Removing Unneeded Parts From a Cube File



Cubefiles can become quite large, especially for very large systems. This can put a severy limit on how many of them can be visualized at the same time or even stored on the same disk. Frequently only a part of them is needed for the visualization, for example with surface slabs, isolated molecules or when looking at localized orbitals. The **trimcube** utility presented here (trimcube.c) allows to do this almost automatically, by trying to cut off parts of a cube file, whose absolute values are all below a given threshold. The version of **cpmd2cube** distributed together CPMD v3.9.2 contains this feature as well, so that you can get the small file immediately. The image on the left shows the original box and then the subboxes for each localized orbital. The file size reduction in this case was more than a factor of 35.

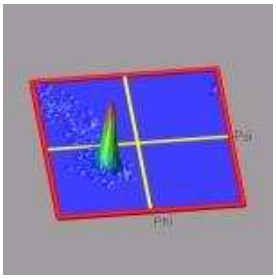
```
trimcube: cut out a part of a cubefile.

usage: trimcube [options] <input cube> [<output cube>]

available options:
-h[elp]                 print this info
-t[hresh] <value>       set trim threshold to <value>, (default 0.005)
-n[orm]                 normalize so that the integral over the cube is zero.

use '-' to read from standard input
program writes to standard output if not output file is given
```

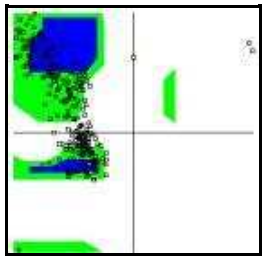
8.4. Extract Some Coordinates with Bounding Box Information

To following script provides the command **extract_sel** (extract_sel.tcl) will write a pdb file from a given selection. The special feature is, that it will shift the selection close to the origin, compute a minimal box that will fit this selection and store it in the CRYST record of the pdb file. Very useful for creating a QM subsystem from a classical MD simulation.

```
proc extract_sel { sel pdbfile {addbox {2.0 2.0 2.0}} } {
  set molid [$sel molid]

  # save original box dimensions and selection coordinates
  set origbox [molinfo $molid get {a b c alpha beta gamma}]
  set origxyz [$sel get {x y z}]

  # get the min/max coordinates from the selection
  set minmax [measure minmax $sel]
  # subtract the coordinates of the lower left front corner.
  $sel moveby [vecscale -1.0 [lindex $minmax 0]]
  # and shift by half the addbox vector to the middle
  $sel moveby [vecscale 0.5 $addbox]

  # box shall be the size of the selection plus the addbox vector
  set box [vecsub [lindex $minmax 1] [lindex $minmax 0]]
  set box [vecadd $addbox $box]
  # update the box size so the pdbfile will get that info
  molinfo $molid set {a b c alpha beta gamma} "$box 90.0 90.0 90.0"
  $sel writepdb "$pdbfile"

  # undo the coordinate shifts from above and reset the box
  $sel set {x y z} $origxyz
  molinfo $molid set {a b c alpha beta gamma} $origbox
}
```

8.5. Creating 3d−Ramachandran Histograms





The ramaplot tool in VMD is very useful for tracking changes of the protein backbone angles from a trajectory. But to get an impression of the statistical distribution of some configurations, the 'show all timesteps' mode in ramaplot can be easily misleading (see the picture on the right). In this example we have a short trajectory of a 12−ALA alpha−helix which is in fact stable during the time of the simulation. A histogram of the data would put an equal weight to each data point and with a so−called 'rubbersheet' representation (see picture on the left) one could could identify the regions of statistical relevance much better and see, that the alpha−helix does indeed not fall apart significantly.

To achieve this, we have to first create a histogram of the alpha−carbon angles and then visualize it. Since VMD does not support this internally, we create an empty dummy molecule and add the surface to it using VMDs graphics primitives. With the following script code, we can create the raw histogram.

```
set sel [atomselect top {protein and name CA}]
set res 36
set w [expr ($res − 1.0)/360.0]

set n [molinfo [$sel molid] get numframes]
for {set i 0 } { $i < $n } { incr i } {
    $sel frame $i
    $sel update

    foreach a [$sel get {phi psi}] {
        set phi [lindex $a 0]
        set psi [lindex $a 1]
        incr data([expr int(($phi + 180.0) * $w)],[expr int(($psi + 180.0) * $w)])
    }
}
```

We then have to normalize the histogram and create the surface by drawing four triangles between the corners of each square of data points and their mid−point. We also use the z−value of the mid−point to colorize the triangles according to a BGR color scale. so that the bottom is blue and the peaks will be green with red tips:

```
color scale method BGR
color scale max 0.9
color scale midpoint 0.3
for {set i 0} {$i < $res} {incr i} {
    for {set j 0} {$j < $res} {incr j} {

        set i2 [expr $i + 1]
        set j2 [expr $j + 1]

        set x1 [expr ($i  − (0.5 * $res)) * $len]
        set x2 [expr ($i2 − (0.5 * $res)) * $len]
        set xm [expr 0.5 * ($x1 + $x2)]

        set y1 [expr ($j  − (0.5 * $res)) * $len]
        set y2 [expr ($j2 − (0.5 * $res)) * $len]
        set ym [expr 0.5 * ($y1 + $y2)]
```

```
            set zm [expr ($data($i,$j) + $data($i2,$j2) \
                        + $data($i2,$j) + $data($i,$j2)) / 4.0]

            graphics $mol color [expr 17 + int (200 * $zm)]

            graphics $mol triangle "$x1 $y1 $data($i,$j)"   \
                     "$xm $ym $zm" "$x2 $y1 $data($i2,$j)"
            graphics $mol triangle "$x1 $y1 $data($i,$j)" " \
                     $x1 $y2 $data($i,$j2)" "$xm $ym $zm"
            graphics $mol triangle "$x2 $y2 $data($i2,$j2)" \
                     "$x2 $y1 $data($i2,$j)" "$xm $ym $zm"
            graphics $mol triangle "$x2 $y2 $data($i2,$j2)" \
                     "$xm $ym $zm" "$x1 $y2 $data($i,$j2)"
        }
    }
```

Completed with a nice border and some labels the full code can be put into a separate subroutine, so that the code to create the picture on the left becomes:

```
mol new {12-ala.pdb} type pdb waitfor all
mol addfile {12-ala.dcd} type dcd waitfor all
set sel [atomselect top {resid > 1 and resid < 12 and name CA}]
mk3drama $sel
```

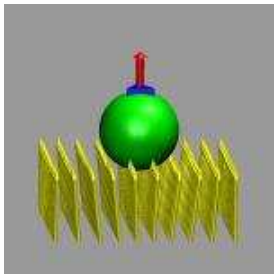
The VMD script code is available for download under mk3drama.tcl. This subroutines also turns off, but does not delete the originally loaded molecule(s), so you can in fact create multiple histograms from different selections and, e.g., view them in turn by clicking on the 'D' symbols in the main VMD window. The visualization and the data files are available under 12-ala-rama.vmd, 12-ala.pdb, and 12-ala.dcd.

9. Misc Tips and Tricks

This chapter is a collection of (hopefully) useful odds and ends that came up while working on these pages and with VMD in general.

9.1. Collected 'draw' Extensions

The following is a collection of extensions to the **draw** command. All of them return a list of graphics ids for the **graphics** primitives they were build of. When stored in a variable, they can be selectively deleted like in the following example (a more elaborate version is availiable for download as test-draw-ext.vmd):



```
# draw one blue prism and a red vector on top of it
set gidlist [draw prism {-0.7 -0.5 0.0} {0.7 -0.5 0.0} {0.0 1.0 0.0}]
draw color red
```

```
append gidlist " " [draw vector2 {0.0 0.0 0.0} {0.0 0.0 2.0} 1.0 20]

# draw a green sphere
draw color green
set sphere [draw sphere {0.0 0.0 -2.0} radius 2.0 resolution 30]

# delete only the sphere.
draw delete $sphere
```

The individual extensions are:

delete [<gid> [<gid>] ...] (vmd_draw_delete.tcl)
: This replaces the regular **draw delete** command with a version that can handle multiple ids and lists as arguments.

arrow {x1 y1 z1} {x2 y2 z2} [<scale> <resolution> {<radius>}] (vmd_draw_arrow.tcl)
: This is an updated version of the **draw arrow** example from the VMD User's Guide, which returns the graphics ids of the components. The optional third argument allows to scale the size of the vector.

vector {x1 y1 z1} {x2 y2 z2} [<scale> <resolution> {<radius>}] (vmd_draw_vector.tcl)
: This is similar to **draw arrow**, but the two vectors specify the center of the vector and the direction.

vector2 {x1 y1 z1} {x2 y2 z2} [<scale> <resolution> {<radius>}] (vmd_draw_vector.tcl)
: This is another version of an arrow, but now the two vectors specify the basepoint and the direction.

prism {x1 y1 z1} {x2 y2 z2} {x3 y3 z3} [<thickness>] (vmd_draw_prism.tcl)
: This is basically like **draw triangle** but the optional parameter gives the "thickness" of the triangle.

vecfield <list of lists> [<scale> <resolution> {<radius>}] (vmd_draw_vector.tcl)
: This is a wrapper around **draw vector** that processes a list of pairs of coordinate triples ({{{x1_1 y1_1 z1_1} {x2_1 y2_1 z2_1}} {{x1_2 y1_2 z1_2} {x2_2 y2_2 z2_2}} ...}) for creating a large number of vectors conveniently.

arrowfield <list of lists> [<scale> <resolution> {<radius>}] (vmd_draw_arrow.tcl)
: This is is like **draw vecfield** only that the arguments follow the **draw arrow** conventions.

unitcell <options> (vmd_draw_unitcell.tcl)
: This will draw add a unitcell graph to the top molecule.
Available options:

cell	(vmd|auto|{a b c alpha beta gamma}), default:"vmd" "vmd" will use the internal values "auto" will build a orthogonal unitcell based on 'measure minmax {all}' else a list of a,b,c,alpha,beta,gamma will be assumed.
origin	({x y z}|auto) default: {0.0 0.0 0.0} or "auto" with "cell auto"
style	(lines|dashed|rod) default: line
width	'width' of the lines/rods, default: 1.0
resolution	resolution of cylinders/spheres for 'style rod',default: 8

9.2. Using a Different Default Visualization

Sometimes one would want to have VMD start with a different default visualization. While this is not directly supported, something very similar can be achieved by tracing either the **vmd_initialize_structure** or the **vmd_trajectory_read** variable, which will be triggered when the loading of a new molecule starts or when it is finished, respectively. The following example code, when placed into your **.vmdrc** file, will add a new representation as defined in the **my_def_viz** procedure to the newly loaded molecule:

```
# delete the default visualization and create another one
```

```
proc my_def_viz {args} {
   lassign $args fname molid
   mol delrep 0 $molid
   mol color Type
   mol representation CPK 1.000000 0.300000 19.000000 16.000000
   mol addrep $molid
}

# add 'default visualization' hack for the first three molecules.
trace variable vmd_read_trajectory(0) w my_def_viz
trace variable vmd_read_trajectory(1) w my_def_viz
trace variable vmd_read_trajectory(2) w my_def_viz
```

Note that this 'hack' does not work for molecules loaded from the commandline as they are loaded, before **.vmdrc** is processed. Also, in case of vmd_init_structure the default visualization (Lines) will still be added to the molecule, as it will be added after **my_def_viz** has been processed so that there is no chance to delete there. In the case of **vmd_read_trajectory** the new visualization will be created after the whole file has been read, which may not be what you want, if you want to see the new visualization alread during the loading of the trajectory.

WARNING: Visualizations other then **Line** can put a severe strain on your available graphics and memory resources when loading very large molecules or structures.

9.3. Changing the Default vdW Radii

VMD does not know the van der Walls radii for many atom types. Since there are not too many of them in biomolecules, it is usually not a big problem. Also the radius can be changed via the script interface so you can change it later, anyways. If you are not working on/with biomolecules, that can become somewhat cumbersome, to always have to reset the values to what you want, so one may want to change the defaults. Using (almost) the same method as in the previous example this can be done on loading a new structure. The following example code, when placed into your **.vmdrc** file, will reset the vdW radii for the first three molecules to the values in the table. The entry on the left is a string handed to atomselect and the entry on the right is the corresponding radius. Please note, that the less specific selections have to come first, so they do not override the more specific ones. Of course this is not restricted to atom names: you can use any selection string you like.

```
fault VDW radii
dii on loading a structure
 {
]
 and matching radii

me C }  { 1.5 }  {name N }  { 1.4 }  {name O }  { 1.3 }  {name F }  { 1.2 }  {name P }  { 2.0 }  {name S }
def_vdw {
 $molid "$selstr"]
d




ze_structure(0) w my_set_def_vdw
ze_structure(1) w my_set_def_vdw
ze_structure(2) w my_set_def_vdw
```

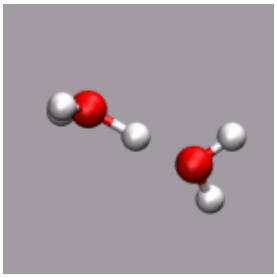
9.4. Reloading the Current Trajectory

When monitoring a running simulation, you may want to update the coordinates from the currently written file without having to recreate the visualization or exiting VMD. The following **reload** procedure (reload.tcl) will delete all frames and then reload the last coordinate file for the current top molecule preserving the current visualization and its orientation and scaling. **reload** also supports the **waitfor** option of the **mol load** command.

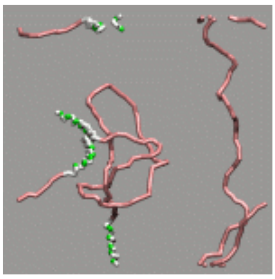
```
proc reload {args} {
    lassign $args arg1 arg2

    set viewpoints {}
    set mol [molinfo top]

    # save orientation and zoom parameters
    set viewpoints [molinfo $mol get {
        center_matrix rotate_matrix scale_matrix global_matrix}]

    # delete all frames and (re)load the latest data set.
    animate delete all
    set files [lindex [molinfo $mol get filename] 0]
    set lf [expr [llength $files] - 1]

    if {$arg1 == "waitfor"} {
        mol addfile [lindex $files $lf] \
            type [lindex [lindex [molinfo $mol get filetype] 0] $lf] \
            $arg1 $arg2
    } else {
        mol addfile [lindex $files $lf] \
            type [lindex [lindex [molinfo $mol get filetype] 0] $lf]
    }

    # restore orientation and zoom
    molinfo $mol set {center_matrix rotate_matrix \
                      scale_matrix global_matrix} $viewpoints
}
```

9.5. Visualize a Trajectory With a Changing Number of Atoms or Bonds

As already stated in the introduction to the visualization of trajectories , the internal design of VMD does not allow for changes of bonds and number of atoms during a trajectory. In case of breaking bonds, this can be handled most of the time by using the **Dynamic Bonds** representation. But if you want to have a changing number of atoms or display only specific bonds (e.g. as given in specific psf files), then one needs some extra scripting magic, to make this work with VMD. For example one can load each individual frame as a separate molecule and then define a new procedure **molmovie** (molmovie.tcl) to turn on only one of them sequentially and thus create an animation.



```
# the molmovie procedure creates an animation by
# sequentially turning the molecules on and off
# default is to wait a 500 miliseconds between 'frames'
proc molmovie {{loops 10} {delay 500}} {
    global molmovie_last

    set nmols [molinfo num]
    if {![info exists molmovie_last]} {
        set molmovie_last [expr $nmols - 1]
    }

    for {set i 0} {$i < $loops} {incr i} {
        for {set n 0} {$n < $nmols} {incr n} {
```



```
            display update
            display update ui
            mol on $n
            mol off $molmovie_last
            set molmovie_last $n
            after $delay
        }
    }
}

# example use of molmovie command
# load frames as separate molecules
for {set i 0} {$i < 15} {incr i} {
    mol new [format "zundel-frame-%02d.pdb" $i]
}

# now disable displaying all molecules
for {set i 0} {$i < [molinfo num]} {incr i} { mol off $i }

# run animation for two loops
molmovie 2
```

The upper example are just some frames of the zundel ion movie from the beginning of the tutorial (zundel-frames.tar.gz), the lower example is more elaborate and from a simulation, where the crossing of the periodic boundaries of long molecules requires new connectivity information in each step after wrapping all atom positions back into the unitcell (psf-molmovie-example.vmd). To get a uniform visualization for all frames, it utilizes the clone_reps command. The data for this example is also available: psf-molmovie-example.tar.gz (1 MByte).

9.6. Set the Unit Cell Information for a Whole Trajectory

Some coordinate file formats (e.g. trr, xtc, some versions of dcd, pdb, cube, or xsf) contain information about the shape and dimensions of the current unit cell. If this is not available or VMD does not support reading the unit cell information for a given file, it is required to set this manually if display of periodic images is desired. For a trajectory, this can become quite tedious, as the unit cell information needs to be set for **every** frame. The following **set_unitcell** procedure (set_unitcell.tcl) will make this more convenient for simulations with fixed cell dimensions.

```
proc set_unitcell {a b c {molid top} {alpha 90.0} {beta 90.0} {gamma 90.0}} {

    if {![string compare $molid top]} {
        set molid [molinfo top]
    }

    set n   [molinfo $molid get numframes]
    for {set i 0} {$i < $n} {incr i} {
        molinfo $molid set frame $i
        molinfo $molid set {a b c alpha beta gamma} \
            [list $a $b $c $alpha $beta $gamma]
    }
}
```

9.7. Directly Print the Current Visualization

VMD does not have an internal *Print This Now* facility, but it can be easily implemented using the integrated rendering capabilites the following is just a rough example which can probably rather easily expanded into a print

dialog plugin.

```
proc hardcopy { {printer "ps"} {renderer "TachyonInternal"}} {

# set the background temporarily to white to save ink and money.
    set oldbg [colorinfo category Display Background]
    color Display Background white

# temporary files
    set tga {/tmp/.hardcopy.tga}
    set ps  {/tmp/.hardcopy.ps}

    render $renderer $tga
    exec convert $tga $ps
    exec lpr -P$printer $ps
    exec rm -f $tga $ps

# reset background color
    color Display Background $oldbg
}
```

After sourcing or (auto−)loading this procedure you only have to type **hardcopy** to print the current visualization. The first argument selects the printer (default is 'ps') and the second the renderer (for this simple script only 'snapshot' and 'TachyonInternal' work). Note: this procedure assumes that you are running on a Unix−like operating system and have the ImageMagick tools installed. A file with this subroutine, hardcopy.tcl, is also available for download.

9.8. Transferring a Visualization from a Molecule to Others

If you have several molecules loaded, you sometimes want to re−use the same (elaborate) visualization that you have created for one molecule for the other molecules. This can be achieved by deleting the current list of representations and then read out and apply the representations from another molecule using the mechanism also used by the 'Save State' functionality. This can be easily put into a tcl procedure. The file clone_reps.tcl provides a new command **clone_reps** which takes the two the molecule ids as arguments. So "cloning" the visualization from molecule 0 to molecules 2 and 3 becomes:

```
clone_reps 0 2
clone_reps 0 3
```

This can also be easily called from a small Tk/TCL GUI (clonerepgui.tcl), which can be registered to the extensions menu. See the next section on how to create and use it.

9.9. Adding a TCL−Plugin to the Extensions Menu

VMD does not only include the TCL−interpreter, but also the Tk−GUI−library which can be used to create graphical user interfaces for your scripts. In this example a small GUI frontend to the clone_reps.tcl script shall be added to VMD's extensions menu.

We start by creating a namespace, defining per−namespace global variables, and providing a package name and version. In this case we need to share access to the base widget of the gui and the two molecule ids.

```
package provide clonerepgui 1.0

namespace eval ::CloneRep:: {
```

```
    namespace export clonerepgui

    variable w;                # handle to the base widget.

    variable fromid  "0";      # molid of the molecule to grab
    variable toid    "1";      # molid to clone the representations to

}
```

Now we need to provide a procedure **::CloneRep::clonerepgui** that will create the GUI and a few callback functions for the active elements of the GUI (e.g. the buttons). Since this should not become a general Tk/TCL tutorial, we will concentrate on one small example (see the comments in the script file clonerepgui.tcl for some more info): the callback for the Clone button. We import the two global variables with the molecule ids and then call the clone_reps procedure explained above.

```
proc ::CloneRep::CloneRepDoClone { args } {
    variable fromid
    variable toid

    clone_reps $fromid $toid
}
```

Since we want to register the GUI in the Extensions menu, we also have to provide a callback function, that will create the GUI and return a handle to the main widget.

```
proc clonerepgui_tk_cb {} {
  ::CloneRep::clonerepgui
  return $::CloneRep::w
}
```

To finally register the plugin with the main VMD GUI, we have to put the following code into our **.vmdrc** file.

```
if { [catch "package require clonerepgui" msg] } {
  puts "CloneRep plugin could not be found:\n$msg"
} elseif { [catch {menu tk register clonerepgui clonerepgui_tk_cb} msg] } {
  puts "CloneRep plugin could not be registered:\n$msg"
} else {
  puts "CloneRep activated"
}
```

As usual the files clone_reps.tcl and clonerepgui.tcl are available for download.

9.10. Turn Off Output in Analysis Scripts

Tcl commands run from the VMD command prompt send their return values also to the standard output (i.e. the screen). This can be annoying if not impeding the execution speed when running large analysis scripts. Simply closing **stdout** does not work in this case (you'll probably get a segmentation fault or worse), but since interative processing is disabled for subroutines, you can execute it from a subroutine. So instead of running **vmd −dispdev text −e get−his.vmd** which would produce the following output.

```
Info) Using plugin gro for structure file AChE.gro
Info) Using plugin gro for coordinates from file AChE.gro
Info) Determining bond structure from distance search ...
Info) Finished with coordinate file AChE.gro.
Info) Analyzing structure ...
Info)    Atoms: 43518
Info)    Residues: 12237
Info)    Waters: 11700
Info)    Segments: 1
```

```
Info)    Fragments: 11701   Protein: 1   Nucleic: 0
0
atomselect0
{41.4300003052 37.7299995422 43.4099998474} {45.8900032043 42.5899963379 48.6999969482}
4.46000289917 4.85999679565 5.28999710083
6.46000289917 6.85999679565 7.28999710083
Info) VMD for LINUX, version 1.8.2 (December 4, 2003)
Info) Exiting normally.
```

You can get rid of the unwanted red part, i.e. the return values, by executing a small wrapper script instead (source_silent.tcl):

```
# define subroutine
proc source_silent {args} {
    foreach f $args {
        source $f
    }
}

# execute the original script
source_silent {get-his.vmd}
```

9.11. Automatically Turn on TCL mode in (X)Emacs for .vmd Files

To have (X)Emacs recognize .vmd files as tcl code you have to add the following code to your **~/.emacs** file:

```
(setq auto-mode-alist
  (append '(("\\.vmd$"  . tcl-mode)) auto-mode-alist))
```

---

10. Credits

The following is a list of tools and persons (in no specific order) that played an important role in making this page reality.

This page was compiled by

Axel Kohlmeyer,
Lehrstuhl für Theoretische Chemie, Ruhr−Universität Bochum,
axel.kohlmeyer@theochem.ruhr−uni−bochum.de

Additional simulation data was contributed by

Holger Langer, Volker Kleinschmidt, Marcel Baer
Lehrstuhl für Theoretische Chemie, Ruhr−Universität Bochum
Heshe Peshkin

Helpful suggesions and encouragement

John E. Stone, Dominik Marx, Roger Rousseau, Volker Kleinschmidt, Holger Langer, Eduard Schreiner, Amalendu Chandra, Marcel Baer

VMD

Humphrey, W., Dalke, A. and Schulten, K.,
VMD − Visual Molecular Dynamics" J. Molec. Graphics 1996, 14.1, 33−38.
http://www.ks.uiuc.edu/Research/vmd/

CPMD

Copyright IBM Corp © 1990−2005,
Copyright MPI für Festkörperforschung Stuttgart © 1997−2001.
http://www.cpmd.org/

ESPRESSO / PWscf

Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso (SISSA, Trieste),
Paolo Giannozzi (Scuola Normale, Pisa) and others.
http://www.pwscf.org/

Gaussian 98 (Revision A.11)

M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, V. G. Zakrzewski, J. A. Montgomery, R. E. Stratmann, J. C. Burant, S. Dapprich, J. M. Millam, A. D. Daniels, K. N. Kudin, M. C. Strain, O. Farkas, J. Tomasi, V. Barone, M. Cossi, R. Cammi, B. Mennucci, C. Pomelli, C. Adamo, S. Clifford, J. Ochterski, G. A. Petersson, P. Y. Ayala, Q. Cui, K. Morokuma, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. Cioslowski, J. V. Ortiz, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. Gomperts, R. L. Martin, D. J. Fox, T. Keith, M. A. Al−Laham, C. Y. Peng, A. Nanayakkara, C. Gonzalez, M. Challacombe, P. M. W. Gill, B. G. Johnson, W. Chen, M. W. Wong, J. L. Andres, M. Head−Gordon, E. S. Replogle and J. A. Pople, Gaussian, Inc., Pittsburgh PA, 1998.
Copyright © 1988, 1990, 1992−1999, Gaussian, Inc.,
and copyright © 1983, Carnegie Mellon University
http://www.gaussian.com/

Tachyon

Copyright © 1994−2005 John E. Stone
http://jedi.ks.uiuc.edu/~johns/raytracer/

Raster3D

Merritt & Bacon (1997) Meth. Enzymol. 277, 505−524.
http://www.bmsc.washington.edu/raster3d/

ImageMagick

Copyright © 1999−2004 ImageMagick Studio LLC.
http://www.imagemagick.org/

Gifsicle

Copyright © 1997−2003 by Eddie Kohler.
http://www.lcdf.org/gifsicle/

htmldoc

Copyright © 1997−2004 by Easy Software Products.
http://www.easysw.com/htmldoc/

Mjpegtools

by Rainer Johanni, Gernot Ziegler, Andrew Stevens, Bernhard Praschinger, Ronald Bultje, Xavier Biquard, Matthew Marjanovic, pHilipp Zabel, Kawamata/Hitoshi, Stefan Fendt, Scott Moser, Shawn Sulma, Mike Bernson, James Klicman
http://mjpeg.sourceforge.net/

WML −− Website META Language

Copyright © 1996−2001 Ralf S. Engelschall
Copyright © 1999−2001 Denis Barbier
http://thewml.org/

If you want to cite or bookmark this page please use the URL
http://www.theochem.ruhr−uni−bochum.de/go/cpmd−vmd.html
as the underlying link might change in the future.

---

11. Script distribution policy

The scripts linked to on this page are made available free of charge for personal use. They still are copyrighted by the author(s) mentioned in the individual files. They may not be used in any commercial software without prior agreement of the author(s).

These scripts are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

All trademarks and registered trademarks are the properties of their respective holders.

All data files used for the examples presented here are available for download. You can look a all the files, if you follow the link to the files section.

GENERATED VIA Website META Language

Disclaimer / E−mail to the webmaster of this hompage: webmaster@theochem.ruhr−uni−bochum.de
Source File: index.wml (Thu Aug 25 19:02:51 2005) ($Revision: 1.44 $) Translated to HTML: Mon Oct 10 00:05:44 2005