**Very Broad Workflow Clarification:**

1. **Initializing Conditions**
   1. **XYZ data**
   2. **Velocity Data (Temperature)**
   3. **Integrators (NVT/NVE/NPT…** [**http://www.quimica.urv.es/~bo/MOLMOD/General/Dynamics/Ensembles.html#NPT**](http://www.quimica.urv.es/~bo/MOLMOD/General/Dynamics/Ensembles.html#NPT) **)**
2. **Running the sim**
3. **Analyzing the sim**

Create Initial Conditions:

1. Use **Avogadro** to build/load in atoms/molecules you need
   1. Save them as pdb or xyz files
2. Create an input file for **packmol** that assembles the molecules into a mixture/lattice, following this structure:
   1. Comment the changes so it’s readable, easily troubleshooted, and you can be certain it’s true to reality
   2. Keep the tolerance as default
   3. Box length??? Process is as follows:
      1. Calculate a weighted average of the components of the sim

|  |
| --- |
| # A mixture of air's 3 most common 'elements' and HCN  # All the atoms from different molecules will be separated at least 2.0  # Angstroms at the solution.  tolerance 2.0  # The file type of input and output files is XYZ (xyz can be used in HOOMD)  filetype xyz  # The name of the output file  output airHCN.xyz  # 7809 Nitrogen molecules, 2095 oxygen molecules, 93 Argon molecules, and one molecule of HCN  # defined by the minimum coordinates x, y and z = 0. 0. 0. and maximum  # coordinates 40. 40. 40. That is, they will be put in a cube of side  # 40. (the keyword "inside cube 0. 0. 0. 40.") could be used as well.  structure N2.pdb  number 7809  inside box 0. 0. 0. 80. 80. 80.  end structure  structure O2.pdb  number 2095  inside box -40. -40. -40. 40. 40. 40.  end structure  structure Ar.pdb  number 93  inside box -40. -40. -40. 40. 40. 40.  end structure  structure HCN.pdb  number 1  inside box -40. -40. -40. 40. 40. 40.  end structure |

\*\*\*this part specifies your output file type. Either xyz or pdb. Xyz is a more simple, direct format

1. Run Packmol using *packmol < input.inp*
   1. Example: *packmol < airHCN.inp*
   2. Check to make sure the box is large enough to fit all the molecules. If it’s not, you won’t stop getting errors. The command *check* should work but it’s prolly easier to just try to turn the output into a gsd and then just try the rest of the process. You’ll get an error that says something like “not all particles fit within the box” if your box length is too small.
2. WARNING!!! Mbuild, a later step in the process, increased the x, y, and z coordinates of our box over by half the length. So we changed the coordinates to “-40. -40. -40. 40. 40. 40.”. Why? Prolly because mbuild centers the new file format @ the origin (aka 0. 0. 0), but packmol has no necessary origin value… Always check the gsd file in OVito to check if the molecules are actually centered within the box
3. Packmol useful links:
   1. Input examples: <http://m3g.iqm.unicamp.br/packmol/examples.shtml>
   2. User guide: <http://m3g.iqm.unicamp.br/packmol/userguide.shtml>
   3. Output example:

|  |
| --- |
|  |

(file generated by vmd because we needed to change original file format, pdb, to an xyz. Should not be the case in the future because we will be obtaining an xyz file from packmol)

1. Now, we’ll move into hoomd for the rest of the work with this simulation. We need to transfer our xyz/pdb into a file type that hoomd can use, that is, a gsd file. To do this, we need to use mbuild. Open a notebook in Jupyter, and follow these following steps (the whole path example can be seen at the following link, but I’ll type out a more detailed explanation of each step in this google doc): http://localhost:8888/notebooks/Clean%20airHCN.ipynb
2. First, we need to import all the modules/programs/whatever we need to use in our notebook. This makes our process a little cleaner, as we can have all our imports at the top of our notebook. This is a personal preference thing, so it doesn’t really matter when you import everything (as long as it’s before you need to use it). Hoomd.context.initialize actually gets the engine up and running, able to follow commands.

|  |
| --- |
| import hoomd  import numpy  import hoomd.md  import sys  import gsd  import gsd.hoomd  hoomd.context.initialize("") |

1. When the sim is done, convert the xyz/pdb file to a gsd or hoomdxml file using mbuild:

|  |
| --- |
| import mbuild as mb  box = mb.load("/home/themotivation/Avo\_atoms/airHCN.pdb")  box.save("/home/themotivation/Avo\_atoms/airHCN.gsd")  print("it worked!")  #print command is to see if our function worked. |

1. Now that we have a gsd, we need to represent that data in a snapshot (hoomd’s version of positional data). Check out: <https://nbviewer.jupyter.org/github/joaander/hoomd-examples/blob/master/Initialization%20-%20Populate%20snapshot.ipynb>

|  |
| --- |
| #first line opens the gsd file so hoomd can read it. second line specifies that we want it to start reading at the first (and only) frame of the gsd file.  #third line confirms frame.  #fourth line shows (‘prints’) us the type/name of particles (atoms/molecules) that hoomd recognizes. We will use this information later  #fifth line shows (‘prints’) us the snapshot at the specified timestep in the gsd file  #the commented commands at the end were experimental and might be of use to someone else, but we deemed them as unimportant for now  t = gsd.hoomd.open(name='/home/themotivation/Avo\_atoms/airHCN.gsd', mode='rb')  snap = t[0]  snap.configuration.step  print(snap.particles.types)  print(snap.particles.position)  #snap.particles.particle\_types  #snap.particles.position |

1. Now that hoomd recognizes and reads this gsd file we created and our snapshot has been established, we need to save the entire snapshot into another snapshot. Why? Because hoomd would not recognize the first snapshot and we found that saving the snapshot into a new one is one way around this problem. The following section saves that original snapshot into a new one. We called the new snapshot “snap2” because we’re super creative.

|  |
| --- |
| snap2 = hoomd.data.make\_snapshot(N=19904, box=hoomd.data.boxdim(L=100), particle\_types=['AR', 'C', 'H', 'N', 'O'])  for x in range(0, 19904):  snap2.particles.position[x] = snap.particles.position[x]  print(snap2.particles.position) |

1. When that string is finished, we can tell hoomd to read the snapshot we just created!!

|  |
| --- |
| hoomd.init.read\_snapshot(snap2); |

“Now HOOMD is ready for pair forces, integrators, and other commands to configure the simulation run.”, aka thermalization (assigning velocities to our sim)

1. Thermalization is *not* a step that we need to be completely anal with in the initialization. The velocities will sort themselves out eventually through equilibration. However, we need the temperature to be accurate. In order to translate temperature into initial velocities of the particles, we need to find a magnitude of velocity and assign random components to each molecule.

1

|  |
| --- |
| import hoomd  import numpy  import hoomd.md  import sys  import gsd  import gsd.hoomd  hoomd.context.initialize("") |

2

|  |
| --- |
| import mbuild as mb  box = mb.load("/home/themotivation/Avo\_atoms/airHCN.pdb")  box.save("/home/themotivation/Avo\_atoms/airHCN.gsd")  print("it worked!") |

3

|  |
| --- |
| #first line opens the gsd file so hoomd can read it. second line specifies the first (and only) frame of the gsd file (this gsd .  #third line confirms frame.  #fourth line shows us the type/name of particles (atoms/molecules) that hoomd recognizes  #first line opens the gsd file so hoomd can read it. second line specifies the first (and only) frame of the gsd file (this gsd .   #third line confirms frame.   #fourth line shows us the type/name of particles (atoms/molecules) that hoomd recognizes  #fifth line shows us the snapshot at the specified timestep in the gsd file   t = gsd.hoomd.open(name='/home/themotivation/Avo\_atoms/airHCN.gsd', mode='rb')  snap = t[0]  snap.configuration.step  print(snap.particles.types)  print(snap.particles.position)  #snap.particles.particle\_types  #snap.particles.position  #fifth line shows us the snapshot at the specified timestep in the gsd file  t = gsd.hoomd.open(name='/home/themotivation/Avo\_atoms/airHCN.gsd', mode='rb')  snap = t[0]  snap.configuration.step  print(snap.particles.types)  print(snap.particles.position)  #snap.particles.particle\_types  #snap.particles.position |

4

|  |
| --- |
| snap2 = hoomd.data.make\_snapshot(N=19904, box=hoomd.data.boxdim(L=100), particle\_types=['AR', 'C', 'H', 'N', 'O'])  for x in range(0, 19904):  snap2.particles.position[x] = snap.particles.position[x]  print(snap2.particles.position) |

5

|  |
| --- |
| hoomd.init.read\_snapshot(snap2); |

6

|  |
| --- |
| print(snap.particles.diameter) |

List of things you need to download/tools we use:

1. Avogadro (used to create the actual molecules. Makes bonds and inter-atomic reaction)
2. Packmol (Used to assemble the molecules in a “lattice” or block)
3. Anaconda-Navigator (A huge umbrella program that contains the Jupyter notebook)
   1. Jupyter Notebook (Already pre-installed in anaconda-navigator)
      1. Hoomd (The engine that runs simulations we set up with the developed script we built. Using input files in Hoomd is a process we still need to fine-tune)
      2. mbuild (used to convert files, typically used as a molecule builder. We use it to convert pdb or xyz files into gsd or hoomdxl)

Install Packmol: http://m3g.iqm.unicamp.br/packmol/download.shtml

Install Anaconda-Navigator: conda install -c anaconda anaconda-navigator

Hoomd:

Mbuild: <http://mosdef-hub.github.io/mbuild/installation.html>

$ conda config --add channels omnia  
$ conda config --add channels mosdef  
$ conda install mbuild

Fourloop: used to make repetitive actions

Snap.particles.position[0] = (x, y, z)

Github: store: supplementary input files, .py of source code, documentation