VISUALIZATION OF MOLECULAR SIMULATIONS USING VMD ON STAMPEDE2

SETTING VMD DISPLAY SETTINGS

VMD builds on supercomputers automatically sets to default display settings. Therefore, several settings need to be adjusted according to the system of study.

Open VMD on your local machine:

Go to Extensions → Tk Console

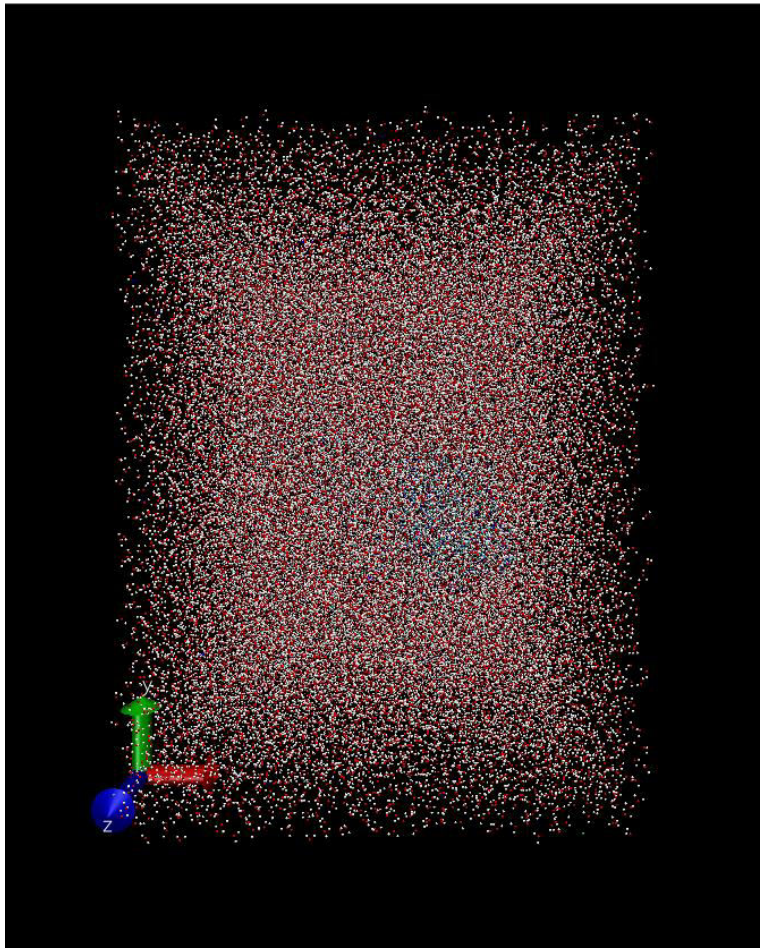
Load the structure file (.prmtop format) and trajectory (.dcd format)

VMD TkConsole

File Console Edit Interp Prefs History Help

```
>Main< (test_mpi_stampede2) 12 % set molid [mol new Imp_wtNLS_solv_ions.prmtop]
3
>Main< (test_mpi_stampede2) 13 % mol addfile prod_all_unwrapped.dcd waitfor all
3
>Main< (test_mpi_stampede2) 14 %
```

mol new assigns and an unique ID to each loaded structure. The variable **$molid** allows to execute commands that apply to the specific molecule

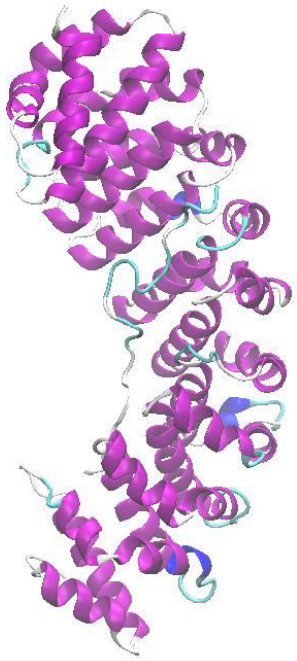


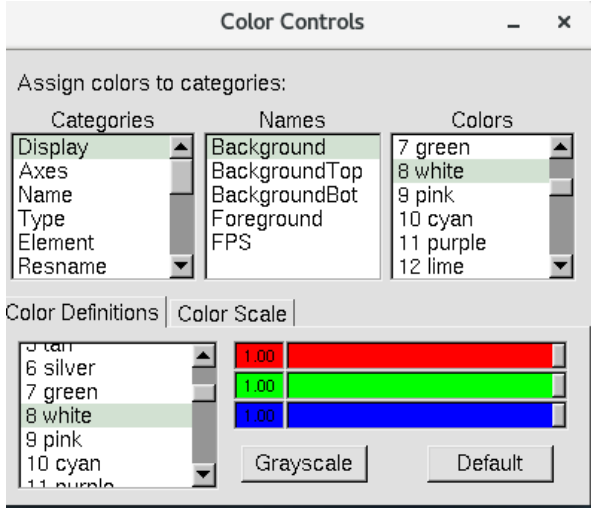
SETTING UP VMD DISPLAY SETTINGS

The additional changes on the display setting are going to be performed through the GUI of VMD. In order to keep track of the command to add to the final rendering script, the following is done:

- Go to File → Log Tcl commands to console
- Set Display → Orthographic
- Set Display → Axes → Off
- Set Graphics → Colors → Categories: Display, Names: Background, Color: White

Visual inspection of the system





Printed commands on console

```
Info) Logging commands to 'console'.
Info) # VMD for LINUXAMD64, version 1.9.4a7 (July 12, 2017)
Info) # Log file 'console', created by user fabiog
Info) display projection Orthographic
Info) axes location Off
Info) menu color off
Info) menu color on
Info) color Display Background white
Info) mol modselect 0 3 protein
Info) display resetview
Info) mol modcolor 0 3 Structure
Info) mol modstyle 0 3 NewCartoon 0.300000 10.000000 4.100000 0
Info) mol modmaterial 0 3 AOShiny
Info) mol modstyle 0 3 NewCartoon 0.300000 15.000000 4.100000 0
>Main< (test_mpi_stampede2) 14 %
```

Graphics → Representations… → Coloring method: Secondary Structure, Drawing Method: New Cartoon, Materials: AOChalky, Resolution: 50

RENDERING USING VMD MPI

The following script describes rendering movies using MPI-supported VMD on Stampede2 supercomputer.

render_movies_mpi.tcl

```
proc blockdecompose { framecount } {
  set noderank  [parallel noderank]
  set nodecount [parallel nodecount]
  set start [expr round($noderank     * $framecount / $nodecount)]
  set end [expr round(($noderank+1) * $framecount / $nodecount) - 1]
  return [list $start $end]
}

proc testgather { num trajectoryfile } {
        set noderank  [parallel noderank]
        # only print messages on node 0
        if {$noderank == 0} {
                puts "Testing parallel gather..."
        }
        # Do a parallel gather resulting in a list of all of the node names
        set datalist [parallel allgather $num]
        # only print messages on node 0
        if {$noderank == 0} {
        set filej [open "${trajectoryfile}_${noderank}.dat" w]
        puts "datalist length: [llength $datalist]"
        puts "datalist: $datalist"
        foreach dataline $datalist {
                foreach line $dataline {
                puts $filej $line
                }
        }
        close $filej
        }
}
```

Decomposition of trajectory in blocks, according to the total number of frames and number of ranks requested.

The *parallel allgather* command allows Tcl MPI analysis scripts to gather the results from all the nodes, in a set of per-node list of results. The output in printed on rank 0.

RENDERING USING VMD MPI

The following script describes rendering movies using MPI-supported VMD on Stampede2 supercomputer.

render_movies_mpi.tcl

```
proc sumreduction { a b } {
        return [expr $a + $b]
}

proc testreduction {} {
        set noderank  [parallel noderank]
         # only print messages on node 0
         if {$noderank == 0} {
                puts "Testing parallel reductions..."
         }
         parallel allreduce sumreduction [parallel noderank]
}
```

Computes parallel reduction across the requested ranks, each rank contributing to one value. The final value is returned to all ranks.

RENDERING USING VMD MPI

```
proc take_picture {args} {
        global take_picture
         # when called with no parameter, render the image
        if {$args == {}} {
        set f [format $take_picture(format) $take_picture(frame)]
        # take 1 out of every modulo images
        if { [expr $take_picture(frame) % $take_picture(modulo)] == 0 } {
                render $take_picture(method) $f
        if { $take_picture(exec) != {} } {
                set f [format $take_picture(exec) $f $f $f $f $f $f $f $f $f $f]
                eval "exec $f"
        }
        }
        # increase the count by one
        incr take_picture(frame)
        return
        }
        lassign $args arg1 arg2
        # reset the options to their initial stat
        # (remember to delete the files yourself)
        if {$arg1 == "reset"} {
                set take_picture(frame)  0
                set take_picture(format) "./animate.%04d.rgb"
                set take_picture(method) snapshot
                set take_picture(modulo) 1
                set take_picture(exec) {}
                return
        }
        # set one of the parameters
        if [info exists take_picture($arg1)] {
                if { [llength $args] == 1} {
                        return "$arg1 is $take_picture($arg1)"
                }
                set take_picture($arg1) $arg2
                return
        }
  # otherwise, there was an error
        error {take_picture: [ | reset | frame | format  | \
                                method  | modulo ]}
}
```

The following script describes rendering movies using MPI-supported VMD on Stampede2 supercomputer.

render_movies_mpi.tcl

Generates the data files for each frame of the trajectory (.dat). After the generation of data files, the image rendering is performed using the Very fast multiprocessor ray tracer **Tachyon.**

RENDERING USING VMD MPI

The following script describes rendering movies using MPI-supported VMD on Stampede2 supercomputer.

render_movies_mpi.tcl

```
proc mpianalyze { framecount trajectoryfile } {
        set noderank  [parallel noderank]
        set nodecount [parallel nodecount]
        set block [blockdecompose $framecount]
        set start [lindex $block 0]
        set end   [lindex $block 1]
        set len   [expr $end - $start + 1]
        parallel barrier; # wait for all nodes to reach this point
        puts "Node $noderank, frame range: $start to $end, $len frames total"
        parallel barrier; # wait for all nodes to reach this point
        mol addfile $trajectoryfile first $start last $end step 1 waitfor all
        parallel barrier; # wait for all nodes to reach this point
        puts "Node $noderank, loaded [molinfo top get numframes]"
        parallel barrier; # wait for all nodes to reach this point
        # Display settings
        source my_ss_colors.tcl
        display projection Orthographic
        display resize 400 800
        display shadows on
        display ambientocclusion on
        axes location Off
        color Display Background white
        mol modselect 0 0 protein
        mol modcolor 0 0 Structure
        mol modstyle 0 0 NewCartoon 0.300000 50.000000 4.100000 0
        mol modmaterial 0 0 AOChalky
        mol smoothrep 0 0 3
        display resetview
        rotate y by -240
        rotate z by -40
        scale by 1.4
        set dir /work/06295/fabiogon/stampede2/test_mpi_stampede2/figs
        set tachyondir /work/06295/fabiogon/stampede2/vmd-1.9.4a43/lib/tachyon
        # Loop over frames and do your analysis here!
        for {set frame 0} {$frame <= [expr $len - 1]} {incr frame 1} {
                set ts [expr $frame + $start]
                set frameinput ${dir}/figure.${ts}.dat
                take_picture reset
                take_picture format ${dir}/figure.${ts}.dat
                take_picture method Tachyon
                animate goto $frame
                puts "Frame...$frame"
                take_picture
                exec ${tachyondir}/tachyon_LINUXAMD64 -aasamples 12 -format TARGA ${dir}/figure.${ts}.dat -o ${dir}/figure.${ts}.dat.tga
        }
        parallel barrier; # wait for all nodes to reach this point
}
```

Frame decomposition on ranks

Load trajectory

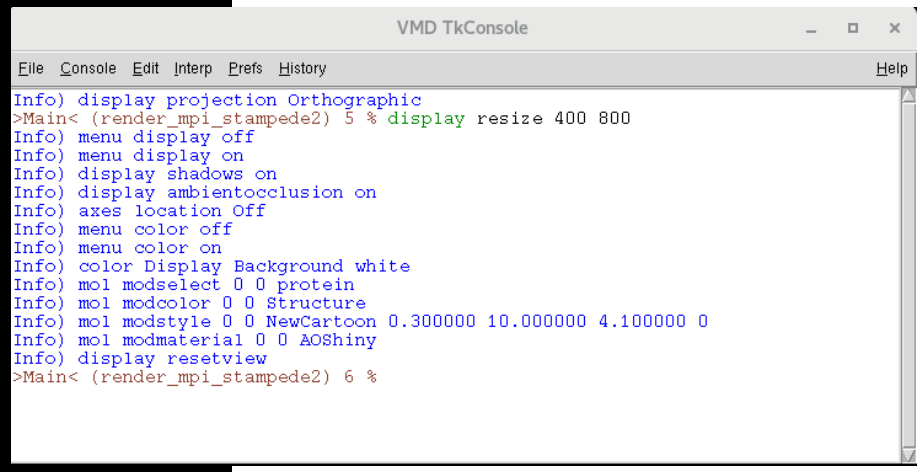
Save visualization state from console to the submission script

Visual settings

Loop and rendering over frames

my_ss_colors.tcl sources a color palette for secondary structure feature in the proteins.

Viewpoints and representations saved on the console are copied to the rendering script in the *mpianalyze* process



RENDERING USING VMD MPI

> The following script describes rendering movies using MPI-supported VMD on Stampede2 supercomputer.
>
> **render_movies_mpi.tcl**

```
proc analyzetest { frame trajectoryfile } {
        parallel barrier; # wait for all nodes to reach this point
        mol new Imp_wtNLS_solv_ions.prmtop
        animate delete all
        parallel barrier; # wait for all nodes to reach this point
        set num [mpianalyze $frame $trajectoryfile]
        parallel barrier
        testgather $frame $trajectoryfile
}

testreduction
set mytraj prod_all_unwrapped.dcd
set nframes 5000
analyzetest $nframes $mytraj
quit
```

- Loads the structure file and preforms the rendering over the total number of ranks requested.
- The example follows a trajectory with 5,000 frames.

RENDERING USING VMD MPI

The script is submitted using Slurm Workload Manager

runbatch_mpi.sh

```
#!/bin/bash

TIME=01:00:00
HOSTS=2
PROCSPERNODE=1
export IBRUN_TASKS_PER_NODE=1
BINDIR=/work/06295/fabiogon/stampede2/VMD_MPI/vmd
RUNDIR=`pwd`

sbatch --job-name=render_mpi --nodes=${HOSTS} --ntasks=$((${HOSTS}*${PROCSPERNODE})) --ntasks-per-node=$PROCSPERNODE --time=${TIME} --partition=normal -o slurm%j.out -e slurm%j.err << ENDINPUT
#!/bin/bash

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/work/06295/fabiogon/stampede2/netcdf-c-4.7.4/build/lib
export myfile=$1

# Go to working directory (submission directory)
cd $RUNDIR
echo "Current working directory is: " `pwd`
# Execute job
time ibrun $BINDIR/vmd_mpi -e \${myfile}

ENDINPUT
```

Frame 0

Frame 9

The resources requested correspond to 2 nodes, each one running a single process. Therefore, each node renders 2,500 frames.

SUMMARY

Setting the simulation display settings.

1.1 Load the structure file and the MD trajectory on a local machine.

1.2 Enable Tcl commands to console.

1.3 Set VMD visualization settings and copy the commands to the render script, under the *mpianalize* procedure.

2. Rendering using VMD MPI

2.1 Set the total resources to request with Slurm workload manager.

2.2 Submit the job rendering the figures with Tachyon.

3. Post image processing

3.1 Convert TARGA format figures to JPG/PNG extension. GNU Parallel is useful for the conversion.

3.2 Render the composition using image post-processing software such as mencoder, ffmpeg or Adobe AfterEffects.

REFERENCES

1. Humphrey, W., Dalke, A., & Schulten, K. (1996). VMD: visual molecular dynamics. *Journal of molecular graphics*, *14*(1), 33-38.
2. Caddigan, E., Cohen, J., Gullingsrud, J., & Stone, J. (2003). Vmd user's guide. *Urbana*, *51*, 61801.
3. Caddigan, E., Cohen, J., Gullingsrud, J., & Stone, J. (2003). VMD Installation Guide.
4. Stone, John E. (1998). An Efficient Library for Parallel Ray Tracing and Animation. Masters Theses.