MD_course

The following is an overview for the practical side of the course. You will find both information, files and instuctions here.

DAY 1

Linux/Bash introduction

When working on a cluster there may not be a GUI. Also on your own machine it may be a lot more efficient to use a terminal.

You can do everything you can do via a GUI by typing commands. Here we will do so using `bash` the currently still most used shell.

Even better you can automate tedious and repetitive processes, which would require a lot of time to do by hand.

Here is a quick overview over important bash commands

Command	Description
`.`	Current directory
`~`	Home directory
`..`	Parent directory (one level up)
`;`	end of line replacement if you want multiple commands in 1 line
`cd destination`	Change directory (e.g. `cd /path/to/dir`) - not writing a destination goes to `~`
`mkdir -p`	Create a directory and any necessary parent directories
`mv`	Move or rename files or directories (`mv source target`)
`cp`	Copy files or directories (`cp source target`)
`cat`	concatenates files (`cat file1 file2 file3`)
`touch`	creates an empty file (or updates times)
`scp`	Secure copy files between hosts (`scp file user@host:/path`)
`pwd`	Print working directory
`sed "s\\|to_substitute\\|replaced_by\\|g"`	Replace all occurrences of `to_substitute` with `replaced_by` using `sed`
`grep string files`	searches for string in files
`man command`	opens the manual page for the command - RTFM (Read the very fine manual…
`ssh user@machine`	connects to machine via ssh for your username "user" on the remote machine
`echo`	prints to stout: e.g. `echo foo bar`
`chmod`	Change file permissions (mode). Examples: `chmod 755 file` or `chmod +x script.sh`
`chown`	Change file owner and group. Example: `chown user:group file`

1) (Bash) Scripts are essentially nothing essentially nothing more storing commands in a text file so you can reuse them (also great to have reproducability - no arbitray behavior due to typos-). You usually start them with the so called shebang `#!/bin/bash` which tells the computer with which interpreter to execute it - here bash. `#!/bin/python3`would be a python script for example. You can of course completely forgo it and manually call the `bash`command in the shell.

2) Wildcards/placeholders `*` any chars, `?` one char.
3) Redirections `>` redirects stout to a new file. `>>` appends to file. `&>` redirects stout and sterr to a file.
4) With loops you can do a lot some examples

For loop over iterations:

```
for i in {1..5}; do
  echo "Iteration $i"
done
```

```
while true; do
    sleep 5
    # do something every 5 seconds, for example:
    echo "Still running - $(date)"
done
```

find all .gro files and do something with them

```
files=$(ls *gro)
for file in $files
do

#what you do actually want to do
echo $file
done
```

4) Variable definitions are as simple as e.g. `myvariable=5`. It can be called then by its name adding a $ before. `$myvariable`.
5) Comments: # is a comment symbol and everything behind it will not be executed.
6) Simple addition `newvalue=$(($oldvalue + $newvalue))`
7) Logic for comparison

Operator / Command	Meaning / Example
`-gt`	Numeric greater than. Example: `[ "$a" -gt "$b" ]`
`-lt`	Numeric less than. Example: `[ "$a" -lt "$b" ]`
`-eq`	Numeric equal. Example: `[ "$a" -eq "$b" ]`
`-ne`	Numeric not equal. Example: `[ "$a" -ne "$b" ]`
`-ge`	Numeric greater than or equal. Example: `[ "$a" -ge "$b" ]`
`-le`	Numeric less than or equal. Example: `[ "$a" -le "$b" ]`
`=`	String equal (POSIX `[ ]`). Example: `[ "$str1" = "$str2" ]`
`!=`	String not equal. Example: `[ "$str1" != "$str2" ]`
< / >	String less/greater (lexicographic) in `[[ ]]` or use `\<'/\>'` in `[ ]`. Example: `[[ "$a" < "$b" ]]`
`-z`	String is empty. Example: `[ -z "$var" ]`
`-n`	String is not empty. Example: `[ -n "$var" ]`
`!`	Logical NOT. Example: `if ! [ -f file ]; then ...`
&&	Logical AND between commands/tests. Example: `[ "$a" -gt 0 ] && echo "pos"`

Operator / Command	Meaning / Example
`||`	Logical OR between commands/tests. Example: `[ -f f ] || echo "missing"`
`-e`	File exists (any type). Example: `[ -e path ]`
`-f`	Regular file exists. Example: `[ -f file ]`
`-d`	Directory exists. Example: `[ -d dir ]`
`-r`	File is readable. Example: `[ -r file ]`
`-w`	File is writable. Example: `[ -w file ]`
`-x`	File is executable. Example: `[ -x script.sh ]`
`-s`	File exists and has non-zero size. Example: `[ -s file ]`

Notes: - Use `[ ... ]` (POSIX) or `[[ ... ]]` (bash). `[[ ]]` is more flexible (allows `==`, `<`, `>` without escaping and pattern matching). - Always quote variables in tests (e.g. `[ "$a" = "$b" ]`) to avoid word-splitting and errors with empty values. - Numeric comparisons require the `-` operators (`-eq`, `-gt`, …). Using `=`/`!=` compares strings, not numbers. - For floating-point comparisons use external tools (e.g. `awk`, `bc`) because `test`/`[` only handle integers. - Combine tests with `&&`, `||`, or use compound tests: `if [ -f a ] && [ -w a ]; then … fi`. - Use `(( ))` for arithmetic expressions: `if (( a > b )); then … fi` (no `$` needed inside). - Example: `if (( a > b && b >= 0 )); then echo "ok"; fi` - Example combining file checks: `if [ -f file ] && [ -s file ]; then echo "exists and not empty"; fi` 8) permissions: you may somethimes need to change permissions. To do this you can use chmod to change permissions for user, group and the rest. Using a common numbering scheme this means, e.g. `chmod 755` gives you all permissions, whereas everyone else may read and run it but not modify. (common for scripts)

Permission	Number	chmod shorthand
`---` (no permissions)	0	`---`
`--x` (execute only)	1	`--x`
`-w-` (write only)	2	`-w-`
`-wx` (write & execute)	3	`-wx`
`r--` (read only)	4	`r--`
`r-x` (read & execute)	5	`r-x`
`rw-` (read & write)	6	`rw-`
`rwx` (read, write & execute)	7	`rwx`

Setting up a Simulation for the combined Protein-DNA system 1J46.pdb

An MD simulation in GROMACS will always need 3 things 1) coordinates, which tell the program where is what (usually a `.gro` file) 2) a "topology" (yes it is defined different in Maths…). This contains the actual forcefield that tells the program how which molecule interacts with all the others and with itself. This can be nicely separated out in a pure `.itp` file for conciseness and to transport it over. Furthermore it wants to know (at the beginning) how the atomtypes are defined and how to deal with LR-interactions in the intra-molecular part. Also at the end you specify how many of which kind of molecules (in the order of the geometry `.gro`) is included. All this makes a `.top`file.

3) Instructions what you expect the program to actually do with those things. (Energy minimization/ MD run, how to deal with temparature (thermostat parameters), pressure (barostat parameters), constraints, how do describe LR-interactions, free energy dH/dlambda calculations,…)

Once you have all those, you can send the instructions to the "preprocessor"

```
gmx grompp -f instuctions.mdp -p topology_file.top -c starting_coordinates.gro -o name_how_runfile_shoul
```

This then checks everything for correctness and yields a `.tpr`file.

This you can then use for a simulation. To simplify the naming conventions and to prevent you from typing differnt stuff for energy, trajectory, log, etc. files you can use the `deffnm` option here, which uses the same name for everything and just changes the file endings. e.g. this starts a (verbose) run with the name chosen above. (HINT DO NOT WRITE .tpr here, that conflicts with other file endings.)

```
gmx mdrun -v -deffnm  name_how_runfile_should_be_called
```

The following are some common GROMACS mdrun output files

File ending	Description
`.gro`	Final finished output geometry: coordinates (and box vectors) in GROMACS `.gro` format, used as the new starting structure.
`.xtc`	Compressed trajectory (coordinates only, reduced precision) for visualization and analysis; small file size.
`.trr`	Full-precision trajectory containing coordinates, velocities, forces; much larger.
`.edr`	Binary energy file with time series of energies and thermodynamic observables; read with `gmx energy -f`.
`.log`	Text log of the mdrun run: settings, progress, performance summary, warnings and errors.
`.cpt`	Checkpoint file for restarting runs: binary snapshot of simulation state (step, random seeds, integrator state); use with `-cpi` to continue.

### pdb preparation

First get the `1J46.pdb` file from the protein database or our course.

You can quickly have a look both at the visual structure (using `vmd 1J46.pdb`) and the actual text file (e.g. using `more`, `less` or an editor like `vi`, `vim` or `nano`).

Think of what protonation state do you expect at neutral conditions? (pH=7.2) What are the amino-acid pKs Values for this? - Does this correspond to your assumptions?

You can use `https://server.poissonboltzmann.org/pdb2pqr` to get an automated generation of charge states of each individual pH for the amino acids. Still you have to be careful and you definitely should check its results.

First you want to split the (mixed) pdb file, which contains 2 DNA parts and a protein part. For this you only need the respective `ATOM` parts in their individual files. To speed this up you can use the privided `split_pdb.py` script. This is a very simple python script that iterates over the lines and separates at the separator. Alternatively you can of course also use a texteditor.

You should be left with 3 `.pdb` files. (Named `part_0.pdb`, `part_1.pdb`, `part_2.pdb` if you used the script to do this.)

Next you want to obtain both a geometry and a "topology" (molecule specific forcefield with the ff-parameters of your chosen force field).

Force field paramterization and geometry files for the system For this part you want to use `gmx pdb2gmx`. This gromacs program allows you to write a topology based on a chosen forcefield and a geometry based on a provided pdb.

For this make shoure you have the `amber99bsc1` folder in your current working director or to know the full path to where you put it. Now make 3 preliminary topologies and geometries based on your pdb inputs. It is recommended to use separated folders. Usage of `gmx pdb2gmx` is

```
gmx pdb2gmx -f pdbfile -ff path/to/forcefield -o output_for_geometry.gro -i output_for_topology.itp
```

Therefore you can use

```
 gmx pdb2gmx -f part_0.pdb -ff amber99bsc1 -ignh -o  helix_part/helix_pt1.gro -p helix_part/helix_topol
i helix_part/posre_h_pt1.itp
```

and choose the option for the tip3p water model.

If you need to do this multiple times you can also echo the selection of your water model. e.g. assuming 1 corresponds to tip3p:

```
echo "1" | gmx pdb2gmx -f part_0.pdb -ff amber99bsc1 -ignh -o  helix_part/helix_pt1.gro -
p helix_part/helix_topol_pt1.top -i helix_part/posre_h_pt1.itp
echo "1" | gmx pdb2gmx -f part_1.pdb -ff amber99bsc1 -ignh -o  helix_part/helix_pt2.gro -
p helix_part/helix_topol_pt2.top -i helix_part/posre_h_pt2.itp
echo "1" | gmx pdb2gmx -f part_2_copy.pdb -ff amber99bsc1 -ignh -o protein_part/protein.gro -
p protein_part/protein_topol.top -i protein_part/posre.itp
```

You see that the `.gro` files still contain the same coordinates as before. That makes it simple to recombine them into files for your simulation. For that just copy the coordinates to a new .gro File. If you instead use `gmx insert molecules` this will put them randomly into the box without preserving their coordinates.

For our simulation we explicitly want to preserve the helix. It would not do to have two separated strands nanometers from one another! To make the combined `.gro` you need to also set the number (line 0) equal to the actual atom number in the file. (It is possible to renumber the atoms starting from 1 to end with `gmx genconf`but not necessary.)

So next make a `.gro` File consisting of the whole helix (or the helix and the protein).

For the corresponding topology you also want to combine all systems into 1 file. You can use this irrespectively how many of strand1, strand2 and protein residues it contains, because you can just set the number at the very bottom.

To obtain this this make a new file `system.top` (or however you want to call it). To have the ff paramters you should start with the header from ‘./amber99bsc1.ff/forcefield.itp. If you have it in a subfolder, then you should of course change the import to the corresponding subfolder of course.

First use the pdb2gmx generated topologies and run `python3 top_to_itp.py --file old.itp > new.itp`. The corresponding `new.itp` files have been cleaned of everything that is a top but not strictly a molecule-specific `.itp` In this file you want to import the corresponding itps generated previously.

Now you still need to add the following, to include the water model, the ion-interactions and possible position restraints. In the `[ system ]` section you can give your topology any name you want. It has no influence on the actual simulation. Opposed to this the `[ molecules ]` section is extremely important. Here you write how many of which of your molecules are within the geometry. Be careful to add the molecules in the correct order. You can also have multiple of the same residues, but they NEED to be in the very same order as in your geometry. e.g. For a system with SOL, PRO and UNL residues:

```
SOl 100
PRO 1
SOL 150
UNL 2
SOL 1
```

opposed to

```
PRO 1
UNL2
SOL 251
```

The latter would expect your system to start with the PRO residue, followed by 2 UNL and 251 SOL residues. You can find such errors later by going over the "Warnings" of `gmx grompp`, but if they get ignored you may end up destroying your system, because the apparent interactions are between the wrong indices of atoms.

So your system topology (e.g.`system.top`) needs to contain also the water forcefield (in our case tip3p) and the ion force field that you want to use. Go ahead and add the following to your `.top` file (replace .itps with however you named your pure `.itp` files.) And also adapt the number of residues and their order in the `[ molecules]` section according to the `.gro` file that you want to use.

```
; Include water topology
#include "./amber99bsc1.ff/tip3p.itp"

#include "YOUR RESPECTIVE NEW .itp file1"

#include "YOUR RESPECTIVE NEW .itp file2"

#include "YOUR RESPECTIVE NEW .itp file3"

#ifdef POSRES_WATER
; Position restraint for each water oxygen
[ position_restraints ]
;  i funct       fcx        fcy        fcz
   1    1       1000       1000       1000
#endif

; Include topology for ions
#include \"./amber99bsc1.ff/ions.itp\"

[ system ]
; Name
Protein and DNA strands

[ molecules ]
; Compound        #mols
DNA_chain_B         1
DNA_chain_C         1
Protein_chain_A     1 ; the order has to be the same as in the gro file!
```

Next first check your geometry visually with vmd `vmd your_geometry.gro`. (once vmd is running type `pbc box` into the command line. This will show you the pbc-box. Is the boxsize ok? The largest distance between two interacting points needs to be less than half the box length - to avoid self-interaction. Even using cutoffs, it is really recommended to make the box more than twice as large in each direction that your largest molecule! If the box is too small you can edit it either manually or using `gmx editconf`.)

If it looks fine, you can also already use the `example.mdp` to initally grompp your system, even if it does not yet contain solvents, just to check whether all the steps so far seem correct. If you get just a warning that the atomnames differ, and it is about different naming schemes of the SAME atoms, you can ignore this by adding a `--maxwarn 1` (ignore 1 warning in grompp). If you get no further warnings or errors, next we will add the solvent. As the forcefield parameters are already speciefied in the topology, you just need to actually add the solvent molecules to the `.gro`file and add the number and order in the `[ molecules ]`section of the `.top`file. You may want to copy your `.gro` and `.top` files for this.

You can now use `gmx solvate` to add water to the system. Because tip3p is a three-point model, you do not need to change the settings from the default (`spc216.gro`), but you should specify your geometry file and your topology file to modify.

It should look like this:

```
gmx solvate -cp full_system.gro -o solvated_full_system.gro -p  full_system.top
```

HINT: ALL GMX commands have their own manual page: e.g. https://manual.gromacs.org/current/online-help/gmx-solvate.html (online) or in the command line via the `man` command. You can (and should!) always check the options for all commands that you are using.

Now look at your system again using vmd.

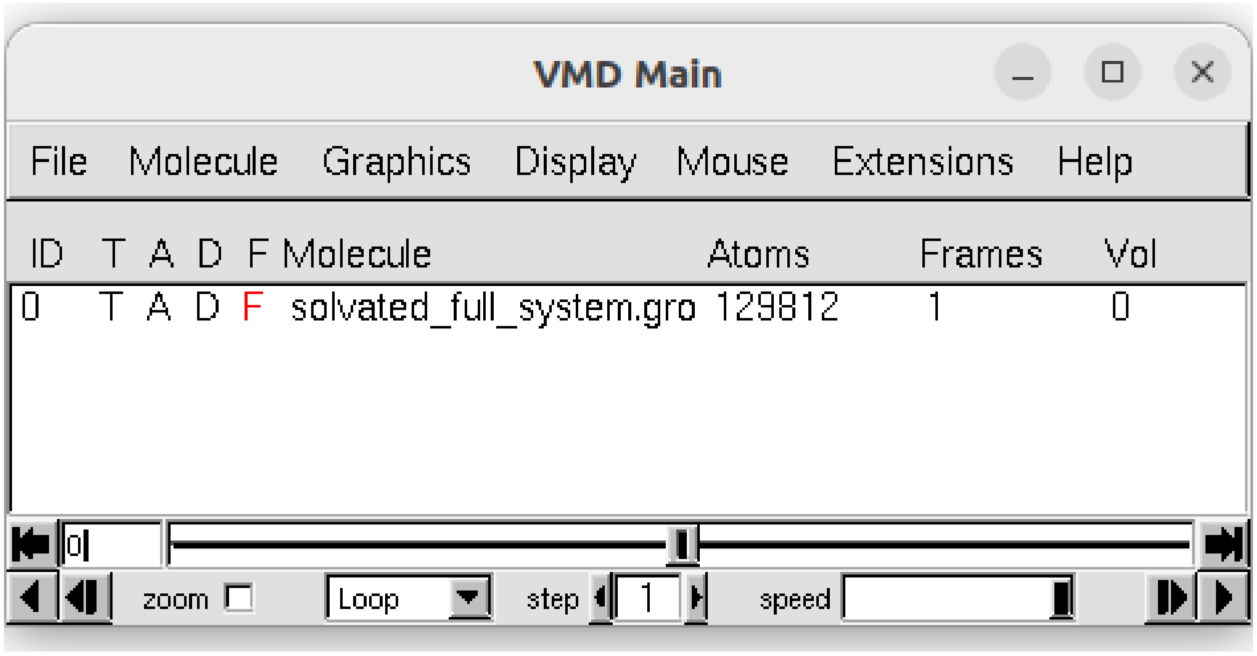
Interlude some quick overview over VMD (VISUAL MOLECULAR DYNAMICS) VMD is a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting. VMD supports computers running MacOS X, Unix, or Windows, is distributed free of charge, and includes source code. (https://www.ks.uiuc.edu/Research/vmd/)

While this is an extremely powerful program that can also be directly be used as a frontend for another MD software, we will only be using it for visualization purposes in this course.

The program can be opened either by itself `vmd`, which requires you to load data manually via the GUI, or you can also directly load data upon program call `vmd grofile.gro` - When you later read trajectories, you will see that `vmd grofile.gro trajectoryfile.xtc` is required, because the pure trajectory file only contains information about coordinates, but not atomtypes, residues etc.

Upon opening your solvated system you should see something like this:

The Main window is the actually controlling window. The Graphical Representation controls how stuff is represented (it can also be closed and reopened from the Main without any issues.)

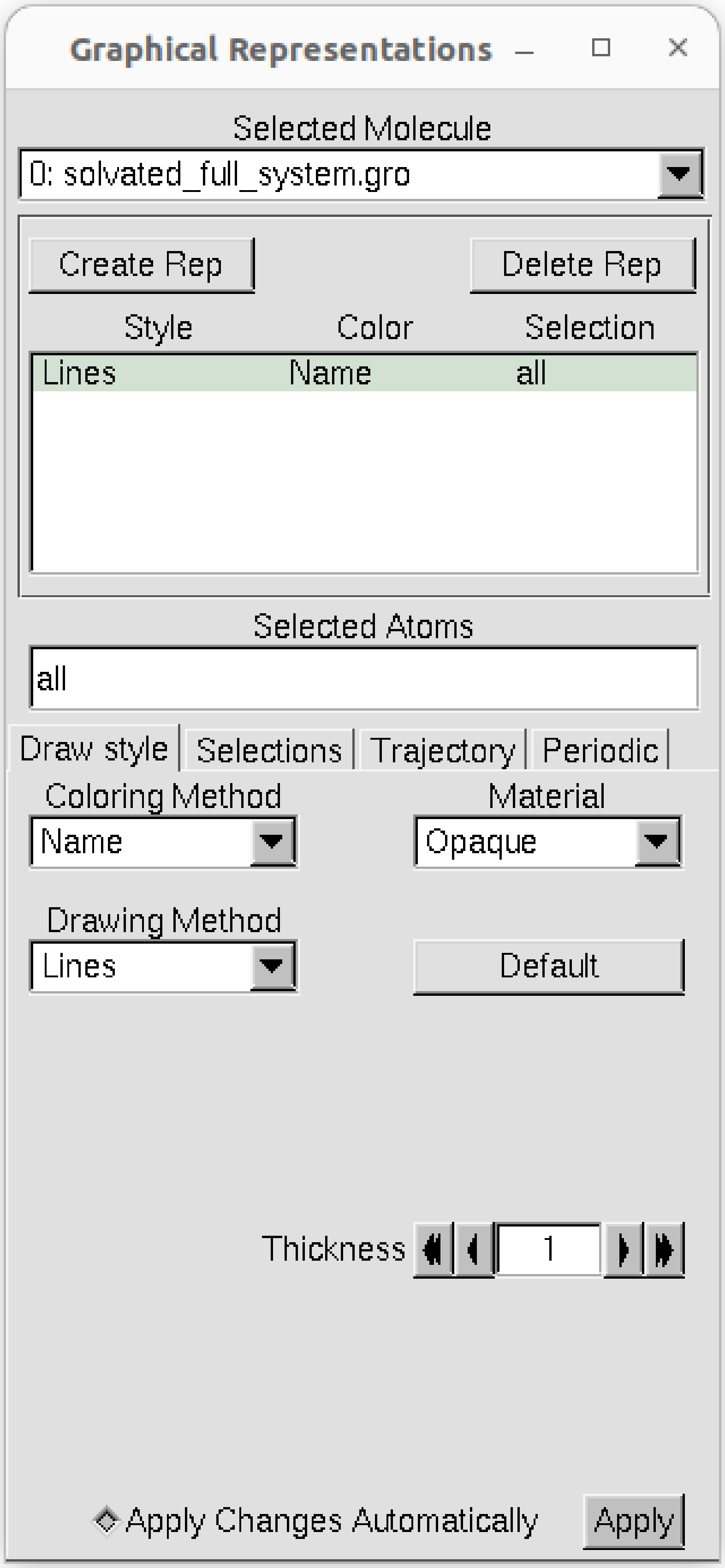


ID	T	A	D	F	Molecule	Atoms	Frames	Vol
0	T	A	D	F	solvated_full_system.gro	129812	1	0

Frame: 0, step: 1

Figure 1: Main Window

Unless other settings have been saved to the `.vmdrc` file, the default will be to show everything (connected by lines). If you need to load data into your molecules you could do so via the “Main” window (you will need that later.) As an example you will now see how to select residues/molecules - for now select only the



Selected Molecule: 0: solvated_full_system.gro

Style	Color	Selection
Lines	Name	all

Selected Atoms: all

Coloring Method: Name

Material: Opaque

Drawing Method: Lines

Thickness: 1

◇ Apply Changes Automatically

Figure 2: Graphical Representations Window

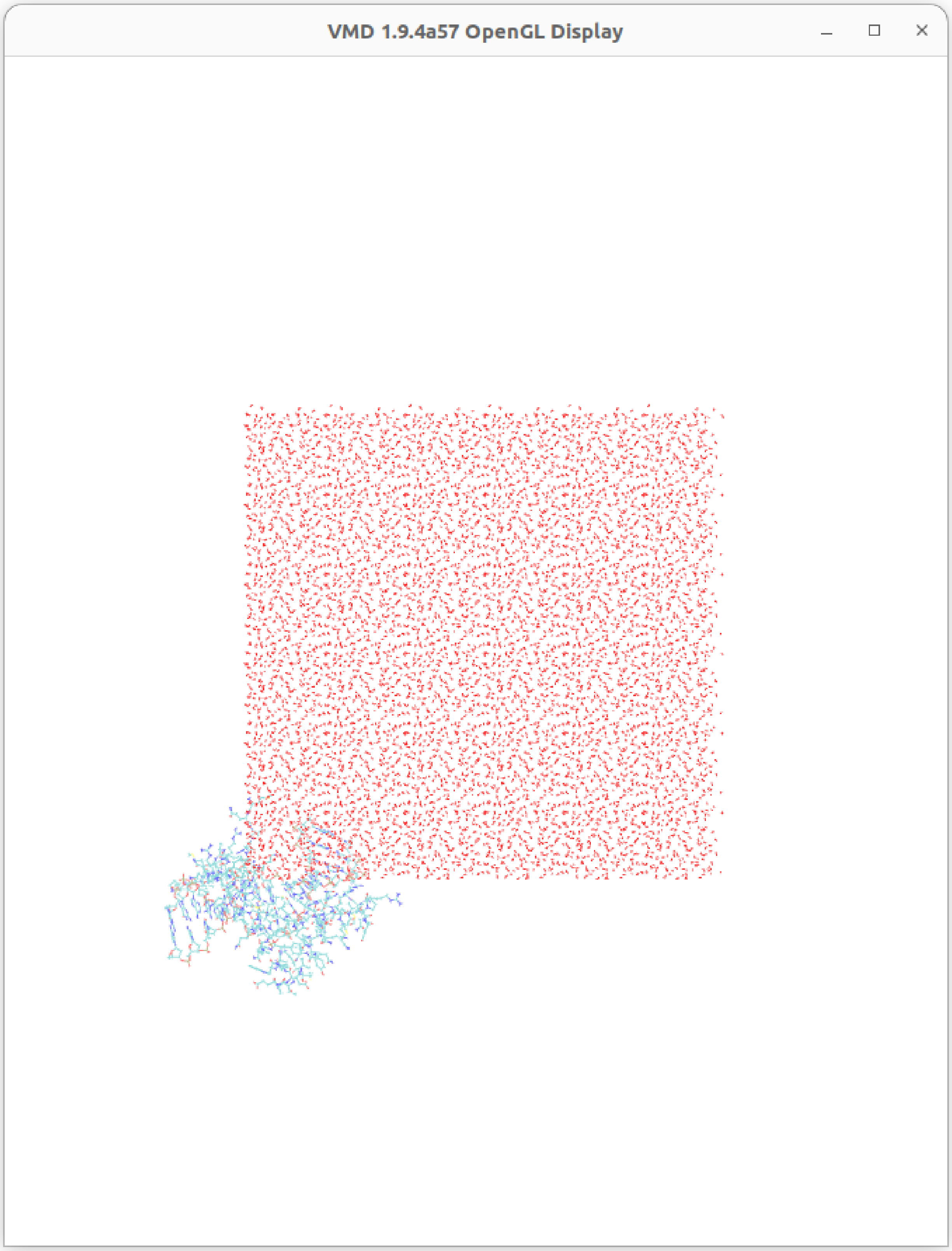
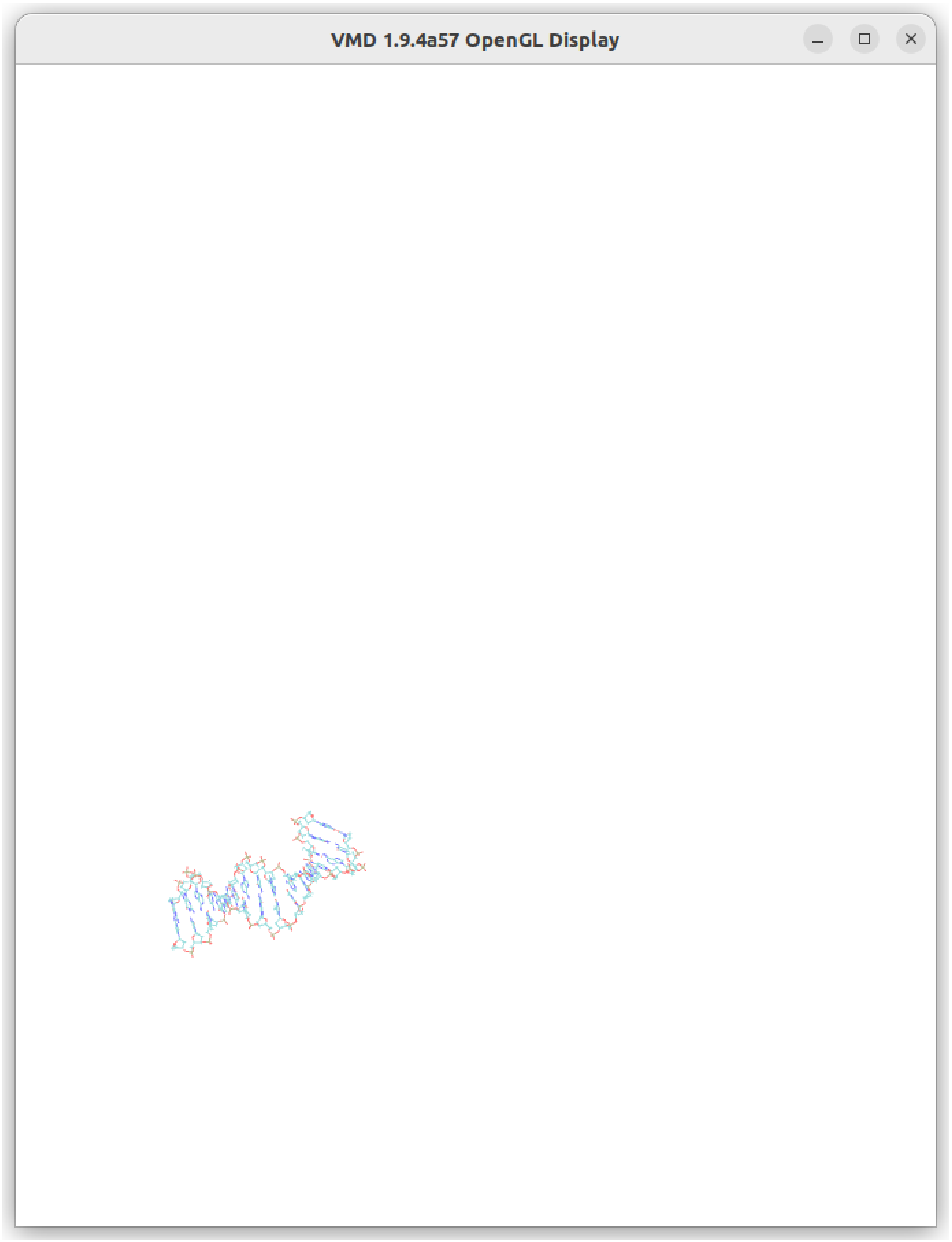


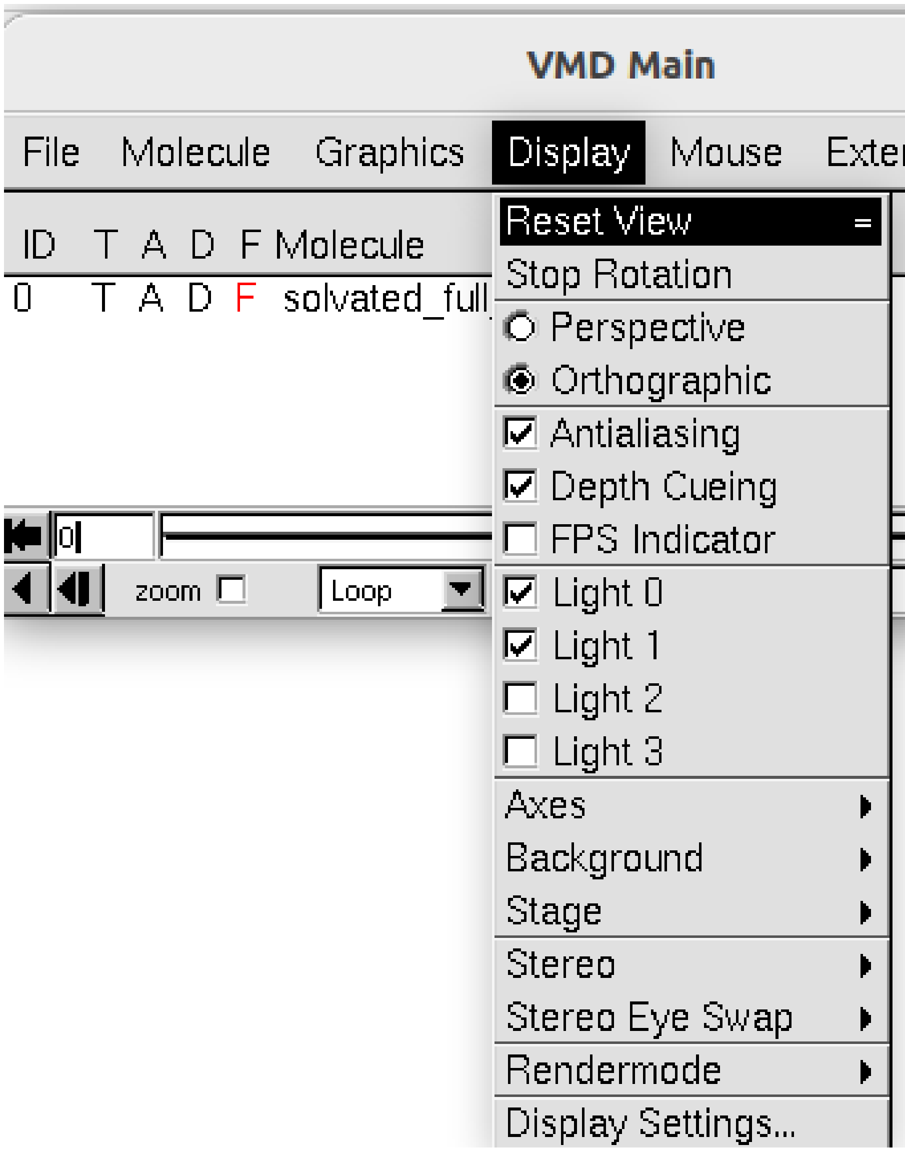
Figure 3: OpenGL Display

`resname DA DC DG DT` which should correspond to your helix. Typical selections are `name` (atomic names), `resname` (residue names - e.g. amino acids, base pairs), `resid` (id of residue), `serial`(index).

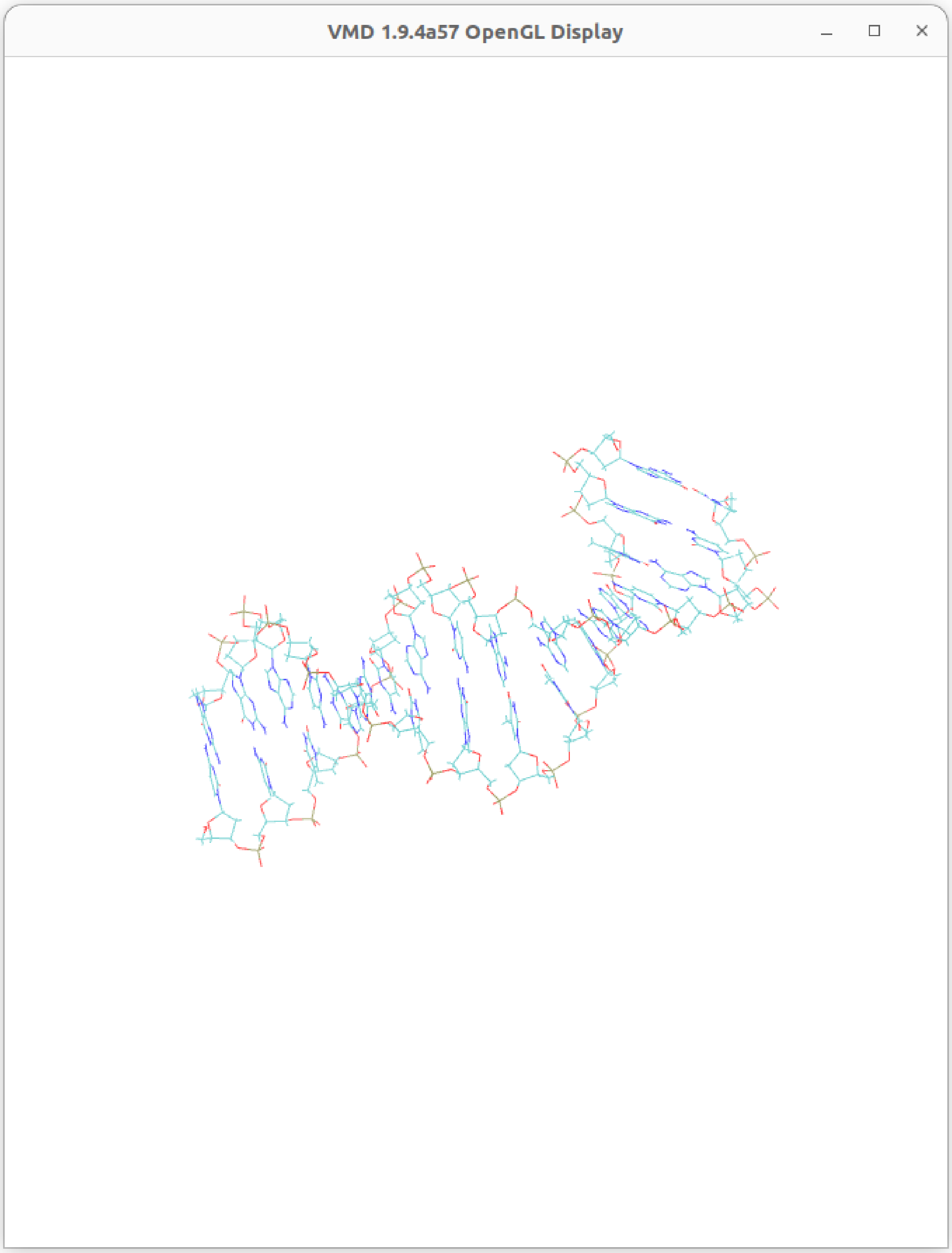
Further HINTS: Logical operations are possible and are used by adding `or` or `and`. Inverting a selection with `not` is also useful (e.g. `not resname SOL`for displaying all non-water.) Maybe you also want to disply the environment (especially useful for H-bond selections) of a group. For this you can use `within X of`(e.g. `(within 5 of resname DA DG DC DT) and (resname SOL)`). If you do not want to cut their residues, you can also use `same resid as`. Using `pbc box` in the vmd console you can show the periodic box. If you later display a non-centered system do not use `Lines`as the Drawing Method. Instead use `DynamicBonds`.



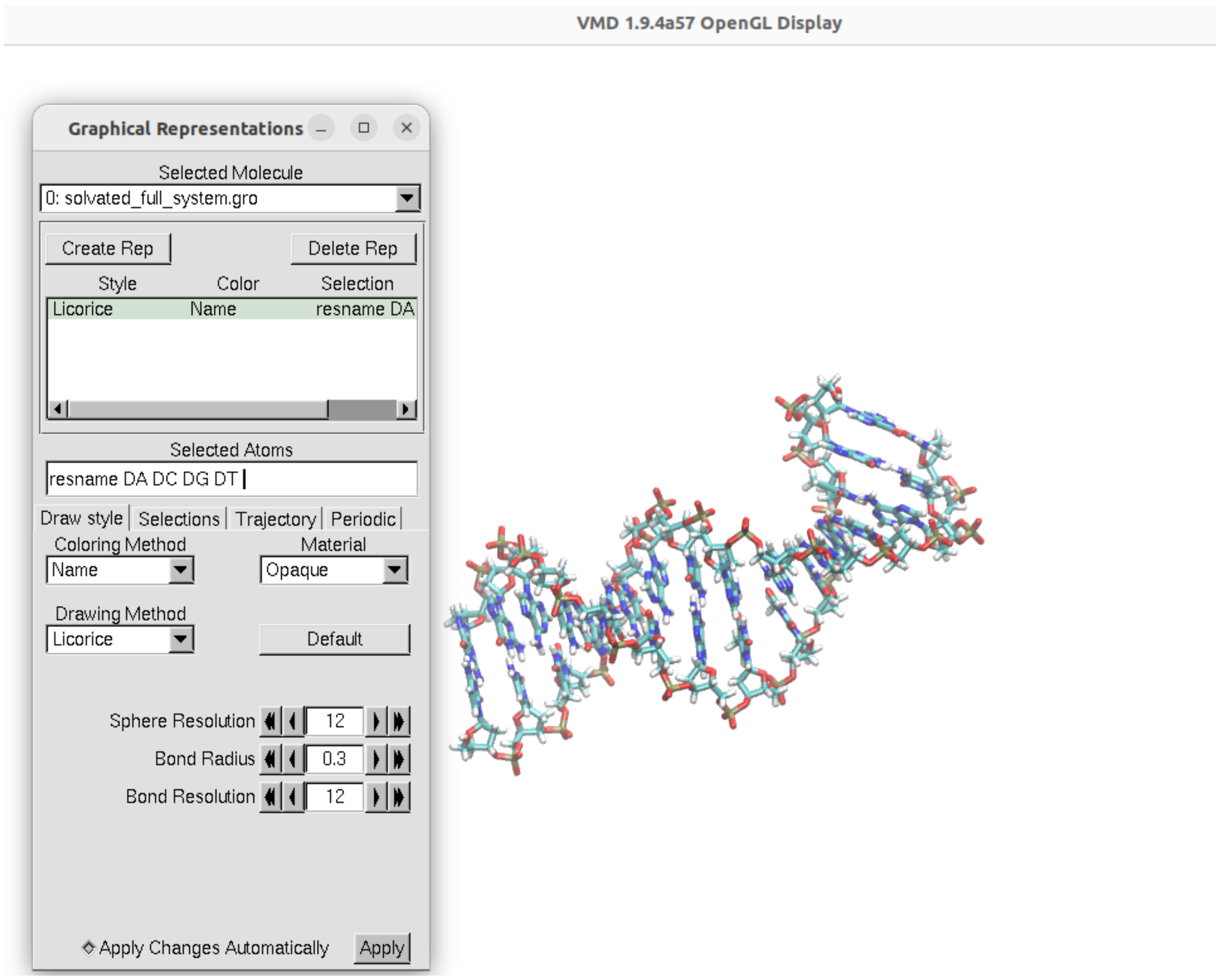
As you see the display stayed the same. This is inconvenient: By using Display - Reset View in the Main window



you can easily center on your desired fragments.



Your Drawing Method is still lines, you can change it in the Graphical Representations window. This may help for visualization purposes or also to render publication quality pictures. (We will not talk about this now. If there is time at the end of the course we can do this.).


VMD 1.9.4a57 OpenGL Display
Graphical Representations
Create Rep
Delete Rep
Draw style
Selections
Trajectory
Periodic
Default
Apply Changes Automatically
Apply

Selected Molecule: 0: solvated_full_system.gro

Style	Color	Selection
Licorice	Name	resname DA

Selected Atoms: resname DA DC DG DT

Coloring Method: Name

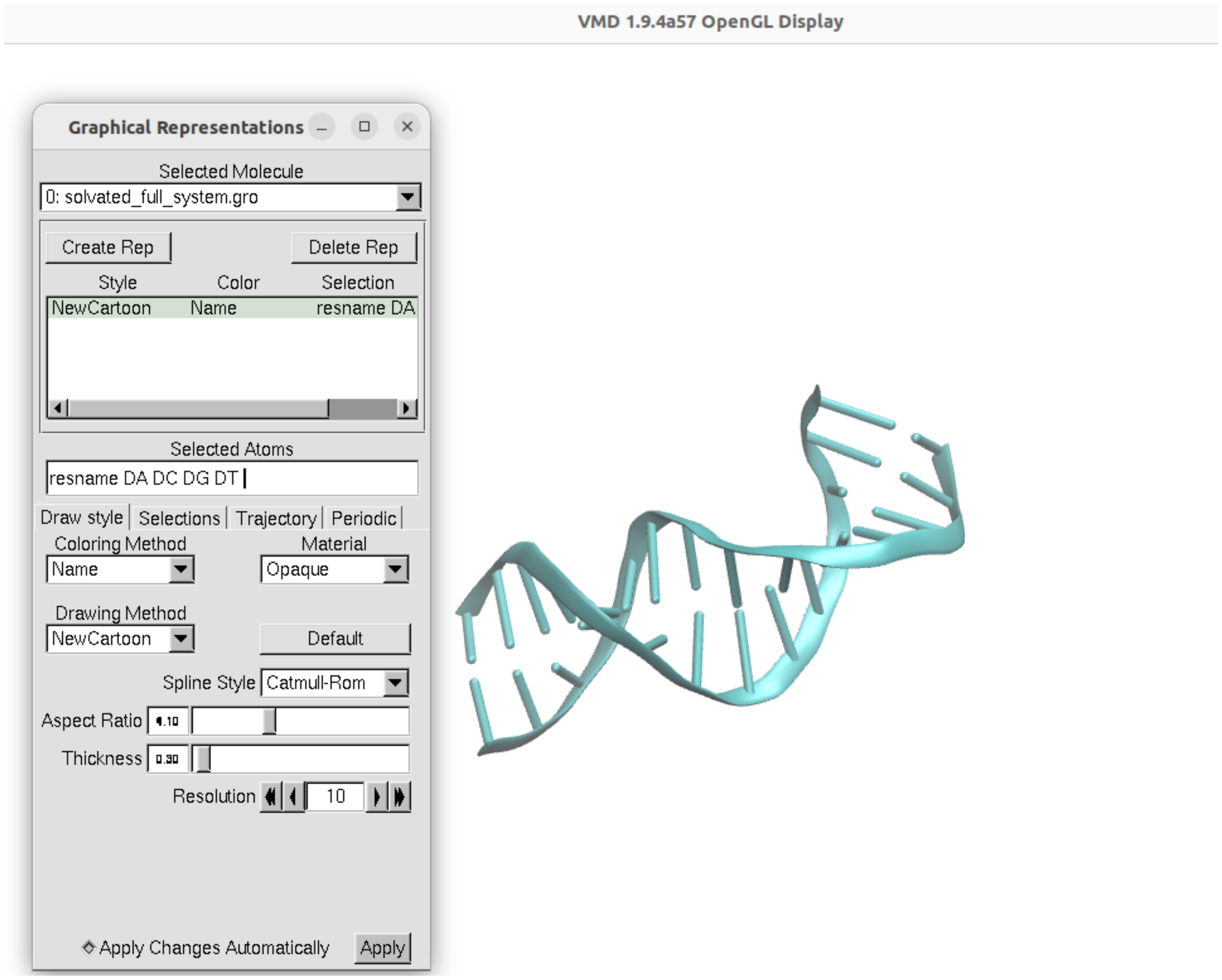
Material: Opaque

Drawing Method: Licorice

Sphere Resolution: 12

Bond Radius: 0.3

Bond Resolution: 12


VMD 1.9.4a57 OpenGL Display

Graphical Representations

Selected Molecule: 0: solvated_full_system.gro

Style	Color	Selection
NewCartoon	Name	resname DA

Selected Atoms: resname DA DC DG DT

Coloring Method: Name

Material: Opaque

Drawing Method: NewCartoon

Spline Style: Catmull-Rom

Aspect Ratio: 4.10

Thickness: 0.30

Resolution: 10

You can also use the `Coloring Method` to e.g. manually color selections, color residues differently,...

Back to the system preparation If your system looks fine, you can now go to prepare the actual simulation.

So far we have obtained 2 of the 3 required inputs. (Geometry and topology) We still need the (`.mdp`) actual instructions what GROMACS is supposed to do with those.

For this you can use the prepared files `steep.mdp`, `nvt.mdp`, `npt_ber.mdp` and `npt.mdp`. In principle (starting from an equilibrium configuration) only the actual instructions for the simulation in the isothermal-isobaric ensemble (`npt.mdp`) would be necessary. However, we do not know how good our starting guess is. Therefore, we try to prepare our system. 1) Using an energy-minimization to get rid of high energy clashes, that would cause huge forces, tearing our system apart. (the corresponding .mdp can have other options, which will be ignored.) 2) Similarly running a canonic ensamble simulation should help distribute the molecules better, preventing the barostat from exploding. 3) Using the Berendsen barostat first, (stable, but not quite the correct ensemble) helps to get the system slowly into a region of phase-space that is already close to the npT ensemble. 4) Using the Parrinello-Rahman barostat gives the correct ensemble, but may explode if the system is not yet well prepared. (if you experience periodic fluctuations of the boxsize in your simulations in the future this may be the cause, for this check a fourier transform as a test.)

Now do an energy-minimization, and the equilibration steps and then run the simulation. For this (as mentioned above) first run `gmx grompp` and then `gmx mdrun`

The final production you will not do on your local machine.

For this we will now discuss how to run your systems on the cluster.

Alternatively – For analysis you may be given a finished system.

Connecting to another machine via ssh Normally you are your user at your local machine. Now you want to run this on another one.

You will connect to int-nano.int.kit.edu using the username username by using

```
ssh -Y username@int-nano.int.kit.edu
```

There you should see your folders. Make a new folder for this course and if you need to remove/move stuff in this course consult your advisors. This is a group machine and we do not want you to accidentally delete data of other users.

you can then copy your inputs from the course pc via `scp` (secure copy) or `rsync`.

Those work by

```
scp target destination
```

or

```
rsync target destination
```

for copying whole folders it is recommendet do use a dry-run first and then run the command again without this option if it looks correct.

```
rsync -a --progress --verbose --dry-run target destination
```

Normally you submit scripts on a cluster using a queuing system. For the course however we have blocked a node, so you can directly run here.

Connect (via ssh) from the login node to the reserved node on the cluster (`ssh user@reserved_machine`) - you can run your gmx run there. For this you should specify how many resources you can use with the assistants.

(This will change the `mdrun options` slightly, e.g. using `-nt`, `-pin` etc.) Before running the jobs here for the first time please get an ok of the assistants.

Generating an index file

Index files allow for a more in-depth selection for later analysis or other operations.

The simples way to generate one based on logical selections is using `gmx make_ndx` - this requires a `.tpr` for full functionality or a `.gro` for reduced functionality. To also use indices that have been defined previously use the `-n` option.

```
gmx make_ndx -f tprfile -n pre_existing_index_file -o new_indexfile
```

You can also simply write the indexfiles by yourself (usually using a script for whatever you need).

The structure is for a group you want to define as "SEL" where the indices correspond to your atom indices.

```
[ SEL ]
 index1 index2 ....


[ OTHERSEL ]

indexS1 indexS2
```

We will use such a ndx file to make our visualisation of the final trajectory easier on our eyes.

Excursion: the .mdp file - Instructions for your GROMACS run

Like with most of the gromacs human-readable files, you can write comments with a ";" here. That also means it is simple to prepare a single `.mdp`file that contains your basic run instructions and modify it for your current runs.

Let's look at some options:

```
integrator              = md
nsteps                  = 500000
nstlist                 = 10
nstcomm                 = 50
dt                      = 0.001
nstxtcout               = 500
nstcalcenergy           = 500
```

`integrator = md` means a Leap frog integrator is used for integrating Newton's equations of motion. It is also specified how many steps the simulation is supposed to run (`nsteps`) and how long each timestep is (`dt` (ps)). nstxcout means the coordinate output is written in compressed form every n steps. (if you need velocity or force use `nstvout`and `nstfout`. We do not do this by default, as this triples the storage space needed.) `coulombtype                 = PME`means we are using a parallel form of ewald summation (particle mesh ewald). For the long-range continuation we use the analytic continuation to the energy and the pressure (`DispCorr                 = EnerPres`).

Temperature gets managed with the v-rescale thermostat.

```
tcoupl                  = v-rescale
tc-grps                 = system
tau-t                   = 1.0
ref-t                   = 300
```

and pressure can be controlled by the barostat. Apart from the number of simulation steps this is the main thing you need to change during your equilibration runs opposed to the production run.

```
pcoupl                  = no;  Berendsen; Parrinello-Rahman
pcoupltype              = isotropic
tau-p                   = 1.0
ref-p                   = 1.0
compressibility         =    4.5e-5
```

To speed up our calculations, we use constraints on our bonds using LINCS (do not use this for planar ring-systems without thinking).

```
constraints             = all-bonds ;h-angles
constraint-algorithm    = LINCS
lincs-order             = 4
lincs-iter              = 2
verlet-buffer-tolerance = 0.0001
```

The following describes how to deal with the real-space cutoff and declares the whole system as a single energy group (for gpu parallelism). If you want to separate nergy groups afterwards use `gmx mdrun -rerun`options with individual energy groups. As this runs on GPU vs CPU you will save most time this way

```
rlist =1.2
rcoulomb =1.2
rvdw                      = 1.2
vdwtype =cutoff

energygrps = System
```

Those options are only for a very basic run. More advanced options can (like everything) be found in the documentation. **A**lways remember **R**ead **T**he very **F**ine **M**anual (**RTFM**)

Displaying multiple frames of the trajectory without diffusion

Select the DC basepairs and then use `gmx trjconv` with the option `-fit rot+trains` to obtain a trajectory where those groups are fitted on top of another (removing its diffusion and rotation.) This allows you to display multiple frames with vmd (change `now` to `startframe:endframe`) for a part of the trajectory and use a slight smoothing factor in the "Trajectory" section of the Graphical Representations window of vmd. This directly shows you how flexible different parts of your molecule are (if your fitted selection is rigid - e.g. a protein backbone.)

If your selection is broken over periodic boundary conditions during the trajectory, you should first center it into the box. This is also something that you can do with the `gmx trjconv` options. The simplest way for this is ususally `-pbc cluster` and `-center`.

NOTES TO MYSELF

`make_structure_for_gromacs.sh`so far generates the topology (CHARGES NOT YET SET TO ZERO-DEPENDING ON WHAT WE WANT TO SIMULATE EITHER SIMULATE AT PH, chargestate etc)

todo: once finished make this as instructions, so the students do this in part by line and others they can execute

`run_mds.sh`is a script that they will have to do themselves- not for the course but for our usage

Course graining (day 2)

Data evaluation (day 3)