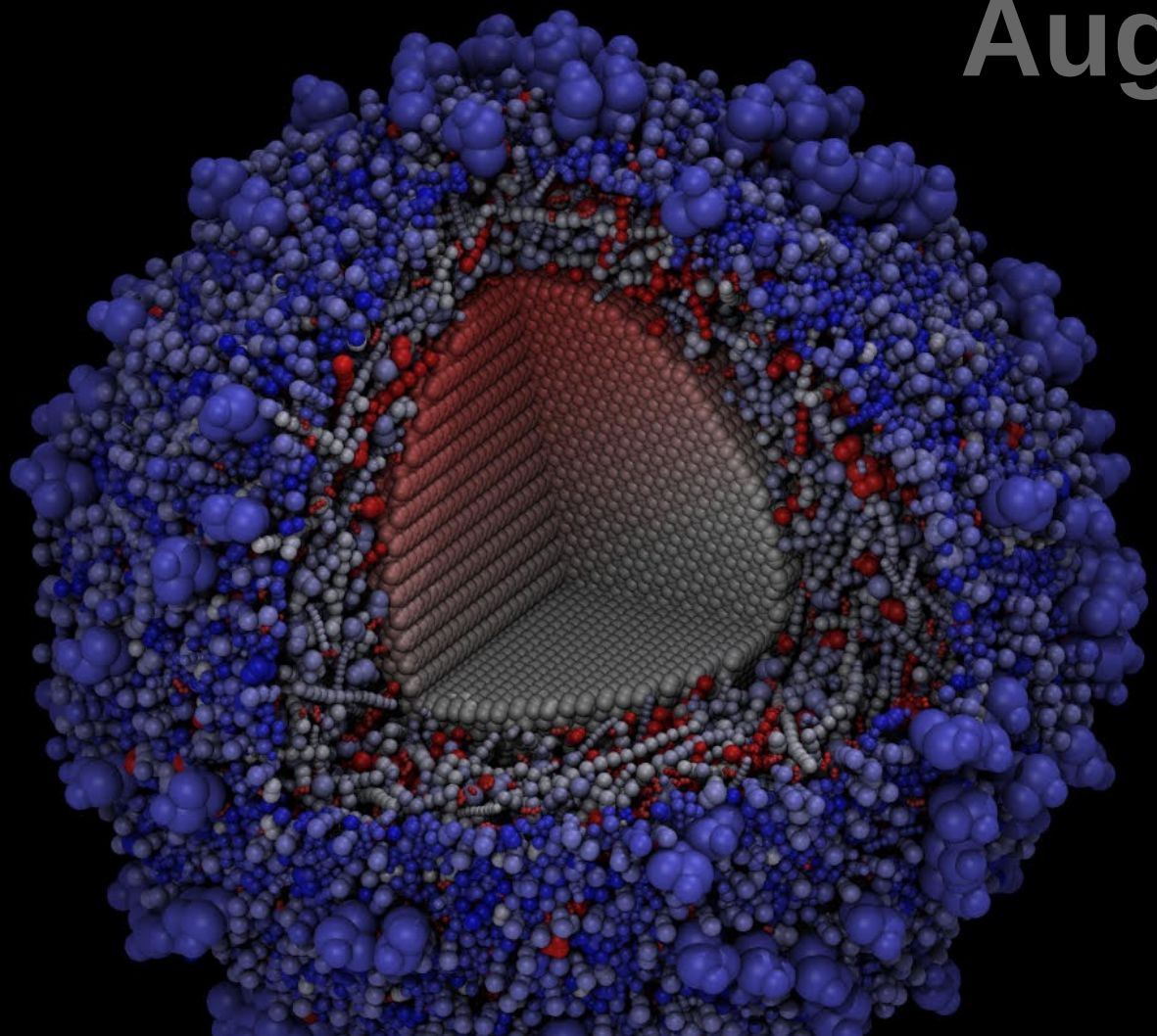
“Creating molecular assemblies and force fields with Moltemplate”

Andrew Jewett **LAMMPS workshop**

August 15th, 2019



Scripps Research

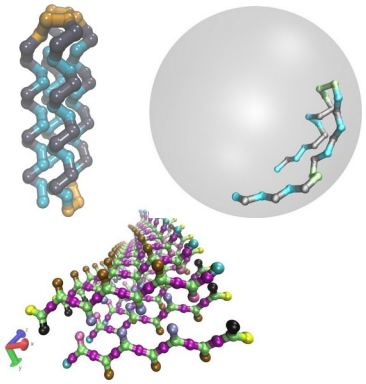
Talk slides, examples, docs:
are at *moltemplate.org*

Contact me at:

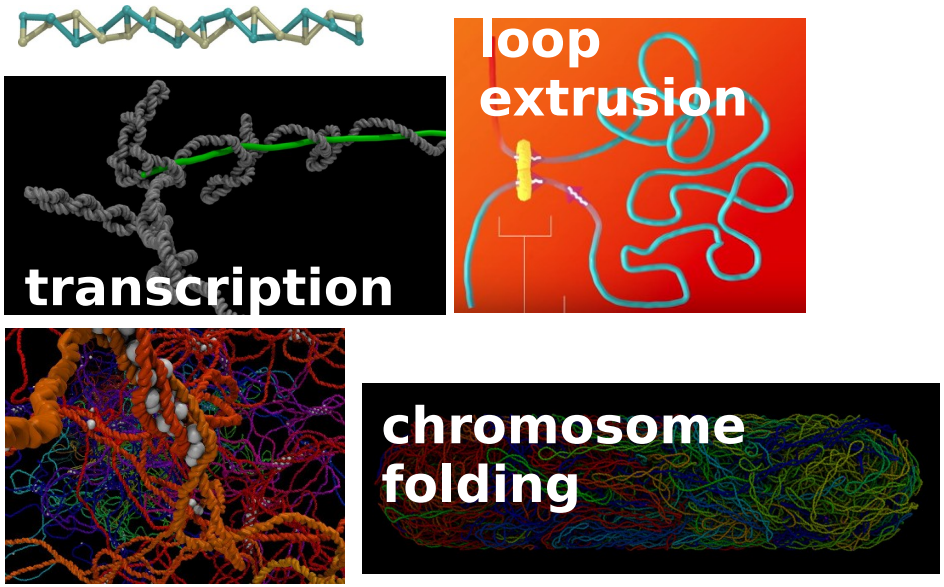
jewett.aij@gmail.com

Coarse grained systems built with MOLTEMPLATE:

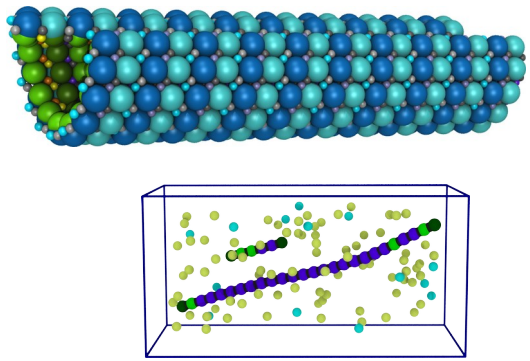
Proteins



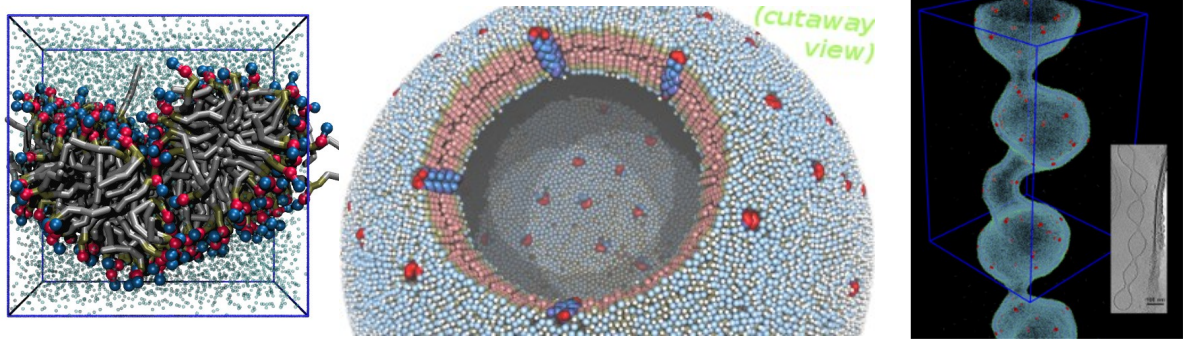
DNA



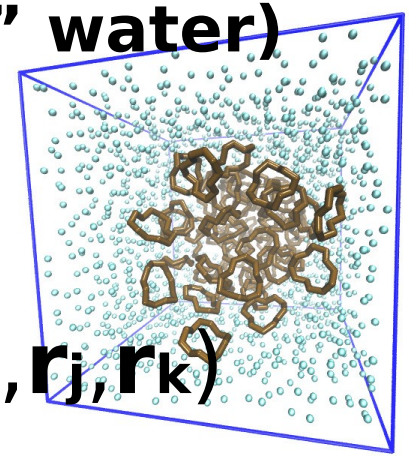
Cytoskeleton



Lipids, Vesicles, Membrane Proteins

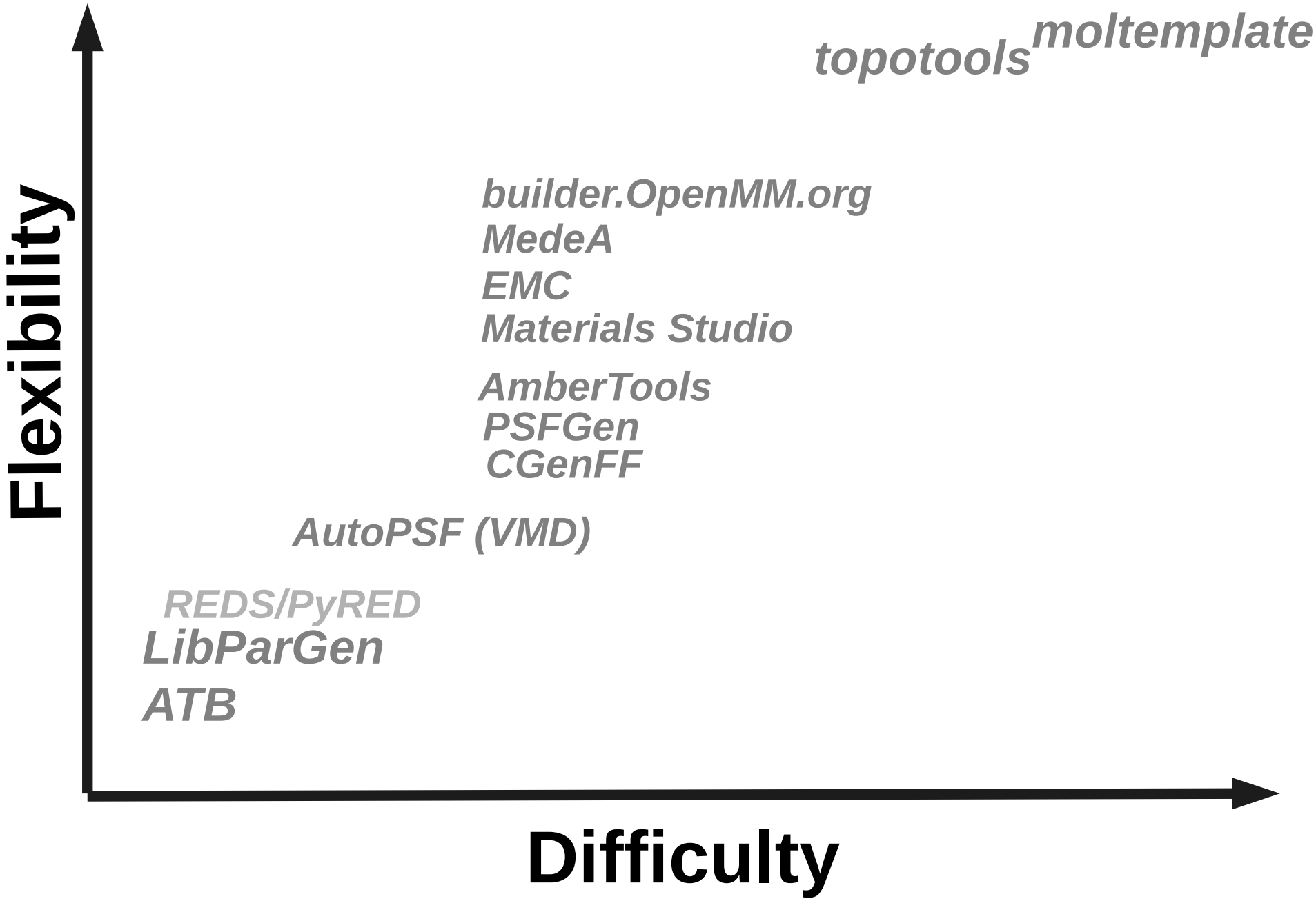


Many-body systems (eg. "mW" water)

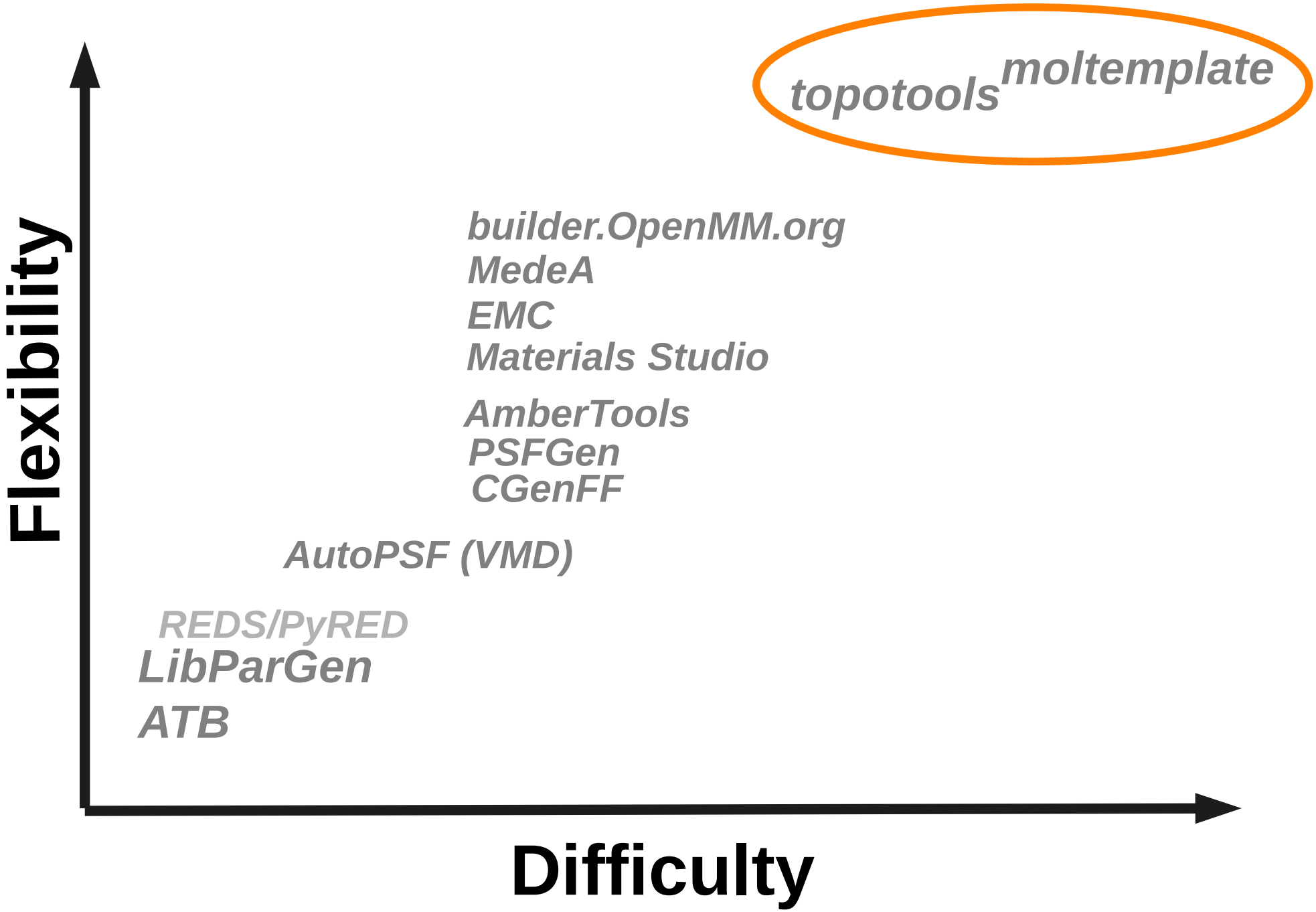


$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

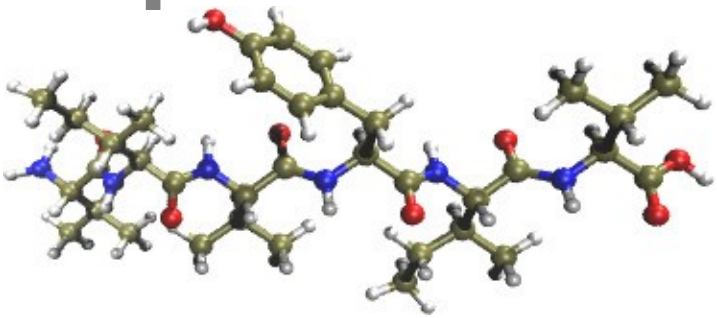
Tradeoff


topotools
moltemplate
builder.OpenMM.org
MedeA
EMC
Materials Studio
AmberTools
PSFGen
CGenFF
AutoPSF (VMD)
REDS/PyRED
LibParGen
ATB
Flexibility
Difficulty

Tradeoff

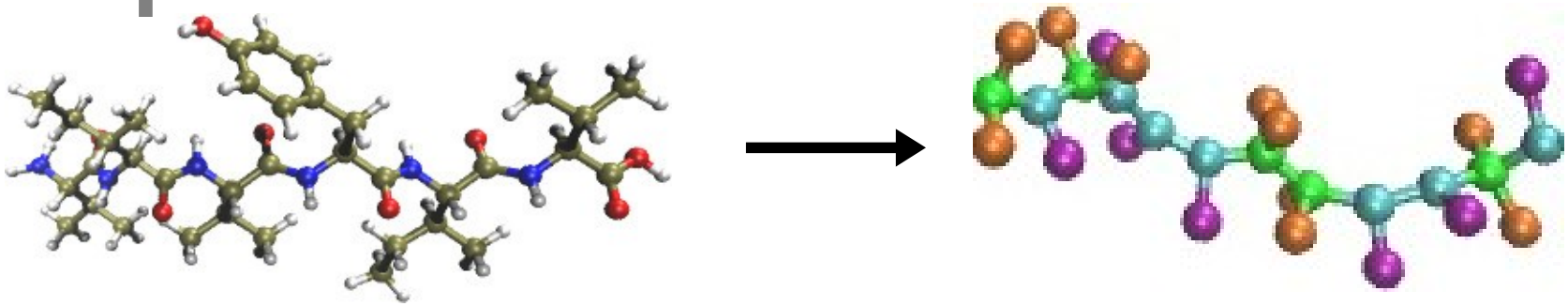

topotools
moltemplate
builder.OpenMM.org
MedeA
EMC
Materials Studio
AmberTools
PSFGen
CGenFF
AutoPSF (VMD)
REDS/PyRED
LibParGen
ATB
Flexibility
Difficulty

Diverse zoo of exotic molecular representations available in LAMMPS



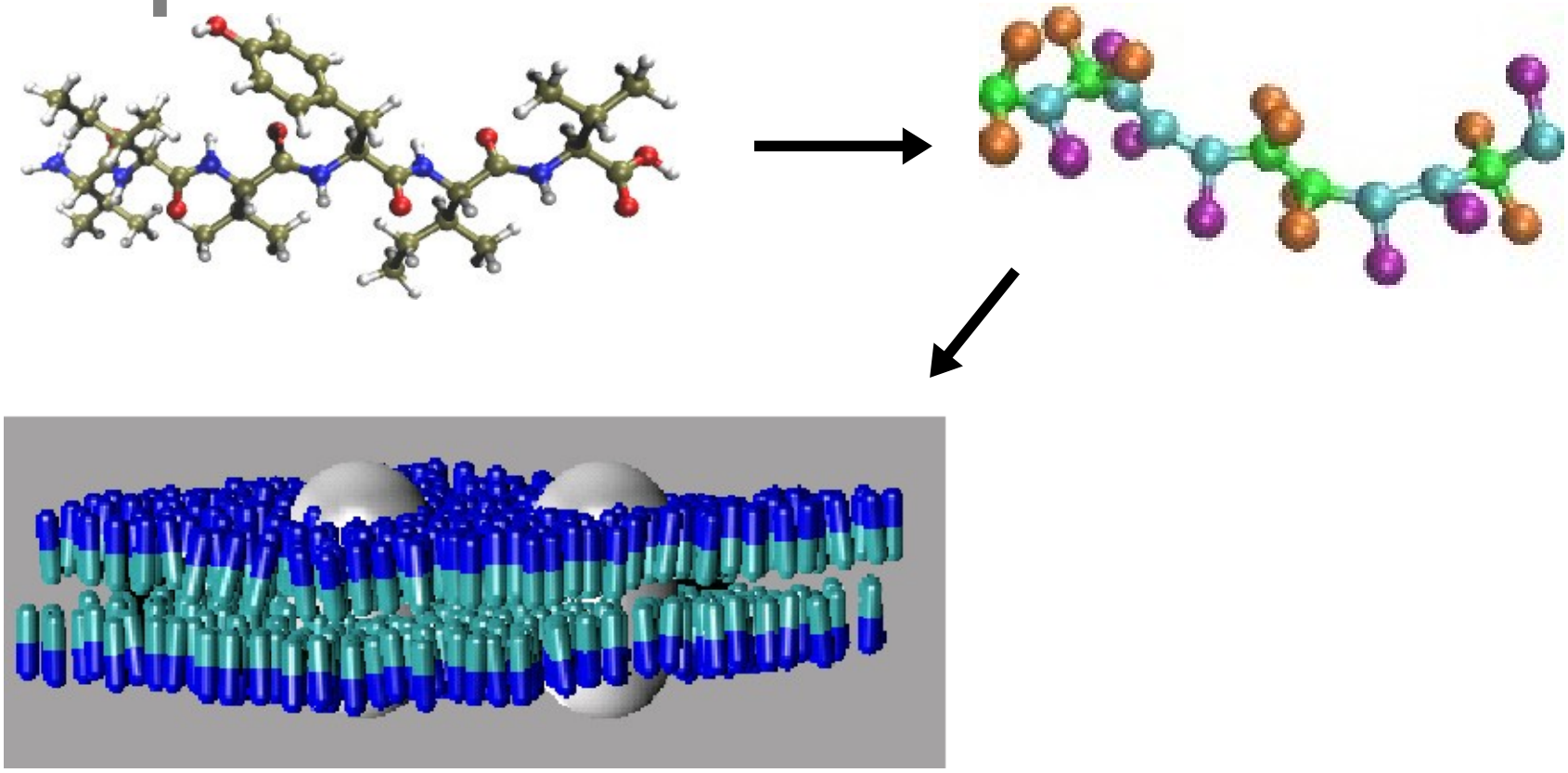
All atom models (explicit solvent)
(particles represent individual atoms)

Diverse zoo of exotic molecular representations available in LAMMPS



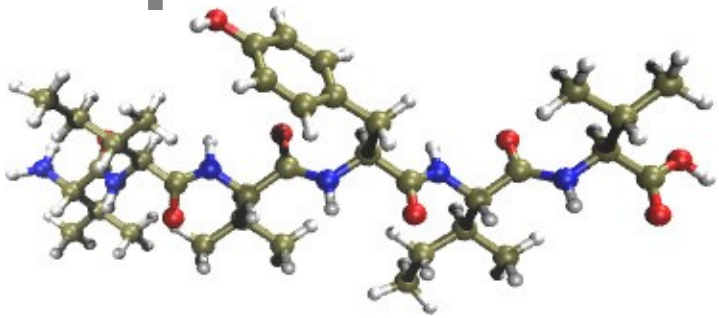
Reduced atom models
(crude implicit solvent, typically)
particles represent chemical groups

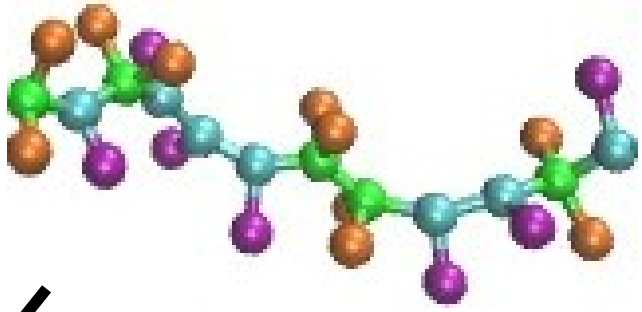
Diverse zoo of exotic molecular representations available in LAMMPS

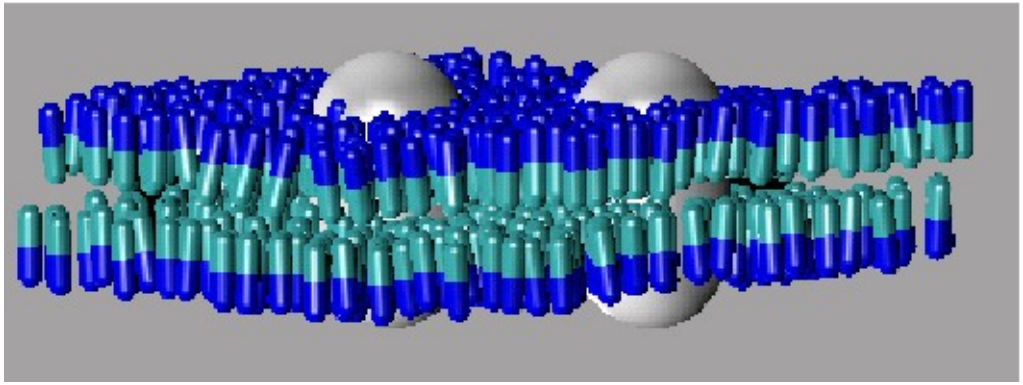


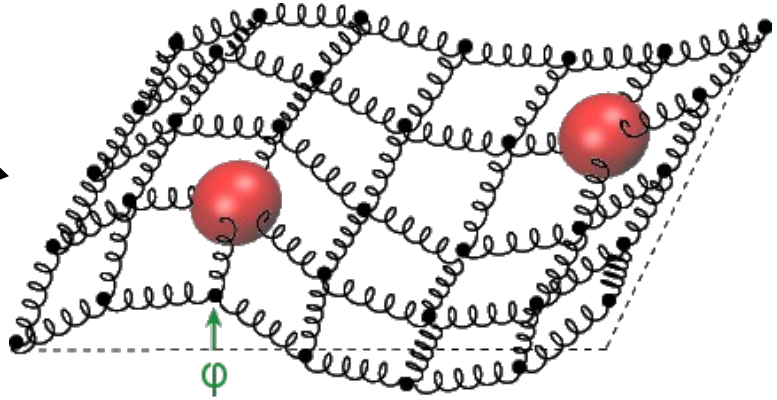
Non-point-like particles. (In this example by Grace Brannigan, ellipsoids and dipoles represent proteins and lipids.)

Diverse zoo of exotic molecular representations available in LAMMPS



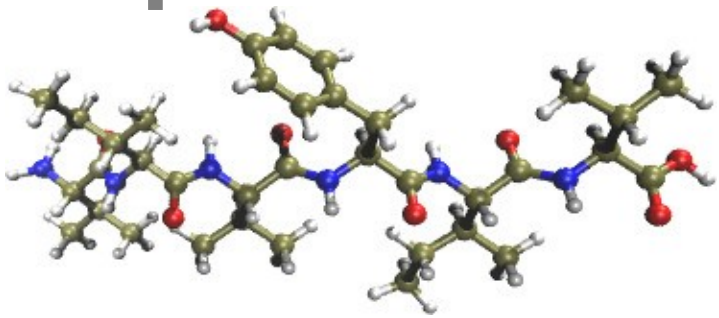


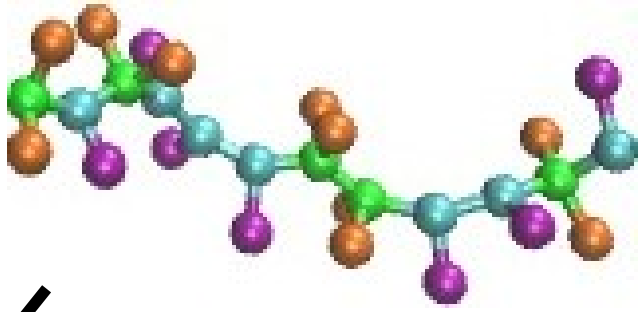


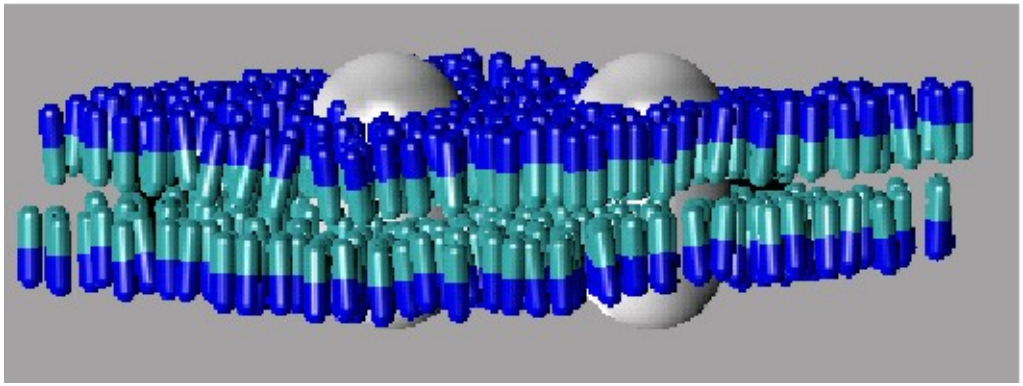


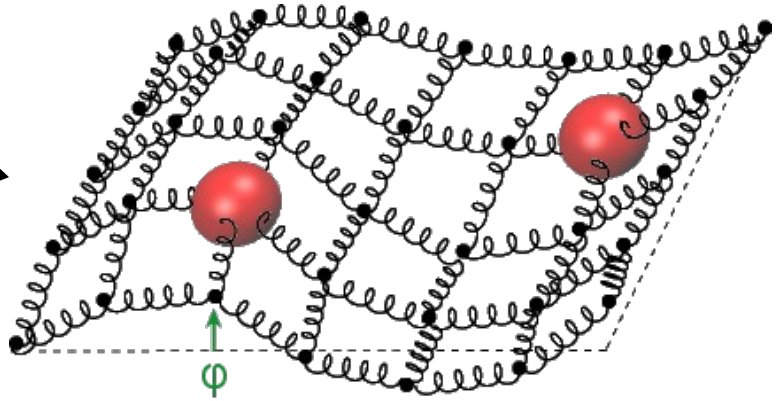
Coupled **continuum-field + discrete** particle hybrid systems

Diverse zoo of exotic molecular representations available in LAMMPS









Coupled **continuum-field + discrete** particle hybrid systems

New force-fields and atom styles are added by the community frequently...

LAMMPS

- **General**

  -Supports atoms, “*exotic” atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

  → -New force-fields and features are constantly submitted by users

  The file format is always changing

LAMMPS

- **General**

  -Supports atoms, "*exotic" atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

  -New force-fields and features are constantly submitted by users

  -**The file format is always changing**

→ **Writing a *general* molecule builder for LAMMPS is hard.**

LAMMPS

- **General**

  -Supports atoms, “*exotic” atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

  -New force-fields and features are constantly submitted by users

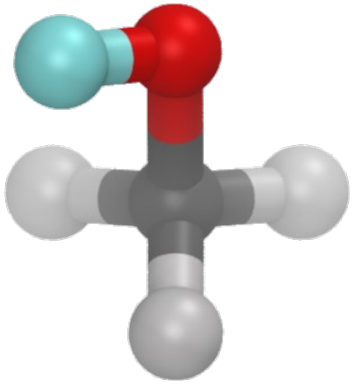
  -**The file format is always changing**

Writing a *general* molecule builder for LAMMPS is hard.

→ **templates**

simple example: methanol

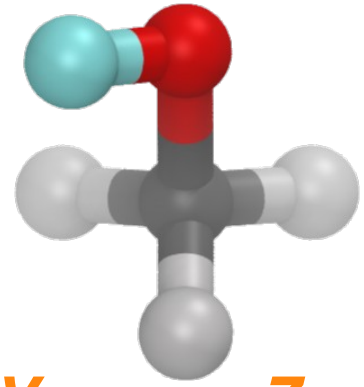
LAMMPS "data" file format
depends on atom_style



```
Atoms

    1          1          1          -0.5988  0.708  0.0    0.0
    2          1          2           0.1167 -0.708  0.0    0.0
    3          1          3           0.0287 -1.073 -0.769  0.685
    4          1          3           0.0287 -1.073 -0.195 -1.011
    5          1          3           0.0287 -1.063  0.979  0.331
    6          1          4           0.3960  0.994 -0.88  -0.298

Bonds
```

⋮ *(bond details omitted)*

simple example: methanol



LAMMPS "data" file format

depends on atom_style

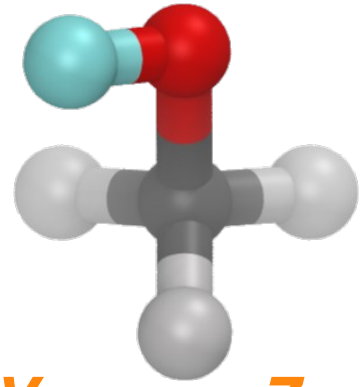
Atoms

Atom-ID	*Mol-ID*	*AtomType*	*charge*	*X*	*Y*	*Z*
1	1	1	-0.5988	0.708	0.0	0.0
2	1	2	0.1167	-0.708	0.0	0.0
3	1	3	0.0287	-1.073	-0.769	0.685
4	1	3	0.0287	-1.073	-0.195	-1.011
5	1	3	0.0287	-1.063	0.979	0.331
6	1	4	0.3960	0.994	-0.88	-0.298

Bonds

⋮ *(bond details omitted)*

simple example: methanol

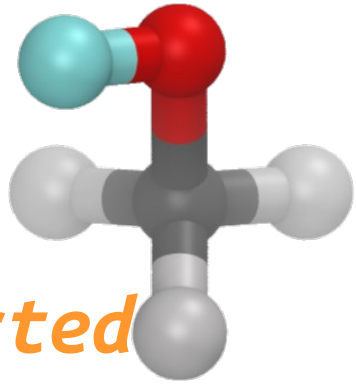


MOLTEMPLATE file format
mimics LAMMPS format exactly

Atom-ID Mol-ID AtomType charge X Y Z

```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

simple example: methanol

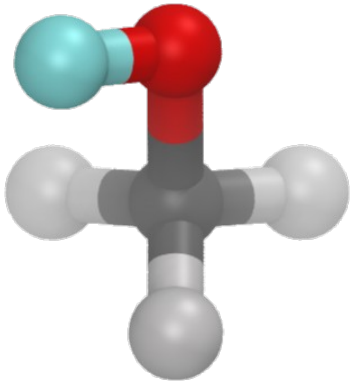


MOLTEMPLATE file format
mimics LAMMPS format exactly
→ All styles & text files are supported
(not only data files)

```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

simple example: methanol

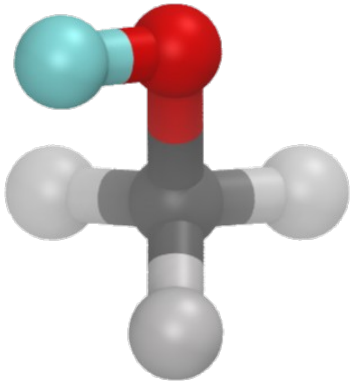
Counter variables
(anything following $ or @)



```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

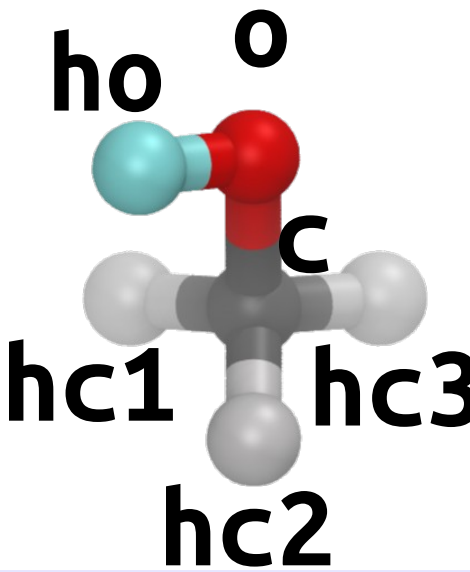
simple example: methanol

$ variables are unique IDs
@ variables are types



```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

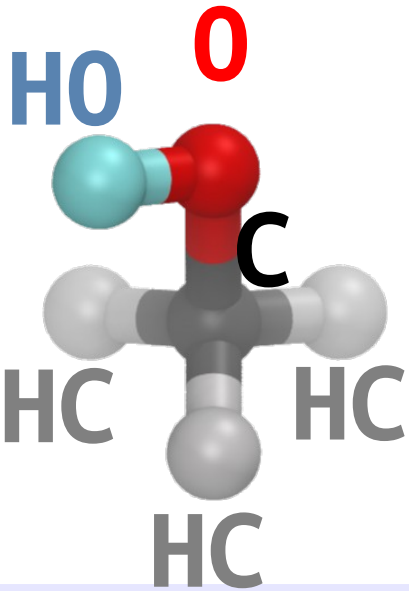
simple example: methanol



Atom-IDs ($atom)

```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

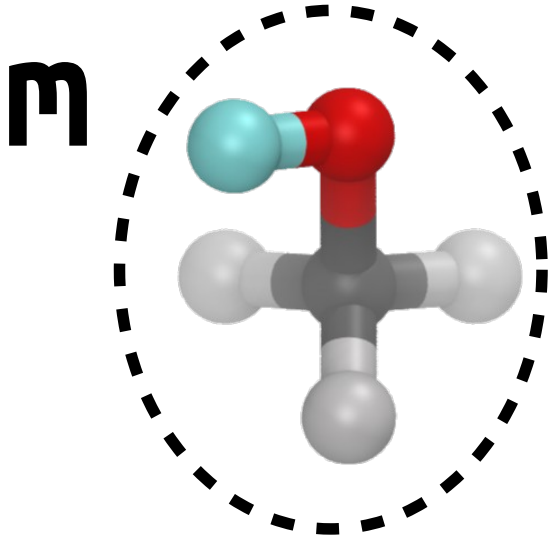
simple example: methanol



Atom-Types
(@atom)

```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

simple example: methanol

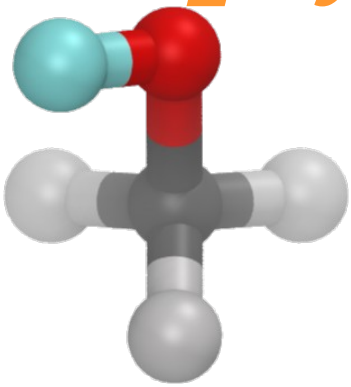


Molecule-IDs ($mol)

```
Methanol {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  ⋮ (bond details omitted for now...)
}
```

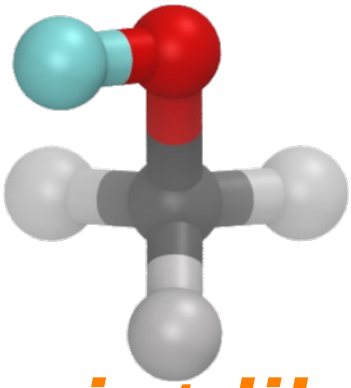
Moltemplate supports exotic styles

Example:
"atom_style hybrid sphere dipole molecular"



```
                atomID atomType    X      Y      Z  diam dens charge mux muy muz molID
Methanol {
  write("Data Atoms") {
    $atom:o   @atom:O   0.708  0.0    0.0   0.0 16.0 -0.5988 0.0 0.0 0.0 $mol:m
    $atom:c   @atom:C  -0.708  0.0    0.0   0.0 12.01 0.1167 0.0 0.0 0.0 $mol:m
    $atom:hc1 @atom:HC -1.073 -0.769  0.685 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc2 @atom:HC -1.073 -0.195 -1.011 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc3 @atom:HC -1.063  0.979  0.331 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:ho  @atom:HO  0.994 -0.88  -0.298 0.0 1.008 0.3960 0.0 0.0 0.0 $mol:m
  }

  ⋮
}
```

Moltemplate supports exotic styles

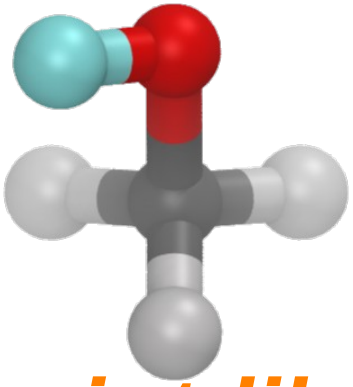


+

point-like ***dipole+sphere***

```
Methanol {
  write("Data Atoms") {
    $atom:o   @atom:O   0.708  0.0    0.0   0.0 16.0 -0.5988 0.0 0.0 0.0 $mol:m
    $atom:c   @atom:C  -0.708  0.0    0.0   0.0 12.01 0.1167 0.0 0.0 0.0 $mol:m
    $atom:hc1 @atom:HC -1.073 -0.769  0.685 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc2 @atom:HC -1.073 -0.195 -1.011 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc3 @atom:HC -1.063  0.979  0.331 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:ho  @atom:HO  0.994 -0.88  -0.298 0.0 1.008 0.3960 0.0 0.0 0.0 $mol:m
  }
}
ELBAwater {
  write("Data Atoms") {
    $atom:w @atom:W  0.0 0.0 0.0 4.080749 0.5063179 0.0 0.5410 0.0 0.0 $mol:m
  }
}
```

Moltemplate supports exotic styles

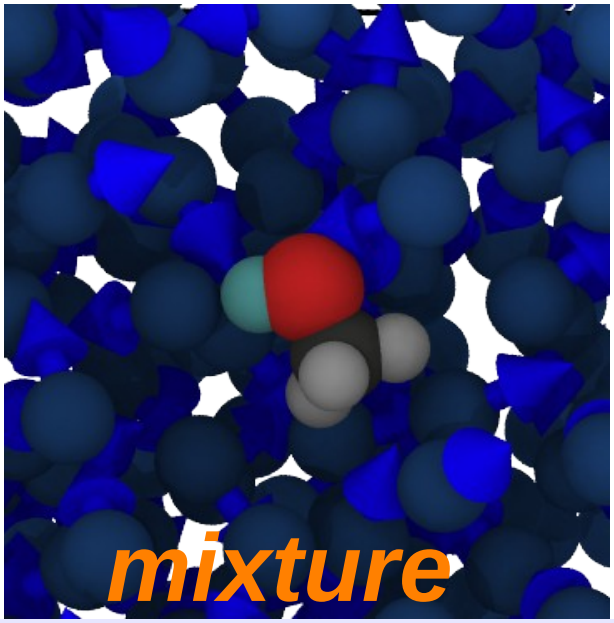


point-like

+

dipole+sphere

→



mixture

```
Methanol {
  write("Data Atoms") {
    $atom:o   @atom:O   0.708  0.0    0.0   0.0 16.0 -0.5988 0.0 0.0 0.0 $mol:m
    $atom:c   @atom:C  -0.708  0.0    0.0   0.0 12.01 0.1167 0.0 0.0 0.0 $mol:m
    $atom:hc1 @atom:HC -1.073 -0.769  0.685 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc2 @atom:HC -1.073 -0.195 -1.011 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:hc3 @atom:HC -1.063  0.979  0.331 0.0 1.008 0.0287 0.0 0.0 0.0 $mol:m
    $atom:ho  @atom:HO  0.994 -0.88  -0.298 0.0 1.008 0.3960 0.0 0.0 0.0 $mol:m
  }
}
ELBAwater {
  write("Data Atoms") {
    $atom:w @atom:W  0.0 0.0 0.0 4.080749 0.5063179 0.0 0.5410 0.0 0.0 $mol:m
  }
}
```

almost all atom styles supported

lammps.sandia.gov/doc/atom_style.html

create_box command
delete_atoms command
delete_bonds command
dielectric command
dihedral_coeff command
dihedral_style command
dimension command
displace_atoms command
dump command
dump vtk command
dump h5md command
dump molfile command
dump netcdf command
dump image command
dump movie command
dump adios command
dump atoms/adios command
dump custom/adios command
dump cfg/uef command
dump h5md command
dump image command
dump movie command
dump_modify command
dump molfile command

atom_style command

```
atom_style style args
```

angle	bonds and angles	bead-spring polymers with stiffness
atomic	only the default values	coarse-grain liquids, solids, metals
body	mass, inertia moments, quaternion, angular momentum	arbitrary bodies
bond	bonds	bead-spring polymers
charge	charge	atomic system with charges
dipole	charge and dipole moment	system with dipolar particles
dpd	internal temperature and internal energies	DPD particles
edpd	temperature and heat capacity	eDPD particles
mdpd	density	mDPD particles
tdpd	chemical concentration	tDPD particles
electron	charge and spin and eradius	electronic force field
ellipsoid	shape, quaternion, angular momentum	aspherical particles
full	molecular + charge	bio-molecules
line	end points, angular velocity	rigid bodies
meso	rho, e, cv	SPH particles
molecular	bonds, angles, dihedrals, impropers	uncharged molecules
peri	mass, volume	mesoscopic Peridynamic models
smd	volume, kernel diameter, contact radius, mass	solid and fluid SPH particles
sphere	diameter, mass, angular velocity	granular models
spin	magnetic moment	system with magnetic particles
template	template index, template atom	small molecules with fixed topology
tri	corner points, angular momentum	rigid bodies

all styles supported

lammps.sandia.gov/doc/pairs.html

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts

Docs » Pair Styles LAMMPS 7 Aug 2019 Website Co

Previous

Pair Styles

- pair_style adp command
- pair_style adp/omp command
- pair_style agni command
- pair_style agni/omp command
- pair_style airebo command
- pair_style airebo/intel command
- pair_style airebo/omp command
- pair_style airebo/morse command

all styles supported

lammps.sandia.gov/doc/angles.html

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts

Docs » Angle Styles LAMMPS 7 Aug 2019 Website Co

Previous

Angle Styles

- angle_style charmm command
- angle_style charmm/intel command
- angle_style charmm/kk command
- angle_style charmm/omp command
- angle_style class2 command
- angle_style class2/kk command
- angle_style class2/omp command
- angle_style class2/p6 command

all styles supported

lammps.sandia.gov/doc/dihedrals.html

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts

Docs » Dihedral Styles LAMMPS 7 Aug 2019 Website Co

Previous

Dihedral Styles

- dihedral_style charmm command
- dihedral_style charmm/intel command
- dihedral_style charmm/kk command
- dihedral_style charmm/omp command
- dihedral_style charmmfsw command
- dihedral_style class2 command
- dihedral_style class2/omp command
- dihedral_style class2/kk command

all styles supported

lammps.sandia.gov/doc/impropers.html

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts

Previous

Improper Styles

- improper_style class2 command
- improper_style class2/omp command
- improper_style class2/kk command
- improper_style cossq command
- improper_style cossq/omp command
- improper_style cvff command
- improper_style cvff/intel command
- improper_style cvff/omp command

all styles supported

lammps.sandia.gov/doc/fixes.html

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts

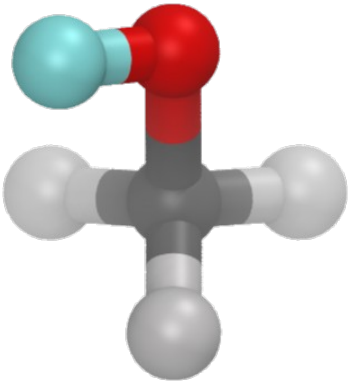
Docs » Fixes LAMMPS 7 Aug 2019 Website Co

Previous

Fixes

- fix adapt command
- fix adapt/fep command
- fix addforce command
- fix addtorque command
- fix append/atoms command
- fix atc command
- fix atom/swap command
- fix ave/atom command

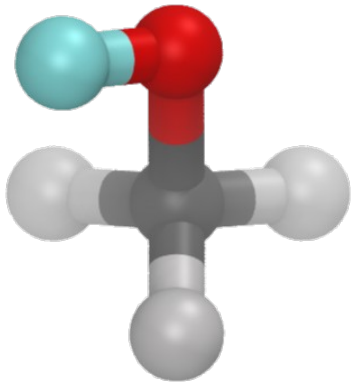
Multiple molecules

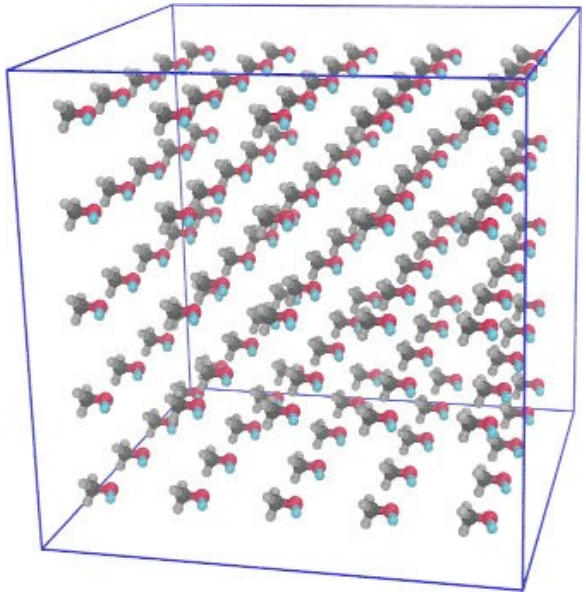


methanol.lt

```
import "methanol.lt"  # <-- defines "Methanol"
```

Multiple molecules

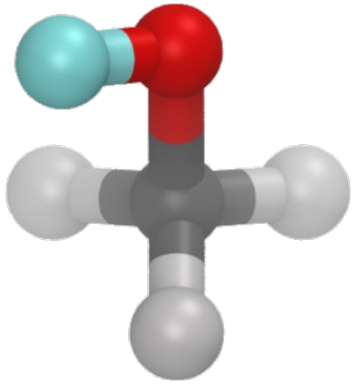


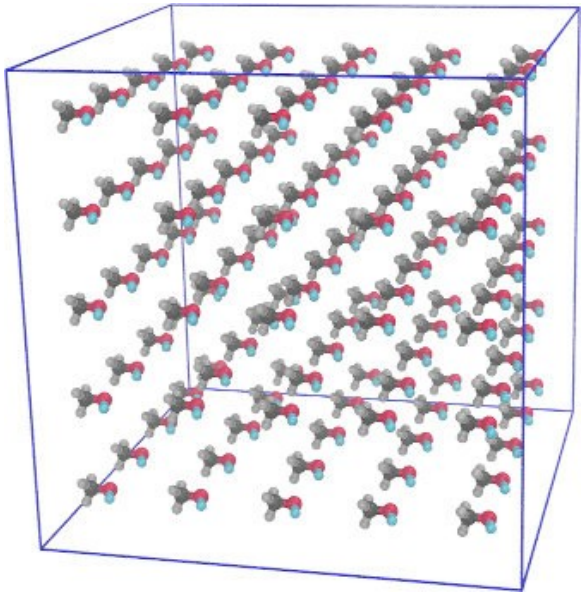


methanol.lt

```
import "methanol.lt"  # <-- defines "Methanol"

methanol1 = new Methanol.move(0.00, 0.00, 0.00)
methanol2 = new Methanol.move(0.00, 0.00, 6.90)
methanol3 = new Methanol.move(0.00, 0.00, 13.8)
methanol4 = new Methanol.move(0.00, 0.00, 20.7)
methanol5 = new Methanol.move(0.00, 0.00, 27.6)
methanol6 = new Methanol.move(0.00, 6.90, 0.00)
    :                    :
methanol26 = new Methanol.move(6.90, 0.00, 0.00)
    :                    :
methanol125 = new Methanol.move(27.6, 27.6, 27.6)
```

Multiple molecules

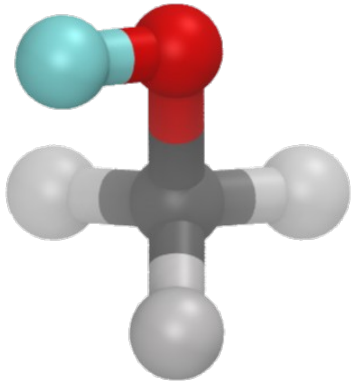


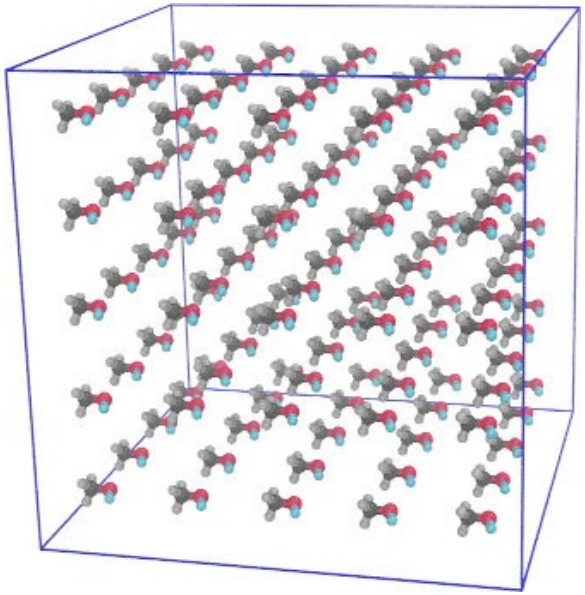


```
import "methanol.lt"  # <-- defines "Methanol"


methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)
```

Shorthand equivalent

How to run moltemplate?

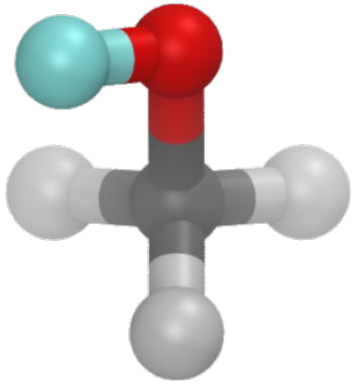




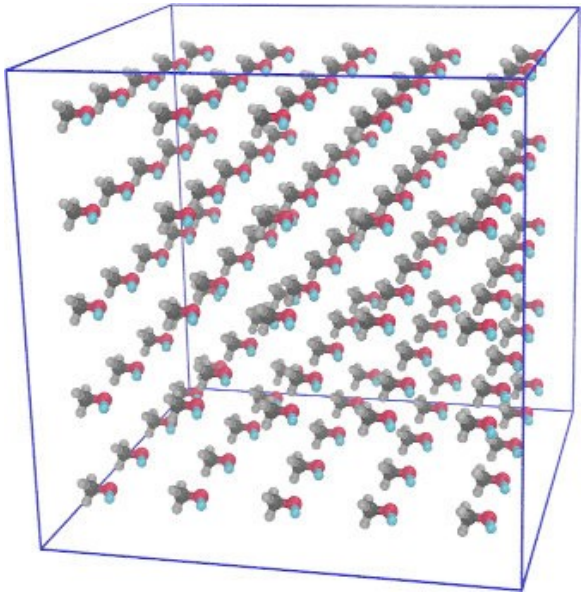
```
import "methanol.lt"  # <-- defines "Methanol"

methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)
```

system.lt file

Run moltemplate in a terminal



moltemplate.sh system.lt



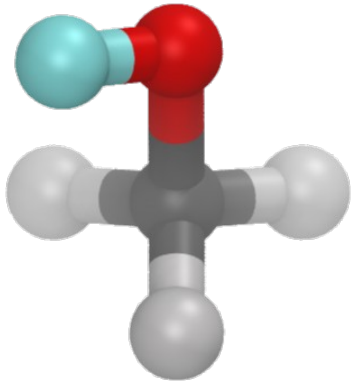
```
import "methanol.lt"  # <-- defines "Methanol"

methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)
```

system.lt file

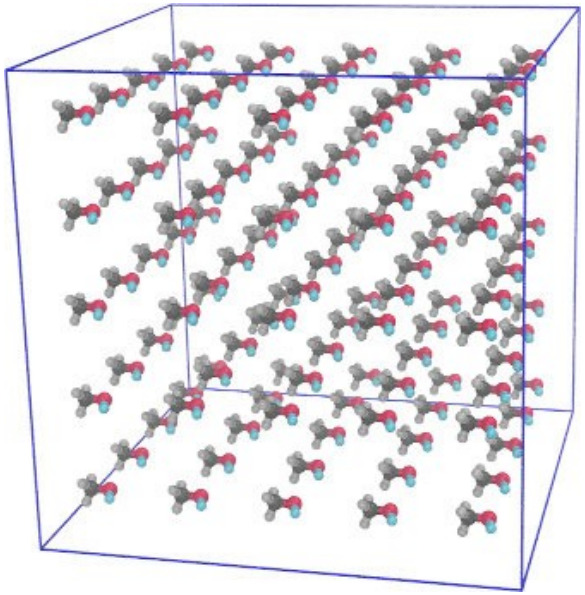
Terminal:

```
$
$ moltemplate.sh system.lt
$
```

Run moltemplate in a terminal



moltemplate.sh system.lt



```
import "methanol.lt"  # <-- defines "Methanol"

methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)
```

system.lt file

Terminal:

```
$
$ moltemplate.sh system.lt -vmd
$
```

(if VMD is installed)

What moltemplate does

- Creates a LAMMPS **data file**

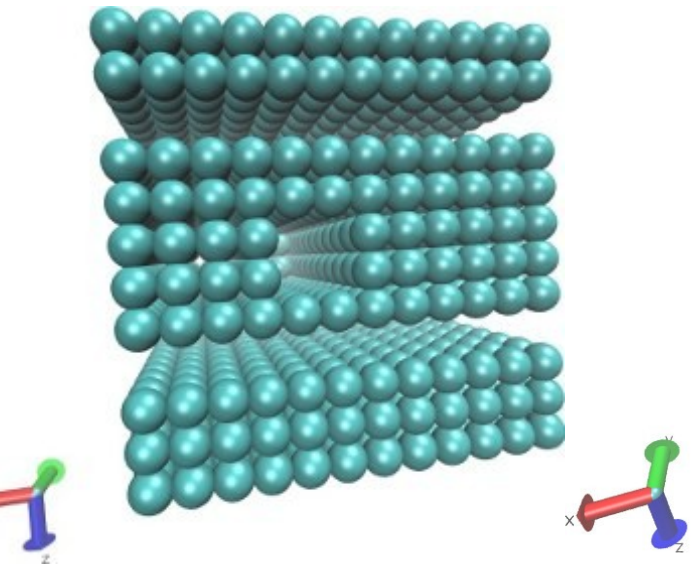
  (eg “system.data”)

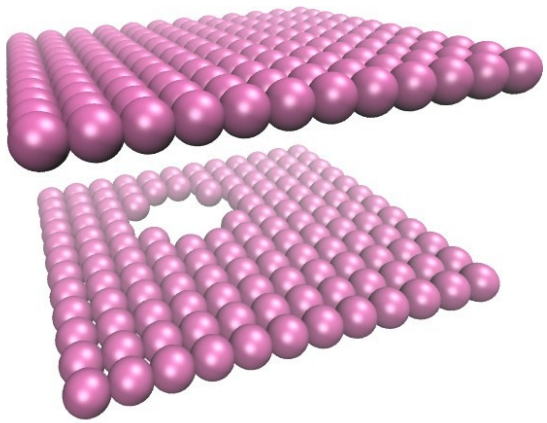
- Creates one or more **input scripts** containing style declarations, force field information, group definitions, fixes, etc...

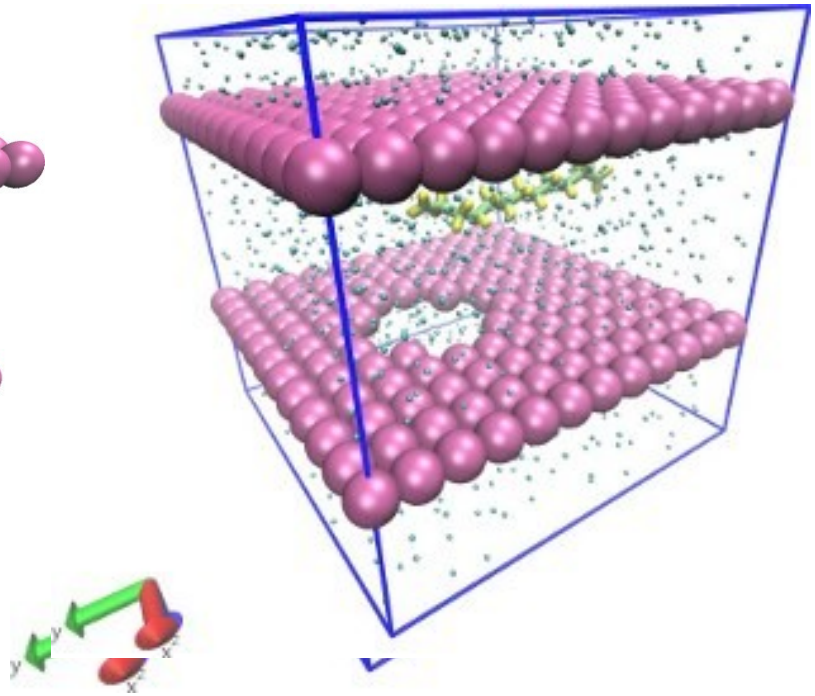
  (eg. fix shake)

- Creates **auxiliary files** needed by your force-field styles or fixes (if applicable)

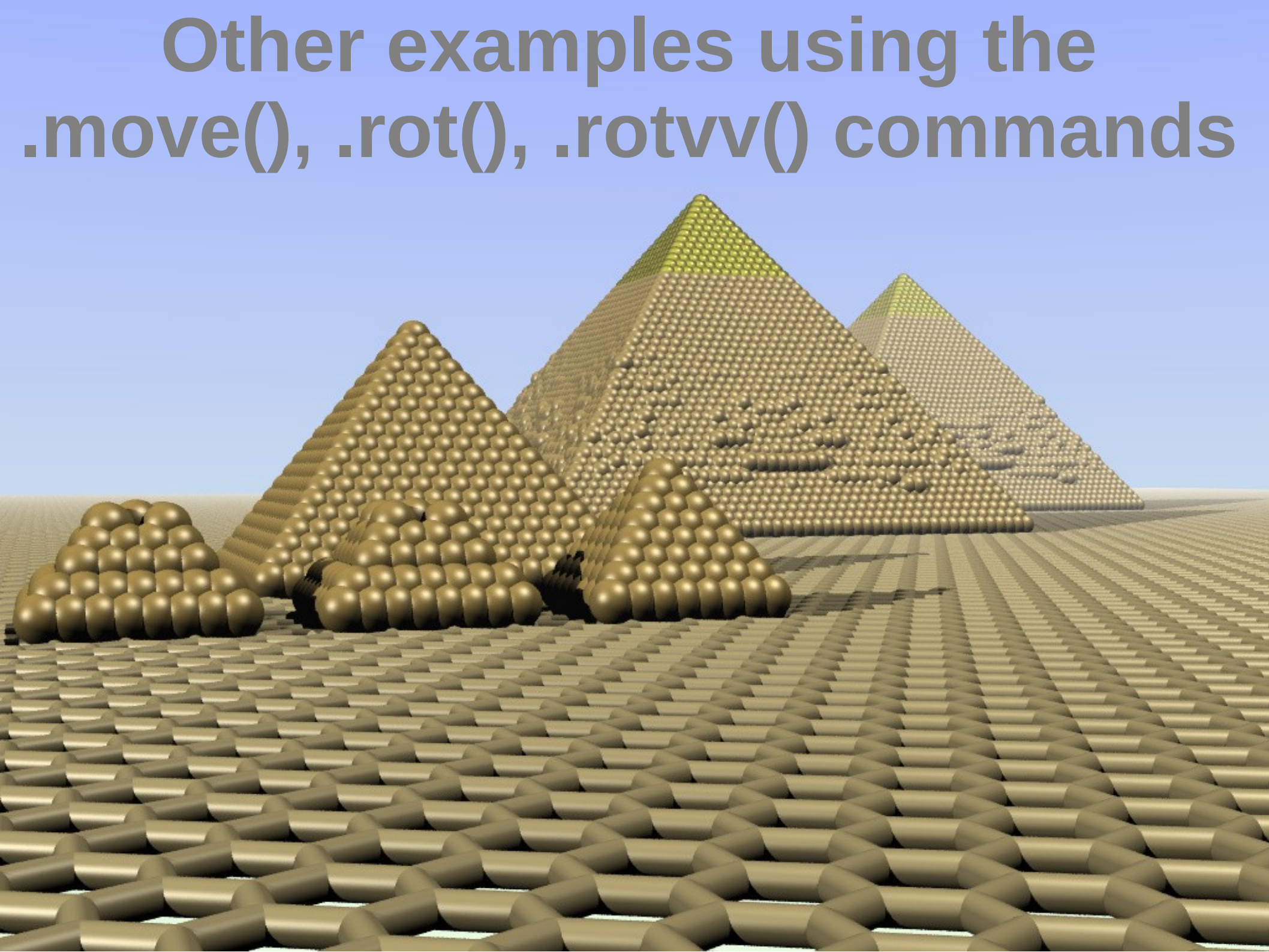
Other examples using the .move(), .rot(), .rotvv() commands



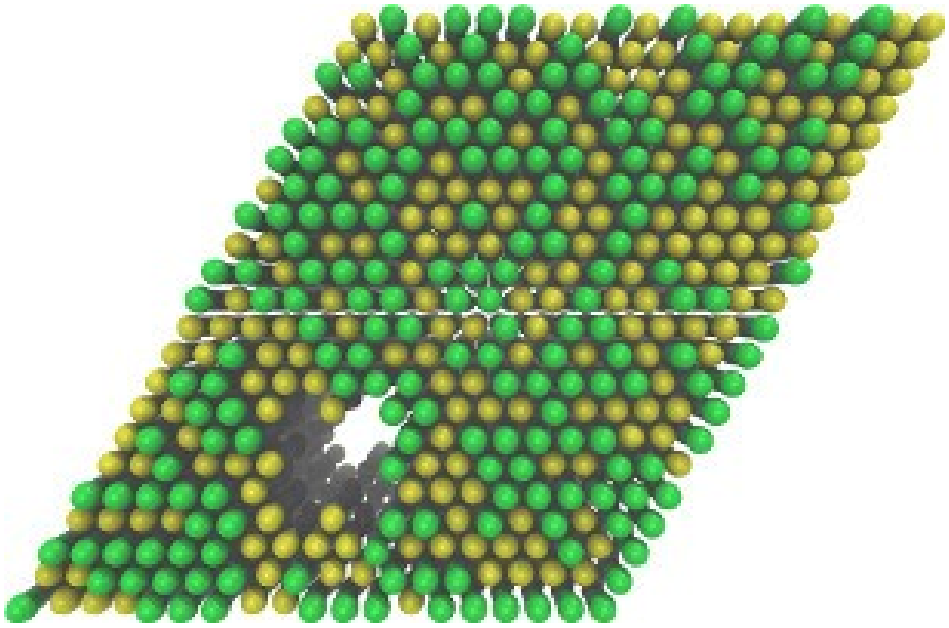




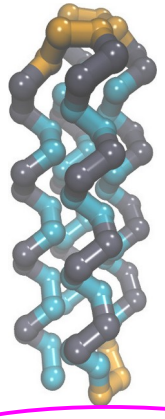
Other examples using the .move(), .rot(), .rotvv() commands



Other examples using the .move(), .rot(), .rotvv() commands



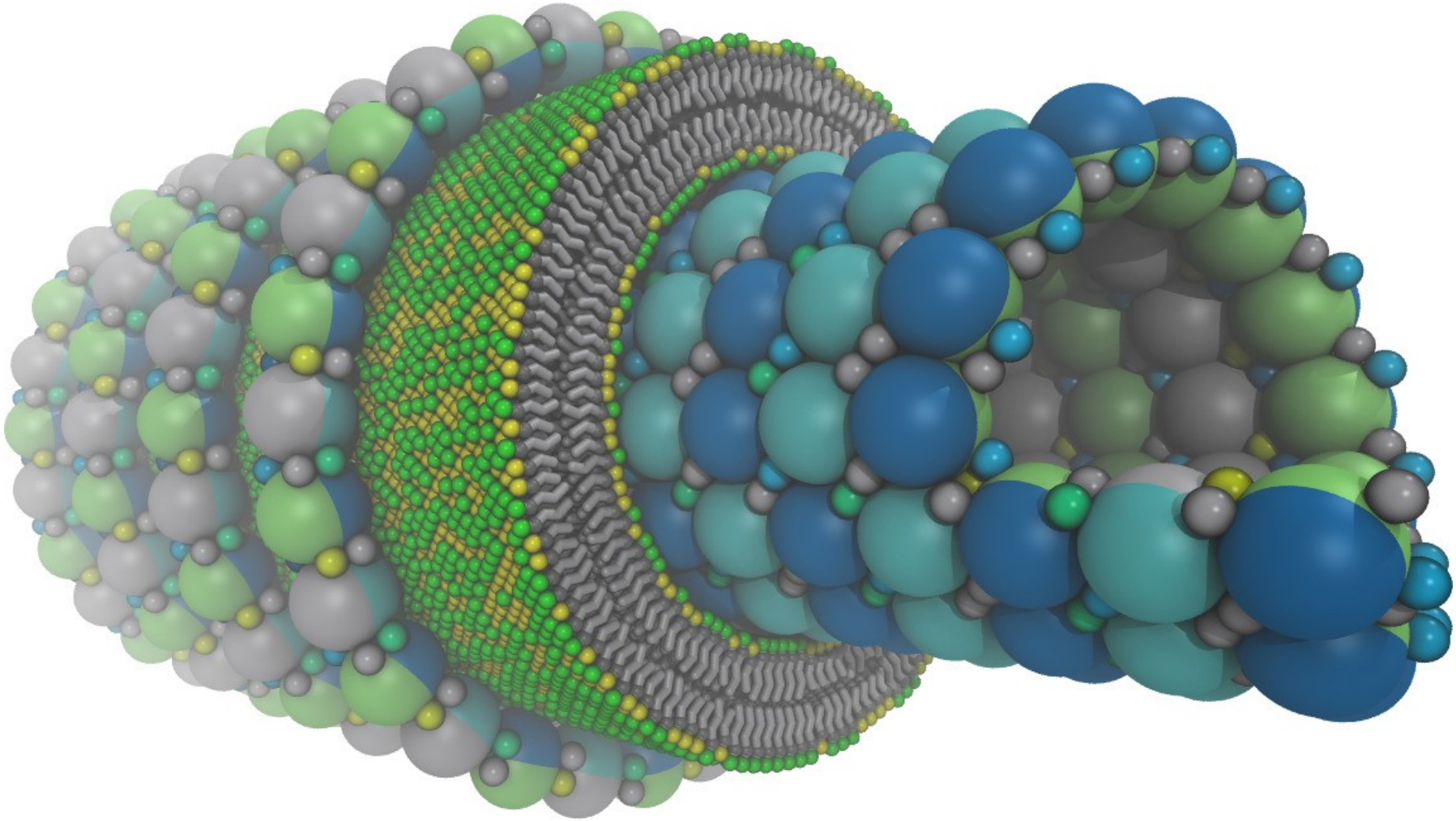
+



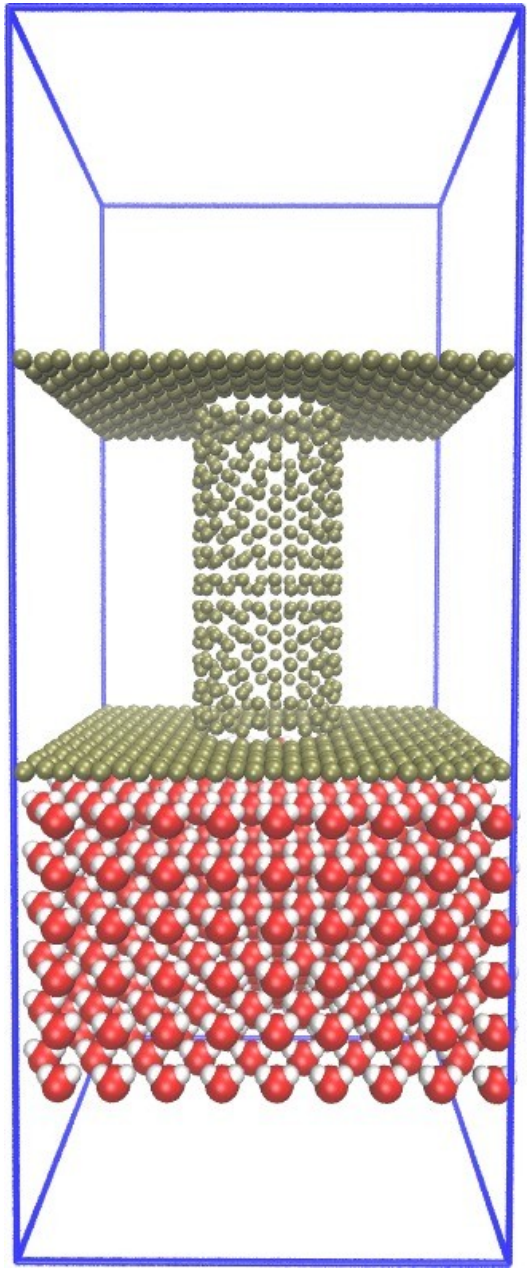
“4HelixBundle”

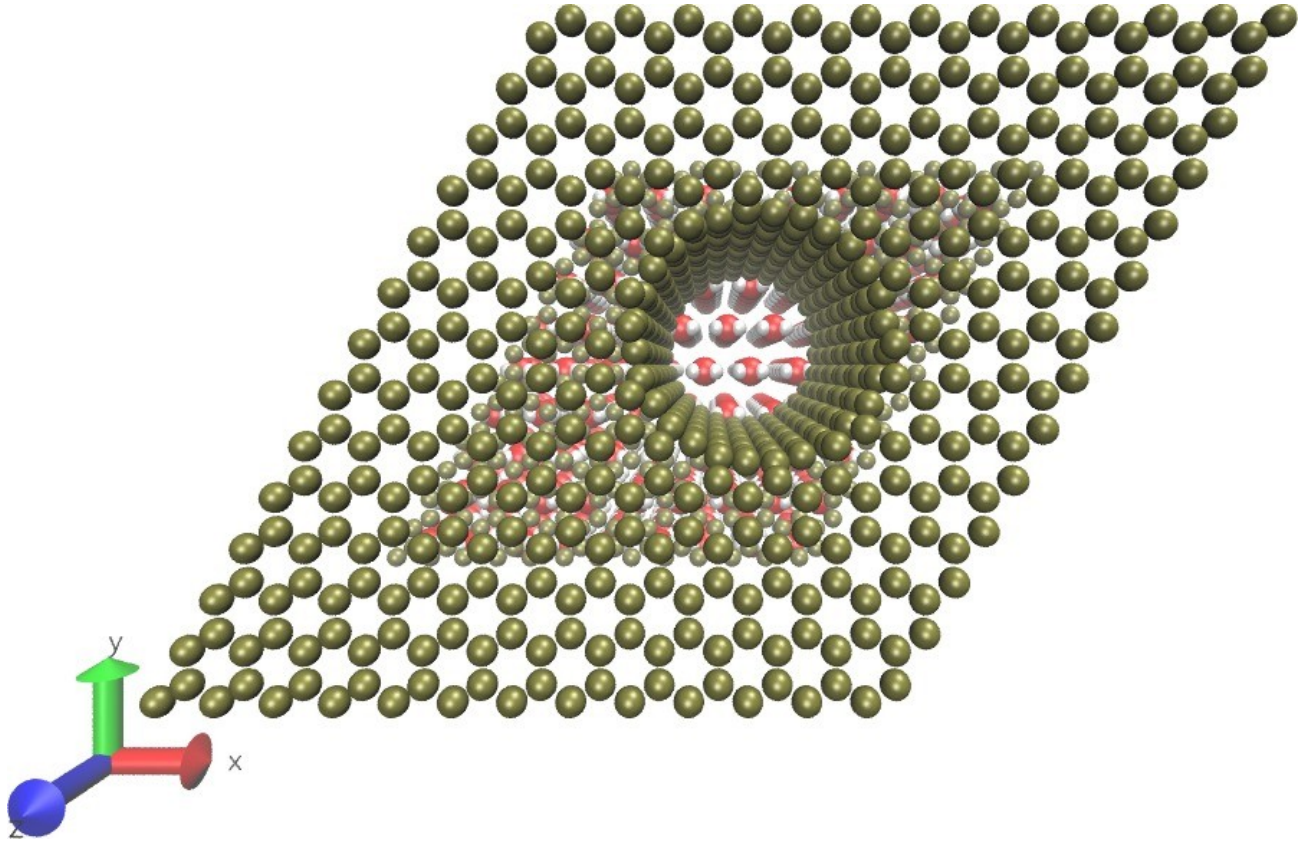
```
lipids = new random([DPPC,DLPC], [0.5,0.5])
                   [19].move(7.5,  0,  0)
                   [22].move(3.75, 6.49519, 0)
                   [2].rot(180, 1, 0, 0)
delete lipids[7][3][*]
delete lipids[9][3][*]
delete lipids[6-9][4][*]
delete lipids[6-8][5][*]
delete lipids[5-8][6][*]
delete lipids[5][7][*]
delete lipids[7][7][*]

protein = new 4HelixBundle
```

Other examples using the .move(), .rot(), .rotvv() commands

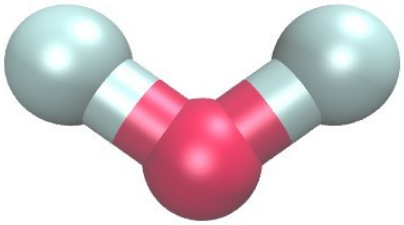


Other examples using the .move(), .rot(), .rotvv() commands





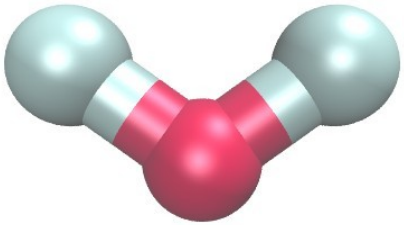
simple example: water

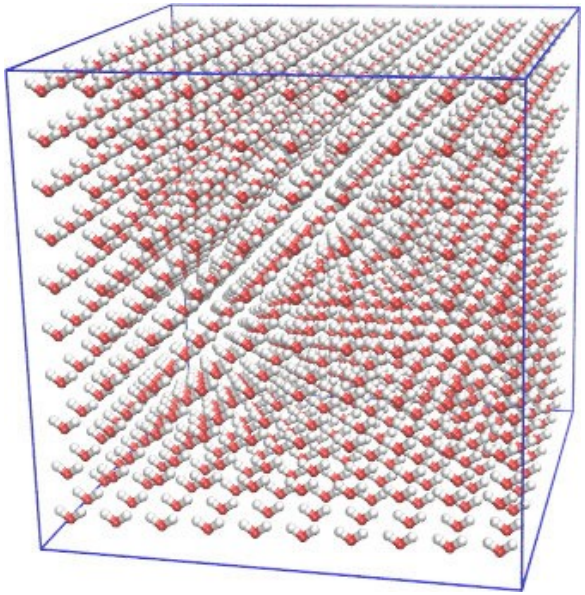


spce.lt

```
SPCE {
  write("Data Atoms") {
    $atom:o    $mol    @atom:O -0.8476  0.0000 0.000 0.0000
    $atom:h1   $mol    @atom:H  0.4238  0.8165 0.000 0.5774
    $atom:h2   $mol    @atom:H  0.4238 -0.8165 0.000 0.5774
  ⋮ (bond details omitted for now...)
}
```

simple example: water

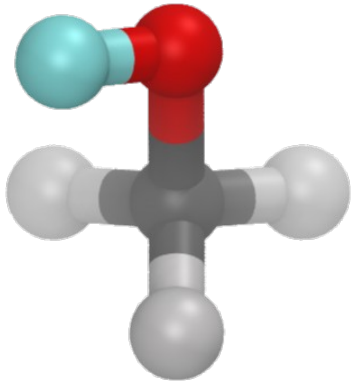


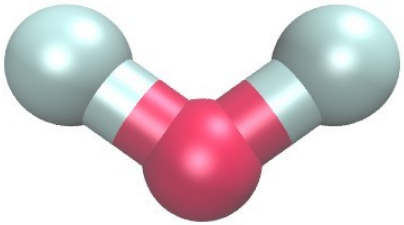


spce.lt

```
import "spce.lt"        # <-- defines "SPCE" (water)



wat  = new SPCE [10].move(0.00, 0.00, 3.45)
                [10].move(0.00, 3.45, 0.01)
                [10].move(3.45, 0.01, 0.01)
```

Creating mixtures

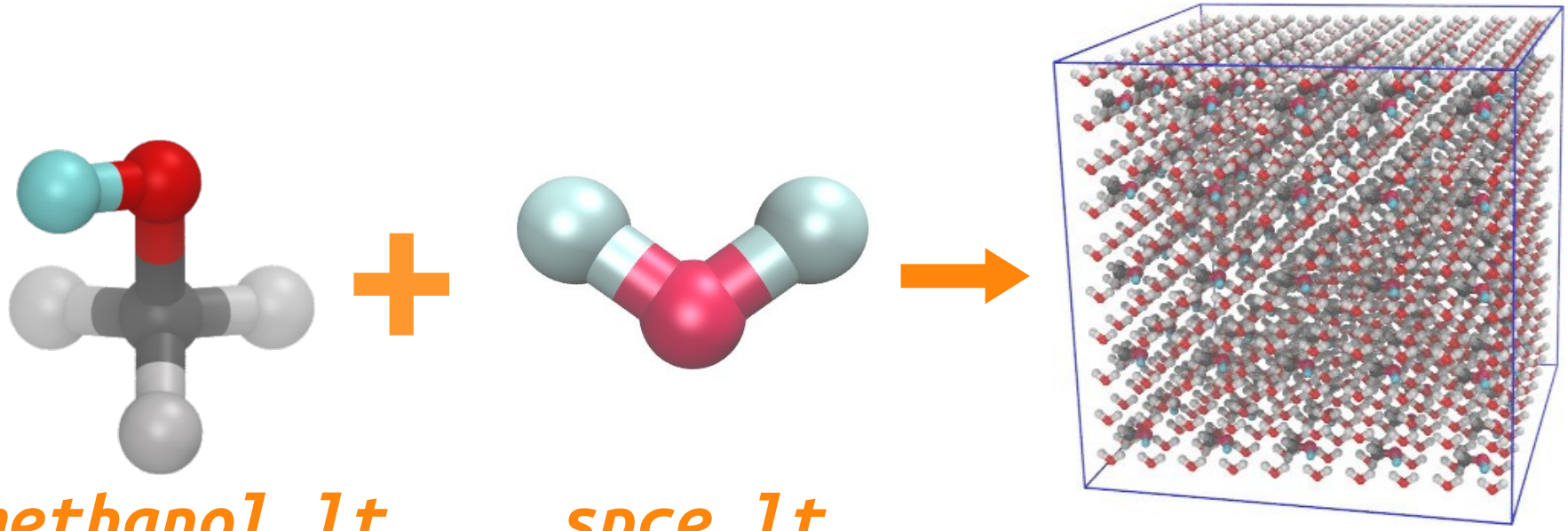




methanol.lt ***spce.lt***

```
import "methanol.lt"  # <-- defines "Methanol"
import "spce.lt"      # <-- defines "SPCE" (water)
```

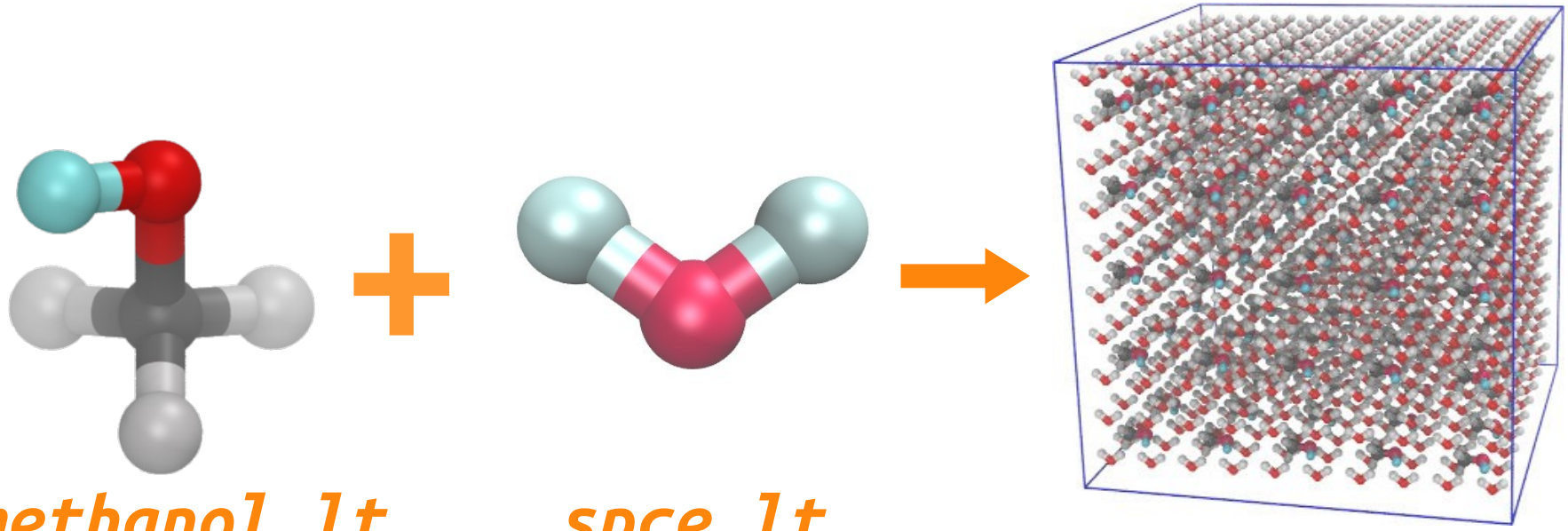
Creating mixtures



methanol.lt *spce.lt*

```
import "methanol.lt"  # <-- defines "Methanol"
import "spce.lt"      # <-- defines "SPCE" (water)

methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)

wat  = new SPCE [10].move(0.00, 0.00, 3.45)
                [10].move(0.00, 3.45, 0.01)
                [10].move(3.45, 0.01, 0.01)
```

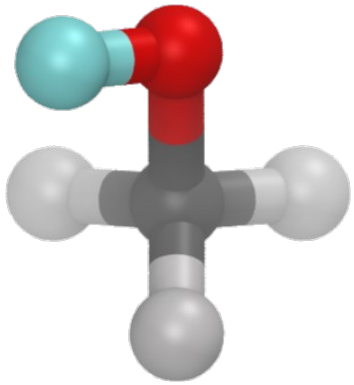
Creating mixtures

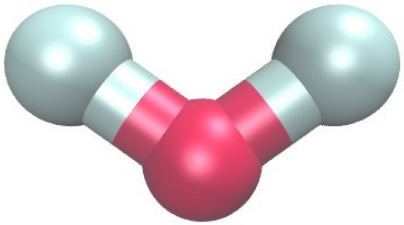


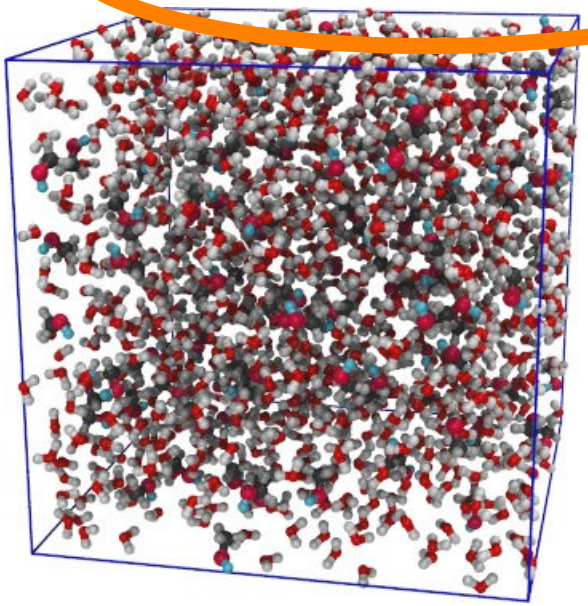
methanol.lt *spce.lt*

```
import "methanol.lt"  # <-- defines "Methanol"
import "spce.lt"      # <-- defines "SPCE" (water)

methanols = new Methanol [5].move(0.00, 0.00, 6.90)
                         [5].move(0.00, 6.90, 0.00)
                         [5].move(6.90, 0.00, 0.00)

wat  = new SPCE [10].move(0.00, 0.00, 3.45)
                [10].move(0.00, 3.45, 0.01)
                [10].move(3.45, 0.01, 0.01)
methanols[*][*][*].move(1.725,1.735,1.725)  #<-avoid overlap
```

Creating mixtures with PACKMOL



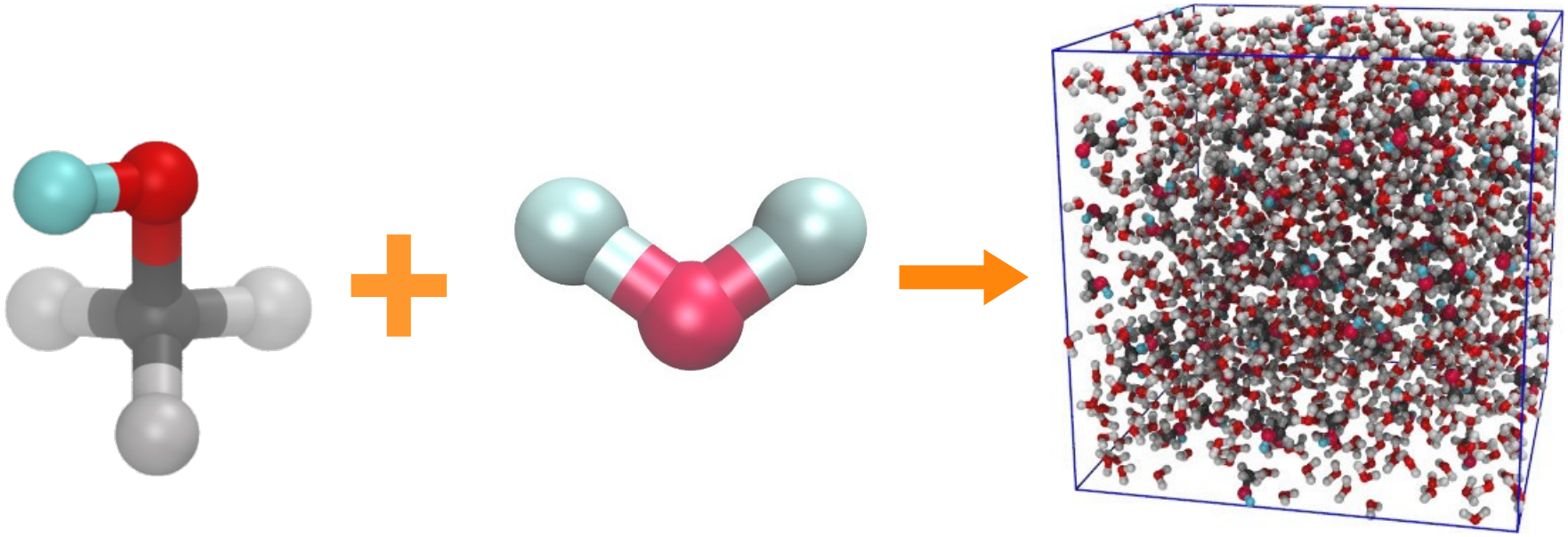




methanol.lt *spce.lt*

```
import "methanol.lt"  # <-- defines "Methanol"
import "spce.lt"      # <-- defines "SPCE" (water)

methanols = new Methanol [125]



wat  = new SPCE [1000]
```

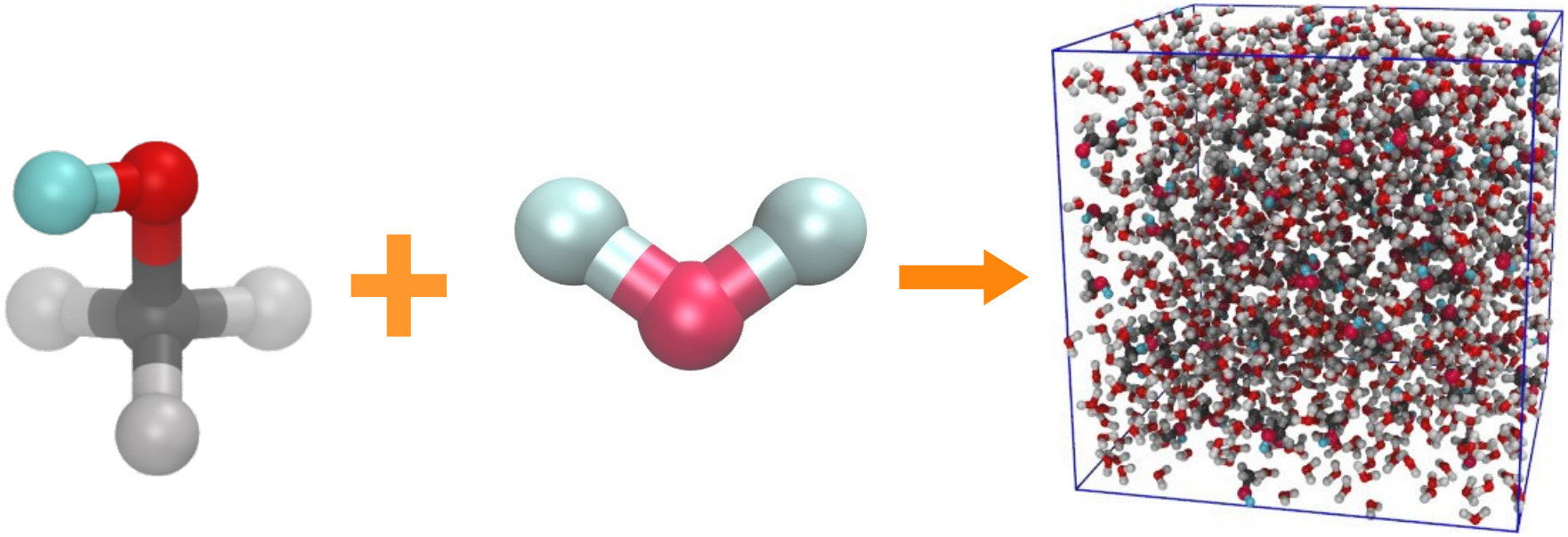
Creating mixtures with *PACKMOL*



```
tolerance 2.5
filetype xyz
output system.xyz

structure methanol.xyz
  number 125
  inside box 0.0 0.0 0.0 34.5 34.5 34.5
end structure
structure water.xyz
  number 1000
  inside box 0.0 0.0 0.0 34.5 34.5 34.5
end structure
```

PACKMOL INPUT FILE EXAMPLE: “input.packmol”

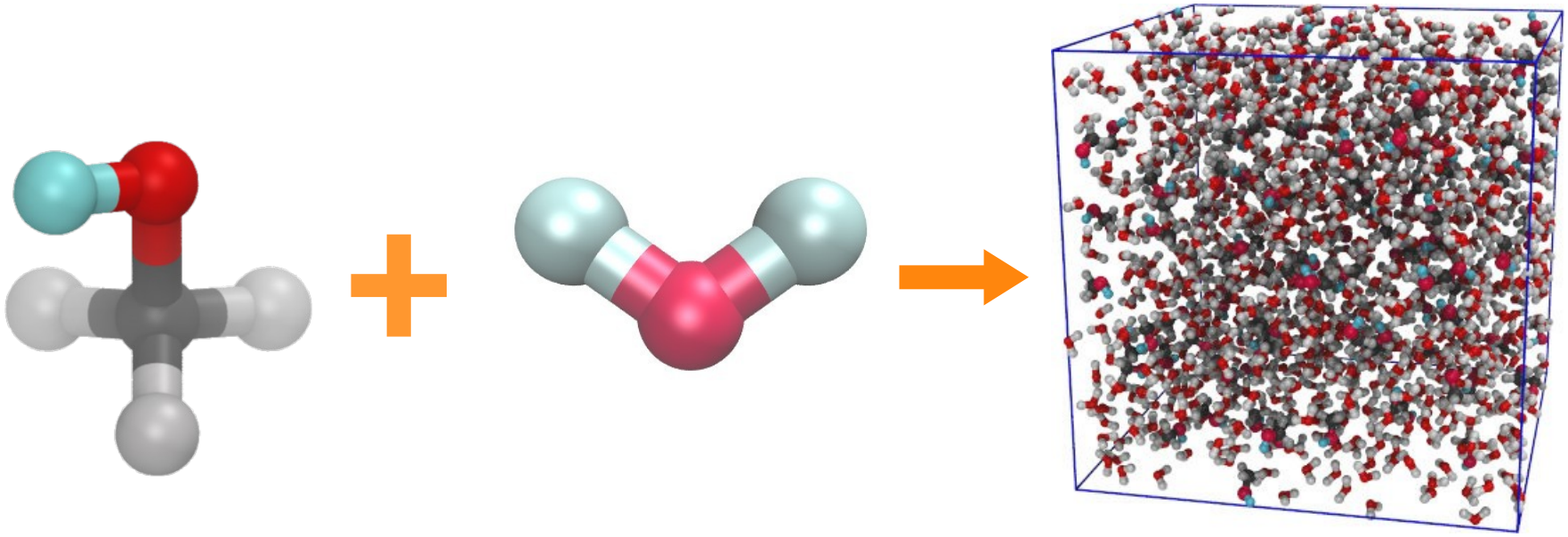
Creating mixtures with *PACKMOL*



```
tolerance 2.5
filetype xyz
output system.xyz

structure methanol.xyz
  number 125
  inside box 0.0 0.0 0.0 34.5 34.5 34.5
end structure
structure water.xyz
  number 1000
  inside box 0.0 0.0 0.0 34.5 34.5 34.5
end structure
```

PACKMOL INPUT FILE EXAMPLE: "input.packmol"

Creating mixtures with *PACKMOL*



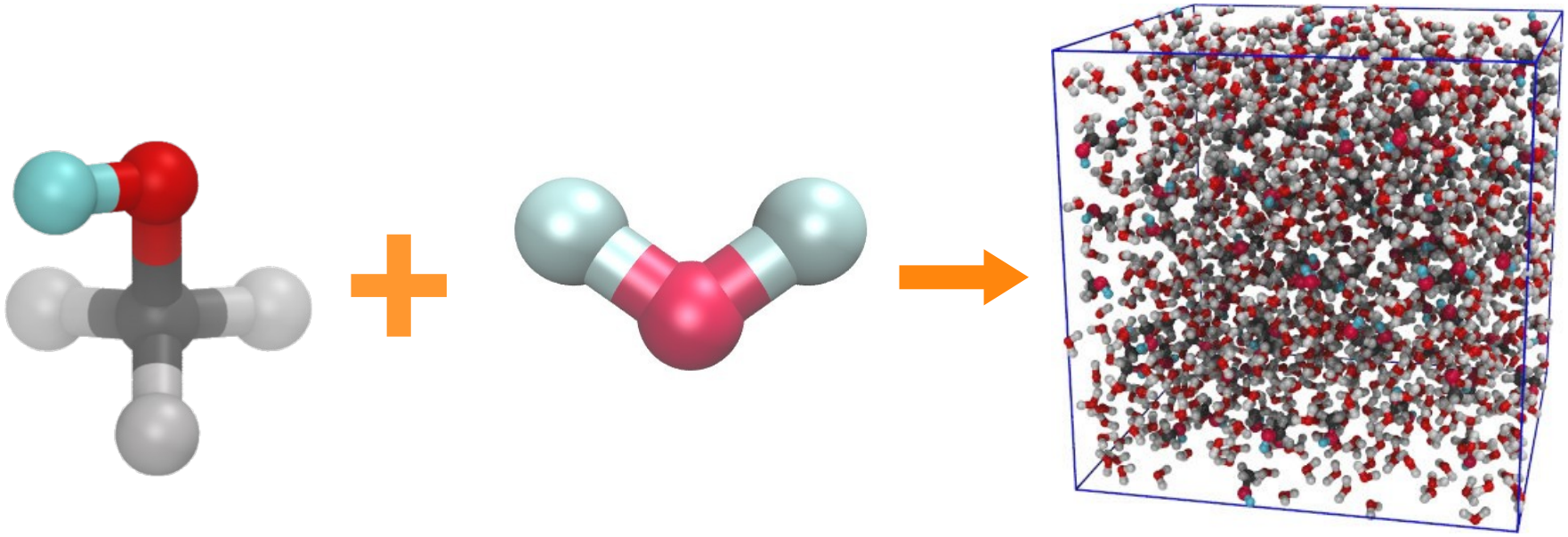
Terminal: Run PACKMOL on this file

```
$
$ packmol < input.packmol
$
```

Tell moltemplate to read coordinates generated by PACKMOL

```
$
$ moltemplate.sh system.lt -xyz system.xyz
$
```

Creating mixtures with *PACKMOL*



Terminal: Run PACKMOL on this file

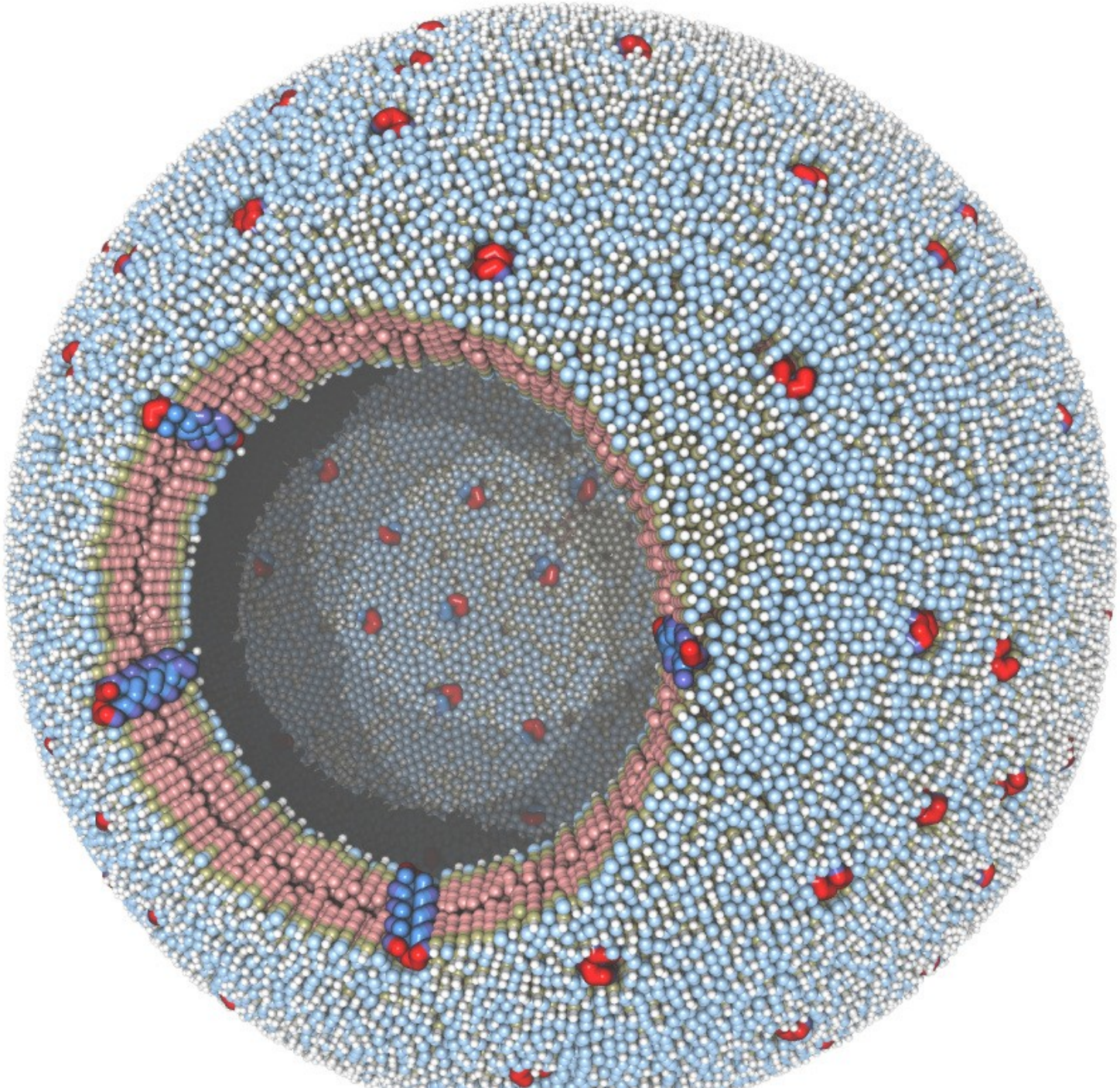
```
$
$ packmol < input.packmol
$
```

Tell moltemplate to read coordinates generated by PACKMOL

```
$
$ moltemplate.sh system.lt -xyz system.xyz
$
```

Other PACKMOL+Moltemplate examples

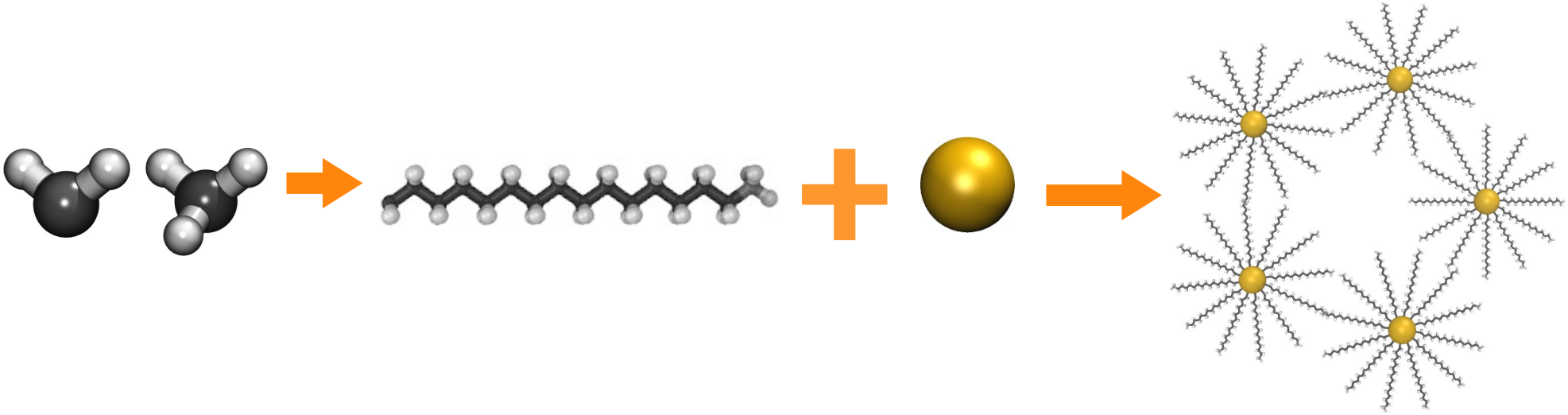
cutaway
-view



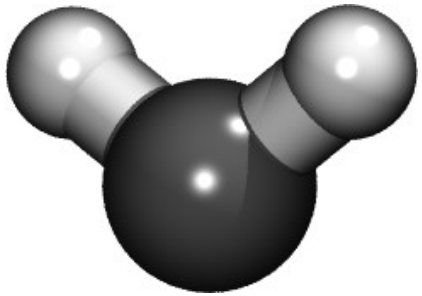
Moltemplate lets you build big things out of small things

(...and use them to build bigger things)

Hierarchical molecules



Hierarchical molecules



CH2

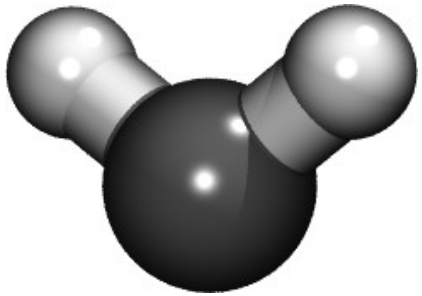
```
import "gaff2.lt" # <-- defines "GAFF2" (AMBER) force field

CH2 inherits GAFF2 {
  write("Data Atoms") {
    $atom:c  $mol:... @atom:c3 -0.12  0.000  0.0000  0.000
    $atom:h1 $mol:... @atom:hc  0.06  0.000  0.6310  0.8924
    $atom:h2 $mol:... @atom:hc  0.06  0.000  0.6310 -0.8924
  }

  write('Data Bond List') {
    $bond:ch1 $atom:c $atom:h1
    $bond:ch2 $atom:c $atom:h2
  }
} # CH2
```

defined in "GAFF2" Force Field

Force Field support *(2019)*

- OPLSAA *(2009 Tinker version)*
- LOPLSAA *(2015)*
- AMBER *(GAFF & GAFF2)*
- COMPASS *(published only)*
- TraPPE *(1998)*
- MARTINI ? *(untested)*
- SDK ? *(untested)*

Hierarchical molecules

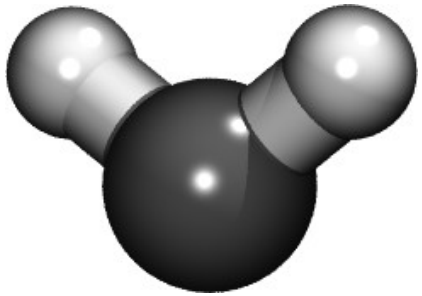


CH2

```
import "gaff2.lt" # <-- defines "GAFF2" (AMBER) force field

CH2 inherits GAFF2 {
  write("Data Atoms") {
    $atom:c  $mol:... @atom:c3 -0.12  0.000  0.0000  0.000
    $atom:h1 $mol:... @atom:hc  0.06  0.000  0.6310  0.8924
    $atom:h2 $mol:... @atom:hc  0.06  0.000  0.6310 -0.8924
  }

  write('Data Bond List') {
    $bond:ch1 $atom:c $atom:h1
    $bond:ch2 $atom:c $atom:h2
  }
} # CH2
```

Part of larger molcule (share mol-IDs)

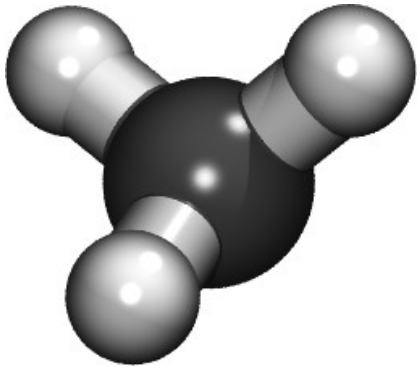
Hierarchical molecules



CH2

```
import "gaff2.lt" # <-- defines "GAFF2" (AMBER) force field

CH2 inherits GAFF2 {
  write("Data Atoms") {
    $atom:c  $mol:... @atom:c3 -0.12  0.000  0.0000  0.000
    $atom:h1 $mol:... @atom:hc  0.06  0.000  0.6310  0.8924
    $atom:h2 $mol:... @atom:hc  0.06  0.000  0.6310 -0.8924
  }

  write('Data Bond List') {
    $bond:ch1 $atom:c $atom:h1
    $bond:ch2 $atom:c $atom:h2
  }
} # CH2
```

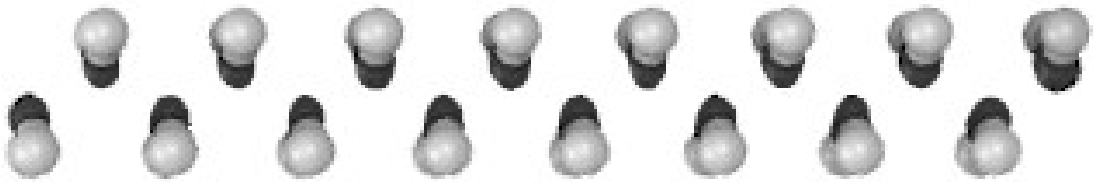
Hierarchical molecules



CH3

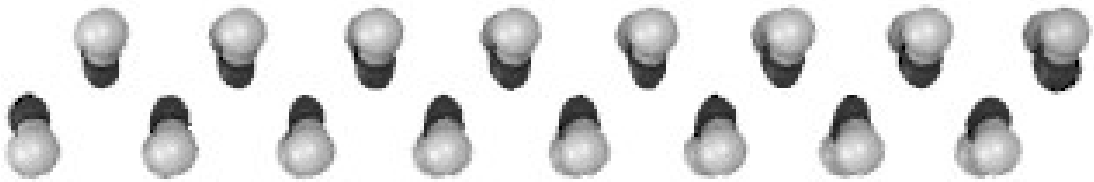
```
import "gaff2.lt" # <-- defines "GAFF2" (AMBER) force field

CH3 inherits GAFF2 {
  write("Data Atoms") {
    $atom:c  $mol:... @atom:c3 -0.18  0.000  0.0000  0.000
    $atom:h1 $mol:... @atom:hc  0.06  0.000  0.6310  0.8924
    $atom:h2 $mol:... @atom:hc  0.06  0.000  0.6310 -0.8924
    $atom:H3 $mol:... @atom:hc  0.06 -0.892 -0.6310  0.0000
  }
  write('Data Bond List') {
    $bond:ch1 $atom:c $atom:h1
    $bond:ch2 $atom:c $atom:h2
    $bond:ch3 $atom:c $atom:h3
  }
} # CH3
```

Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"



monomer1 = new CH2
monomer2 = new CH2.rot(180,1,0,0).move(1.253,0,0)
monomer3 = new CH2.rot(  0,1,0,0).move(2.506,0,0)
monomer4 = new CH2.rot(180,1,0,0).move(3.759,0,0)
monomer5 = new CH2.rot(  0,1,0,0).move(5.012,0,0)
      ⋮                                   ⋮
monomer16 = new CH2.rot(180,1,0,0).move(18.795,0,0)
```

Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"


monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
```

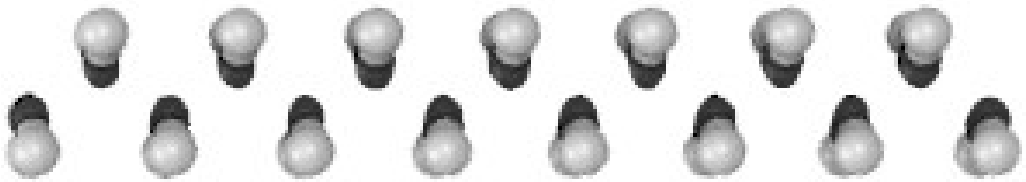
Shorthand equivalent

Hierarchical molecules

```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"

monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
delete monomers[15]
```

Delete the last entry in the array

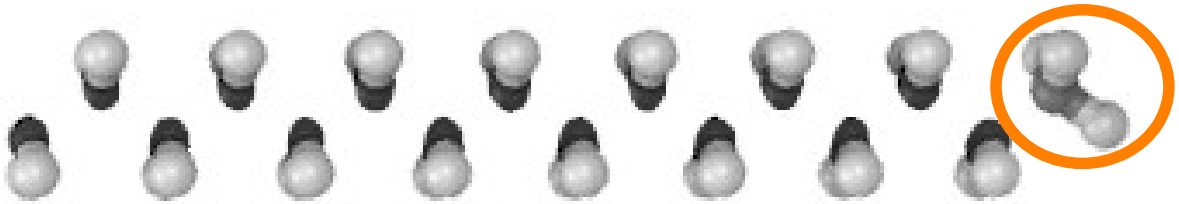
Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"

monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
delete monomers[15]
```

Delete the last entry in the array

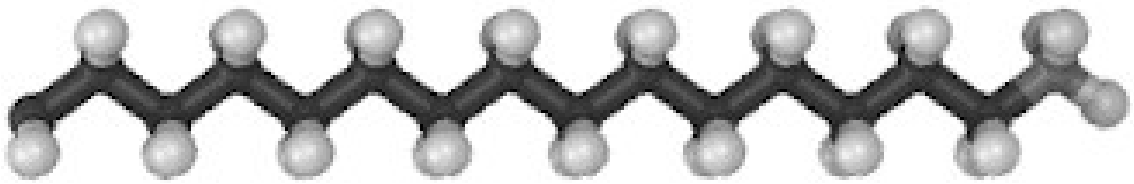
Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"

monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
delete monomers[15]
monomers[15] = new CH3.rot(180.0,0,0,1).move(18.795,0,0)
```

Replace it with a "CH3" and move it into position

Hierarchical molecules

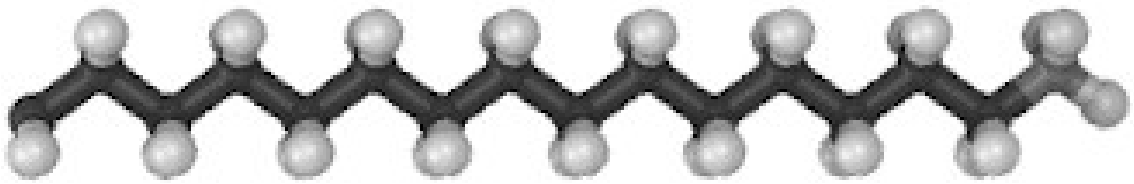


```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"
```

Add bonds

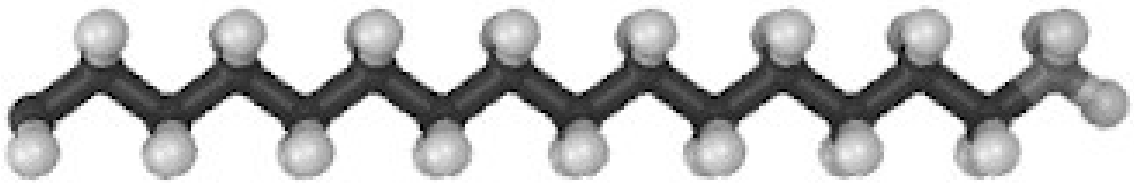
```
monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
delete monomers[15]
monomers[15] = new CH3.rot(180.0,0,0,1).move(18.795,0,0)

write('Data Bond List') {
  $bond:b1  $atom:monomers[0]/c $atom:monomers[1]/c
  $bond:b2  $atom:monomers[1]/c $atom:monomers[2]/c
  $bond:b3  $atom:monomers[2]/c $atom:monomers[3]/c
  ⋮                ⋮                    ⋮
  $bond:b15  $atom:monomers[14]/c $atom:monomers[15]/c
}
```

Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"



  monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
  delete monomers[15]
  monomers[15] = new CH3.rot(180.0,0,0,1).move(18.795,0,0)

  write('Data Bond List') {
    $bond:b1  $atom:monomers[0]/c $atom:monomers[1]/c
    $bond:b2  $atom:monomers[1]/c $atom:monomers[2]/c
    $bond:b3  $atom:monomers[2]/c $atom:monomers[3]/c
       :                :                  :
    $bond:b15  $atom:monomers[14]/c $atom:monomers[15]/c
  }
```

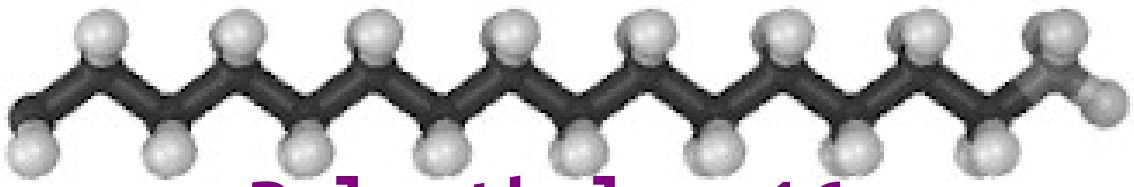
Hierarchical molecules



```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"

Polyethylene16  {

  monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
  delete monomers[15]
  monomers[15] = new CH3.rot(180.0,0,0,1).move(18.795,0,0)

  write('Data Bond List') {
    $bond:b1  $atom:monomers[0]/c $atom:monomers[1]/c
    $bond:b2  $atom:monomers[1]/c $atom:monomers[2]/c
    $bond:b3  $atom:monomers[2]/c $atom:monomers[3]/c
       ⋮              ⋮                  ⋮
    $bond:b15  $atom:monomers[14]/c $atom:monomers[15]/c
  }
```

Give this new construct a name

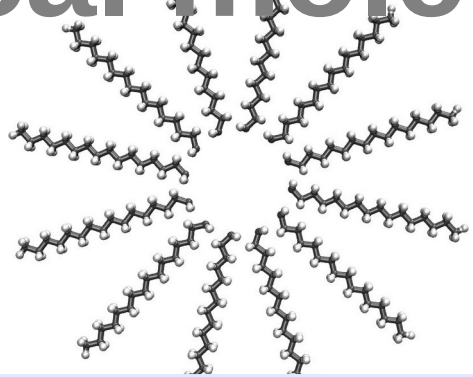
Hierarchical molecules



Polyethylene16

```
import "ch3group.lt"  # <-- defines "CH3"
import "ch2group.lt"  # <-- defines "CH2"

Polyethylene16   {

  monomers = new CH2[16].rot(180,1,0,0).move(1.253,0,0)
  delete monomers[15]
  monomers[15] = new CH3.rot(180.0,0,0,1).move(18.795,0,0)

  write('Data Bond List') {
    $bond:b1  $atom:monomers[0]/c $atom:monomers[1]/c
    $bond:b2  $atom:monomers[1]/c $atom:monomers[2]/c
    $bond:b3  $atom:monomers[2]/c $atom:monomers[3]/c

    $bond:b15  $atom:monomers[14]/c $atom:monomers[15]/c
  }
```

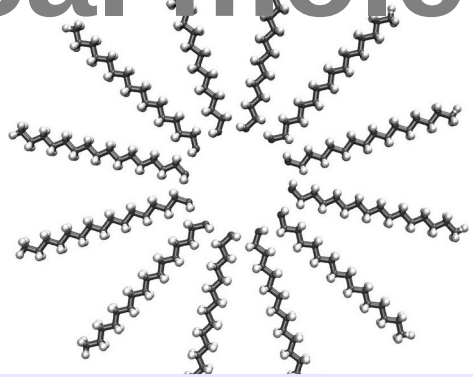
Hierarchical molecules



```
import "polyethylene16.lt"  # <-- defines "Polyethylene16"

StarPolymer {


  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1

}
```

Use it to create even bigger molecules

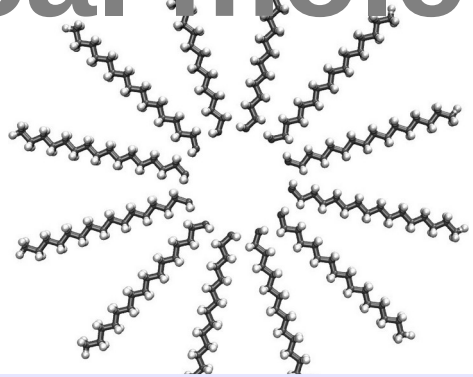
Hierarchical molecules



```
import "polyethylene16.lt"  # <-- defines "Polyethylene16"

StarPolymer {

  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1

}
```

Use it to create even bigger molecules

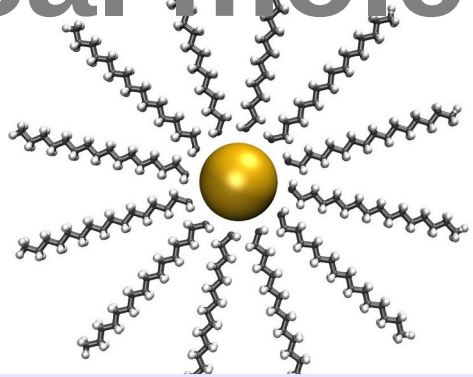
Hierarchical molecules



```
import "polyethylene16.lt"  # <-- defines "Polyethylene16"

StarPolymer {

  create_var { $mol }
  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1

}
```

All atoms in "StarPolymer" share the same molecule-ID

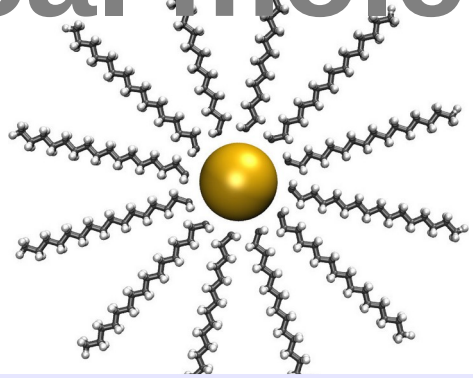
Hierarchical molecules



```
import "polymer.lt"  # <-- defines "Polymer16"

StarPolymer {

  create_var { $mol }
  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1
  write('Data Atoms') {
    $atom:cen $mol @atom:Center 0.0   0.00  0.00  0.00
  }
```

Add an atom at the center ("cen")

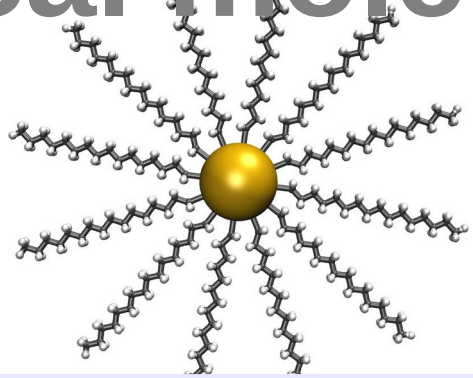
Hierarchical molecules



```
import "polyethylene16.lt"  # <-- defines "Polyethylene16"

StarPolymer {

  create_var { $mol }
  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1
  write('Data Atoms') {
    $atom:cen $mol @atom:Center 0.0   0.00  0.00  0.00
  }
  write_once("In Settings") {
    pair_coeff @atom:Center @atom:Center  0.01000  6.0000
  }
  write_once("Data Masses") {
    @atom:Center 500.0
  }
  ⋮
```

specify properties of new atom type

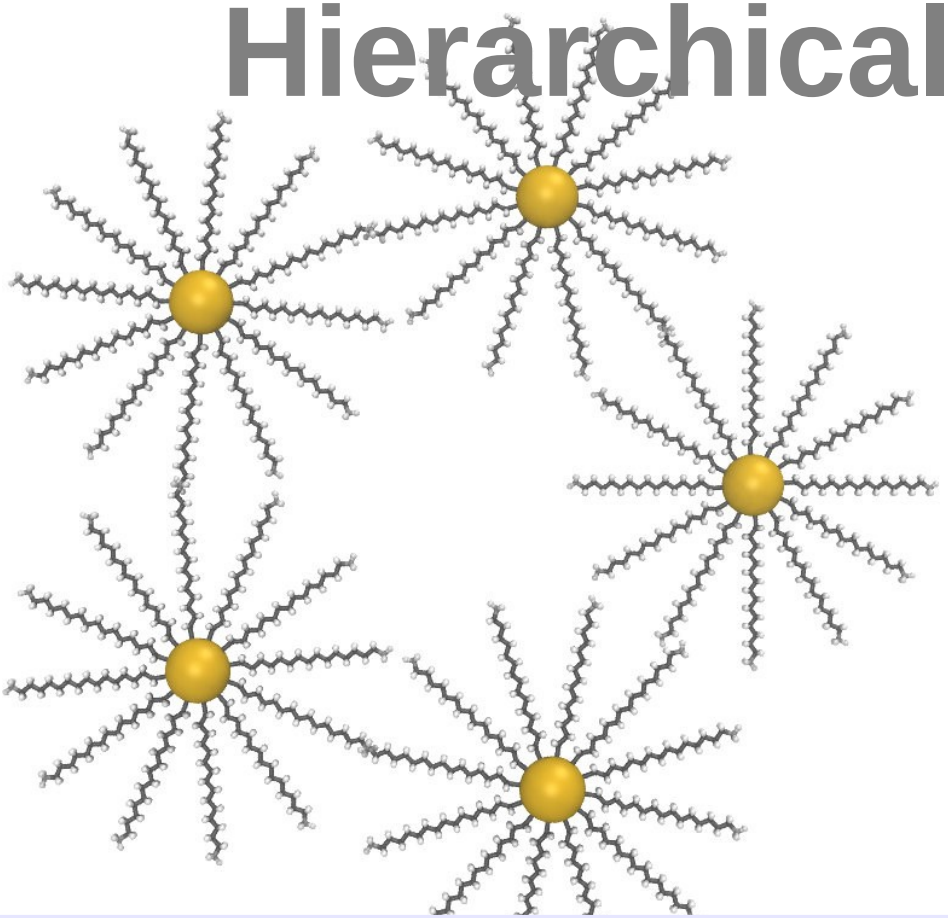
Hierarchical molecules



```
import "polyethylene16.lt"  # <-- defines "Polyethylene16"

StarPolymer {

  create_var { $mol }
  polys = new Polyethylene16.move(6.0,0,0) [12].rot(30,0,0,1
  ⋮
  write('Data Bonds') {
    $bond:b1 @bond:C $atom:cen $atom:polys[0]/monomers[0]/c
    $bond:b2 @bond:C $atom:cen $atom:polys[1]/monomers[0]/c
    ⋮        ⋮        ⋮        ⋮
    $bond:b12 @bond:C $atom:cen $atom:polys[11]/monomers[0]/
  }

}
```

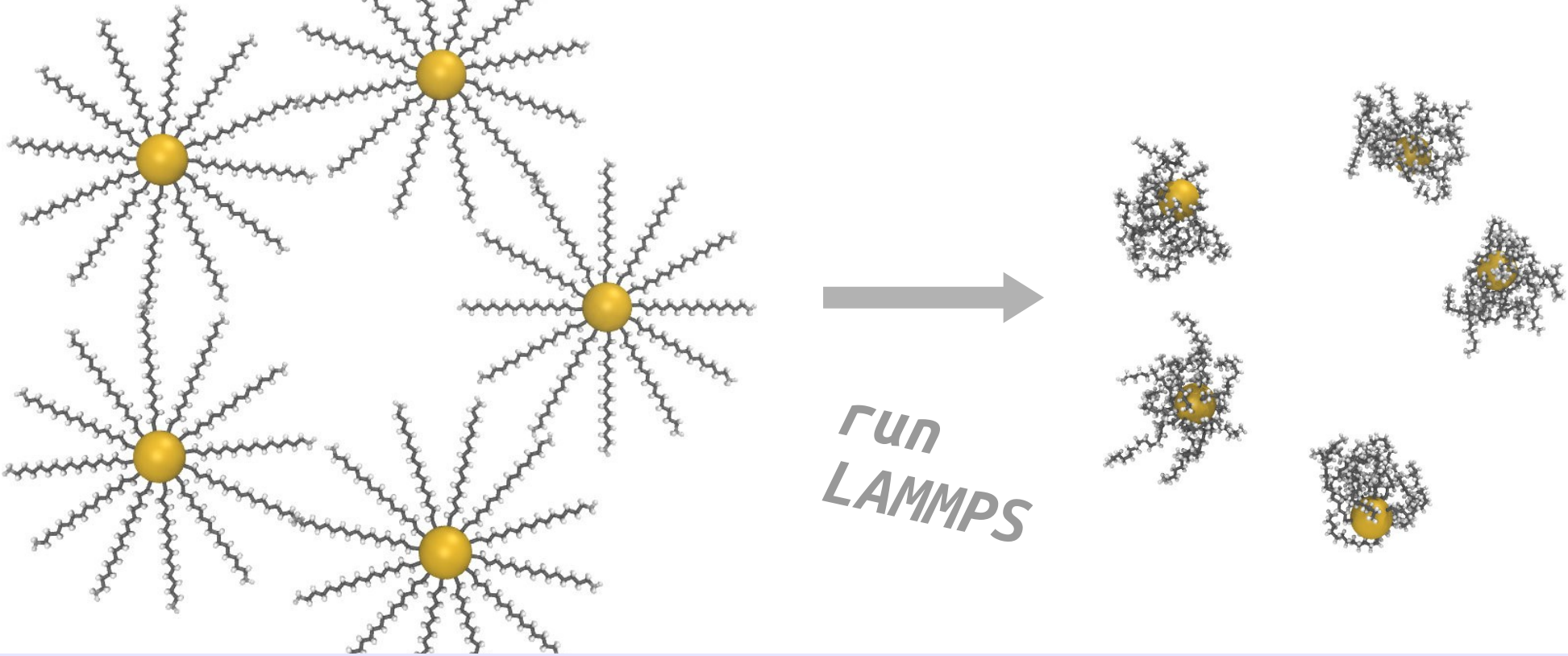
Connect each polymer to this central particle ("cen")

Hierarchical molecules



```
import "StarPolymer.lt"  # <-- defines "StarPolymer"

star1 = new StarPolymer.move(42.0,0,0)
star2 = new StarPolymer.move(42.0,0,0).rot(72.0,0,0,1)
star3 = new StarPolymer.move(42.0,0,0).rot(144.0,0,0,1)
star4 = new StarPolymer.move(42.0,0,0).rot(216.0,0,0,1)
star5 = new StarPolymer.move(42.0,0,0).rot(288.0,0,0,1)
```

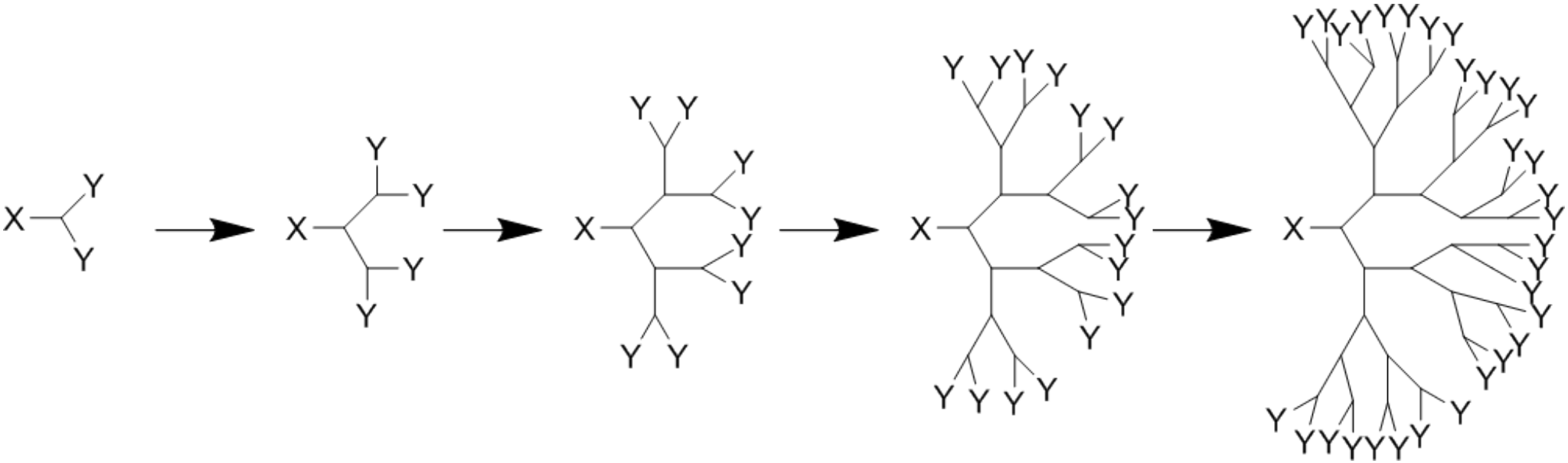
Hierarchical molecules




```
import "StarPolymer.lt"  # <-- defines "StarPolymer"

star1 = new StarPolymer.move(42.0,0,0)
star2 = new StarPolymer.move(42.0,0,0).rot(72.0,0,0,1)
star3 = new StarPolymer.move(42.0,0,0).rot(144.0,0,0,1)
star4 = new StarPolymer.move(42.0,0,0).rot(216.0,0,0,1)
star5 = new StarPolymer.move(42.0,0,0).rot(288.0,0,0,1)
```

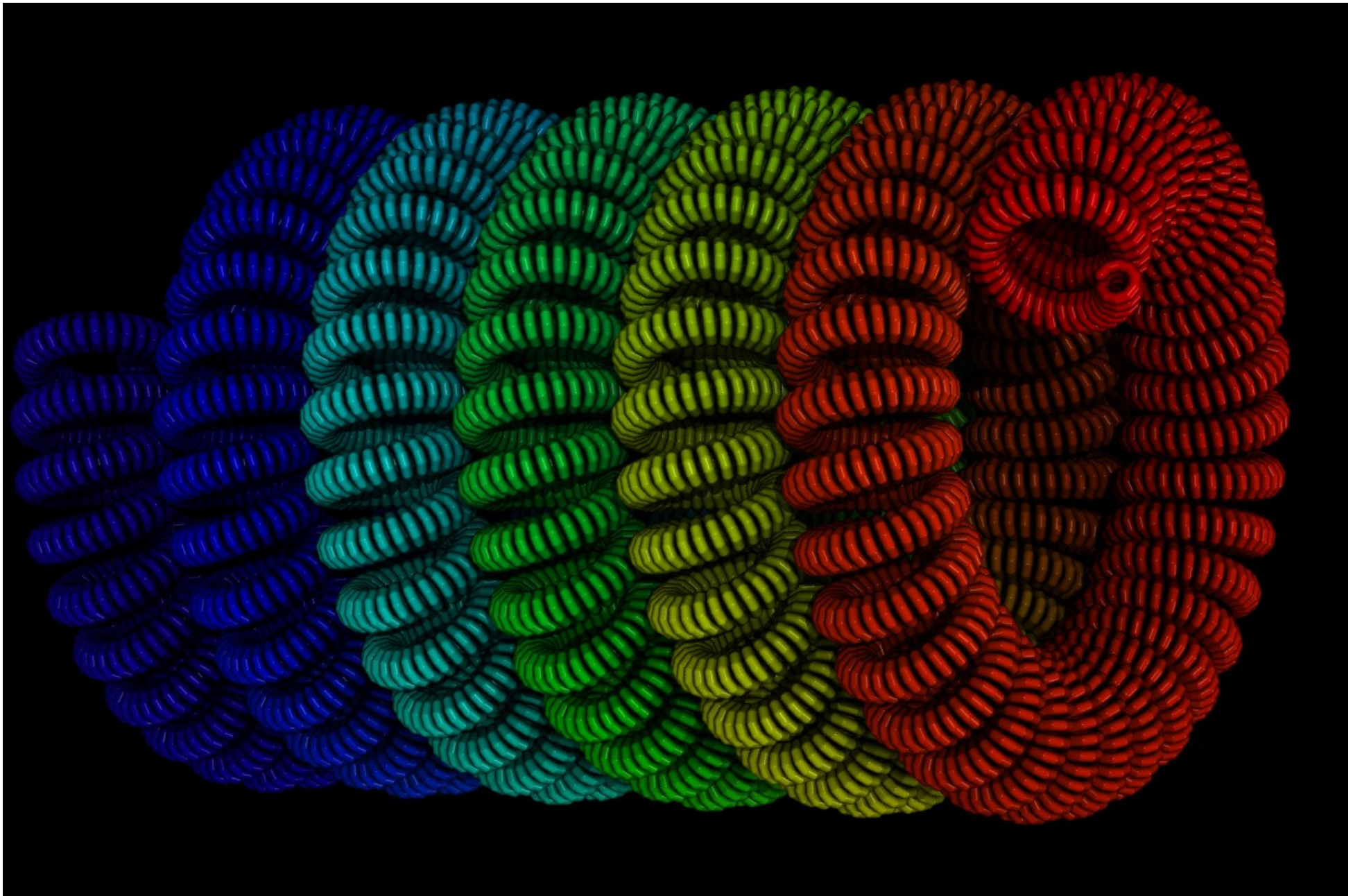
Hierarchical molecules

(example: dendrimers)


X
Y
Y
X
Y
Y
Y
Y
X
Y
Y
Y
Y
Y
Y
Y
Y
X
Y
Y
Y
Y
Y
Y
Y
Y

Hierarchical systems

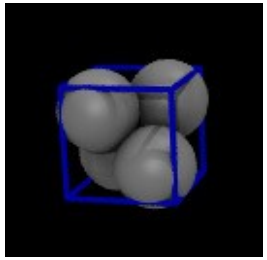
(example: useless supercoil fractal)



Hierarchical systems

Objects can be built from smaller objects

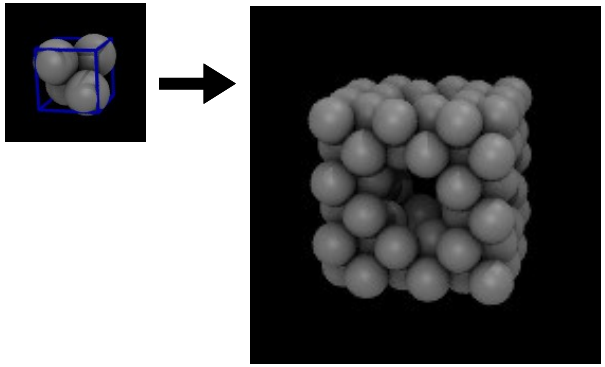
AlCell



```
# "AlCell" defines the 4-atom FCC unit cell
# of Aluminum (with a 4.05 angstrom spacing)

AlCell
{
  # AtomID     AtomType  Charge    X     Y     Z

  write("Data Atoms") {
    $atom:AlC  @atom:Al    0.0   0.000 0.000 0.000
    $atom:AlX  @atom:Al    0.0   0.000 2.025 2.025
    $atom:AlY  @atom:Al    0.0   2.025 0.000 2.025
    $atom:AlZ  @atom:Al    0.0   2.025 2.025 0.000
  }

  write_once("In Settings") {
    pair_coeff  * *  eam/alloy  Al99.eam.alloy Al
  }

  write_once("Data Masses") {
    @atom:Al 27.0
  }

}  # AlCell
```

Hierarchical systems

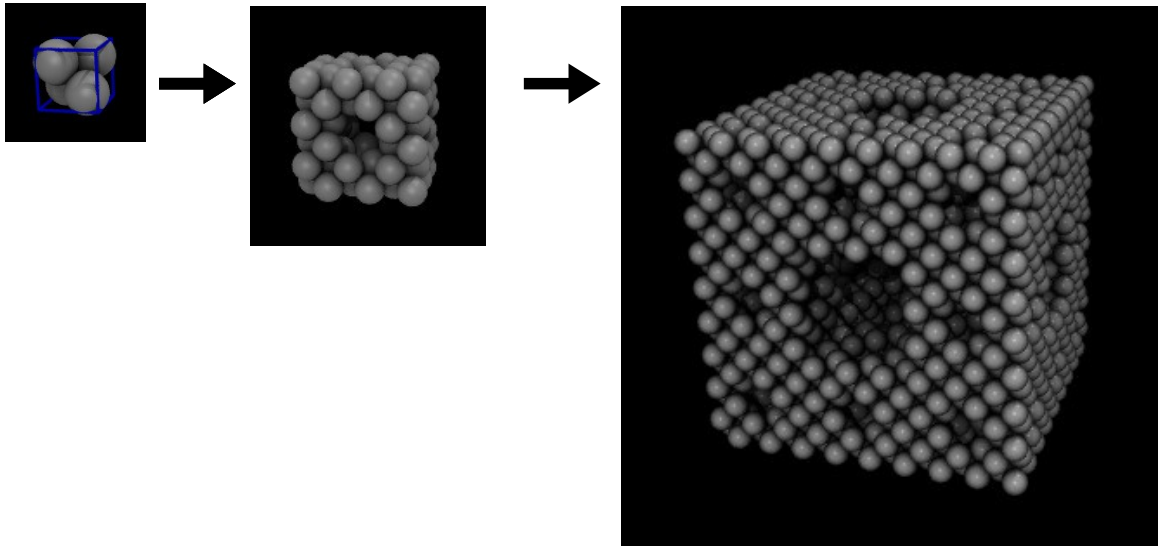
AlCell *SierpinskiCubeLvl1*



```
SierpinskiCubeLvl1
{
  cells = new AlCell [3].move(0.00, 0.00, 4.05)
                     [3].move(0.00, 4.05, 0.00)
                     [3].move(4.05, 0.00, 0.00)
  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```

Hierarchical systems

AlCell *SierpinskiCubeLvl1* *SierpinskiCubeLvl2*

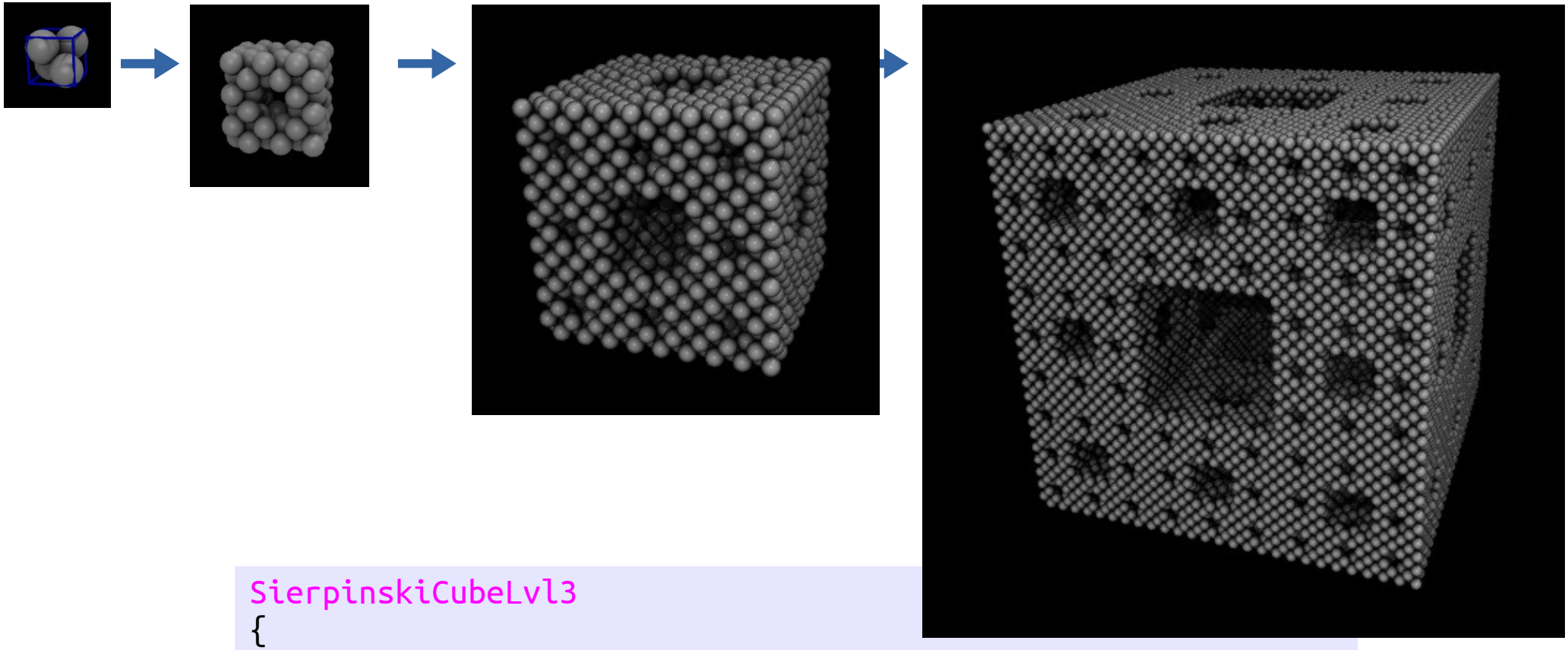


```
SierpinskiCubeLvl2
{
  cells = new SierpinskiCubeLvl1 [3].move(0.00, 0.00, 12.15)
                                 [3].move(0.00, 12.15, 0.00)
                                 [3].move(12.15, 0.00, 0.00)

  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```

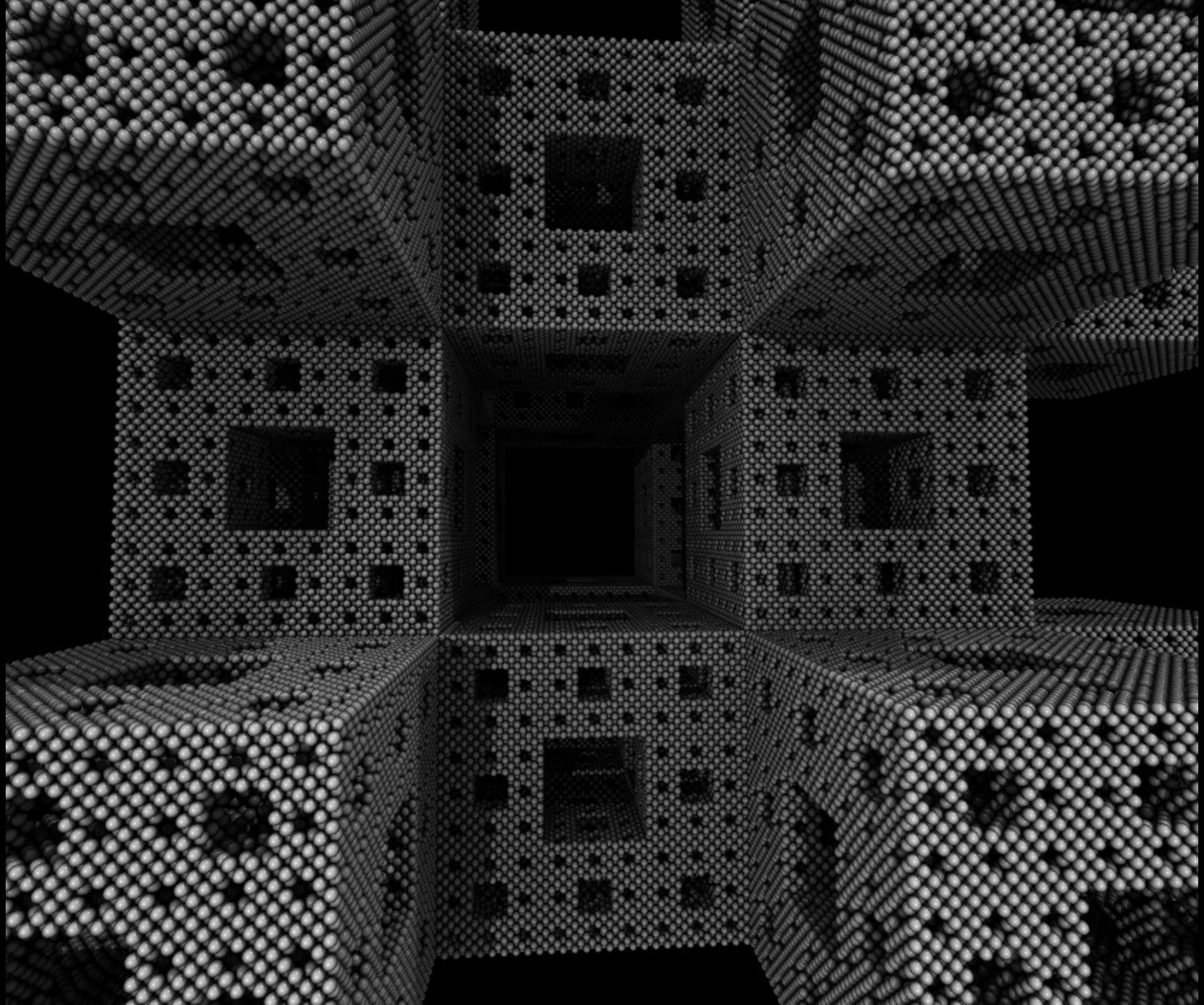
Hierarchical systems

(example: low density aluminum fractal)

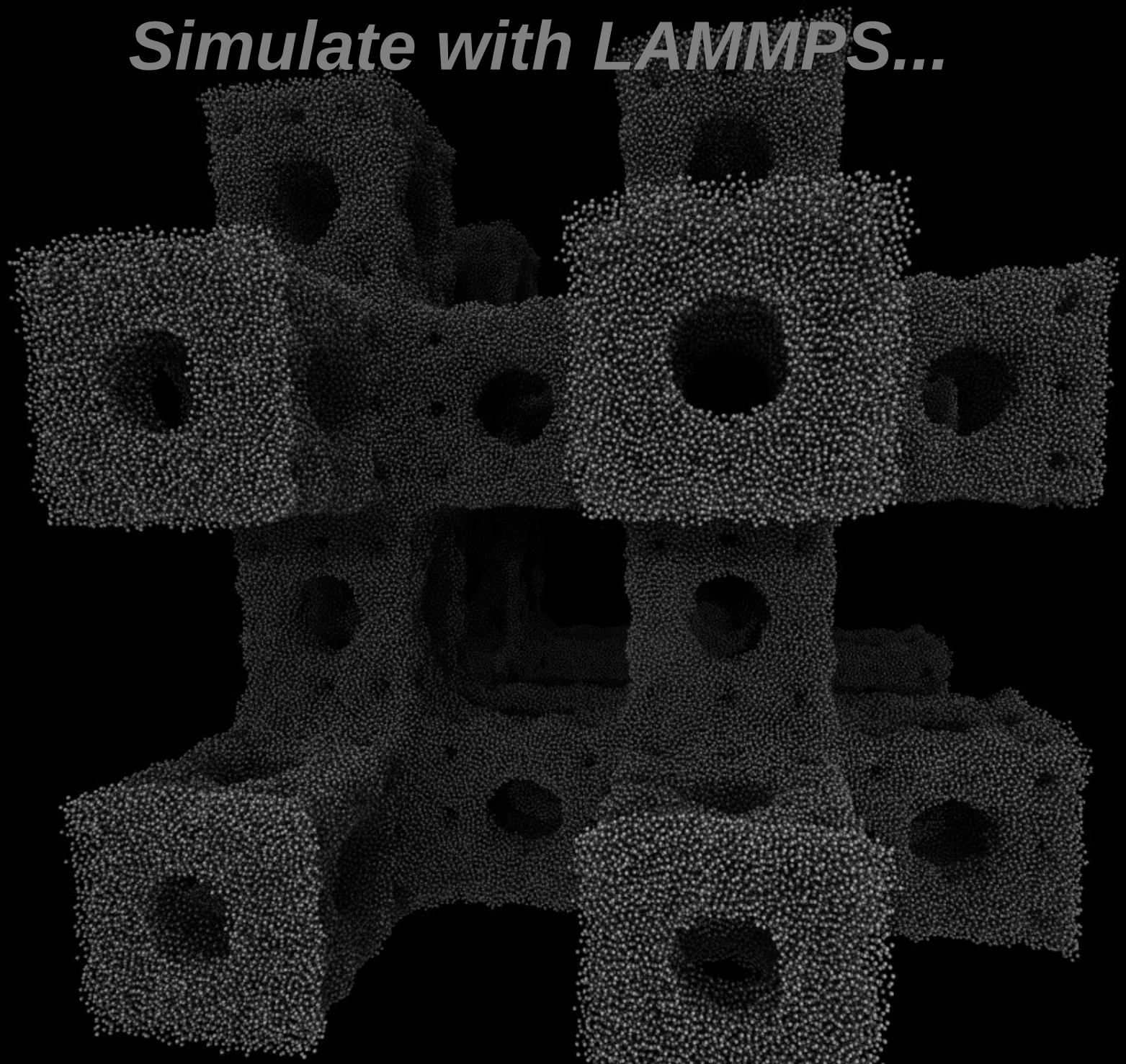
AlCell *SierpinskiCubeLvl1* *SierpinskiCubeLvl2* *SierpinskiCubeLvl3*



```
SierpinskiCubeLvl3
{
  cells = new SierpinskiCubeLvl2 [3].move(0.00, 0.00, 36.45)
                                 [3].move(0.00, 36.45, 0.00)
                                 [3].move(36.45, 0.00, 0.00)

  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```



Simulate with LAMMPS...



(...this arrangement of atoms is not stable)

Hierarchical systems

(example: hierarchical text file generator)

```
Msg {
  write() {
   On the ${day}th day of Christmas, my true love gave to me:
  }
}
Gifts1 {
  write(){
    @day partridge in a pear tree.
  }
}
Gifts2 {
  write(){
    @day turtle doves, and
  }
  gifts = new Gifts1
}
Gifts3 {
  write(){@day french hens,
  }
  gifts = new Gifts2
}
⋮
Msg1 inherits Msg{g = new Gifts1}
Msg2 inherits Msg{g = new Gifts2}
⋮
```

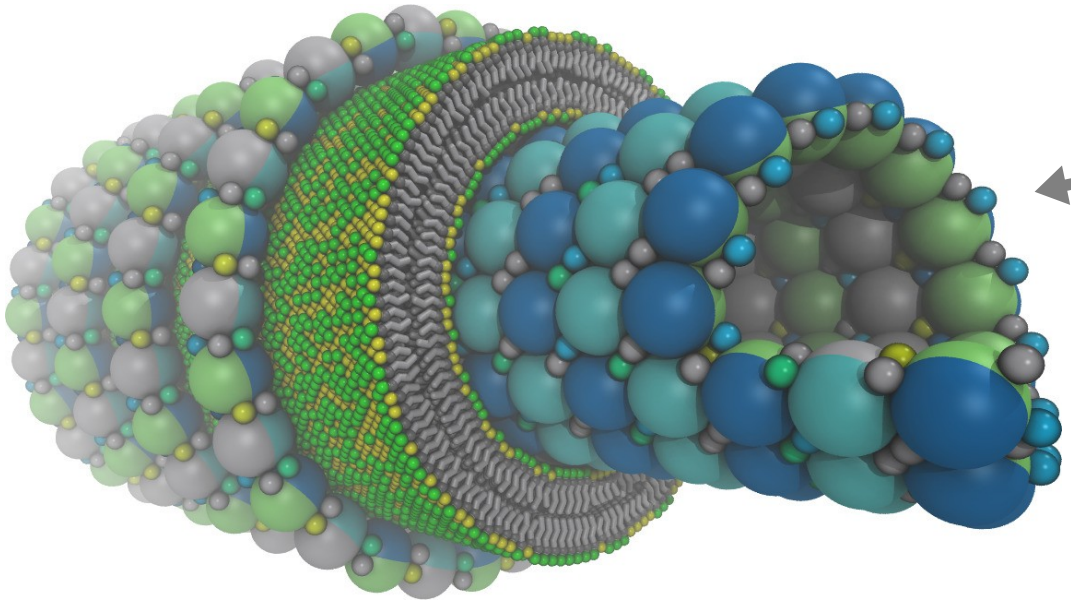
Hierarchical systems

song lyrics →

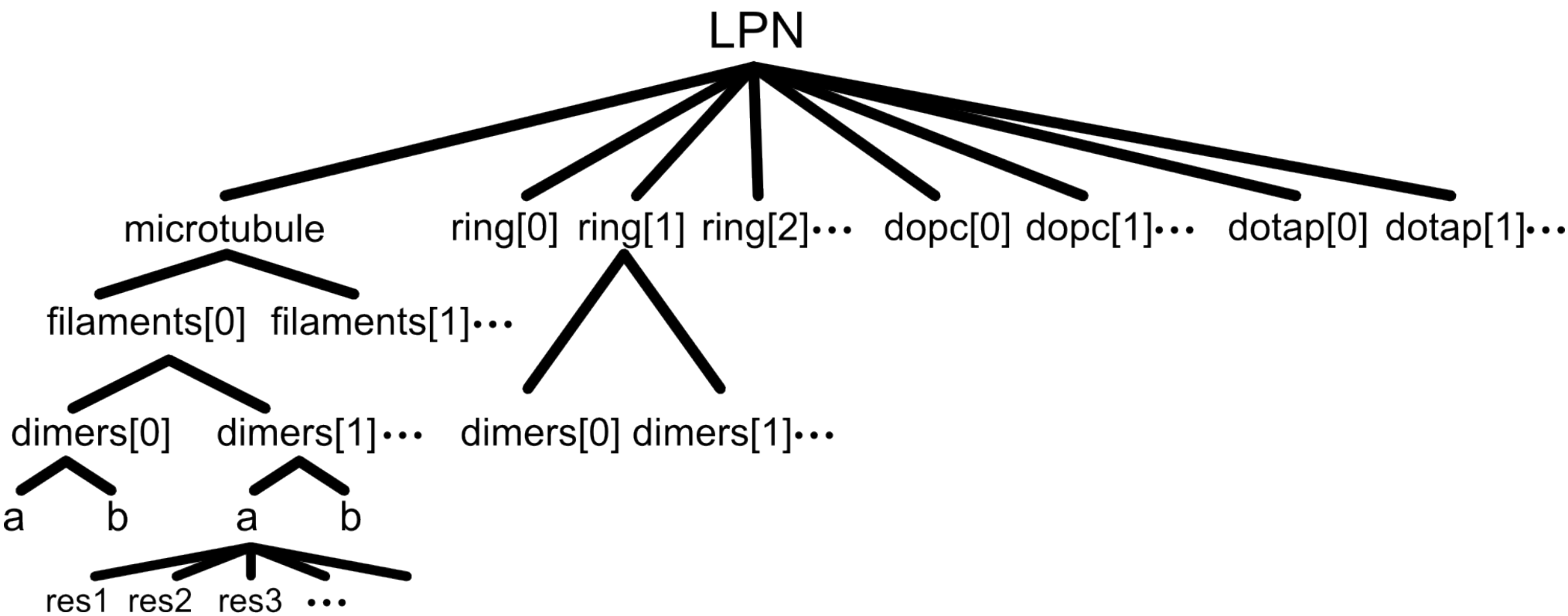
```
Msg {
  write() {
   On the ${day}th day of Christmas, my true love gave to me:
  }
}
Gifts1 {
  write(){
    @day partridge in a pear tree.
  }
}
Gifts2 {
  write(){
    @day turtle doves, and
  }
  gifts = new Gifts1
}
Gifts3 {
  write(){@day french hens,
  }
  gifts = new Gifts2
}
⋮
Msg1 inherits Msg{g = new Gifts1}
Msg2 inherits Msg{g = new Gifts2}
⋮
```

```
On the 1th day of Christmas, my true love gave to me:
1 partridge in a pear tree.

   On the 2th day of Christmas, my true love gave to me:
2 turtle doves, and
1 partridge in a pear tree.

   On the 3th day of Christmas, my true love gave to me:
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 4th day of Christmas, my true love gave to me:
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 5th day of Christmas, my true love gave to me:
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 6th day of Christmas, my true love gave to me:
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 7th day of Christmas, my true love gave to me:
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 8th day of Christmas, my true love gave to me:
8 maids a-milking,
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 9th day of Christmas, my true love gave to me:
9 ladies dancing,
8 maids a-milking,
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 10th day of Christmas, my true love gave to me:
10 lords a-leaping,
9 ladies dancing,
8 maids a-milking,
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 11th day of Christmas, my true love gave to me:
11 pipers piping,
10 lords a-leaping,
9 ladies dancing,
8 maids a-milking,
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.

   On the 12th day of Christmas, my true love gave to me:
12 drummers drumming,
11 pipers piping,
10 lords a-leaping,
9 ladies dancing,
8 maids a-milking,
7 swans a-swimming,
6 geese a-laying,
5 golden rings,
4 calling birds,
3 french hens,
2 turtle doves, and
1 partridge in a pear tree.
```

Hierarchical molecules



LPN example (again)



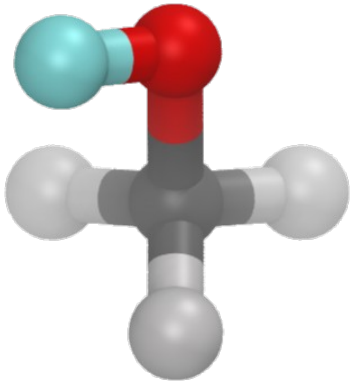
File format details

Note:
The next 50 slides were omitted from the talk I gave at the LAMMPS conference.

If you do not care about the details of how bonded interactions are described, skip these slides.

File format details

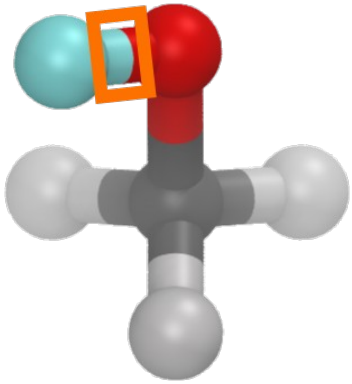
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType   AtomID1  AtomID2
    $bond:b1 @bond:O_H  $atom:o  $atom:ho
    $bond:b2 @bond:C_H  $atom:c  $atom:hc1
    $bond:b3 @bond:C_H  $atom:c  $atom:hc2
    $bond:b4 @bond:C_H  $atom:c  $atom:hc3
    $bond:b5 @bond:C_O  $atom:c  $atom:o
  }
  ⋮
}
```

File format details

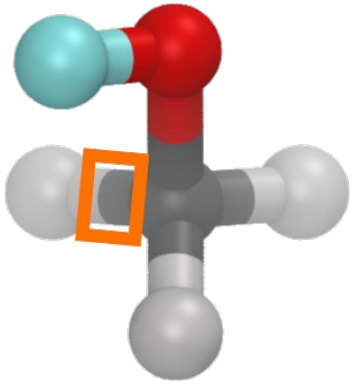
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType    AtomID1   AtomID2
    $bond:b1 @bond:O_H   $atom:o   $atom:ho
    $bond:b2 @bond:C_H   $atom:c   $atom:hc1
    $bond:b3 @bond:C_H   $atom:c   $atom:hc2
    $bond:b4 @bond:C_H   $atom:c   $atom:hc3
    $bond:b5 @bond:C_O   $atom:c   $atom:o
  }
  ⋮
}
```

File format details

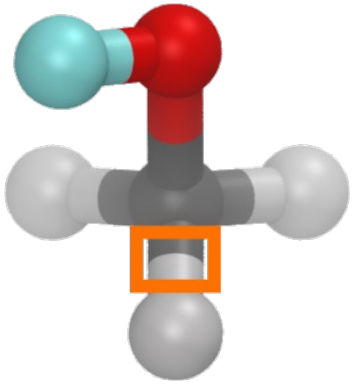
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType   AtomID1  AtomID2
    $bond:b1 @bond:O_H  $atom:o   $atom:ho
    $bond:b2 @bond:C_H  $atom:c   $atom:hc1
    $bond:b3 @bond:C_H  $atom:c   $atom:hc2
    $bond:b4 @bond:C_H  $atom:c   $atom:hc3
    $bond:b5 @bond:C_O  $atom:c   $atom:o
  }
  ⋮
}
```

File format details

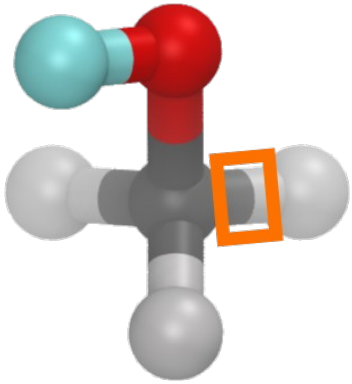
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType    AtomID1   AtomID2
    $bond:b1 @bond:O_H   $atom:o   $atom:ho
    $bond:b2 @bond:C_H   $atom:c   $atom:hc1
    $bond:b3 @bond:C_H   $atom:c   $atom:hc2
    $bond:b4 @bond:C_H   $atom:c   $atom:hc3
    $bond:b5 @bond:C_O   $atom:c   $atom:o
  }
  ⋮
}
```

File format details

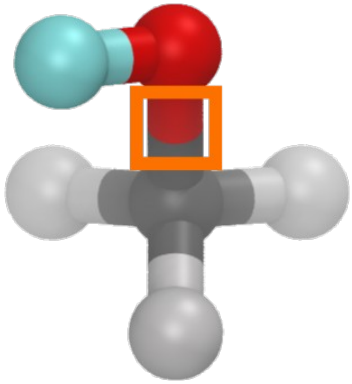
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType    AtomID1   AtomID2
    $bond:b1 @bond:O_H   $atom:o   $atom:ho
    $bond:b2 @bond:C_H   $atom:c   $atom:hc1
    $bond:b3 @bond:C_H   $atom:c   $atom:hc2
    $bond:b4 @bond:C_H   $atom:c   $atom:hc3
    $bond:b5 @bond:C_O   $atom:c   $atom:o
  }
  ⋮
}
```

File format details

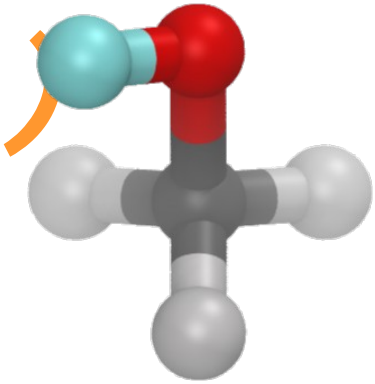
Bonds (must be included)



```
Methanol {
  ⋮
  write("Data Bonds") {
    # BondID BondType    AtomID1   AtomID2
    $bond:b1 @bond:O_H   $atom:o   $atom:ho
    $bond:b2 @bond:C_H   $atom:c   $atom:hc1
    $bond:b3 @bond:C_H   $atom:c   $atom:hc2
    $bond:b4 @bond:C_H   $atom:c   $atom:hc3
    $bond:b5 @bond:C_O   $atom:c   $atom:o
  }
  ⋮
}
```

File format details

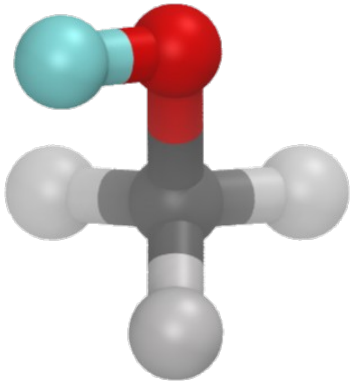
Bond (force field parameters)



```
Methanol {
  ⋮
  write_once("In Settings") {
    #           BondType  bond_style  k    r0
    bond_coeff @bond:C_O harmonic 316.7 1.4233
    bond_coeff @bond:O_H harmonic 371.4 0.973
    bond_coeff @bond:C_H harmonic 330.6 1.0969
  }
```

File format details

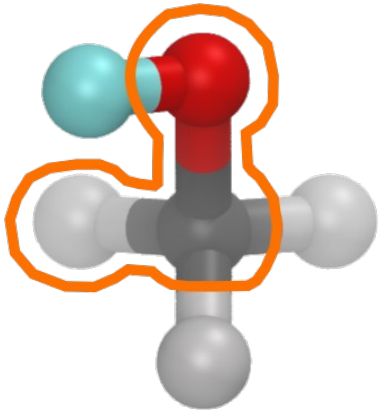
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

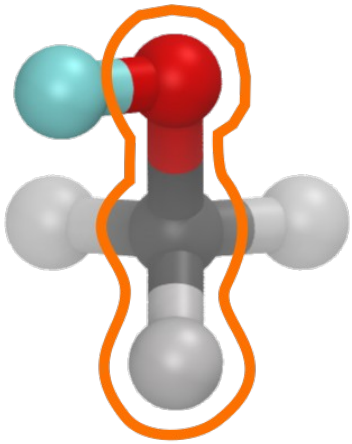
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

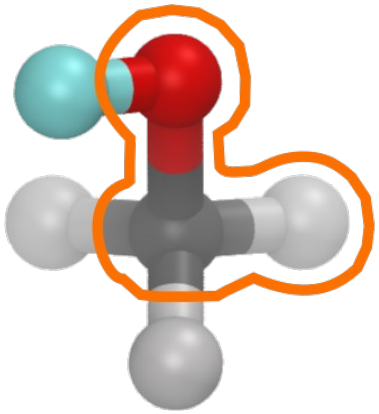
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

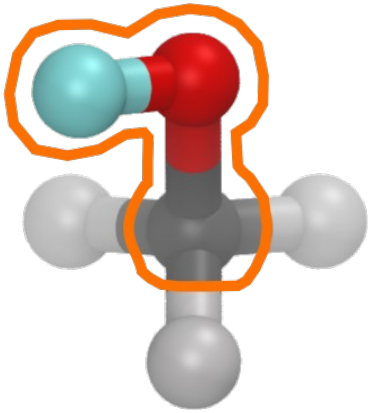
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

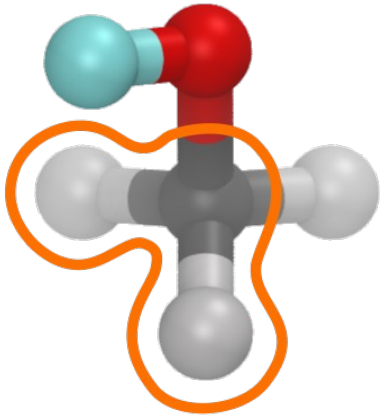
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

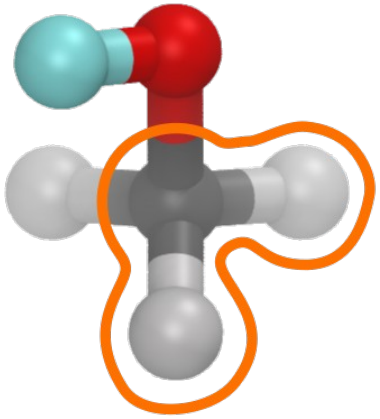
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID   AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

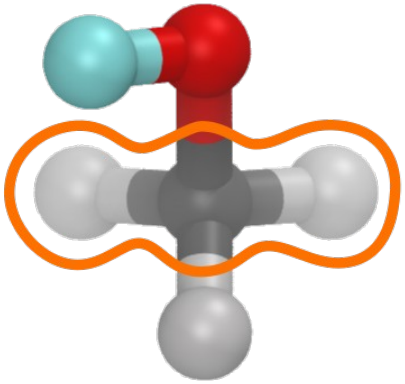
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

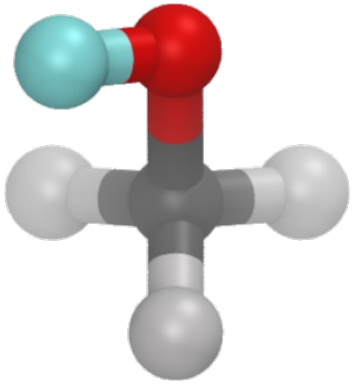
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

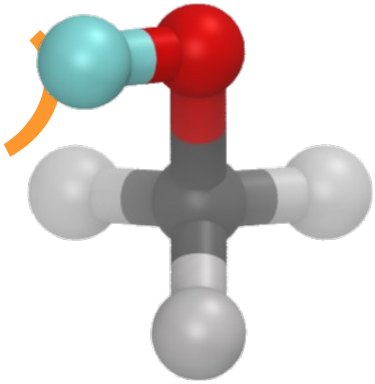
Angles (optional)



```
Methanol {
  ⋮
  write("Data Angles") {
    # AngleID  AngleType      AtomID1 AtomID2 AtomID3
    $angle:an1 @angle:O_C_H  $atom:o $atom:c $atom:hc1
    $angle:an2 @angle:O_C_H  $atom:o $atom:c $atom:hc2
    $angle:an3 @angle:O_C_H  $atom:o $atom:c $atom:hc3
    $angle:an4 @angle:C_O_H  $atom:c $atom:o $atom:ho
    $angle:an5 @angle:H_C_H  $atom:hc1 $atom:c $atom:hc2
    $angle:an6 @angle:H_C_H  $atom:hc2 $atom:c $atom:hc3
    $angle:an7 @angle:H_C_H  $atom:hc3 $atom:c $atom:hc1
  }
```

File format details

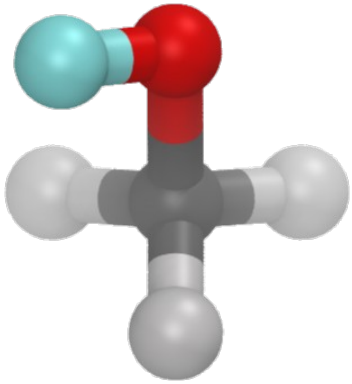
Angle (force field parameters)



```
Methanol {
  ⋮

  write_once("In Settings") {
    #            AngleType  angle_style  k    theta0
    angle_coeff @angle:O_C_H harmonic 50.93 110.26
    angle_coeff @angle:C_O_H harmonic 47.38 107.26
    angle_coeff @angle:H_C_H harmonic 39.24 108.46
  }

  ⋮
```

File format details

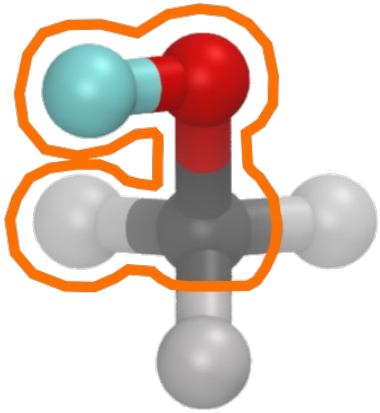
Dihedrals (optional)



```
Methanol {
  ⋮

  write("Data Dihedrals") {
    # DihedralID  DihedralType   AtomID1 AtomID2 AtomID3 AtomID4
    $dihedral:d1 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc1
    $dihedral:d2 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc2
    $dihedral:d3 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc3
  }

  ⋮
}
```

File format details

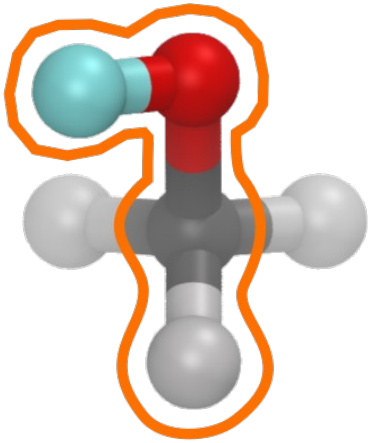
Dihedrals (optional)



```
Methanol {
  ⋮

  write("Data Dihedrals") {
    # DihedralID  DihedralType   AtomID1 AtomID2 AtomID3 AtomID4
    $dihedral:d1 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc1
    $dihedral:d2 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc2
    $dihedral:d3 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc3
  }

  ⋮
}
```

File format details

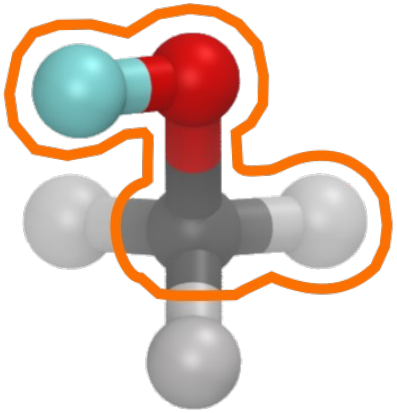
Dihedrals (optional)



```
Methanol {
  ⋮

  write("Data Dihedrals") {
    # DihedralID  DihedralType   AtomID1 AtomID2 AtomID3 AtomID4
    $dihedral:d1 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc1
    $dihedral:d2 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc2
    $dihedral:d3 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc3
  }

  ⋮
}
```

File format details

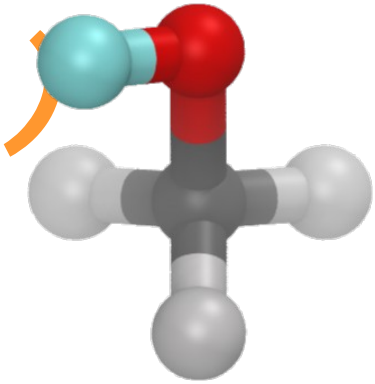
Dihedrals (optional)



```
Methanol {
  ⋮

  write("Data Dihedrals") {
    # DihedralID  DihedralType   AtomID1 AtomID2 AtomID3 AtomID4
    $dihedral:d1 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc1
    $dihedral:d2 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc2
    $dihedral:d3 @dihedral:HOCH $atom:ho $atom:o $atom:c $atom:hc3
  }

  ⋮
}
```

File format details

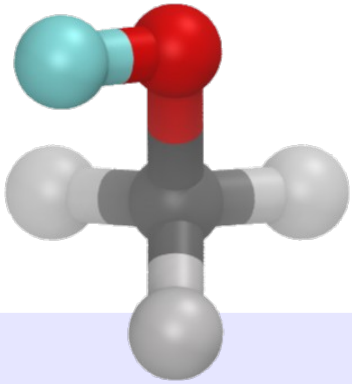
Dihedral (force field parameters)



```
Methanol {
  ⋮

  write_once("In Settings") {
    #                DihedralType   dihedral_style  K   d n
    dihedral_coeff @dihedral:H_O_C_H harmonic 0.1667 1 3
  }

  ⋮

}
```

File format details

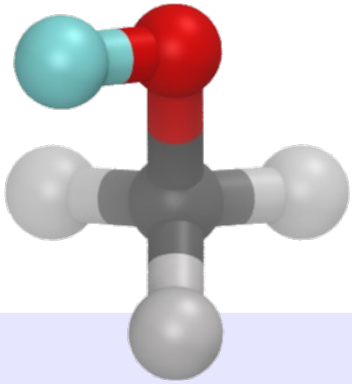
force field parameters can be included in the molecule definition



```
Methanol {
  ⋮
  Write_once("In Settings") {
    #          BondType  bond_style  k    r0
    bond_coeff @bond:C_O harmonic 316.7 1.4233
    bond_coeff @bond:O_H harmonic 371.4 0.973
    bond_coeff @bond:C_H harmonic 330.6 1.0969
    #           AngleType  angle_style  k    theta0
    angle_coeff @angle:O_C_H harmonic 50.93 110.26
    angle_coeff @angle:C_O_H harmonic 47.38 107.26
    angle_coeff @angle:H_C_H harmonic 39.24 108.46
    #              DihedralType   dihedral_style  K  d n
    dihedral_coeff @dihedral:H_O_C_H harmonic 0.1667 1 3
    #         AtomType1 AtomType2     pair_style    epsilon sigma
    pair_coeff @atom:O  @atom:O  lj/charmm/coul/long 0.2104 3.066473
    pair_coeff @atom:C  @atom:C  lj/charmm/coul/long 0.1094 3.399670
```

File format details

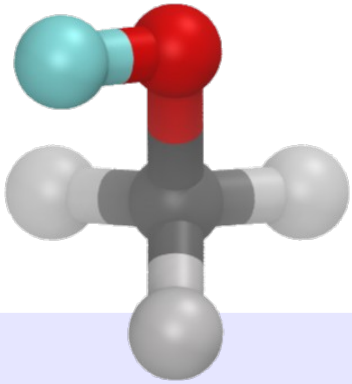
Miscellaneous other settings and even auxiliary files can also be included here too.



```
Methanol {
  ⋮

  write_once("In Init") {
    units real
    atom_style full
    pair_style hybrid lj/charmm/coul/long 11.0 12.0
    bond_style hybrid harmonic
    angle_style hybrid harmonic
    dihedral_style hybrid harmonic
    kspace_style pppm/cg 1.0e-5
    pair_modify mix arithmetic
    special_bonds amber
  }
```

File format details

Miscellaneous other settings and even auxiliary files can also be included here too.

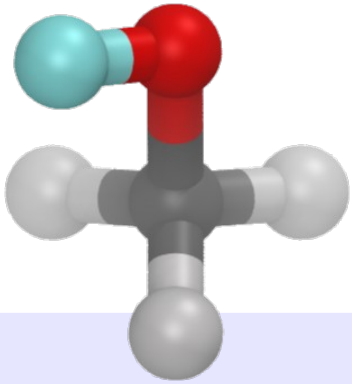


```
Methanol {
  ⋮

  write_once("In Init") {
    units real
    atom_style full
    pair_style hybrid lj/charmm/coul/long 11.0 12.0
    bond_style hybrid harmonic
    angle_style hybrid harmonic
    dihedral_style hybrid harmonic
    kspace_style pppm/cg 1.0e-5
    pair_modify mix arithmetic
    special_bonds amber
  }
```

Not Recommended

File format details

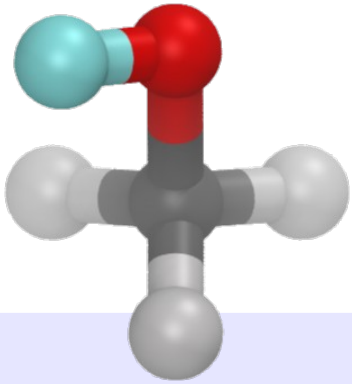
Force field parameters can be stored separately



```
ForceField {
  ⋮
}
Methanol inherits ForceField {
  write("Data Atoms") {
    $atom:o    $mol:m  @atom:O  -0.5988  0.708  0.0    0.0
    $atom:c    $mol:m  @atom:C   0.1167 -0.708  0.0    0.0
    $atom:hc1  $mol:m  @atom:HC  0.0287 -1.073 -0.769  0.685
    $atom:hc2  $mol:m  @atom:HC  0.0287 -1.073 -0.195 -1.011
    $atom:hc3  $mol:m  @atom:HC  0.0287 -1.063  0.979  0.331
    $atom:ho   $mol:m  @atom:HO  0.3960  0.994 -0.88  -0.298
  }
  write("Data Bonds") {
    $bond:b1 @bond:O_H  $atom:o  $atom:ho
    $bond:b2 @bond:C_H  $atom:c  $atom:hc1
    $bond:b3 @bond:C_H  $atom:c  $atom:hc2
    $bond:b4 @bond:C_H  $atom:c  $atom:hc3
    $bond:b5 @bond:C_O  $atom:c  $atom:o
```

File format details

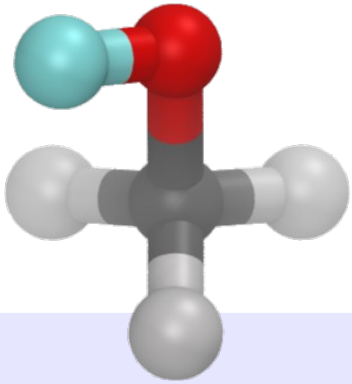
Force field parameters can be stored separately



```
ForceField {
  ⋮
}

Methanol inherits ForceField {
  write("Data Atoms")     {...}
  write("Data Bond List") {...}
}
```

Creating a Force Field

Force field parameters
*can be **stored separately***
*and **shared** between molecules*

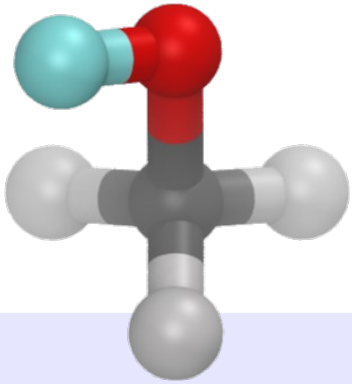


```
ForceField {
  ⋮
}

Methanol inherits ForceField {
  write("Data Atoms")     {...}
  write("Data Bond List") {...}
}

Ethyltoluene3 inherits ForceField {
  write("Data Atoms")     {...}
  write("Data Bond List") {...}
}
⋮
```

only atom coords and bonds are needed

Creating a Force Field

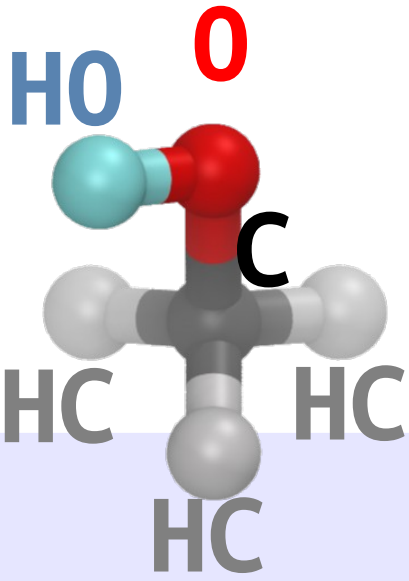
Force field parameters
can be stored separately



```
ForceField {

  Write_once("In Settings") {
    #          BondType  bond_style  k    r0
    bond_coeff @bond:C_O harmonic 316.7 1.4233
    bond_coeff @bond:O_H harmonic 371.4 0.973
    bond_coeff @bond:C_H harmonic 330.6 1.0969
    #           AngleType  angle_style  k    theta0
    angle_coeff @angle:O_C_H harmonic 50.93 110.26
    angle_coeff @angle:C_O_H harmonic 47.38 107.26
    angle_coeff @angle:H_C_H harmonic 39.24 108.46
    #              DihedralType   dihedral_style  K  d n
    dihedral_coeff @dihedral:H_O_C_H harmonic 0.1667 1 3
    #         AtomType1 AtomType2     pair_style    epsilon sigma
    pair_coeff @atom:O  @atom:O  lj/charmm/coul/long 0.2104 3.066473
    pair_coeff @atom:C  @atom:C  lj/charmm/coul/long 0.1094 3.399670
```

Creating a Force Field

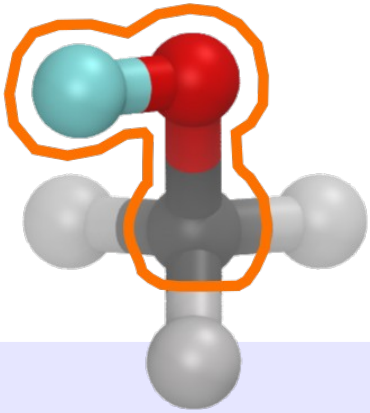
Rules for generating angles (by atom type)




```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮



}
```

Creating a Force Field

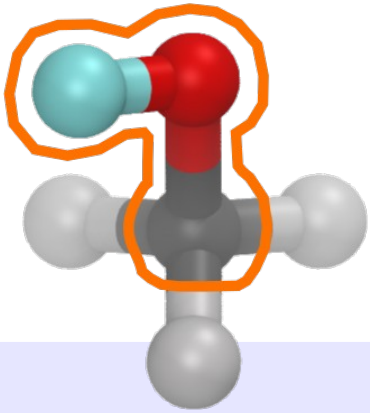
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H    $atom:O  $atom:C  $atom:HC
    @angle:C_O_H    $atom:C  $atom:O  $atom:HO
    @angle:H_C_H    $atom:HC $atom:C  $atom:HC
  }
  ⋮
}
```

Creating a Force Field

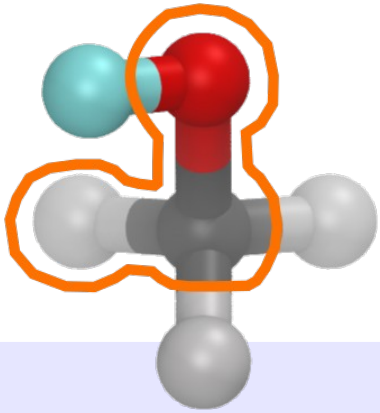
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮
}
```

Creating a Force Field

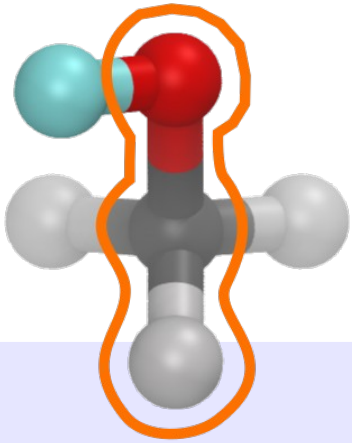
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮



}
```

Creating a Force Field

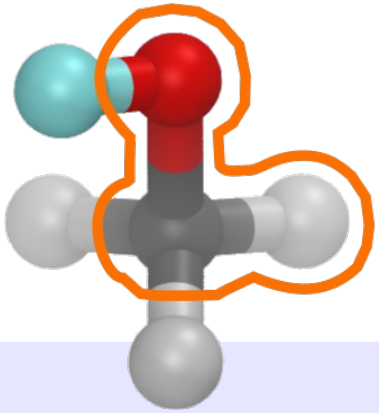
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮



}
```

Creating a Force Field

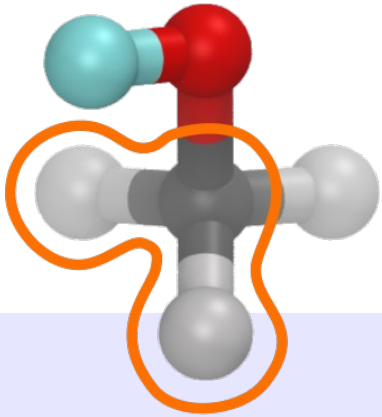
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮



}
```

Creating a Force Field

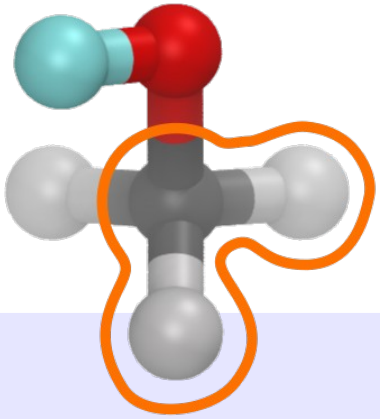
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮
}
```

Creating a Force Field

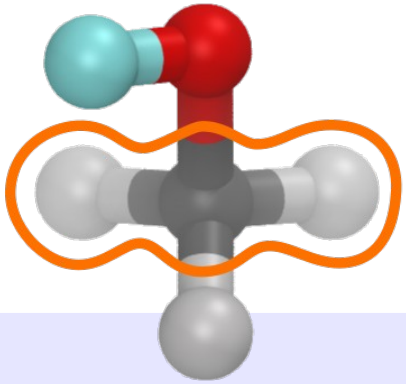
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮
}
```

Creating a Force Field

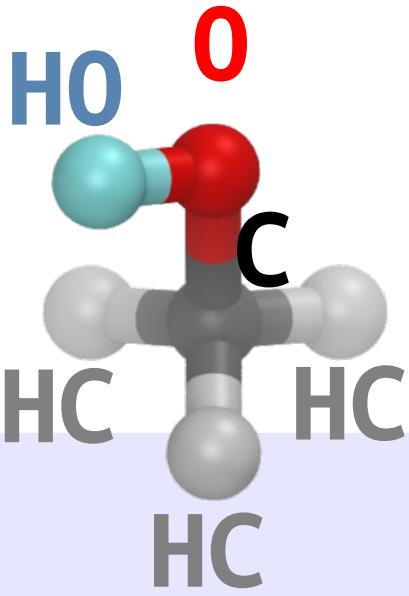
Angles (by atom type)



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮
}
```

Creating a Force Field

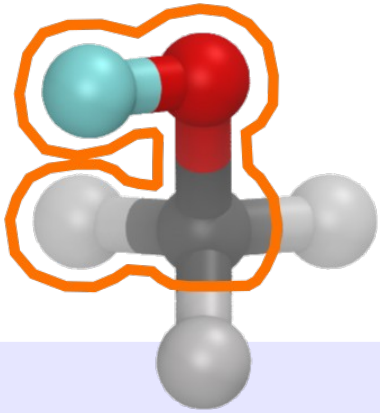
Dihedrals (by atom type)




```
ForceField {
  ⋮
  write("Data Dihedrals By Type") {
    @dihedral:HOCH $atom:HO $atom:O $atom:C $atom:HC
  }
  ⋮
}
```

Creating a Force Field

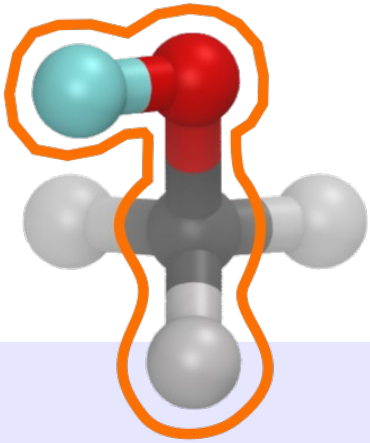
Dihedrals (by atom type)



```
ForceField {
  ⋮
  write("Data Dihedrals By Type") {
    @dihedral:HOCH $atom:HO $atom:O $atom:C $atom:HC
  }
  ⋮
}
```

Creating a Force Field

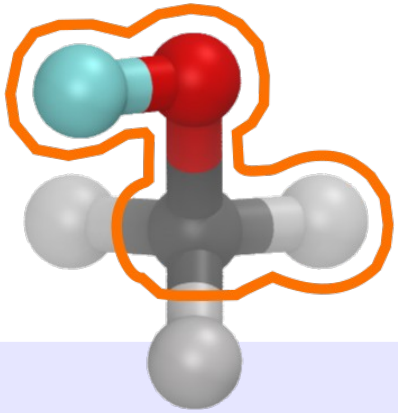
Dihedrals (by atom type)



```
ForceField {
  ⋮
  write("Data Dihedrals By Type") {
    @dihedral:HOCH $atom:HO $atom:O $atom:C $atom:HC
  }
  ⋮
}
```

Creating a Force Field

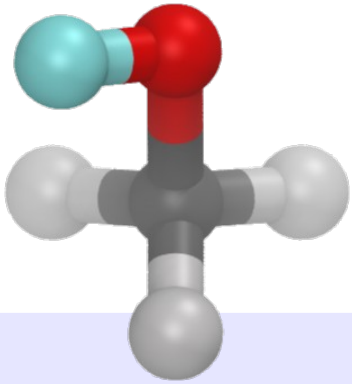
Dihedrals (by atom type)



```
ForceField {
  ⋮
  write("Data Dihedrals By Type") {
    @dihedral:HOCH $atom:HO $atom:O $atom:C $atom:HC
  }
  ⋮
}
```

Creating a Force Field

Force-field settings go here too

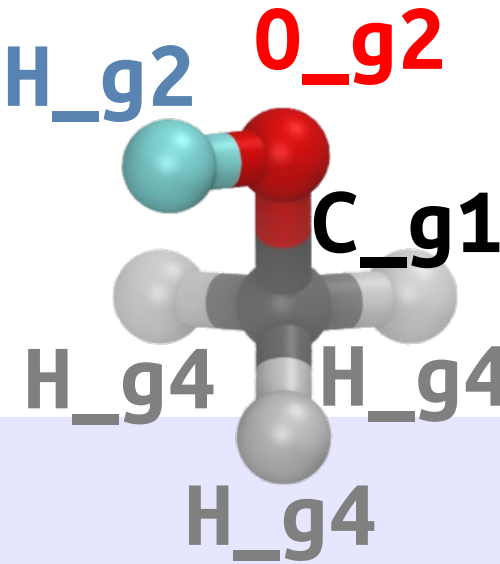


```
ForceField {
  ⋮
  write_once("In Init") {
    units real
    atom_style full
    pair_style hybrid lj/charmm/coul/long 11.0 12.0
    bond_style hybrid harmonic
    angle_style hybrid harmonic
    dihedral_style hybrid harmonic
    kspace_style pppm/cg 1.0e-5
    pair_modify mix arithmetic
    special_bonds amber
  }
}
```

Force Field Details

"equivalence groups"

Atoms belong to different groups (eg. g1,g2,g3,g4) used for force field lookup

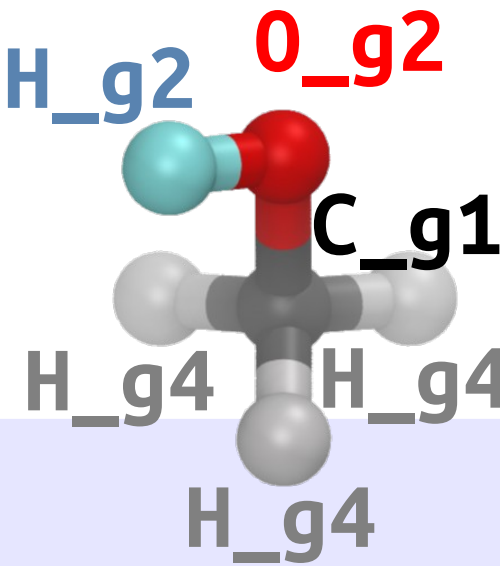



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:*_g2 $atom:*_g1 $atom:*_g4
    @angle:C_O_H   $atom:*_g1 $atom:*_g2 $atom:*_g3
    @angle:H_C_H   $atom:*_g4 $atom:*_g1 $atom:*_g4
  }
  ⋮
}
```

(Note: OPLSAA and COMPASS were implemented this way in moltemplate)

Force Field Details

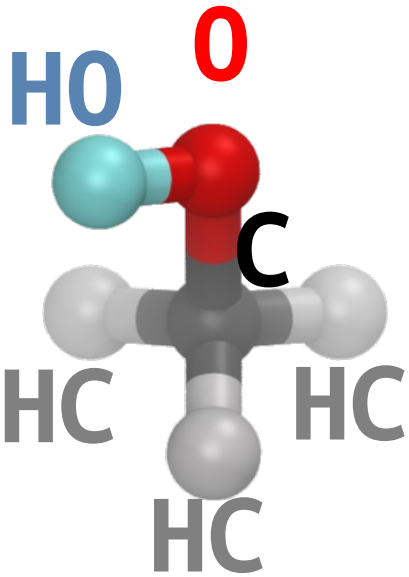
"equivalence groups"
= longer atom type names
+ wild cards ("")*




```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:*_g2 $atom:*_g1 $atom:*_g4
    @angle:C_O_H   $atom:*_g1 $atom:*_g2 $atom:*_g3
    @angle:H_C_H   $atom:*_g4 $atom:*_g1 $atom:*_g4
  }
  ⋮
}
```

Optional: Equivalence groups

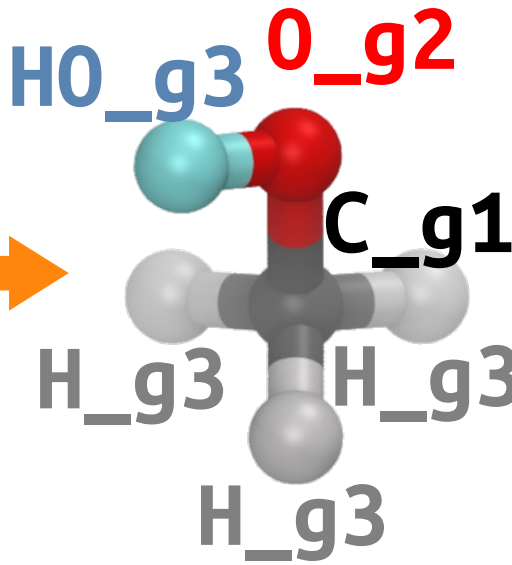
4 atom types
(HO,O,C,HC)




```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O  $atom:C  $atom:HC
    @angle:C_O_H   $atom:C  $atom:O  $atom:HO
    @angle:H_C_H   $atom:HC $atom:C  $atom:HC
  }
  ⋮
```

Optional: Equivalence groups

only 3 group types
(g1,g2,g3)

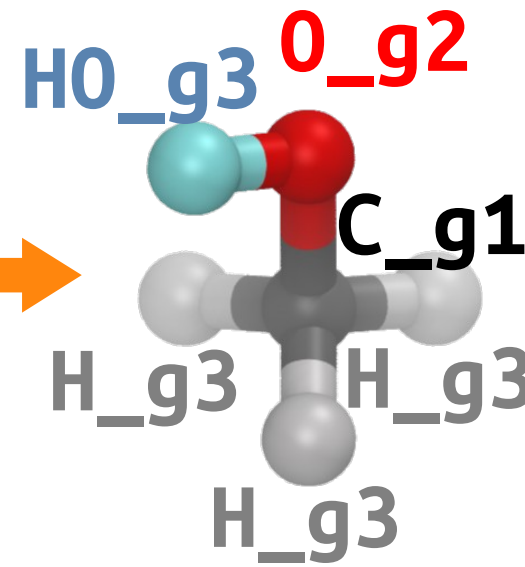



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O_g2 $atom:C_g1 $atom:H_g3
    @angle:C_O_H   $atom:C_g1 $atom:O_g2 $atom:HO_g3
    @angle:H_C_H   $atom:H_g3 $atom:C_g1 $atom:H_g3
  }
  ⋮
}
```

Optional: Equivalence groups

only 3 group types (g1,g2,g3)

Atom type names are longer (HO_g3, O_g2, C_g1, H_g3)

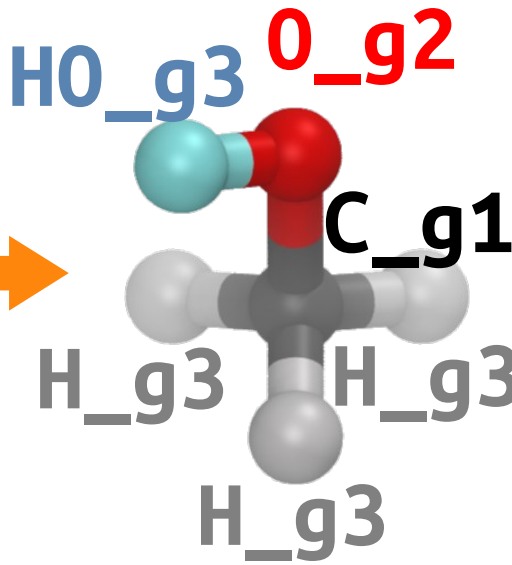



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:O_g2 $atom:C_g1 $atom:H_g3
    @angle:C_O_H   $atom:C_g1 $atom:O_g2 $atom:HO_g3
    @angle:H_C_H   $atom:H_g3 $atom:C_g1 $atom:H_g3
  }
  ⋮
}
```

Optional: Equivalence groups

only 3 group types (g1,g2,g3)

Atom type names are longer (HO_g3, O_g2, C_g1, H_g3)

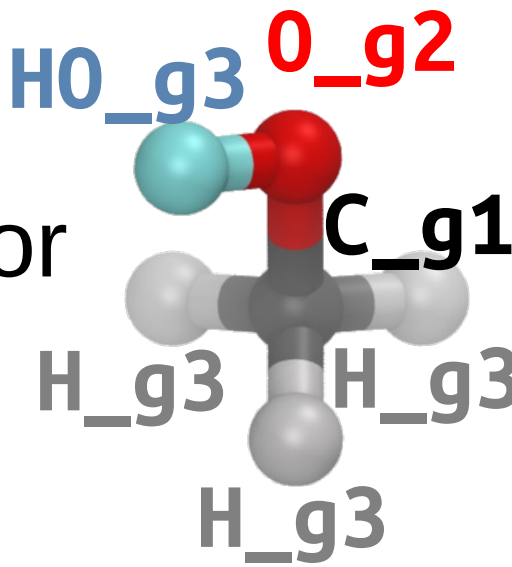



```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:*_g2 $atom:*_g1 $atom:*_g3
    @angle:C_O_H   $atom:*_g1 $atom:*_g2 $atom:*_g3
    @angle:H_C_H   $atom:*_g3 $atom:*_g1 $atom:*_g3
  }
  ⋮
}
```

*Moltemplate expects wildcards *

Optional: Equivalence groups

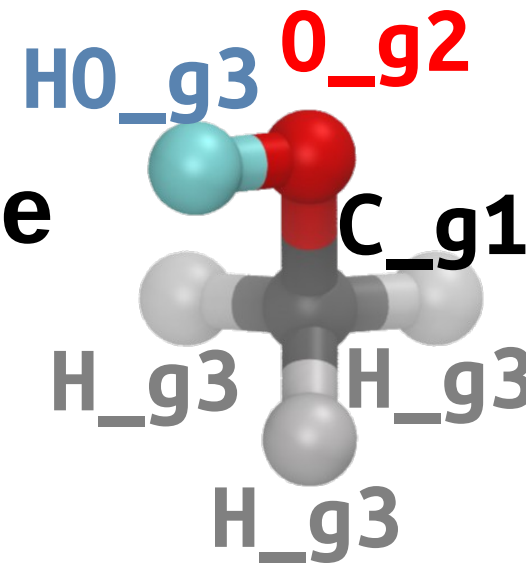
- Put atoms into to a small number of “groups” (eg. g1,g2,g3)
- The group-type (g1,g2,g3) is used for force-field lookup.
- *Usually* this reduces the number of lookup rules. *(not true in this example)*




```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:*_g2 $atom:*_g1 $atom:*_g3
    @angle:C_O_H   $atom:*_g1 $atom:*_g2 $atom:*_g3
    @angle:H_C_H   $atom:*_g3 $atom:*_g1 $atom:*_g3
  }
  ⋮
}
```

Optional: Equivalence groups

- **The OPLSAA and COMPASS force fields use equivalence groups** *(a.k.a. "equivalence tables")*




```
ForceField {
  ⋮
  write("Data Angles By Type") {
    @angle:O_C_H   $atom:*_g2 $atom:*_g1 $atom:*_g3
    @angle:C_O_H   $atom:*_g1 $atom:*_g2 $atom:*_g3
    @angle:H_C_H   $atom:*_g3 $atom:*_g1 $atom:*_g3
  }
  ⋮
}
```

Creating a Force Field

- For more details, examples of two simple force-field files (“forcefield.lt” and “oplsaa_simple.lt”) can be found here:

  http://moltemplate.org/examples/force_fields/

  (...along with usage examples)

- Or contact me at:

jewett.aij@gmail.com

Force Field support *(2019)*

- OPLSAA *(2009 Tinker version)*
- LOPLSAA *(2015)*
- AMBER *(GAFF & GAFF2)*
- COMPASS *(published only)*
- TraPPE *(1998)*
- MARTINI? *(untested)*
- SDK? *(untested)*

jewett.aij@gmail.com

How to get molecule files in moltemplate format

- Create them by hand

How to get molecule files in moltemplate format

- Create them by hand (edit an example from the moltemplate github page)

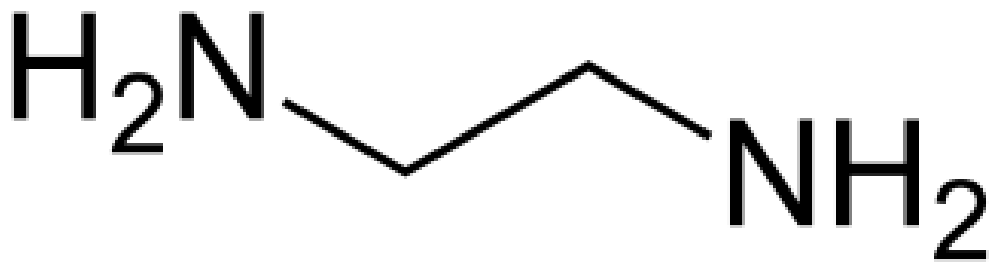
How to get molecule files in moltemplate format

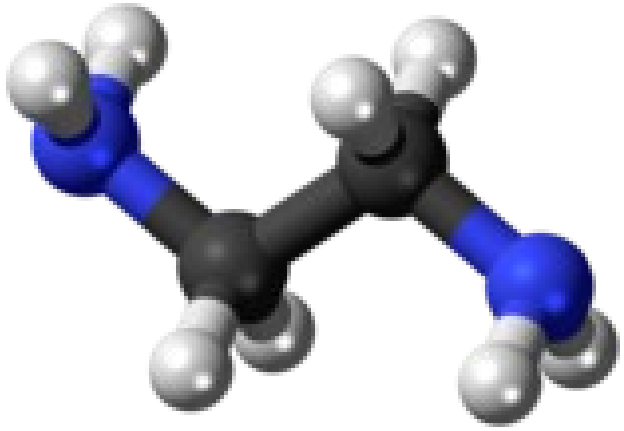
- Create them by hand (edit an example from the moltemplate github page)
- Use ***ltemplify.py*** to import LAMMPS data files created by 3rd-party tools (eg. **LibParGen**, **topotools**, vipster, msi2lmp, amber2lmp, ch2lmp, etc…)

How to get molecule files in moltemplate format

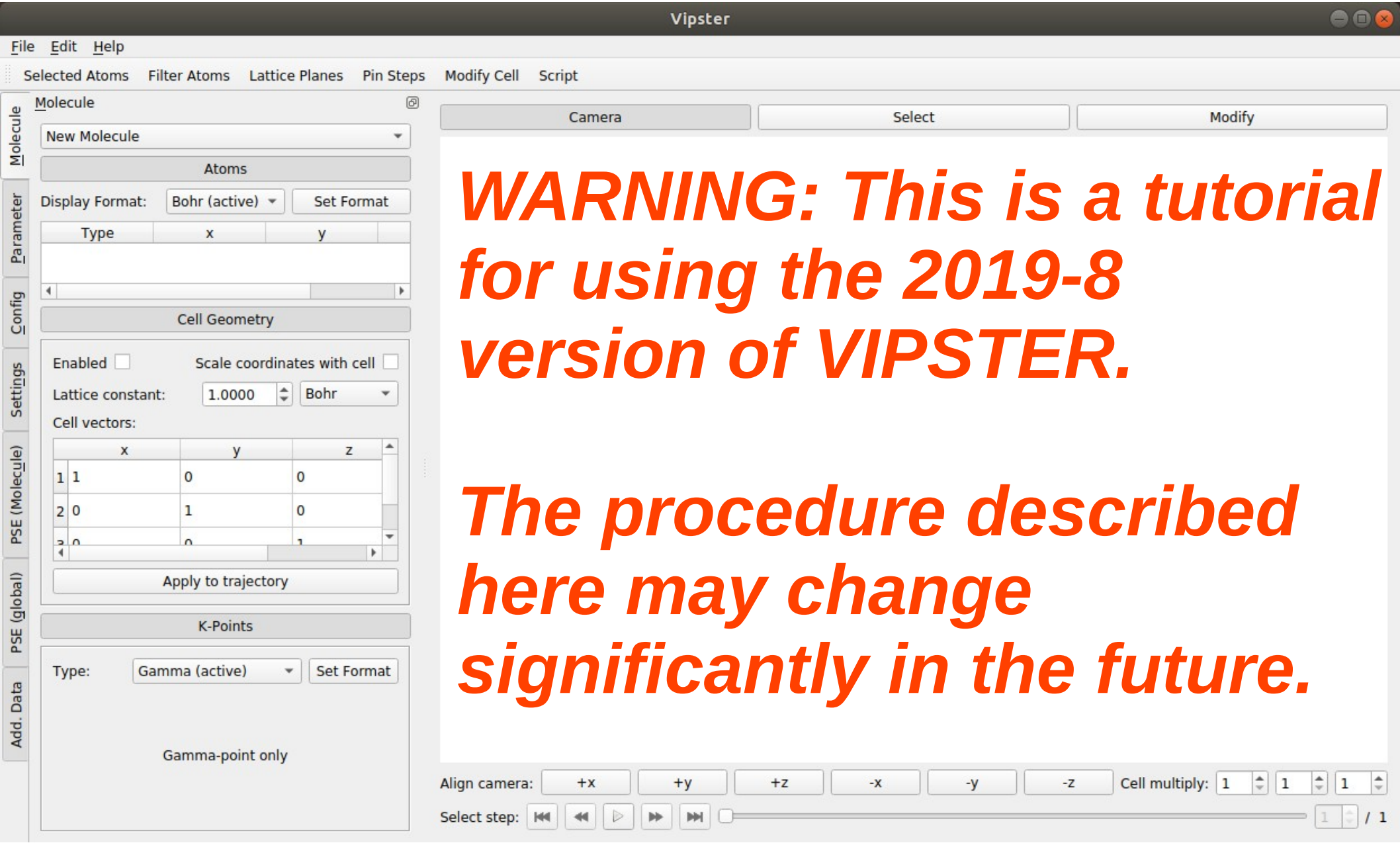
- Create them by hand (edit an example from the moltemplate github page)
- Use ***ltemplify.py*** to import LAMMPS data files created by 3rd-party tools (eg. **LibParGen**, **topotools**, vipster, msi2lmp, amber2lmp, ch2lmp, etc…)
- Download them from **ATB** (GROMOS FF) https://atb.uq.edu.au/

Example: Ethylenediamine





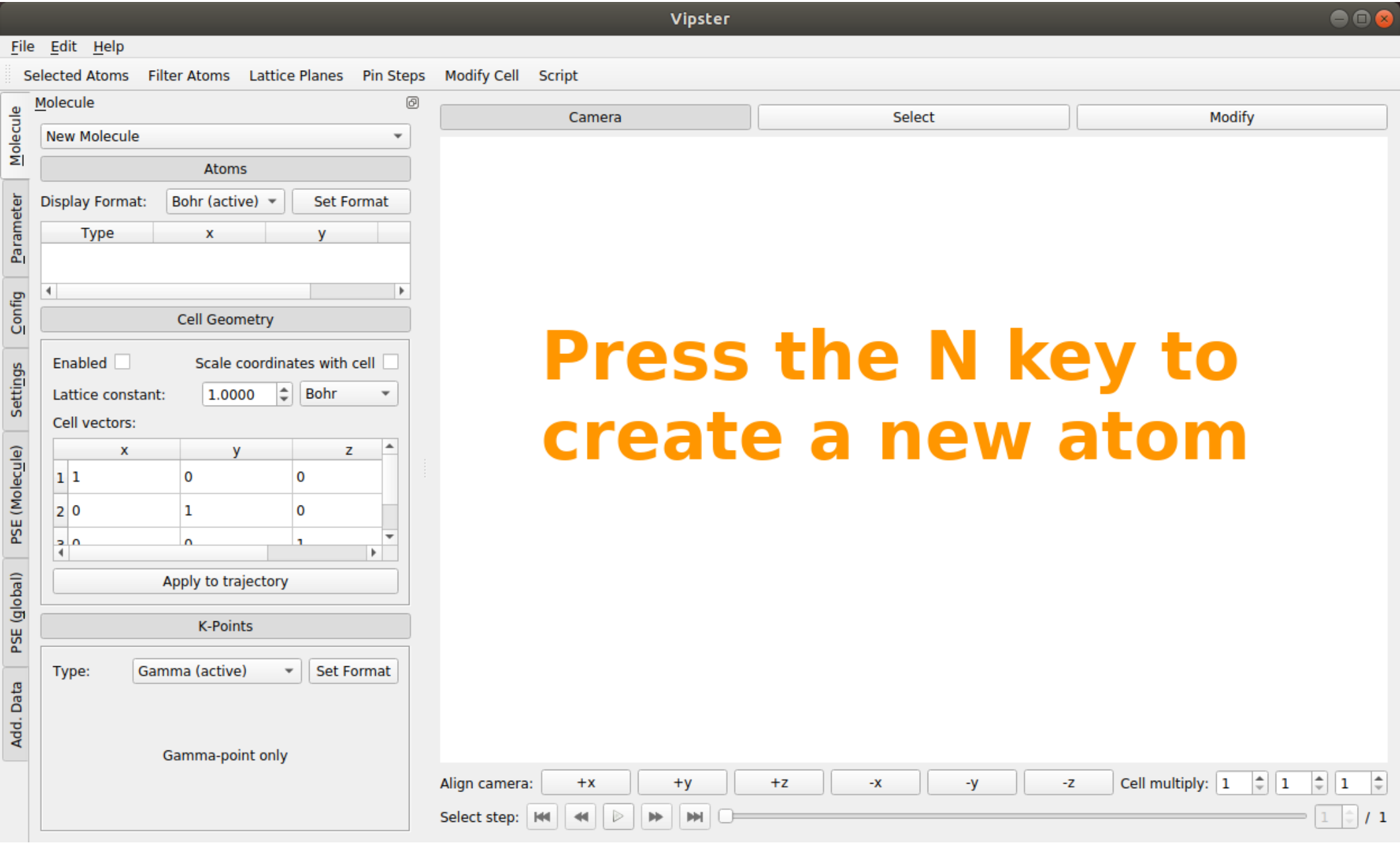
using vipster



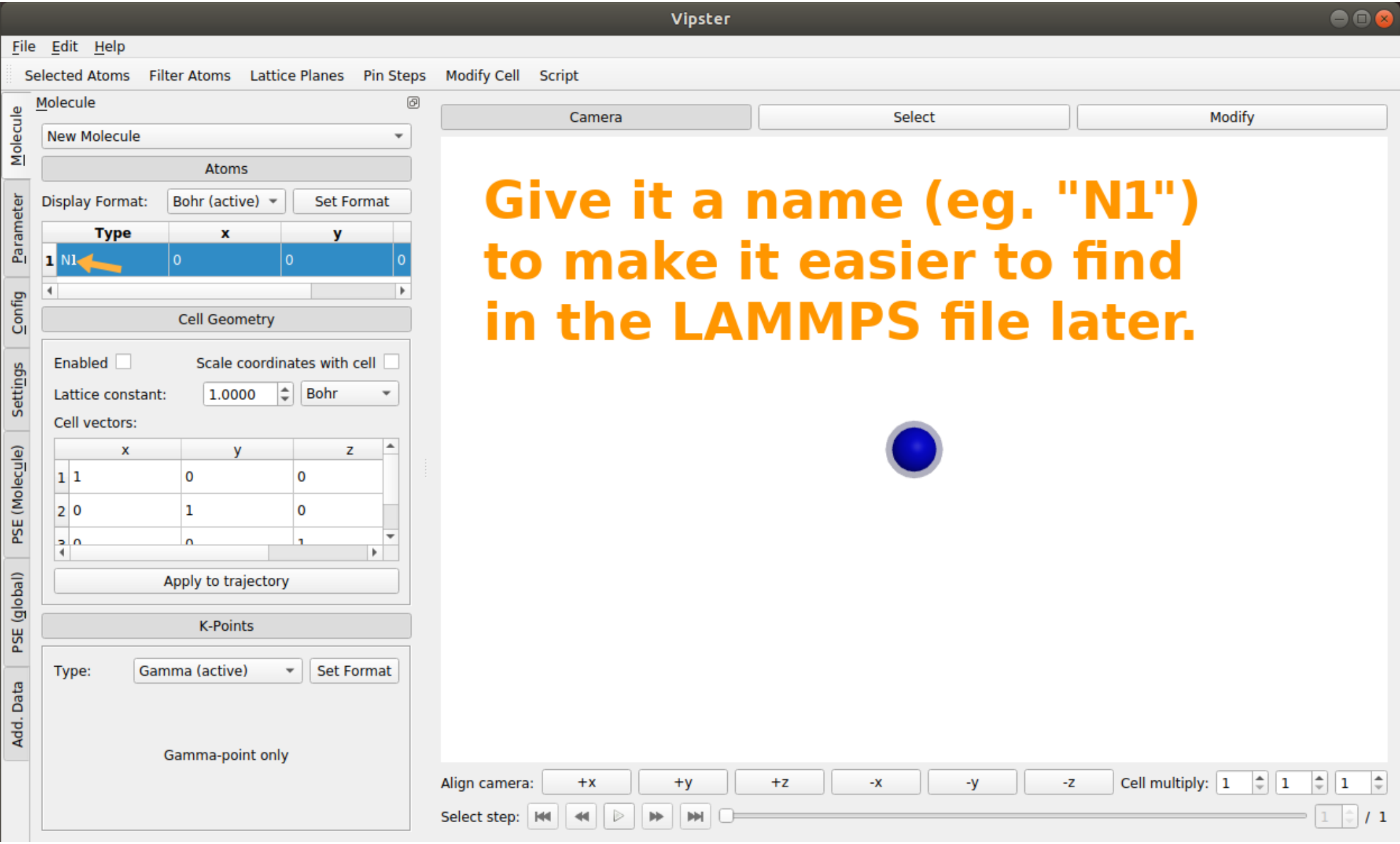
WARNING: This is a tutorial for using the 2019-8 version of VIPSTER.

The procedure described here may change significantly in the future.

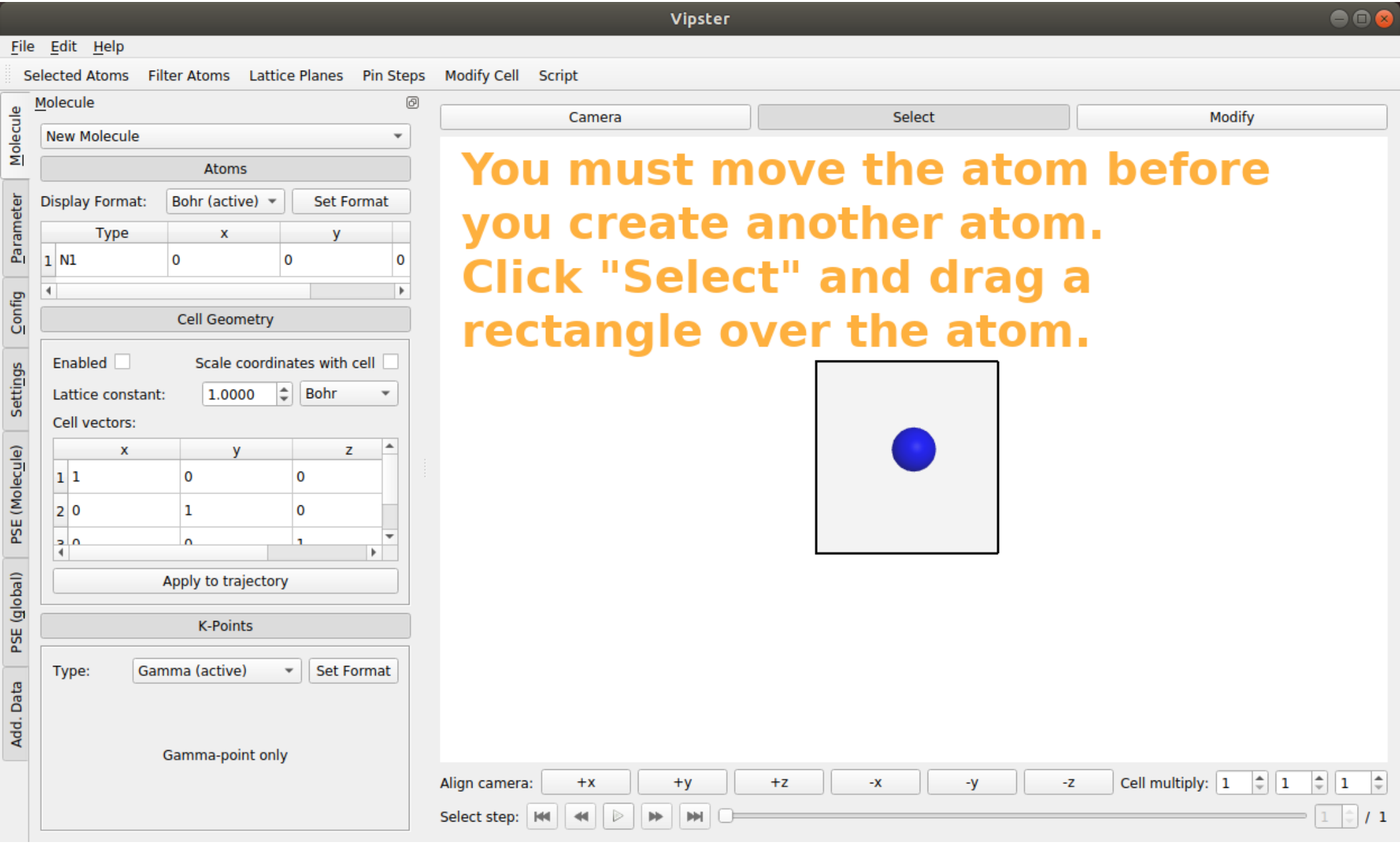
using vipster



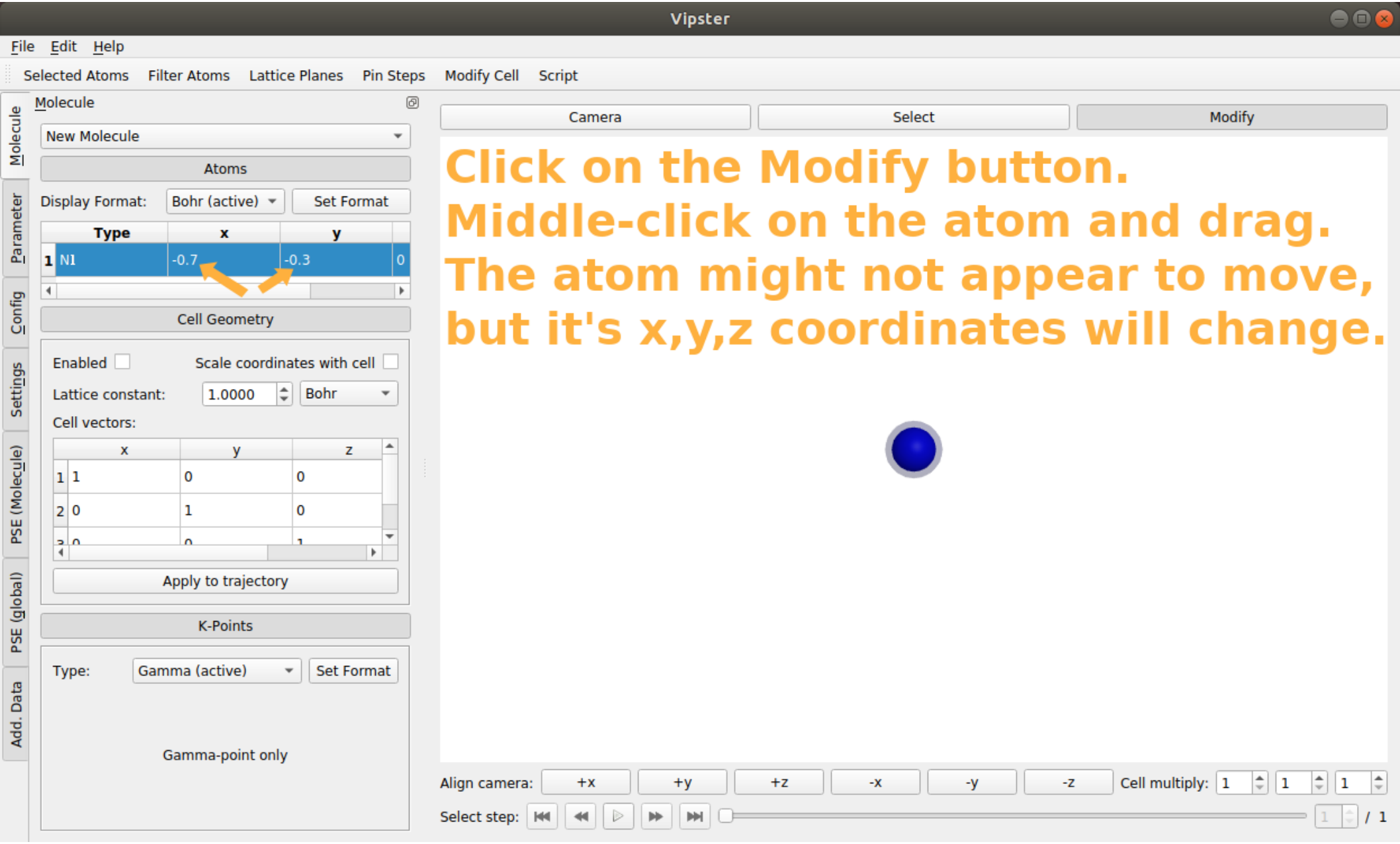
using vipster



using vipster



using vipster



using vipster

Vipster

File Edit Help

Selected Atoms | Filter Atoms | Lattice Planes | Pin Steps | Modify Cell | Script

Molecule | Parameter | Config | Settings | PSE (Molecule) | PSE (global) | Add. Data

Molecule

New Molecule

Atoms

Display Format: Bohr (active) Set Format

	Type	x	y
2	C1	0.869999	0.49

Cell Geometry

Enabled Scale coordinates with cell

Lattice constant: 1.0000 Bohr

Cell vectors:

	x	y	z
1	1	0	0
2	0	1	0
3	0	0	1

Apply to trajectory

K-Points

Type: Gamma (active) Set Format

Gamma-point only

Camera | Select | Modify

Repeat the process.
Add new atoms.
Change their type name.
Drag them to a new position.

Note that bonds will be generated between nearby atoms automatically. You can control atom sizes and distances by atom type, by clicking on the "PSE (global)" tab (left side).

Align camera: +x +y +z -x -y -z Cell multiply: 1 1 1

Select step: 1 / 1

using vipster

Vipster

File Edit Help

Selected Atoms | Filter Atoms | Lattice Planes | Pin Steps | Modify Cell | Script

Molecule | Parameter | Config | Settings | PSE (Molecule) | PSE (global) | Add. Data

Molecule

New Molecule

Atoms

Display Format: Bohr (active) Set Format

	Type	x	y
12	HN22	4.26	0.58

Cell Geometry

Enabled ☐ Scale coordinates with cell ☐

Lattice constant: 1.0000 Bohr

Cell vectors:

	x	y	z
1	1	0	0
2	0	1	0
3	0	0	1

Apply to trajectory

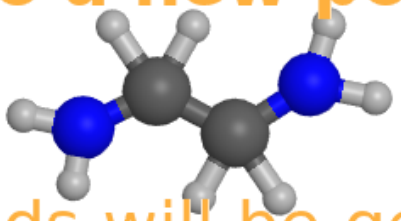
K-Points

Type: Gamma (active) Set Format

Gamma-point only

Camera | Select | Modify

Repeat the process.
Add new atoms.
Change their type name.
Drag them to a new position.

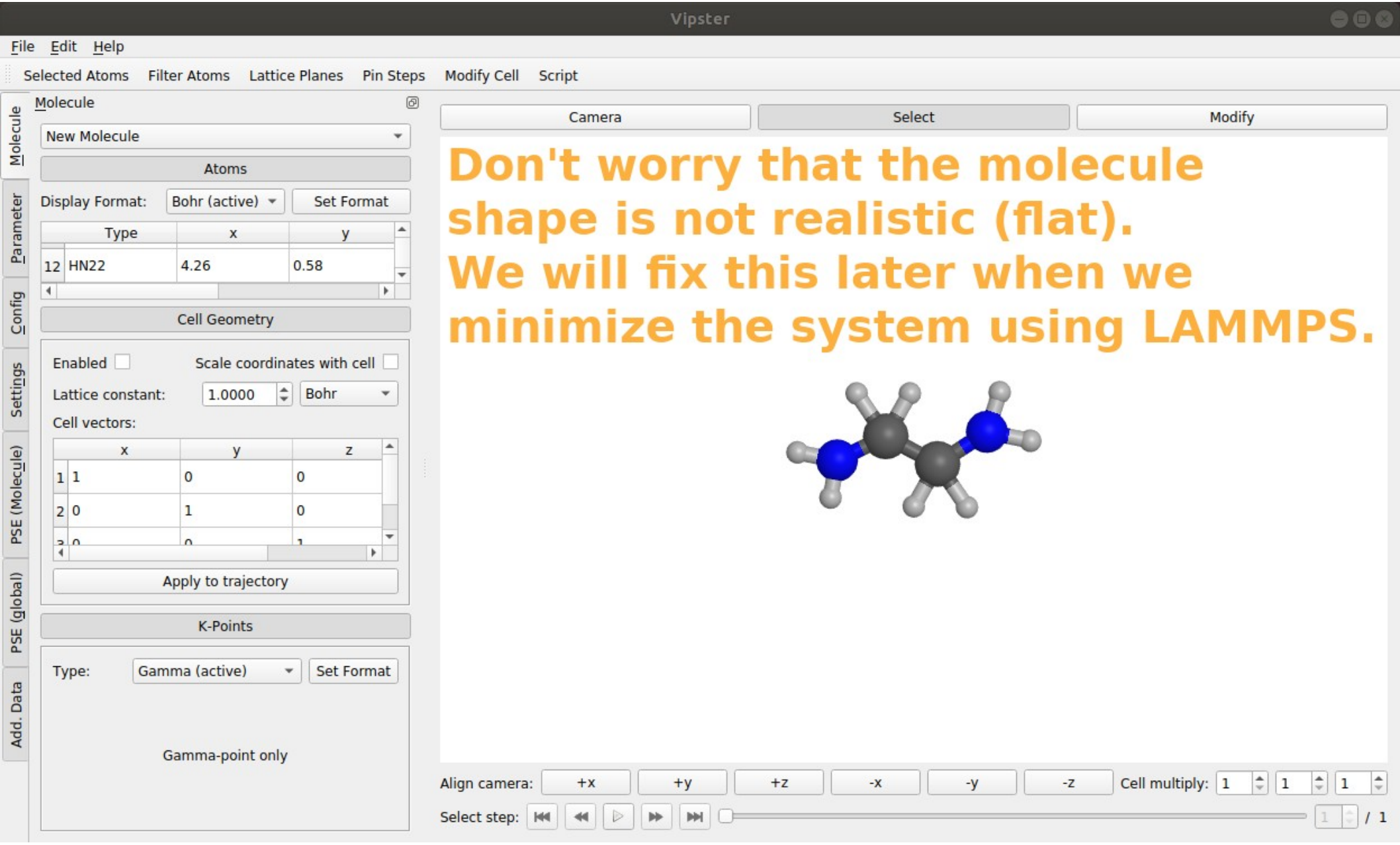


Note that bonds will be generated between nearby atoms automatically. You can control atom sizes and distances by atom type, by clicking on the "PSE (global)" tab (left side).

Align camera: +x +y +z -x -y -z Cell multiply: 1 1 1

Select step: 1 / 1

using vipster



using vipster

Vipster

File Edit Help

Selected Atoms | Filter Atoms | Lattice Planes | Pin Steps | Modify Cell | Script

Molecule | Parameter | Config | Settings | PSE (Molecule) | PSE (global) | Add. Data

Molecule

New Molecule

Atoms

Display Format: Bohr (active) Set Format

	Type	x	y
12	HN22	4.26	0.58

Cell Geometry

Enabled ☐ Scale coordinates with cell ☐

Lattice constant: 1.0000 Bohr

Cell vectors:

	x	y	z
1	1	0	0
2	0	1	0
3	0	0	1

Apply to trajectory

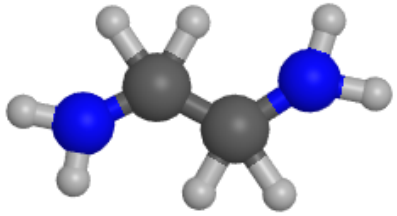
K-Points

Type: Gamma (active) Set Format

Gamma-point only

Camera | Select | Modify

Now, save the molecule (File->Save)
-Select "LAMMPS" format
-Select atom_style "full"
-Request "bonds"

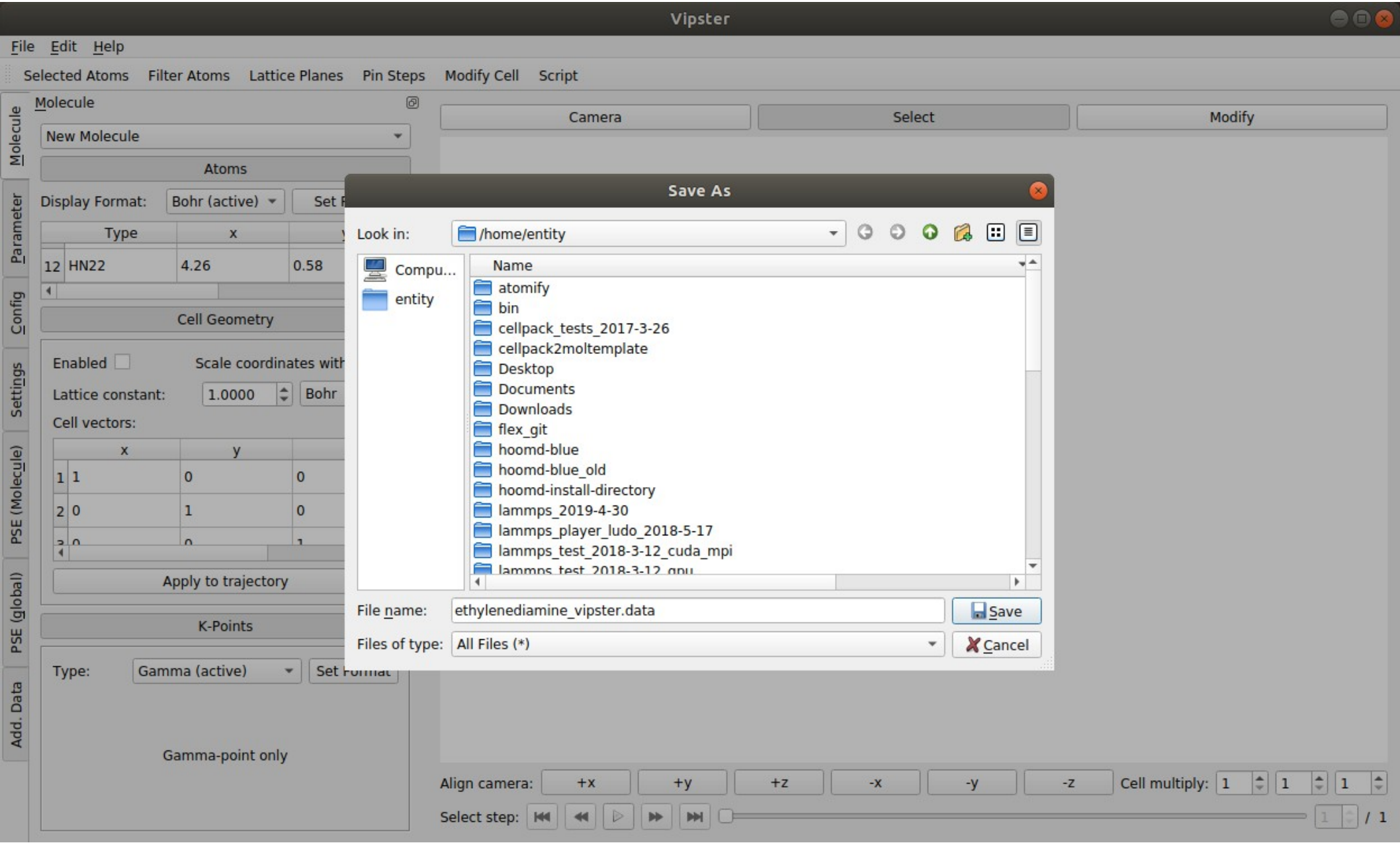


(Do not request angles, dihedrals or impropers because moltemplate will generate these automatically from the bond network and force field rules.)

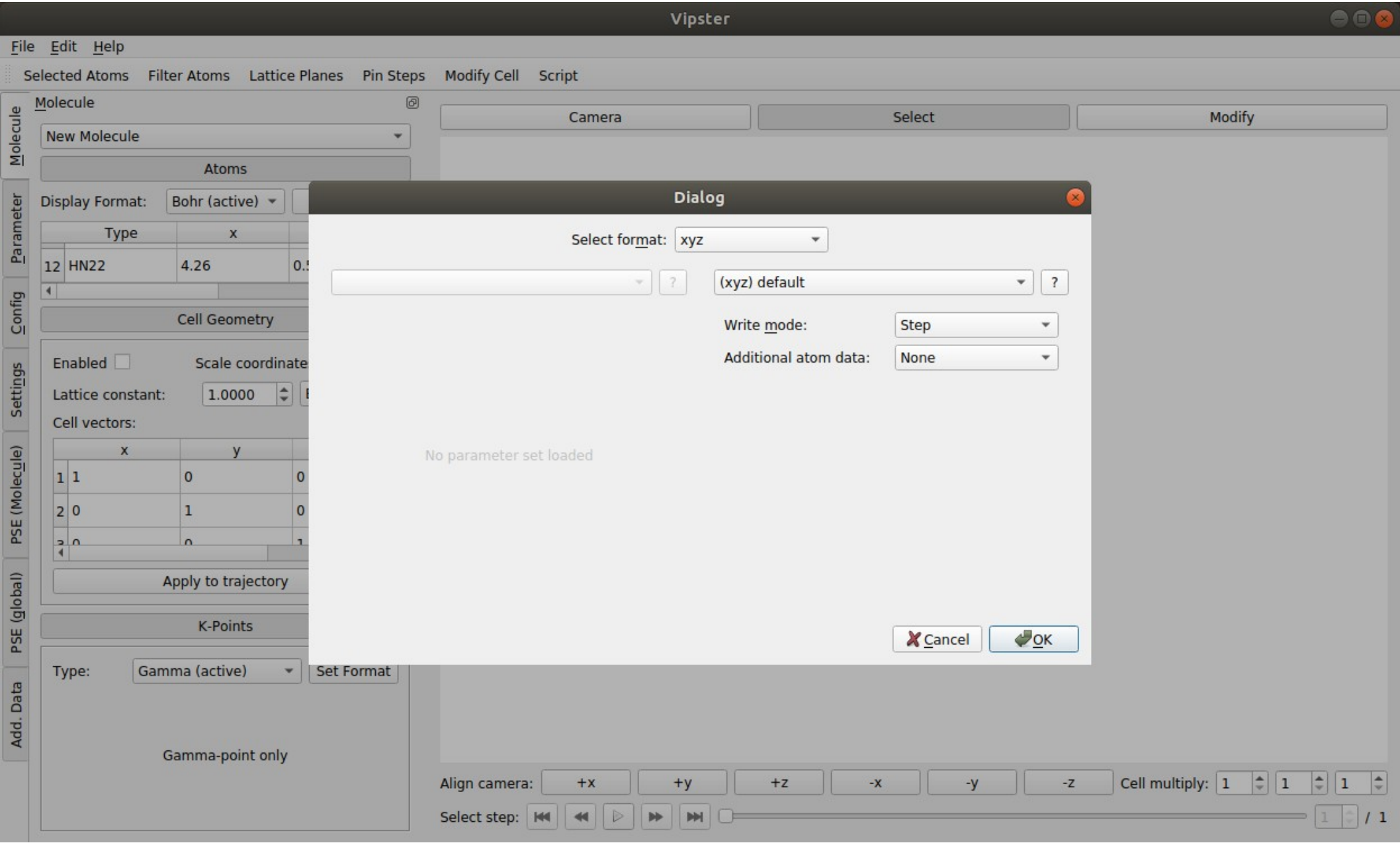
Align camera: +x +y +z -x -y -z Cell multiply: 1 1 1

Select step: 1 / 1

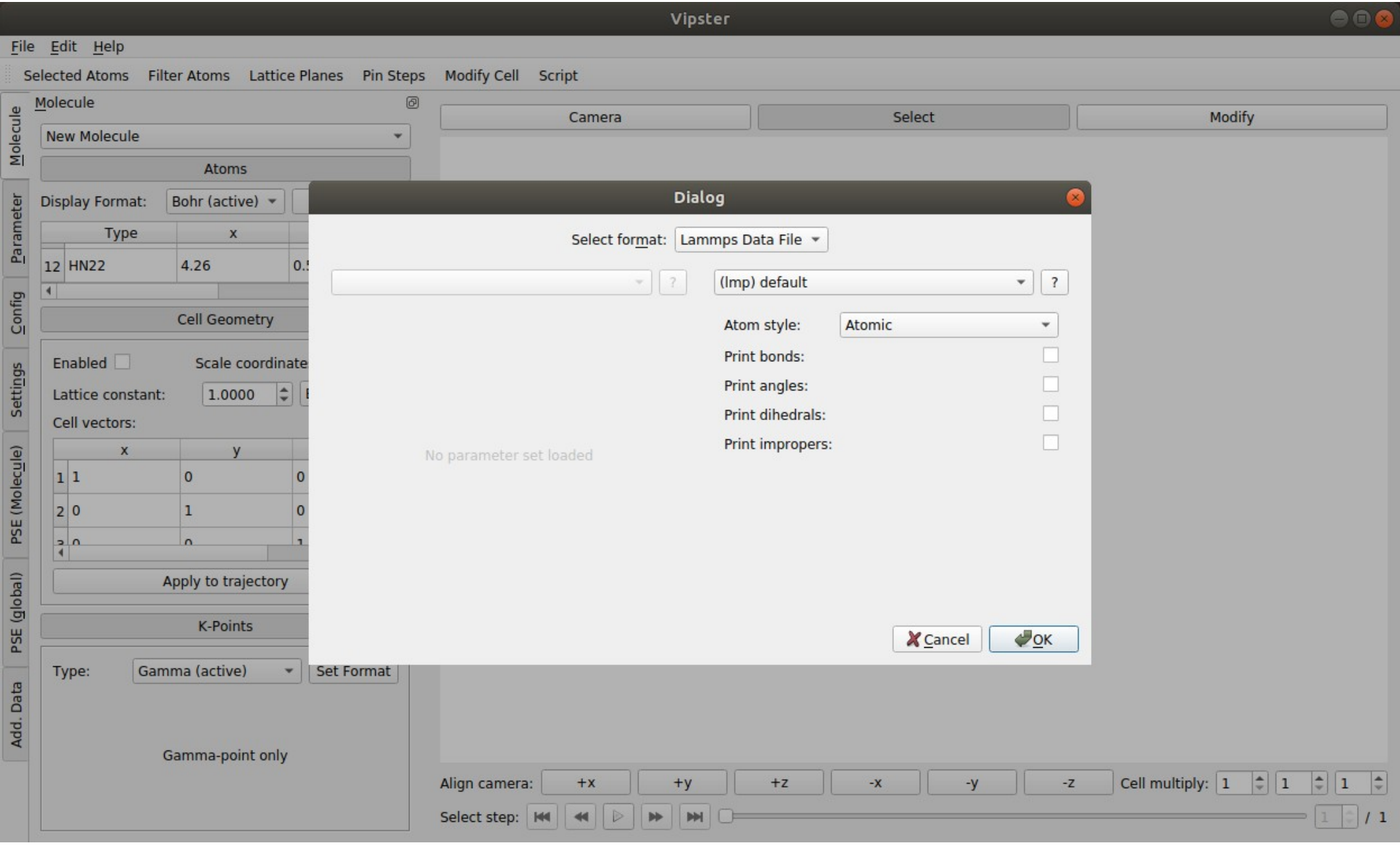
using vipster



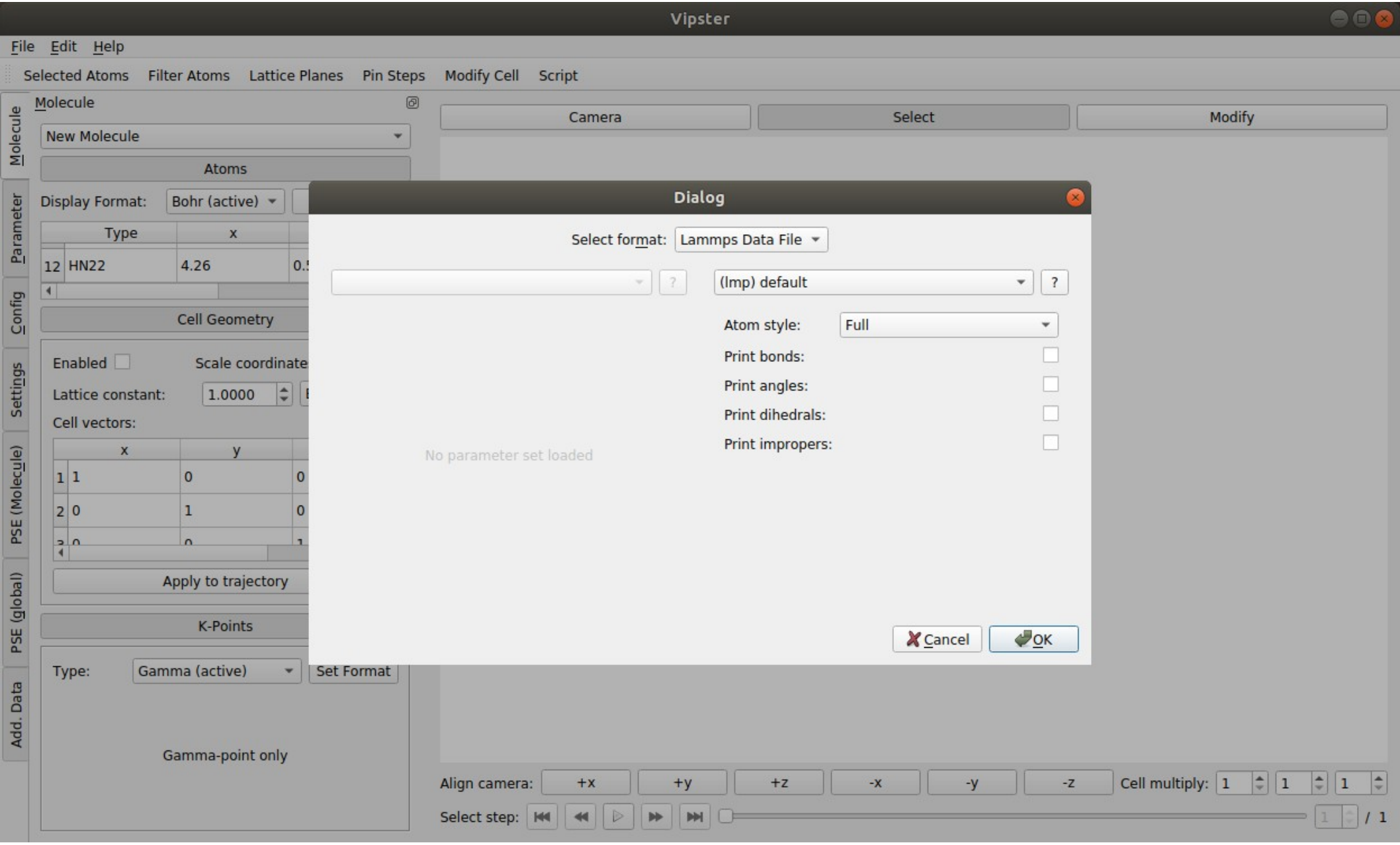
using vipster



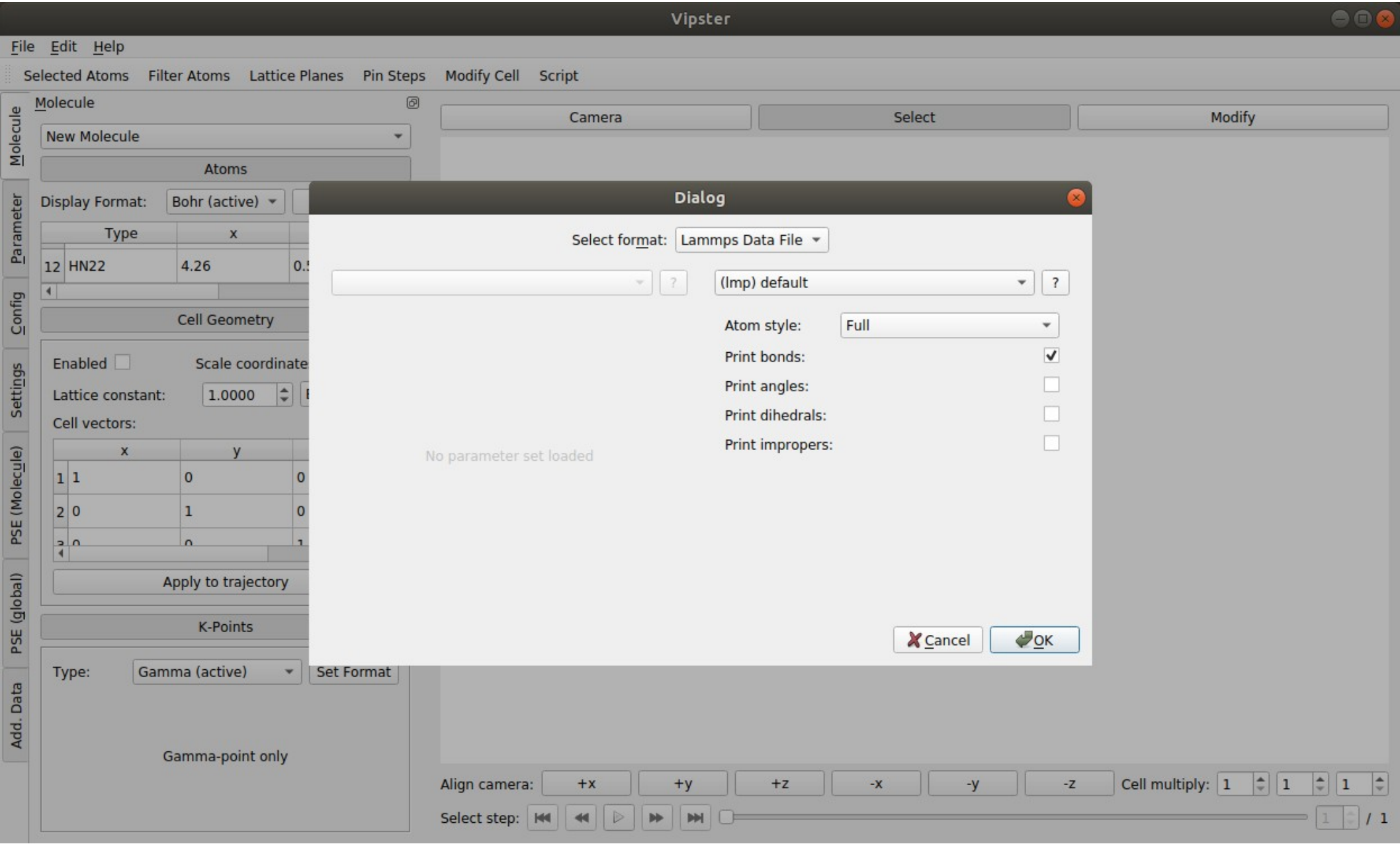
using vipster



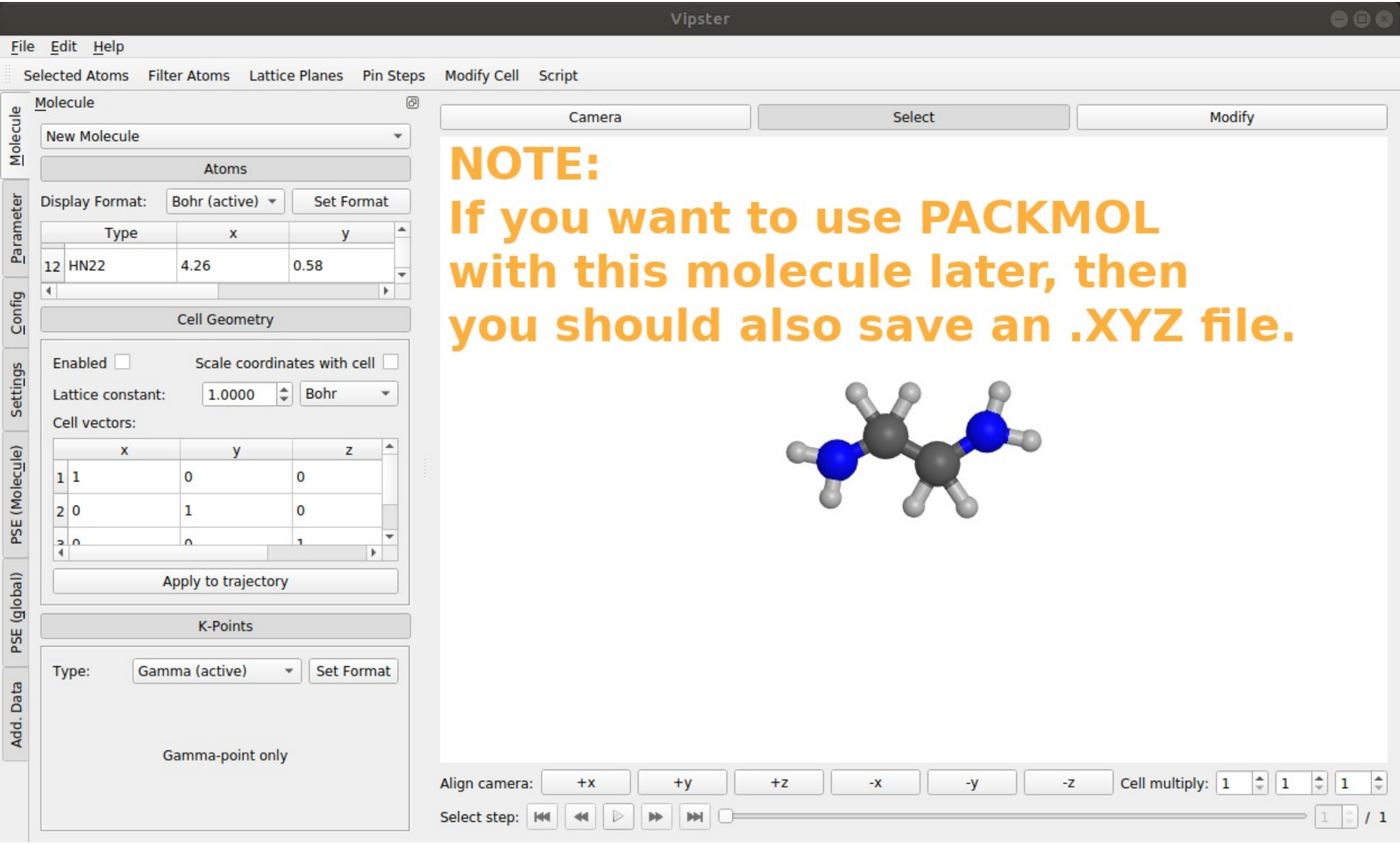
using vipster



using vipster



using vipster



using vipster

Vipster

File Edit Help

Selected Atoms | Filter Atoms | Lattice Planes | Pin Steps | Modify Cell | Script

Molecule | Parameter | Config | Settings | PSE (Molecule) | PSE (global) | Add. Data

Molecule

New Molecule

Atoms

Display Format: Bohr (active) Set Format

	Type	x	y
12	HN22	4.26	0.58

Cell Geometry

Enabled ☐ Scale coordinates with cell ☐

Lattice constant: 1.0000 Bohr

Cell vectors:

	x	y	z
1	1	0	0
2	0	1	0
3	0	0	1

Apply to trajectory

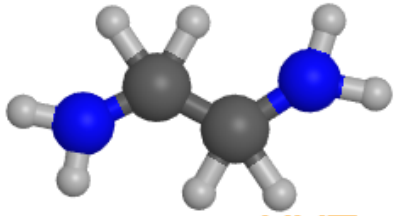
K-Points

Type: Gamma (active) Set Format

Gamma-point only

Camera | Select | Modify

NOTE:
If you want to use PACKMOL with this molecule later, then you should also save an .XYZ file.



Currently VIPSTER can create XYZ and LAMMPS data files for this molecule. **ltemplify.py** (which comes with moltemplate) can convert the LAMMPS data file to moltemplate format. PACKMOL can read XYZ files. You can use PACKMOL and moltemplate to replicate the molecule and build the final LAMMPS data file.

Align camera: +x +y +z -x -y -z Cell multiply: 1 1 1

Select step: 1 / 1

using ltemplify.py

- Included with moltemplate

using ltemplify.py

- Included with moltemplate
- Converts lammps data files *and* input scripts (if present) into moltemplate .LT files.

using ltemplify.py

- Included with moltemplate
- Converts lammps data files *and* input scripts (if present) into moltemplate .LT files.
- Runs in the terminal. Usage examples:

```
ltemplify.py -name MolName DATA.txt > FILE.lt

ltemplify.py -name MolName In1.txt In2.txt ... DATA.txt > FILE.lt
```

using ltemplify.py

- Included with moltemplate
- Converts lammps data files *and* input scripts (if present) into moltemplate .LT files.
- Runs in the terminal. Usage examples:

```
ltemplify.py -name MolName DATA.txt > FILE.lt

ltemplify.py -name MolName In1.txt In2.txt ... DATA.txt > FILE.lt
```

- Arguments:

  -name MoleculeName (optional)

  -LAMMPS input script(s) (optional)

  -one LAMMPS data file (last argument)

Run ltemplify.py

...on the file created by VIPSTER
("ethylenediamine_vipster.data")

Example:

```
$
$ ltemplify.py -name Ethylenediamine   \
  ethylenediamine_vipster.data > ethylenediamine.lt
$
```

(Note: This program will usually generate several warnings. These can be safely ignored.)

Edit the new .LT file to add the missing information

Atom-ID Mol-ID AtomType charge X Y Z

```
Ethylenediamin {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮ (bonds omitted)
```

Edit the new .LT file to add the missing information

AtomType charge

```
Ethylenediamine {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }
```

⋮

WRONG

Edit the new .LT file to add the missing information

```
Ethylenediamine {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮
```

Select a force field

```
import "oplsaa.lt"

Ethylenediamine {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮
```

Select a force field

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object

Ethylenediamine inherits OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮
```

Choose the atom types

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object

Ethylenediamine inherits OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮
```

Read the force-field file to find the appropriate atom types

```
OPLSAA {

  # Below we will use lammps "set" command to assign atom charges
  # by atom type.  http://lammps.sandia.gov/doc/set.html

  write_once("In Charges") {
    set type @atom:1 charge -0.22  # "Fluoride -CH2-F (UA)"
    set type @atom:2 charge 0.22  # "Fluoride -CH2-F (UA)"
    set type @atom:3 charge 0.55  # "Acetic Acid -COOH (UA)"
    set type @atom:4 charge -0.5  # "Acetic Acid >C=O (UA)"
    set type @atom:5 charge -0.58  # "Acetic Acid -OH (UA)"
    set type @atom:6 charge 0.08  # "Acetic Acid CH3- (UA)"
    set type @atom:7 charge 0.45  # "Acetic Acid -OH (UA)"
    set type @atom:8 charge 0.0  # "Methane CH4 (UA)"
    set type @atom:9 charge 0.0  # "Ethane CH3- (UA)"
    set type @atom:10 charge 0.0  # "N-Alkane CH3- (UA)"
    set type @atom:11 charge 0.0  # "Isobutane CH3- (UA)"
-:---  oplsaa.lt        1% L49   Git-master  (Fundamental)
```

Read the force-field file to find the appropriate atom types

```
set type @atom:729 charge -0.598  # "Nitrate Ion NO3-"
set type @atom:730 charge -0.9  # "Amine RNH2"
set type @atom:731 charge -0.78  # "Amine R2NH"
set type @atom:732 charge -0.63  # "Amine R3N"
set type @atom:733 charge 0.0  # "Amine CH3-NH2"
set type @atom:734 charge 0.02  # "Amine CH3-NHR"
set type @atom:735 charge 0.03  # "Amine CH3-NR2"
set type @atom:736 charge 0.06  # "Amine RCH2-NH2"
set type @atom:737 charge 0.08  # "Amine RCH2-NHR"
set type @atom:738 charge 0.09  # "Amine RCH2-NR2"
set type @atom:739 charge 0.36  # "Amine RNH2"
set type @atom:740 charge 0.38  # "Amine R2NH"
set type @atom:741 charge 0.06  # "Amine H-C-N"
set type @atom:742 charge 0.12  # "Amine R2CH-NH2"
set type @atom:743 charge 0.18  # "Amine R3C-NH2"
set type @atom:744 charge 0.14  # "Amine R2CH-NHR"
set type @atom:745 charge 0.15  # "Amine R2CH-NR2"
set type @atom:746 charge 0.18  # "Aniline C-NH2"
-:---  oplsaa.lt       8% L791  Git-master  (Fundamental)
```

Choose the atom types

```
set type @atom:729 charge -0.598  # "Nitrate Ion NO3-"
set type @atom:730 charge -0.9  # "Amine RNH2"
set type @atom:731 charge -0.78  # "Amine R2NH"
set type @atom:732 charge -0.63  # "Amine R3N"
set type @atom:733 charge 0.0  # "Amine CH3-NH2"
set type @atom:734 charge 0.02  # "Amine CH3-NHR"
set type @atom:735 charge 0.03  # "Amine CH3-NR2"
set type @atom:736 charge 0.06  # "Amine RCH2-NH2"
set type @atom:737 charge 0.08  # "Amine RCH2-NHR"
set type @atom:738 charge 0.09  # "Amine RCH2-NR2"
set type @atom:739 charge 0.36  # "Amine RNH2"
set type @atom:740 charge 0.38  # "Amine R2NH"
set type @atom:741 charge 0.06  # "Amine H-C-N"
set type @atom:742 charge 0.12  # "Amine R2CH-NH2"
set type @atom:743 charge 0.18  # "Amine R3C-NH2"
set type @atom:744 charge 0.14  # "Amine R2CH-NHR"
set type @atom:745 charge 0.15  # "Amine R2CH-NR2"
set type @atom:746 charge 0.18  # "Aniline C-NH2"
-:---  oplsaa.lt        8% L791  Git-master  (Fundamental)
```

Choose the atom types

```
set type @atom:729 charge -0.598  # "Nitrate Ion NO3-"
set type @atom:730 charge -0.9  # "Amine RNH2"
set type @atom:731 charge -0.78  # "Amine R2NH"
set type @atom:732 charge -0.63  # "Amine R3N"
set type @atom:733 charge 0.0  # "Amine CH3-NH2"
set type @atom:734 charge 0.02  # "Amine CH3-NHR"
set type @atom:735 charge 0.03  # "Amine CH3-NR2"
set type @atom:736 charge 0.06  # "Amine RCH2-NH2"
set type @atom:737 charge 0.08  # "Amine RCH2-NHR"
set type @atom:738 charge 0.09  # "Amine RCH2-NR2"
set type @atom:739 charge 0.36  # "Amine RNH2"
set type @atom:740 charge 0.38  # "Amine R2NH"
set type @atom:741 charge 0.06  # "Amine H-C-N"
set type @atom:742 charge 0.12  # "Amine R2CH-NH2"
set type @atom:743 charge 0.18  # "Amine R3C-NH2"
set type @atom:744 charge 0.14  # "Amine R2CH-NHR"
set type @atom:745 charge 0.15  # "Amine R2CH-NR2"
set type @atom:746 charge 0.18  # "Aniline C-NH2"
-:---  oplsaa.lt        8% L791  Git-master  (Fundamental)
```

type *charge*

Choose the atom types

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object
                                Type    charge
Ethylenediamine inherit OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:N1    0  -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:C1    0  -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:C2    0  0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:N2    0  1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:HN11  0  -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:HN12  0  -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:HC11  0  -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:HC12  0  0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:HC21  0  0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:HC22  0  1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:HN21  0  1.7092  1.169    0
    $atom:HN22 $mol:m @atom:HN22  0  2.2543  0.3069   0
  }

  ⋮
```

Check atom types *carefully*!!

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object

Ethylenediamine inherits OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:730 -0.9 -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:736 0.06 -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:736 0.06 0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:730 -0.9 1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:739 0.36 -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:739 0.36 -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:741 0.06 -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:741 0.06 0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:741 0.06 0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:741 0.06 1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:739 0.36 1.7092  1.169    0
    $atom:HN22 $mol:m @atom:739 0.36 2.2543  0.3069   0
  }

  ⋮
```

Check atom types *carefully*!!

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object

Ethylenediamine inherits OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:730 -0.9 -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:736 0.06 -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:736 0.06 0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:730 -0.9 1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:739 0.36 -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:739 0.36 -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:741 0.06 -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:741 0.06 0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:741 0.06 0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:741 0.06 1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:739 0.36 1.7092  1.169    0
    $atom:HN22 $mol:m @atom:739 0.36 2.2543  0.3069   0
  }
```

sum should equal the net charge of the molecule

Remove bond types from bond list

```
Ethylenediamine {
  ⋮
  write("Data Bonds") {
    $bond:b1  @bond:type1  $atom:N1 $atom:C1
    $bond:b2  @bond:type2  $atom:N1 $atom:HN11
    $bond:b3  @bond:type3  $atom:N1 $atom:HN12
    $bond:b4  @bond:type4  $atom:C1 $atom:C2
    $bond:b5  @bond:type5  $atom:C1 $atom:HC11
    $bond:b6  @bond:type6  $atom:C1 $atom:HC12
    $bond:b7  @bond:type7  $atom:C2 $atom:N2
    $bond:b8  @bond:type8  $atom:C2 $atom:HC21
    $bond:b9  @bond:type9  $atom:C2 $atom:HC22
    $bond:b10 @bond:type10 $atom:N2 $atom:HN21
    $bond:b11 @bond:type11 $atom:N2 $atom:HN22
  }
} # end of "Ethylenediamine" type definition
```

Moltemplate will determine bond types from force field rules.

```
Ethylenediamine {
  ⋮
  write("Data Bond List") {
    $bond:b1  $atom:N1 $atom:C1
    $bond:b2  $atom:N1 $atom:HN11
    $bond:b3  $atom:N1 $atom:HN12
    $bond:b4  $atom:C1 $atom:C2
    $bond:b5  $atom:C1 $atom:HC11
    $bond:b6  $atom:C1 $atom:HC12
    $bond:b7  $atom:C2 $atom:N2
    $bond:b8  $atom:C2 $atom:HC21
    $bond:b9  $atom:C2 $atom:HC22
    $bond:b10 $atom:N2 $atom:HN21
    $bond:b11 $atom:N2 $atom:HN22
  }
} # end of "Ethylenediamine" type definition
```

This .LT file is ready for use...

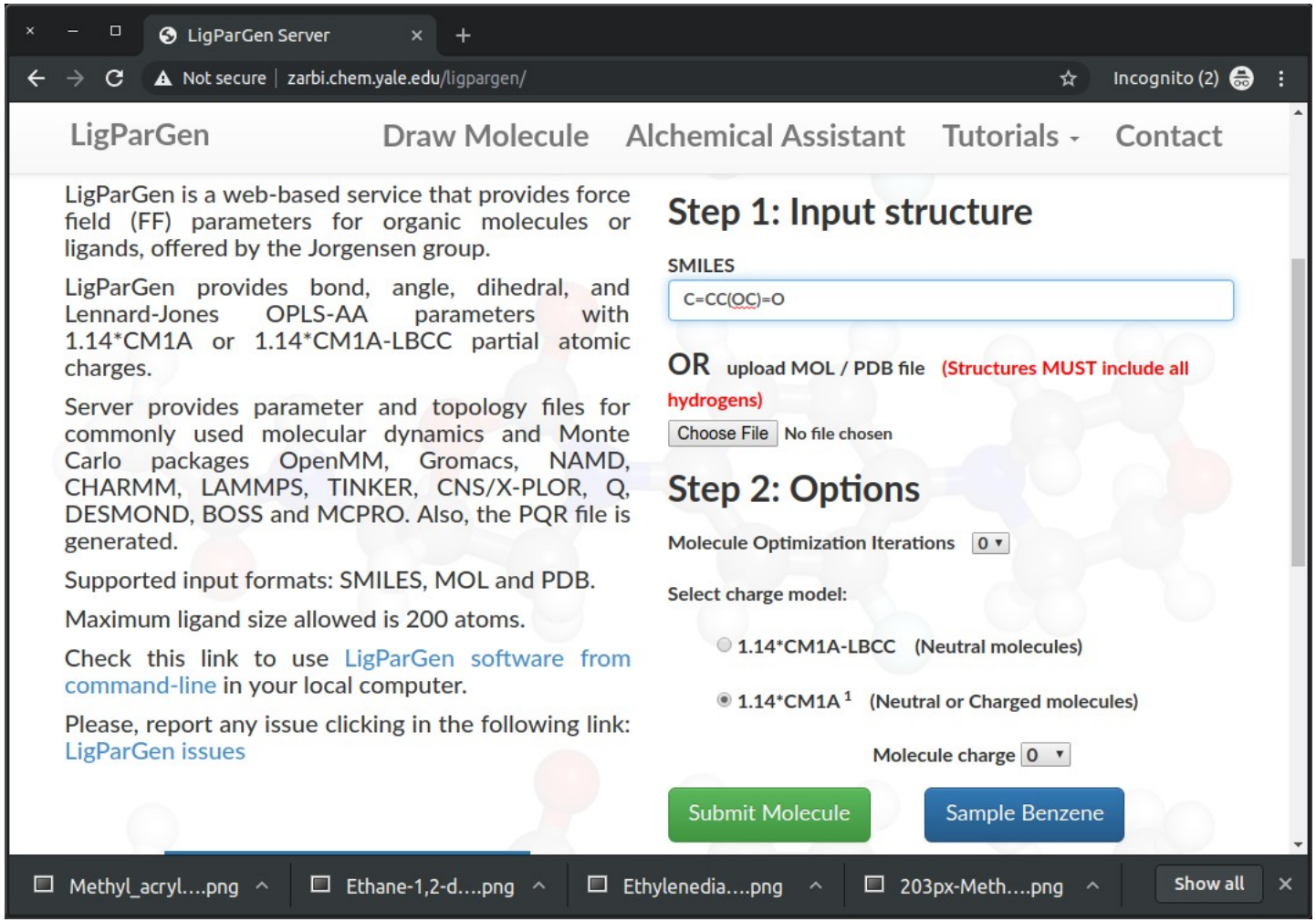
```
Ethylenediamine {
  ⋮
  write("Data Bond List") {
    $bond:id1 $atom:N1 $atom:C1
    $bond:id2 $atom:N1 $atom:HN11
    $bond:id3 $atom:N1 $atom:HN12
    $bond:id4 $atom:C1 $atom:C2
    $bond:id5 $atom:C1 $atom:HC11
    $bond:id6 $atom:C1 $atom:HC12
    $bond:id7 $atom:C2 $atom:N2
    $bond:id8 $atom:C2 $atom:HC21
    $bond:id9 $atom:C2 $atom:HC22
    $bond:id10 $atom:N2 $atom:HN21
    $bond:id11 $atom:N2 $atom:HN22
  }
} # end of "Ethylenediamine" type definition
```

This .LT file is ready for use...

```
import "oplsaa.lt" # ← defines the OPLSAA forcefield object

Ethylenediamine inherits OPLSAA {
  write("Data Atoms") {
    $atom:N1   $mol:m @atom:730 -0.9 -1.1854 -0.04233 0
    $atom:C1   $mol:m @atom:736 0.06 -0.31751 0.38630 0
    $atom:C2   $mol:m @atom:736 0.06 0.60326 -0.10584 0
    $atom:N2   $mol:m @atom:730 -0.9 1.4817  0.46568  0
    $atom:HN11 $mol:m @atom:739 0.36 -1.3018 -0.70381 0
    $atom:HN12 $mol:m @atom:739 0.36 -1.8892  0.1005 0
    $atom:HC11 $mol:m @atom:741 0.06 -0.83610 1.1483  0
    $atom:HC12 $mol:m @atom:741 0.06 0.11113 1.1483  0
    $atom:HC21 $mol:m @atom:741 0.06 0.17992 -0.85727 0
    $atom:HC22 $mol:m @atom:741 0.06 1.1271  -0.87314 0
    $atom:HN21 $mol:m @atom:739 0.36 1.7092  1.169    0
    $atom:HN22 $mol:m @atom:739 0.36 2.2543  0.3069   0
  }

  ⋮
```

LibParGen → moltemplate

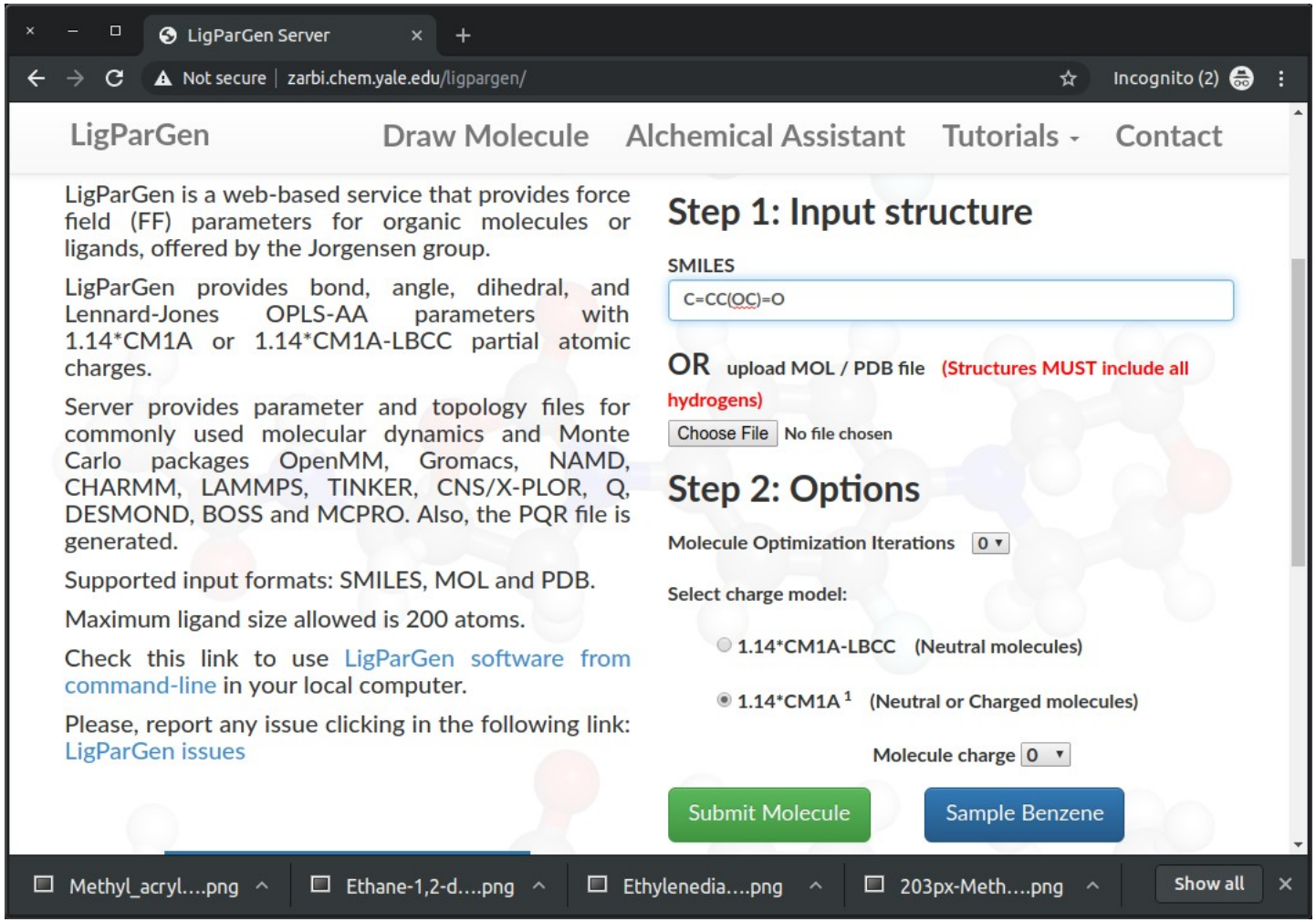
http://zarbi.chem.yale.edu/libpargen



LibParGen → moltemplate

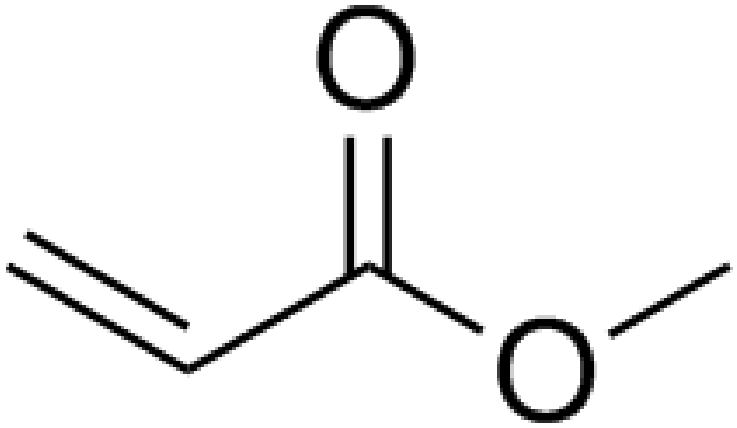
http://zarbi.chem.yale.edu/libpargen

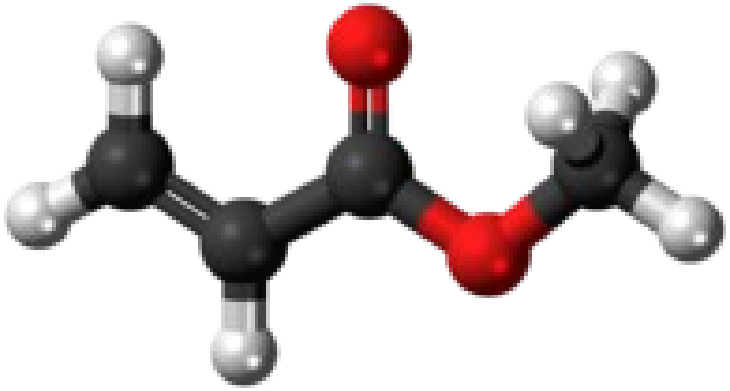
Note: OPLSAA force field only



LibParGen → moltemplate

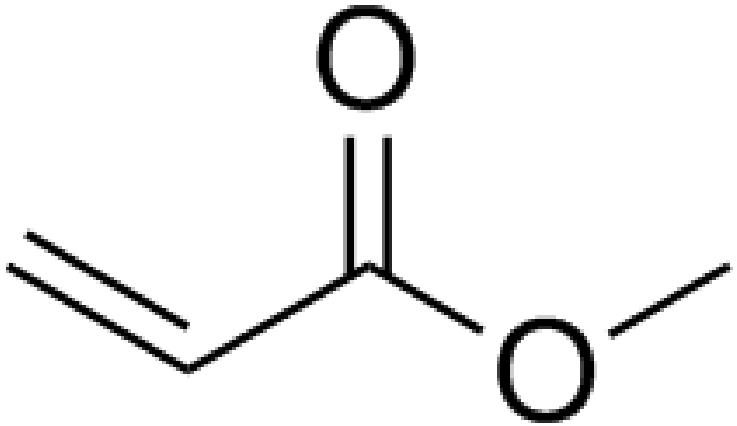
Example: methyl acrylate

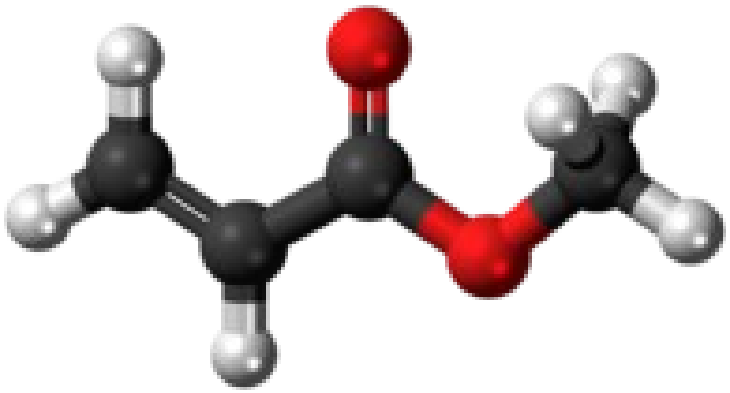




LibParGen → moltemplate

Example: methyl acrylate





SMILES string: ***C=CC(OC)=O***

LibParGen → moltemplate

LigParGen Server

Not secure | zarbi.chem.yale.edu/ligpargen/

Incognito (2)

LigParGen | Draw Molecule | Alchemical Assistant | Tutorials | Contact

LigParGen is a web-based service that provides force field (FF) parameters for organic molecules or ligands, offered by the Jorgensen group.

LigParGen provides bond, angle, dihedral, and Lennard-Jones OPLS-AA parameters with 1.14*CM1A or 1.14*CM1A-LBCC partial atomic charges.

Server provides parameter and topology files for commonly used molecular dynamics and Monte Carlo packages OpenMM, Gromacs, NAMD, CHARMM, LAMMPS, TINKER, CNS/X-PLOR, Q, DESMOND, BOSS and MCPRO. Also, the PQR file is generated.

Supported input formats: SMILES, MOL and PDB.

Maximum ligand size allowed is 200 atoms.

Check this link to use LigParGen software from command-line in your local computer.

Please, report any issue clicking in the following link: LigParGen issues

Step 1: Input structure

SMILES

C=CC(OC)=O

OR upload MOL / PDB file (Structures MUST include all hydrogens)

Choose File No file chosen

Step 2: Options

Molecule Optimization Iterations 0

Select charge model:

- 1.14*CM1A-LBCC (Neutral molecules)
- 1.14*CM1A [1] (Neutral or Charged molecules)

Molecule charge 0

Submit Molecule | Sample Benzene

Methyl_acryl....png | Ethane-1,2-d....png | Ethylenedia....png | 203px-Meth....png | Show all

LibParGen → moltemplate

LigParGen Server

Not secure | zarbi.chem.yale.edu/ligpargen/

LigParGen | Draw Molecule | Alchemical Assistant | Tutorials | Contact

LigParGen is a web-based service that provides force field (FF) parameters for organic molecules or ligands, offered by the Jorgensen group.

LigParGen provides bond, angle, dihedral, and Lennard-Jones OPLS-AA parameters with 1.14*CM1A or 1.14*CM1A-LBCC partial atomic charges.

Server provides parameter and topology files for commonly used molecular dynamics and Monte Carlo packages OpenMM, Gromacs, NAMD, CHARMM, LAMMPS, TINKER, CNS/X-PLOR, Q, DESMOND, BOSS and MCPRO. Also, the PQR file is generated.

Supported input formats: SMILES, MOL and PDB.

Maximum ligand size allowed is 200 atoms.

Check this link to use LigParGen software from command-line in your local computer.

Please, report any issue clicking in the following link: LigParGen issues

Step 1: Input structure

SMILES

C=CC(OC)=O

OR upload MOL / PDB file (Structures MUST include all hydrogens)

Choose File No file chosen

Step 2: Options

Molecule Optimization Iterations 0

Select charge model:

1.14*CM1A-LBCC (Neutral molecules)

1.14*CM1A [1] (Neutral or Charged molecules)

Molecule charge 0

Submit Molecule | Sample Benzene

Methyl_acryl....png | Ethane-1,2-d....png | Ethylenedia....png | 203px-Meth....png | Show all

LibParGen → moltemplate

LigParGen Server

Not secure | zarbi.chem.yale.edu/cgi-bin/results_lpg.py

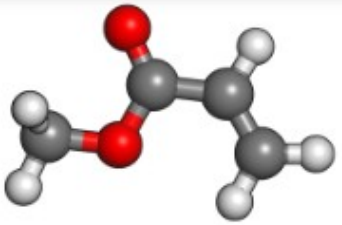
Incognito (2)

LigParGen

Draw Molecule Contact

Parameter and topology files successfully generated!!!

SAFARI USERS: Downloaded files will have the wrong name (results_lpg.py) and must be renamed apropriately



Ligand submitted

Try another molecule

Downloads

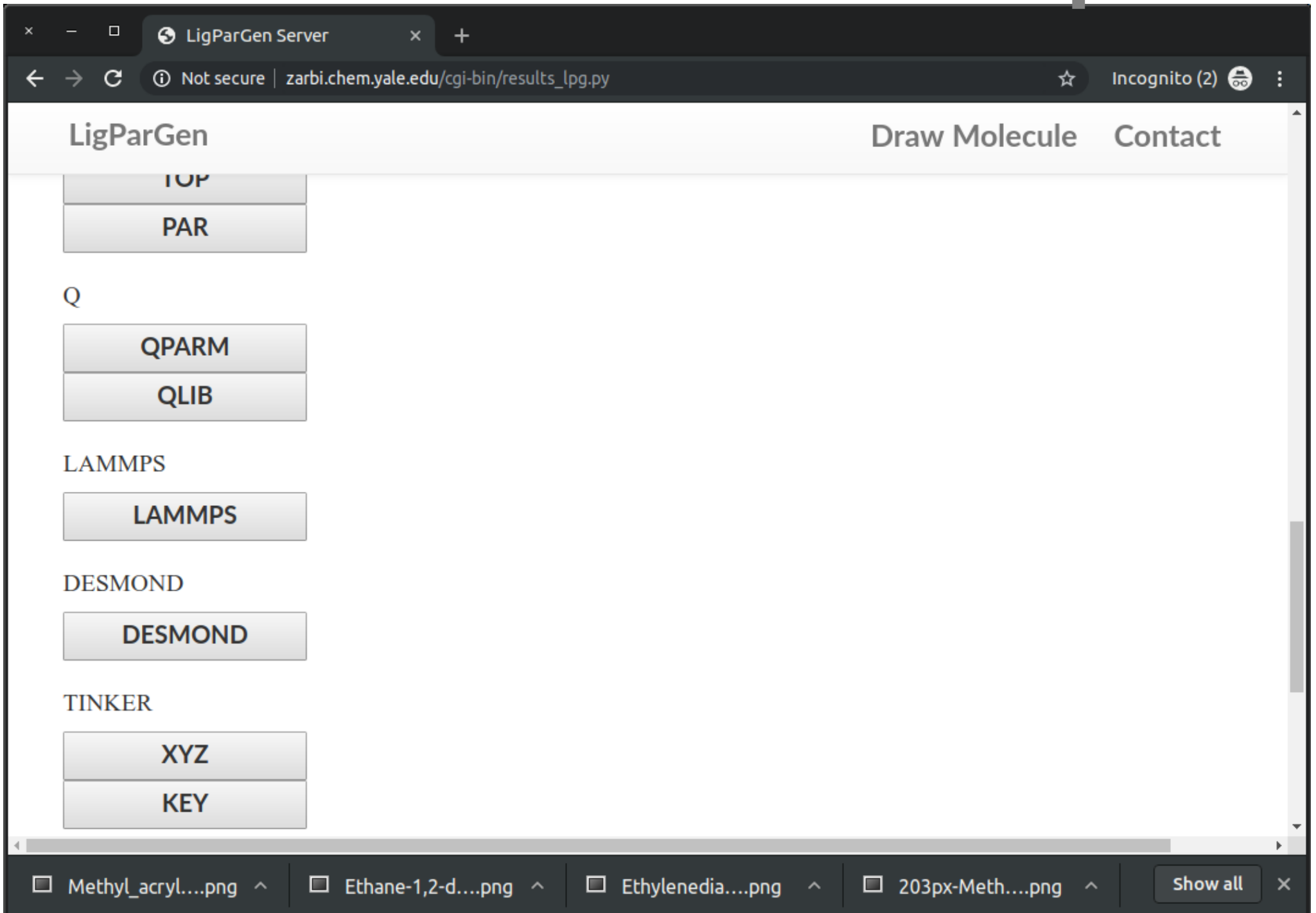
OpenMM

XML

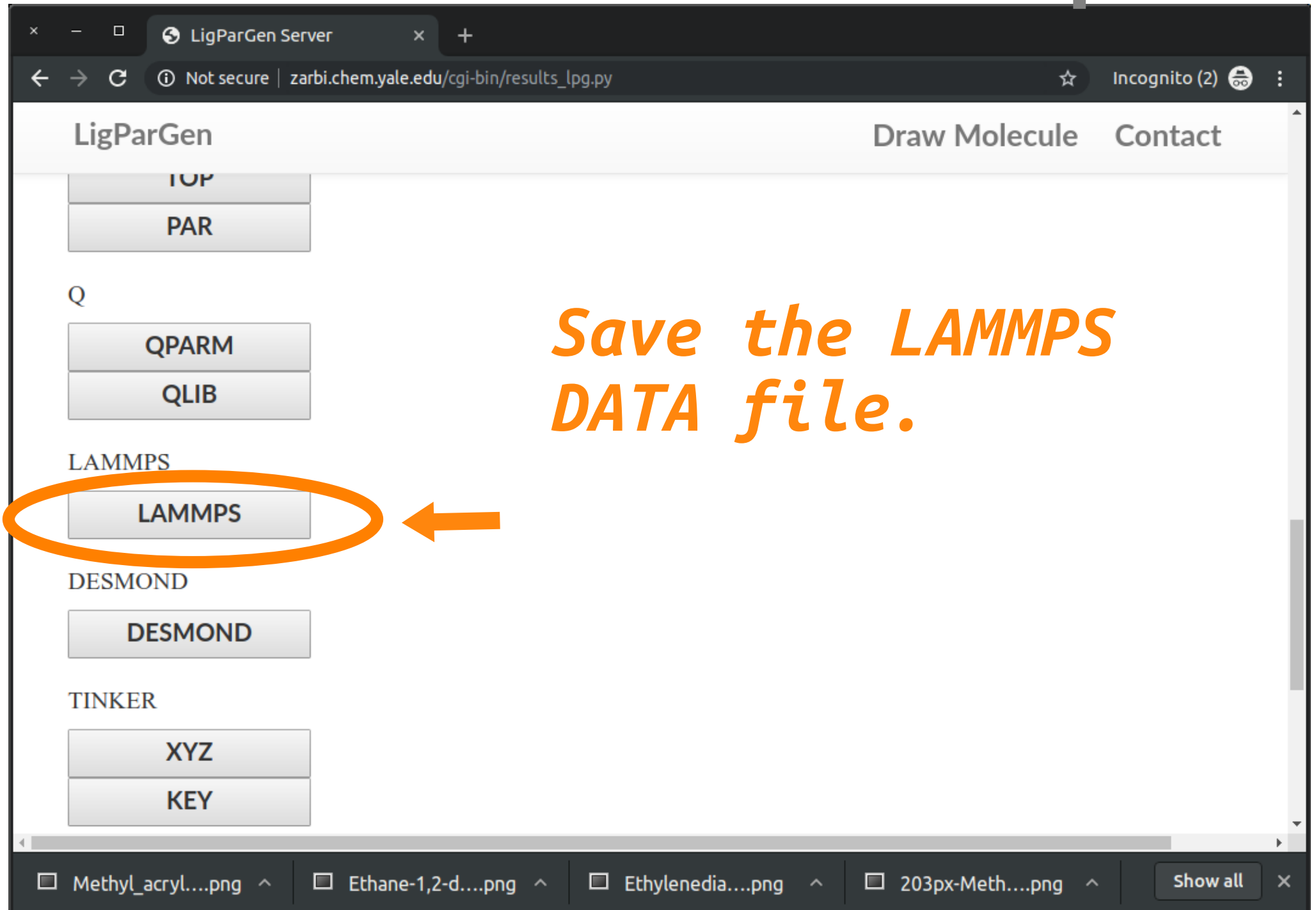
PDB

Methyl_acryl....png | Ethane-1,2-d....png | Ethylenedia....png | 203px-Meth....png | Show all

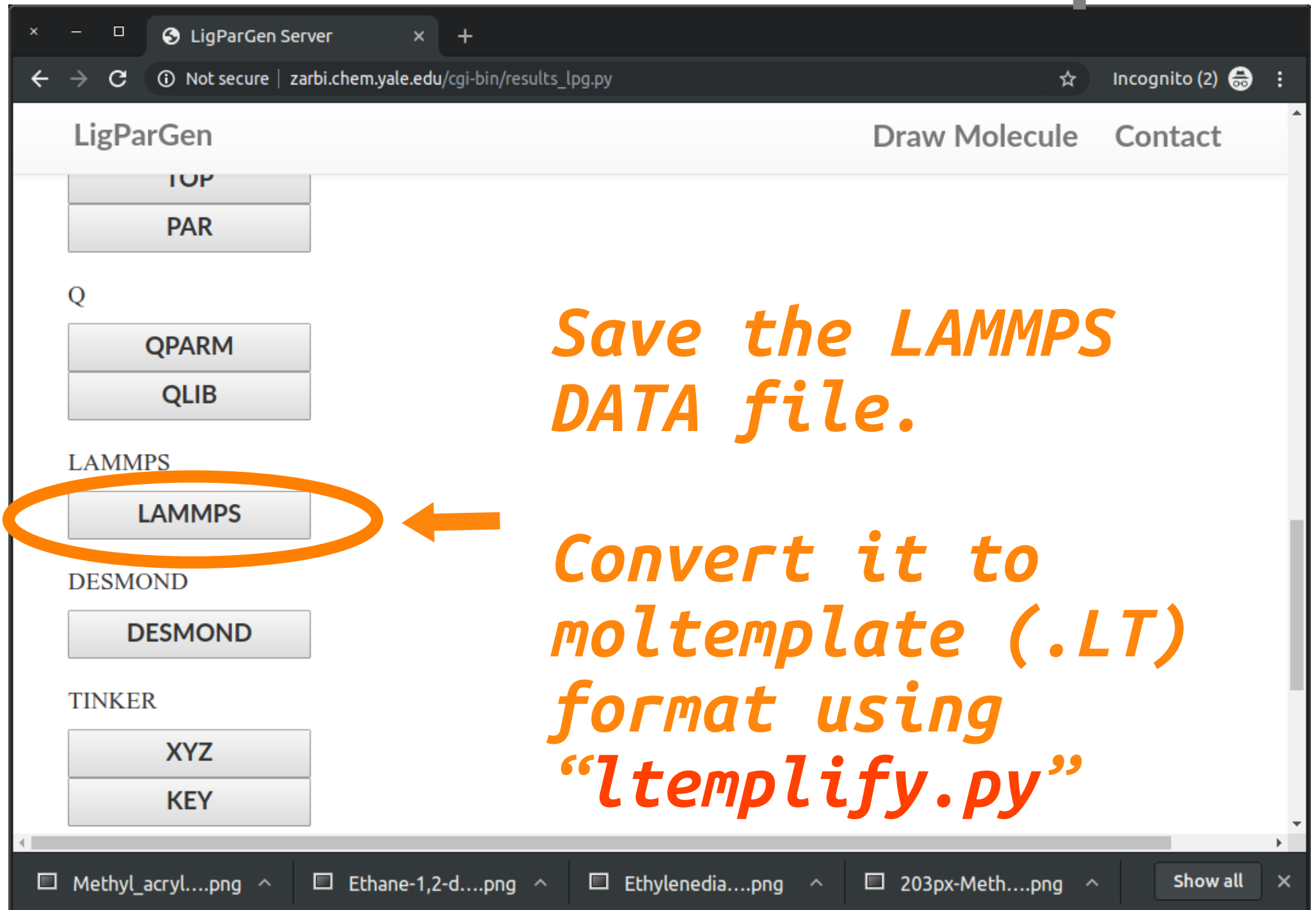
LibParGen → moltemplate



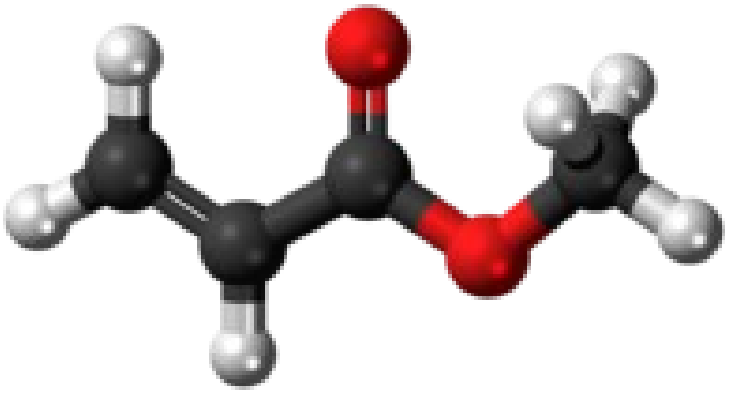
LibParGen → moltemplate



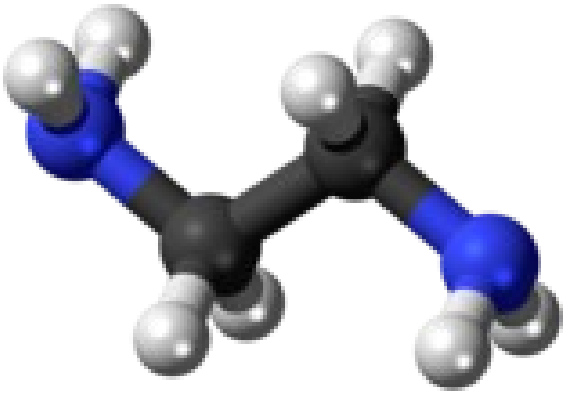
LibParGen → moltemplate



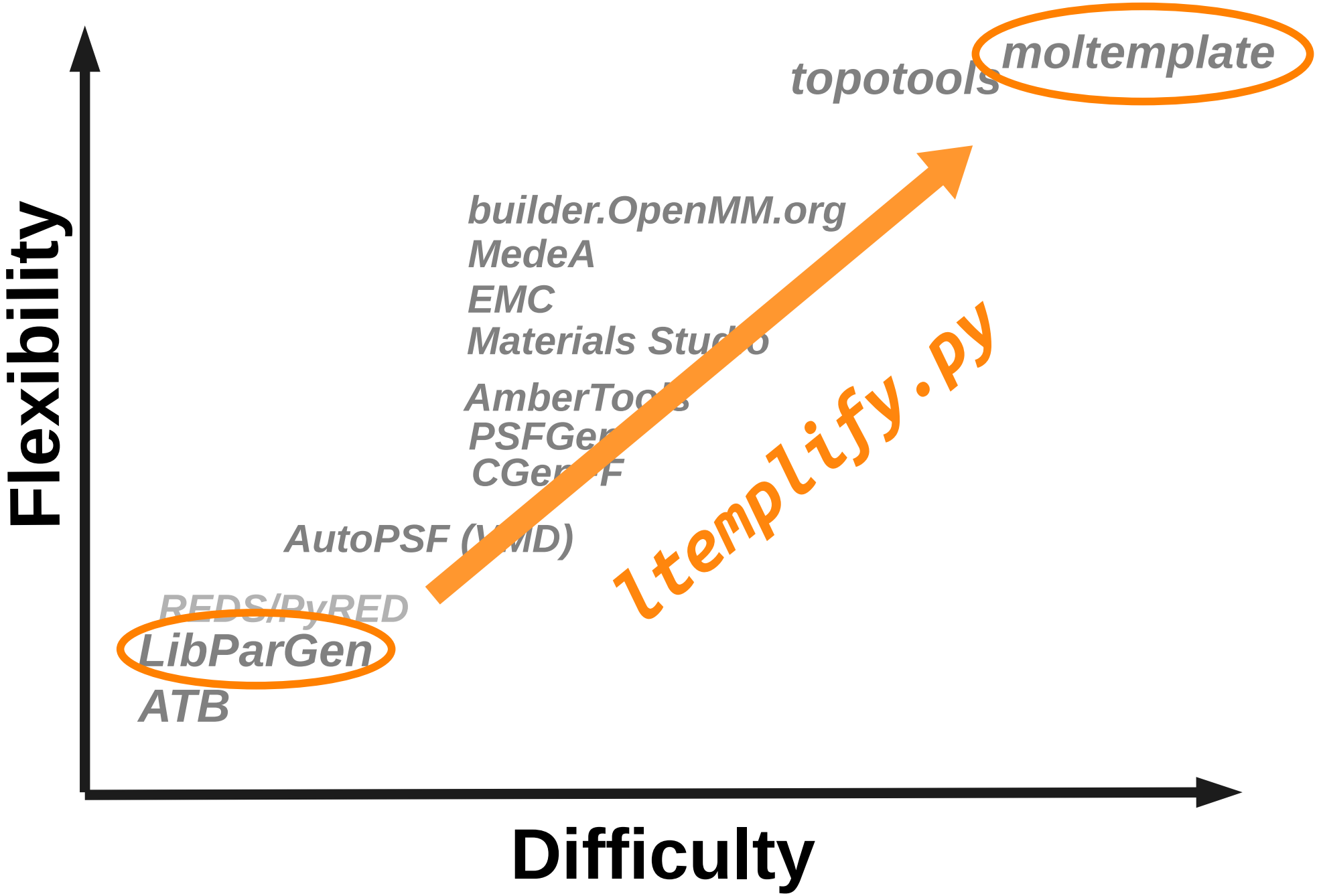
Mixtures of molecules: Atom types will not clash



+



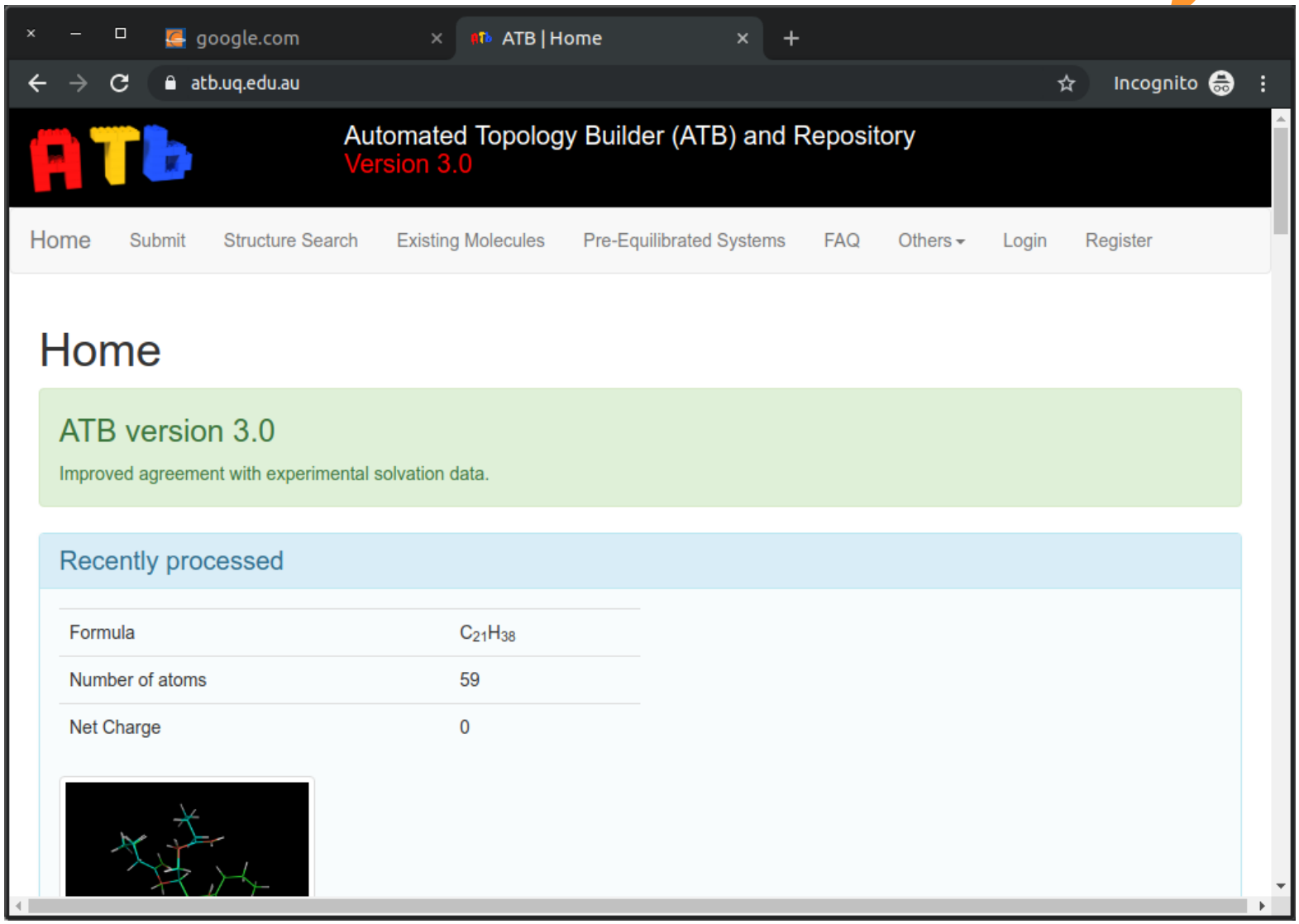
Tradeoff


moltemplate
topotools
builder.OpenMM.org
MedeA
EMC
Materials Studio
AmberTools
PSFGen
CGenFF
AutoPSF (VMD)
REDS/PyRED
LibParGen
ATB
ltemplify.py
Flexibility
Difficulty

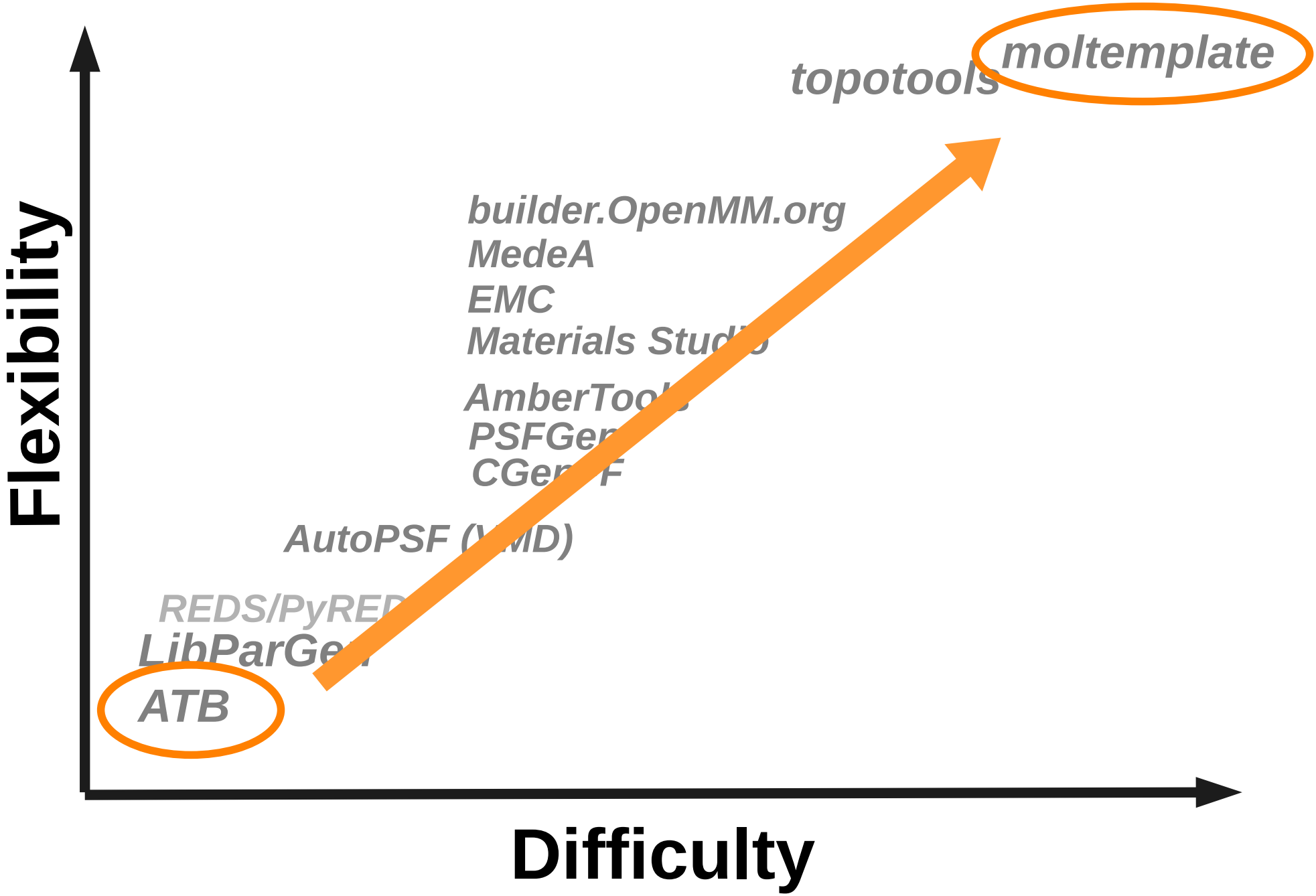
ATB → moltemplate

https://atb.uq.edu.au

GROMOS force field only



Tradeoff


Flexibility
Difficulty
moltemplate
topotools
builder.OpenMM.org
MedeA
EMC
Materials Studio
AmberTools
PSFGen
CGenFF
AutoPSF (VMD)
REDS/PyRED
LibParGen
ATB

VMD/topotools ➞ moltemplate

- Load a PDB file in VMD

VMD/topotools → moltemplate

- Load a PDB file in VMD
- Generate a PSF file with PSFGEN

VMD/topotools → moltemplate

- Load a PDB file in VMD
- Generate a PSF file with PSFGEN
- Use topotools to create a LAMMPS data file (with correct atom type names)

VMD/topotools ➞ moltemplate

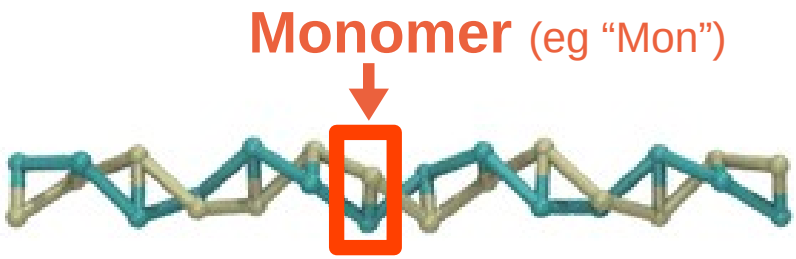
- Load a PDB file in VMD
- Generate a PSF file with PSFGEN
- Use topotools to create a LAMMPS data file (with correct atom type names)
- Use ltemplify.py to convert it to moltemplate format.

VMD/topotools → moltemplate

- Load a PDB file in VMD
- Generate a PSF file with PSFGEN
- Use topotools to create a LAMMPS data file (with correct atom type names)
- Use ltemplify.py to convert it to moltemplate format.

TO-DO: convert CHARMM formatted force fields into moltemplate format.

Making long polymers:

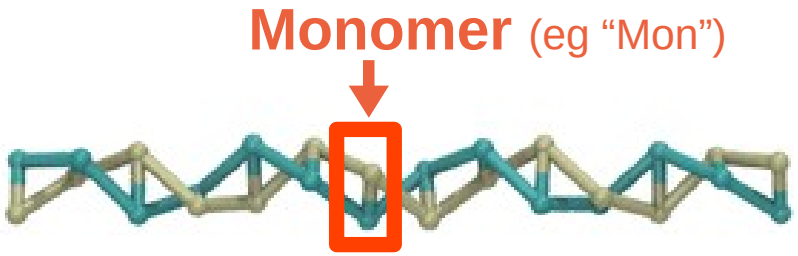


polymer

```
mon1 = new Mon.rotvv(1,0,0, 0.75,-0.2,0.65).move(6,-1.3,41.2)
mon2 = new Mon.rotvv(1,0,0, 0.37,0.67,0.63).move(10.2,2.1,36.1)
mon3 = new Mon.rotvv(1,0,0, 0.36,0.91,0.12).move(11.1,6.7,34)
    :

write("Data Bonds") {
  $bond:a1 @bond:Backbone  $atom:mon1/O3p_a  $atom:mon2/P_a
  $bond:b1 @bond:Backbone  $atom:mon1/P_b    $atom:mon2/O2p_b
  $bond:a2 @bond:Backbone  $atom:mon2/O3p_a  $atom:mon3/P_a
  $bond:b2 @bond:Backbone  $atom:mon2/P_b    $atom:mon3/O2p_b
     :        :                 :                 :
}
```

Limitation: moltemplate needs a “for loop”

Making long polymers:

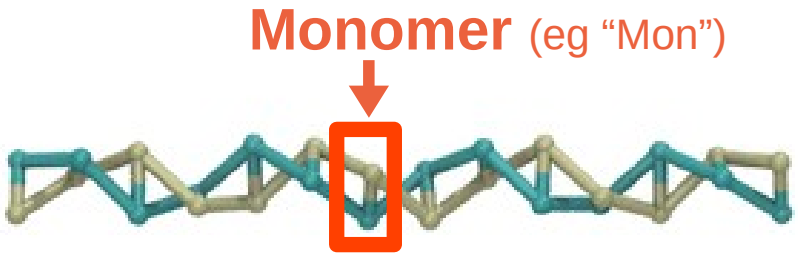


polymer

Write a script to generate the text that moltemplate will read.

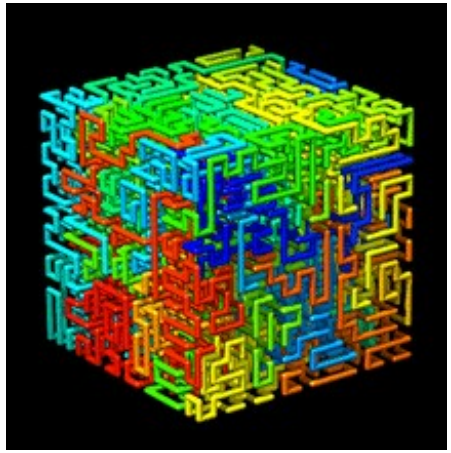
(not an ideal solution)

Polymers that follow a curve:



polymer

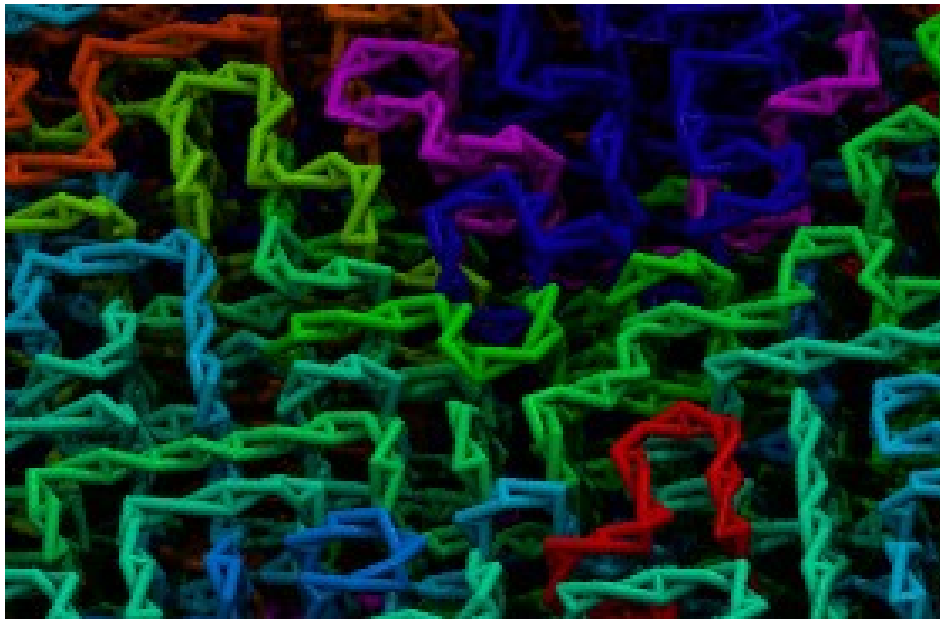
+



arbitrary curve

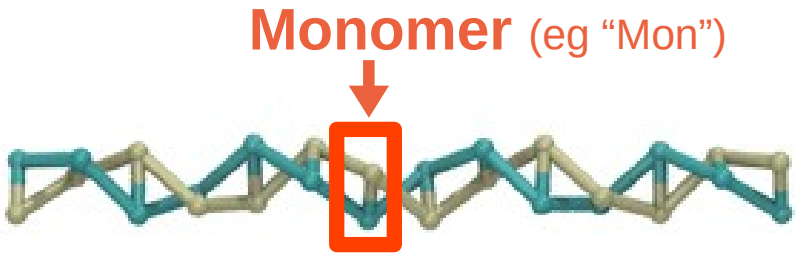
genpoly_lt.py

(included with moltemplate)



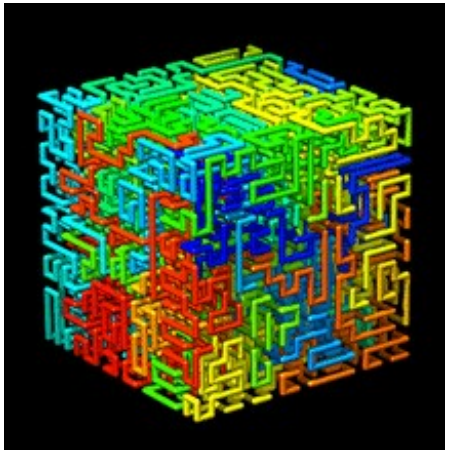
polymer follows curve

Polymers that follow a curve:



polymer

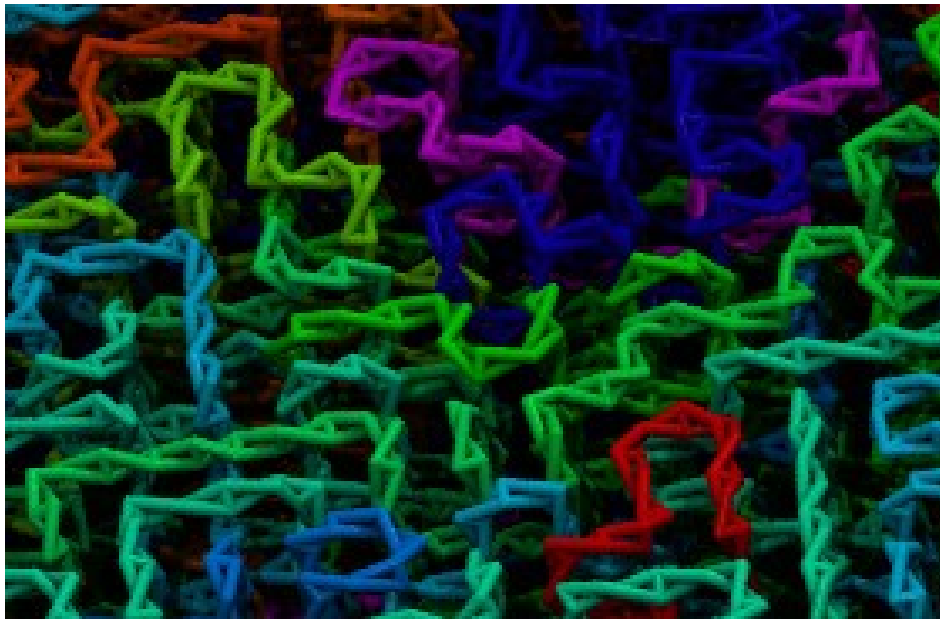
+



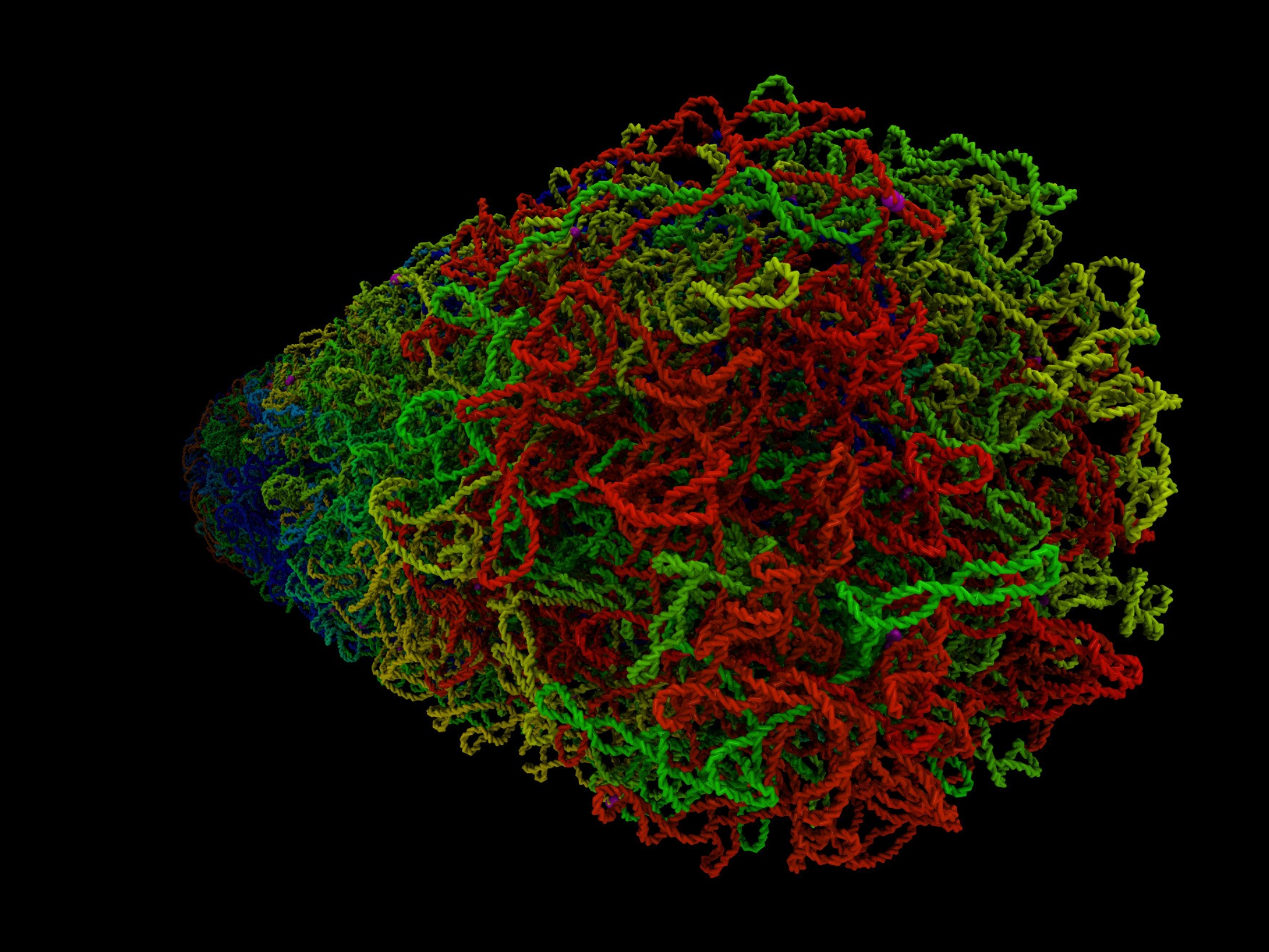
arbitrary curve

genpoly_lt.py

```
$  genpoly_lt.py -helix 34.2857 \
      -monomer-name "Mon"
      -bond Backbone O3p_a  P_a \
      -bond Backbone P_b  O3p_b \
       < curve_coordinates.txt \
       > polymer.lt
```



genpoly_lt.py output



Polymer MELTS in moltemplate

- Generate a space filling curve.
  (eg. *with ndlattice)*

- Smooth it (eg. *with interpolate_mt.py*)

- Decide where to cut it
  (if you have multiple polymers)

- Use ***gen_poly_lt.py***

- Use pair_style lj/soft/…
  to equilibrate

Survey: Possible Future Directions

- Make moltemplate run within python?

  (for loops, procedurally generated molecules, speed improvements)
- Infer bonds between nearby atoms?
- CHARMM Force Field?
- GROMOS Force Field?
- Directly export VMD → moltemplate?
- Read OpenFF/OpenMM files?
- Read PSF (CHARMM, OPLSAA) files?
- Read PRMTOP (AMBER) files?
- Graphical interphase (GUI)?
- More tutorials + better documentation?

Talk slides, examples, docs:
are at *moltemplate.org*

Contact me at:

jewett.aij@gmail.com