TUTORIAL 1 (by M. J. Demkowicz, all rights reserved)

VISUALIZING ATOMIC STRUCTURES

A big part of understanding atomistic models is being able to visualize them. This tutorial will explain how to make use of some basic visualization software, build models of perfect single crystals, and create vacancies and free surfaces in them.

Connecting to HPRC

Throughout this tutorial, we will be using a virtual portal to the ada computer at HPRC. To connect, you will need a valid NetID, active VPN and Duo access setup, and a basic HPRC allocation. I hope you were all able to get this arranged prior to this tutorial!

Start out by logging in to VPN. Next, navigate the to ada portal:

portal-ada.hprc.tamu.edu

If you would like to learn more about how this portal works, you can view an online primer here: https://youtu.be/dqa2ZzsEmQs. The portal includes an easy interface for file transfer and a built-in editor. You can upload and download files using the "files" menu. Note: please use your scratch directory, not your home directory for file upload/download!

Go to the "interactive apps" menu and select VNC. You will be prompted for some input. Please put in the following:

- Number of hours: 4

- Number of cores: 1
- Memory per core (GB): 2
- Node type: any
- Account, email: you can leave these fields blank

Press “launch” and wait a minute. Press the “Launch noVNC in New Tab” button once it shows up. Now you should see a virtual desktop with one open command line interface in your browser.

Go to the command line and issue the following two commands:

- cd /scratch/user/***user name***, where ***user name*** is your NetID
- ml AtomEye/A3

Now you are all set up for the present tutorial. Once you are done with it, please log yourself out completely by doing the following two things:

- Type “exit” in the command line
- Go to the tab in your web browser where the VNC session is running and press “delete” (note: don’t do this in a tab with your file transfer or you’ll inadvertently delete files!).

AtomEye

There are numerous visualization packages available as freeware online. We will go over how to use a specific package called AtomEye. The package has already been pre-installed on ada, but, if you want to run it on your own computer, you can visit this site and see if there’s a version you can download:

http://li.mit.edu/Archive/Graphics/A/

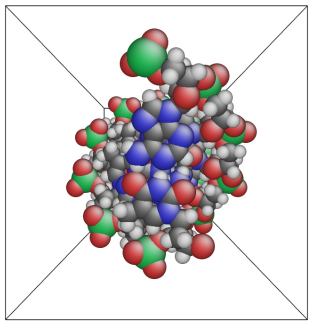
This site also contains a lot of useful info, including manuals, a gallery, and sample structure files.

Now let's visualize something. Click "gallery" on the top/center of the AtomEye home page. This gives you a bunch of pictures labeled with links to files that contain the structure file that was used to create the picture. Download one of the structure files to the directory with your AtomEye executable. I chose the file "DNA.pdb."

In your AtomEye directory, issue the command

```
A3 DNA.pdb
```

A window should open up with the following view:



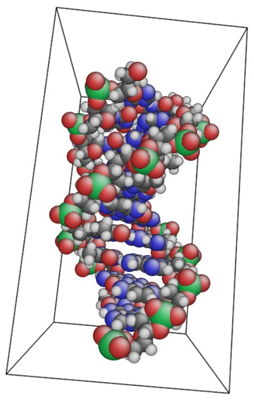
NOTES:

- When it's running, AtomEye will occupy your command line. You may want to open a new terminal window, if you want to issue additional commands while you have AtomEye on. You can do that by right-clicking and choosing the appropriate entry in the pop-up menu. Alternatively, you can open a new window before running AtomEye by issuing the following command:

```
xterm &
```

- If you are using some different platform (e.g., your own computer) and your AtomEye executable has some name other than A3, then substitute that name for "A3" in the above command. Similarly, if DNA.pdb is not the structure file you downloaded, then substitute the name of your file for "DNA.pdb" in the command given above.

This picture doesn't look much like a DNA molecule because you're looking along its axis. To see it for the double helix that it actually is, rotate the picture by clicking and dragging with your mouse. After re-orienting the image, I get something that looks more like DNA:



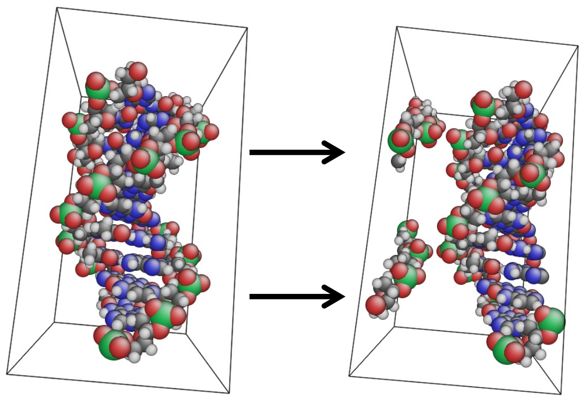
One of the reasons why I like AtomEye is that it's a simple program that offers a lot of functionality, but isn't too complicated to use. For example, to create a .jpg picture of what you see in AtomEye, simply press the button "j." You will be prompted to enter the name of your file (the program appends the ".jpg" suffix automatically). And that's it, you're done. You should now see a .jpg file with the name your chose in the directory from which you ran AtomEye.

On the top/center of the AtomEye home page, you can find a link labeled "manual" that will show you how easy it is to do things like change between perspective to parallel projections, change the sizes and colors of atoms, visualize bonds, etc. I highly recommend that you spend a few minutes toying around with the possibilities to get familiar with the program.

Periodic boundary conditions

In the above two pictures, you see that the DNA structure is enclosed in a box. This is called the "simulation cell" or "supercell" of the atomic model. It defines a region inside of which the atoms of the model are located. What happens at the walls of the simulation cell?

Like most atomic modeling packages, AtomEye imposes "periodic boundary conditions" at the simulation cell walls. That means that atoms that leave the cell through one wall return back into the cell through the opposite wall. The easiest way to see this is to try it: hold down "shift" and drag the DNA molecule to the right in AtomEye. You will see atoms disappearing on the far right side and re-emerging on the far left side, as shown below. That's how periodic boundary conditions work. Similar things happen if you try moving the molecule up: atoms disappear on the top and come out on the bottom.



Although they may seem a little strange at first, periodic boundary conditions are actually a great convenience in modeling atomic structures. Thanks to them, you can create a model containing a relatively small number of atoms that behaves as a "quasi-infinite" system. This can be achieved if you construct your model such that the atoms at one surface of the simulation cell align perfectly with the atoms at the opposite surface. To see what I mean, open file "Nanotube8x3x1.pdb" from the AtomEye gallery and translate it parallel to the nanotube axis (as before, drag with your mouse while holding "shift").

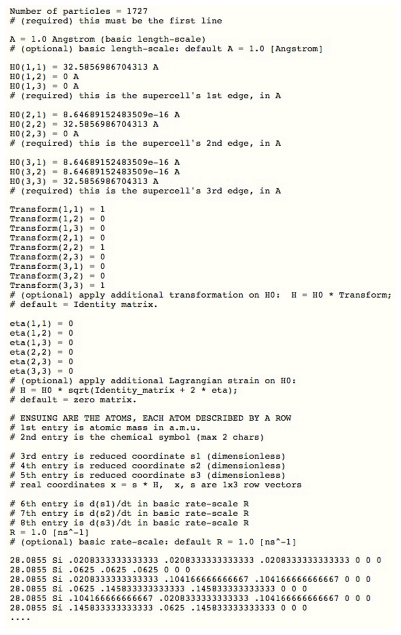
<u>**Making a model of a crystal**</u>

Whenever you apply periodic boundary conditions (PBCs, for short) to a simulation cell, you are in fact turning a finite system in a quasi-infinite system. For example, the file with the DNA is really not a single section of DNA, but an infinite array of such sections repeated throughout 3-D space. The array is symmetric with respect to translations equal to the lengths of the simulation cell.

If this makes you think of the repeating unit cells of crystals, then you're on the right track. Periodic boundary conditions are very useful as a way of creating "quasi-infinite" atomic models of single crystals without having to specify the locations of an infinite number of atoms. Here's how to create such a model.

Let's make a model of a simple cubic crystal under periodic boundary conditions. The only simple cubic element is polonium (Po), which has a cubic lattice parameter of 3.36Å. Theoretically, we could make the model using just one unit cell under periodic boundary conditions, but since I later want to use this model to make some crystal defects, I'll construct it using 10x10x10 unit cells.

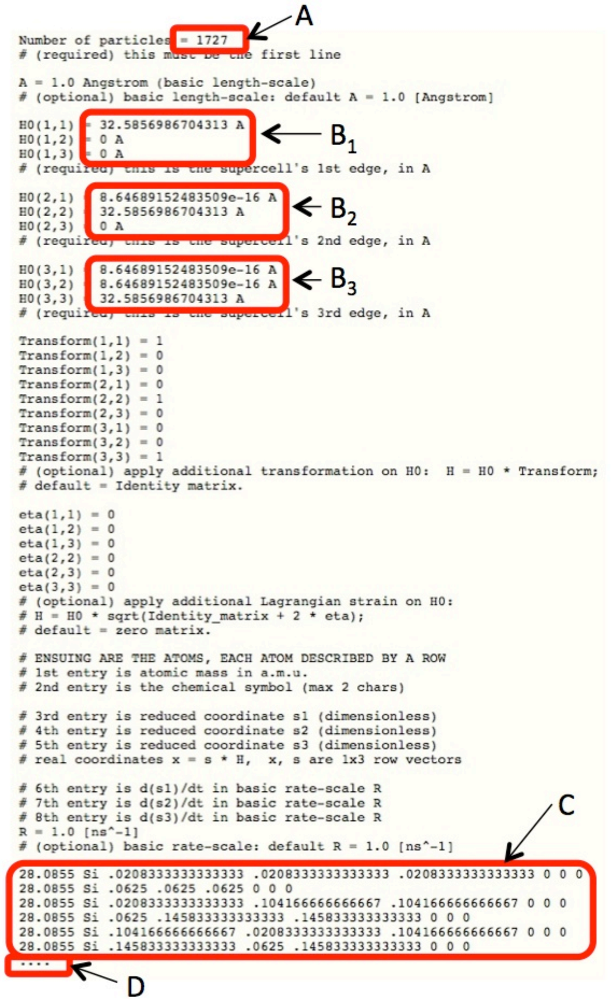
I will construct the model using (what I consider to be) the simplest structure file format used by AtomEye: the .cfg file. To learn about it, go to AtomEye home page and click "file formats" on the top/left and scroll down to the heading "standard CFG format." You will see a brief explanation of the format followed by a sample that looks like this:



```
Number of particles = 1727
# (required) this must be the first line

A = 1.0 Angstrom (basic length-scale)
# (optional) basic length-scale: default A = 1.0 [Angstrom]

H0(1,1) = 32.5856986704313 A
H0(1,2) = 0 A
H0(1,3) = 0 A
# (required) this is the supercell's 1st edge, in A

H0(2,1) = 8.64689152483509e-16 A
H0(2,2) = 32.5856986704313 A
H0(2,3) = 0 A
# (required) this is the supercell's 2nd edge, in A

H0(3,1) = 8.64689152483509e-16 A
H0(3,2) = 8.64689152483509e-16 A
H0(3,3) = 32.5856986704313 A
# (required) this is the supercell's 3rd edge, in A

Transform(1,1) = 1
Transform(1,2) = 0
Transform(1,3) = 0
Transform(2,1) = 0
Transform(2,2) = 1
Transform(2,3) = 0
Transform(3,1) = 0
Transform(3,2) = 0
Transform(3,3) = 1
# (optional) apply additional transformation on H0:  H = H0 * Transform;
# default = Identity matrix.

eta(1,1) = 0
eta(1,2) = 0
eta(1,3) = 0
eta(2,2) = 0
eta(2,3) = 0
eta(3,3) = 0
# (optional) apply additional Lagrangian strain on H0:
# H = H0 * sqrt(Identity_matrix + 2 * eta);
# default = zero matrix.

# ENSUING ARE THE ATOMS, EACH ATOM DESCRIBED BY A ROW
# 1st entry is atomic mass in a.m.u.
# 2nd entry is the chemical symbol (max 2 chars)

# 3rd entry is reduced coordinate s1 (dimensionless)
# 4th entry is reduced coordinate s2 (dimensionless)
# 5th entry is reduced coordinate s3 (dimensionless)
# real coordinates x = s * H,  x, s are 1x3 row vectors

# 6th entry is d(s1)/dt in basic rate-scale R
# 7th entry is d(s2)/dt in basic rate-scale R
# 8th entry is d(s3)/dt in basic rate-scale R
R = 1.0 [ns^-1]
# (optional) basic rate-scale: default R = 1.0 [ns^-1]

28.0855 Si .0208333333333333 .0208333333333333 .0208333333333333 0 0 0
28.0855 Si .0625 .0625 .0625 0 0 0
28.0855 Si .0208333333333333 .104166666666667 .104166666666667 0 0 0
28.0855 Si .0625 .145833333333333 .145833333333333 0 0 0
28.0855 Si .104166666666667 .0208333333333333 .104166666666667 0 0 0
28.0855 Si .145833333333333 .0625 .145833333333333 0 0 0
....
```

Copy and paste this into a new text file: this will be a template that we will modify to create our single crystal model. Use any text editor you like, but make sure that you save this as a simple, ASCII text file and not something else like an MS-Word file. Also, make sure that the suffix is ".cfg" and not ".txt" or anything else. I saved my file as "sc_Po.cfg" to help me remember what's in it.

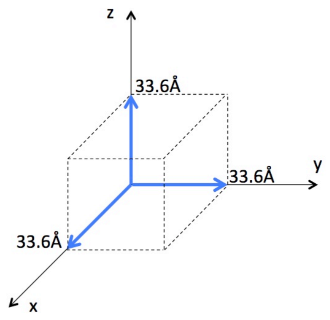
We will focus on modifying parts A, $B_1/B_2/B_3$, C, and D in the .cfg file, indicated below.



```
Number of particles = 1727
# (required) this must be the first line

A = 1.0 Angstrom (basic length-scale)
# (optional) basic length-scale: default A = 1.0 [Angstrom]

H0(1,1) = 32.5856986704313 A
H0(1,2) = 0 A
H0(1,3) = 0 A
# (required) this is the supercell's 1st edge, in A

H0(2,1) = 8.64689152483509e-16 A
H0(2,2) = 32.5856986704313 A
H0(2,3) = 0 A
# (required) this is the supercell's 2nd edge, in A

H0(3,1) = 8.64689152483509e-16 A
H0(3,2) = 8.64689152483509e-16 A
H0(3,3) = 32.5856986704313 A
# (required) this is the supercell's 3rd edge, in A

Transform(1,1) = 1
Transform(1,2) = 0
Transform(1,3) = 0
Transform(2,1) = 0
Transform(2,2) = 1
Transform(2,3) = 0
Transform(3,1) = 0
Transform(3,2) = 0
Transform(3,3) = 1
# (optional) apply additional transformation on H0:  H = H0 * Transform;
# default = Identity matrix.

eta(1,1) = 0
eta(1,2) = 0
eta(1,3) = 0
eta(2,2) = 0
eta(2,3) = 0
eta(3,3) = 0
# (optional) apply additional Lagrangian strain on H0:
# H = H0 * sqrt(Identity_matrix + 2 * eta);
# default = zero matrix.

# ENSUING ARE THE ATOMS, EACH ATOM DESCRIBED BY A ROW
# 1st entry is atomic mass in a.m.u.
# 2nd entry is the chemical symbol (max 2 chars)

# 3rd entry is reduced coordinate s1 (dimensionless)
# 4th entry is reduced coordinate s2 (dimensionless)
# 5th entry is reduced coordinate s3 (dimensionless)
# real coordinates x = s * H,  x, s are 1x3 row vectors

# 6th entry is d(s1)/dt in basic rate-scale R
# 7th entry is d(s2)/dt in basic rate-scale R
# 8th entry is d(s3)/dt in basic rate-scale R
R = 1.0 [ns^-1]
# (optional) basic rate-scale: default R = 1.0 [ns^-1]
28.0855 Si .0208333333333333 .0208333333333333 .0208333333333333 0 0 0
28.0855 Si .0625 .0625 .0625 0 0 0
28.0855 Si .0208333333333333 .104166666666667 .104166666666667 0 0 0
28.0855 Si .0625 .145833333333333 .145833333333333 0 0 0
28.0855 Si .104166666666667 .0208333333333333 .104166666666667 0 0 0
28.0855 Si .145833333333333 .0625 .145833333333333 0 0 0
....
```

Part A is the number of atoms in our model. Since we are making a 10x10x10 simple cubic structure and since there is only one atom per unit cell in this structure, the total number of atoms is 1000. Thus, change the entry in A to 1000.

Parts B_1, B_2, and B_3 describe the size and shape of the simulation cell. Each of these entries is actually a 3-D vector: it contains three numbers. Each vector describes the length and orientation of one of the simulation cell edges. We want a model containing 10x10x10 unit cells, so the simulation cell lengths “l” are equal to ten times the lattice parameter of Po: l=10*3.36Å=33.6Å. Since we are building a simple cubic crystal, the three edges of our unit cell will be oriented parallel to the x-, y-, and z-directions. The three vectors that describe the simulation cell are illustrated below.



Change the entries in B_1, B_2, and B_3 to reflect this geometry as follows:

For B_1:

```
H0(1,1) = 33.6 A
  H0(1,2) = 0 A
  H0(1,3) = 0 A
```

For B_2:

```
  H0(2,1) = 0 A
H0(2,2) = 33.6 A
  H0(2,3) = 0 A
```

For B_3:

```
  H0(3,1) = 0 A
  H0(3,2) = 0 A
H0(3,3) = 33.6 A
```

Note that the units of length are Angstroms (Å), where $1Å=10^{-10}$m. These units are very convenient for working with atomic systems, e.g. the radius of a hydrogen atom is about 0.5Å and the typical nearest-neighbor distance between metal atoms in a solid is 2-3Å. We will always use Å unless otherwise stated.

Part C are the locations of all the atoms in the model. Each row stands for one atom. In our model, there will be 1000 atoms, so there should be 1000 rows in this section. Please, please, PLEASE do not type out all 1000 rows by hand! Write yourself a program or a script to write all those rows for you. I like to do this using MATLAB. To run MATLAB in your VNC session, issue the following two commands:

- ml Matlab/R2019a
- matlab &

The MATLAB GUI should open up in your VNC tab.

Each row has 8 entries. The first is the mass of the atom in atomic units. For Po, this is 209.0. The second entry is the chemical symbol of the atom, so in our case just "Po." The next three entries are the atom coordinates in *reduced units* (more on this below) and the last three entries are the three components of the atomic velocity vector, all of which we will set to zero. Thus, the entry for a Po atom located at coordinates (0.2,0.0,0.9) would be

```
209.0 Po 0.2 0.0 0.9 0 0 0
```

To compute the locations of all the atoms in our model, we simply replicate a simple cubic unit cell of Po 10x10x10 times. I will choose a unit cell description where the atom is located in a corner at the origin. Thus, the coordinates of my first atom are

(0.0Å, 0.0Å, 0.0Å)

I next create a replica of my unit cell displaced by one lattice parameter in the x-direction. Thus, the location of my second atom is

(3.36Å, 0.0Å, 0.0Å)

I continue to replicate the unit cell in the x-direction until I've put in 10 atoms. The coordinates of my 10th atom are

(30.24Å, 0.0Å, 0.0Å)

Notice that I *do not* put an atom at the far corner of the simulation cell, (33.6Å, 0.0Å, 0.0Å), because under periodic boundary conditions this location is identical to the location of the origin, (0.0Å, 0.0Å, 0.0Å)! If I had put an atom at the origin and at (33.6Å, 0.0Å, 0.0Å) then I would have created two atoms at the same location. This is a big no-no because it's unphysical. Suppose you tried to bring two real atomic nuclei so close together that they actually overlap. Since both nuclei are positively charged, the potential energy of the pair goes to infinity as you bring them closer and closer together. Thus, overlapping atoms are states of matter with "infinite energy." Later when we start doing molecular dynamics (MD) simulations, you may see that this kind of oversight generates error messages or, in some cases, makes the code crash.

After I've finished replicating the cubic unit cell 10 times in the x-direction, I advance the y-coordinate by 3.36Å and again replicate the unit cell 10 times in the x-direction. I continue to repeat this process until I've advanced my y-coordinate value to 30.24Å (not 33.6Å, for the same reasons as mentioned before!). Next I advance my z-coordinate by 3.36Å and continue replicating the unit cells until I've filled the simulation cell. In pseudo code, the process of creating all the atomic coordinates can be viewed as three nested loops:

```
a=3.36
N=0
for k=0:9
        for j=0:9
                for i=0:9
                        N=N+1
                        X(N)=i*a
                        Y(N)=j*a
                        Z(N)=k*a
                end
        end
end
```

This creates arrays X, Y, and Z of size 1000 containing all the atomic coordinates in Å units. Now we just have to convert these to coordinates in reduced units, (sx, sy, sz). Reduced units are defined using a matrix-vector product:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{H0} \cdot \begin{bmatrix} sx \\ sy \\ sz \end{bmatrix}$$

H0 is a 3x3 matrix whose columns are the entries in B_1, B_2, and B_3 (i.e. the vectors that define the edges of the simulation cell):

$$\mathbf{H0} = \begin{bmatrix} h0(1,1) & h0(2,1) & h0(3,1) \\ h0(1,2) & h0(2,2) & h0(3,2) \\ h0(1,3) & h0(2,3) & h0(3,3) \end{bmatrix}$$

Note that, unfortunately, the .cfg format does not use the usual convention for indexing matrix rows and columns. Instead, the first index (i) in H0(i,j) denotes the column while the second one (j) denotes the row. If we know the coordinate in Å, (X,Y,Z) and the matrix **H0**, we can compute the reduced coordinates as

$$\begin{bmatrix} sx \\ sy \\ sz \end{bmatrix} = \mathbf{H0^{-1}} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

where $\mathbf{H0^{-1}}$ is the inverse of $\mathbf{H0}$. In our structure,

$$\mathbf{H0} = \begin{bmatrix} 33.6 & 0 & 0 \\ 0 & 33.6 & 0 \\ 0 & 0 & 33.6 \end{bmatrix} \text{Å}$$

Thus,

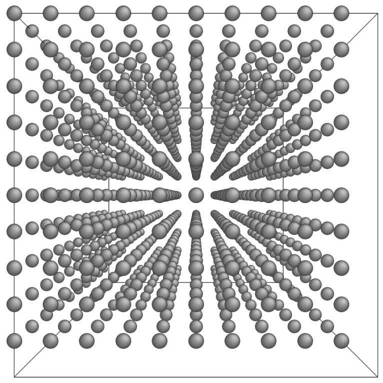
$$\mathbf{H0^{-1}} = \begin{bmatrix} 1/33.6 & 0 & 0 \\ 0 & 1/33.6 & 0 \\ 0 & 0 & 1/33.6 \end{bmatrix} / \text{ Å}$$

and

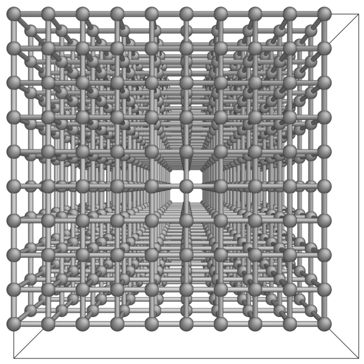
$$\begin{bmatrix} sx \\ sy \\ sz \end{bmatrix} = \begin{bmatrix} X/33.6 \\ Y/33.6 \\ Z/33.6 \end{bmatrix}.$$

Notice that all the reduced coordinates have values between zero and one. You can imagine the process of converting to reduced coordinates as the process of squeezing down your simulation cell into a perfect cube of unit dimensions in all directions. You are now ready to write the atomic positions to your .cfg file.

Finally, there is part D: delete this part and save your file. Now you're done. I have posted the completed structure file online so you can compare it with the file you created. You should now be able to open your file using AtomEye and see something like this (use the "page up" and "page down" buttons to adjust atom radii, if necessary):



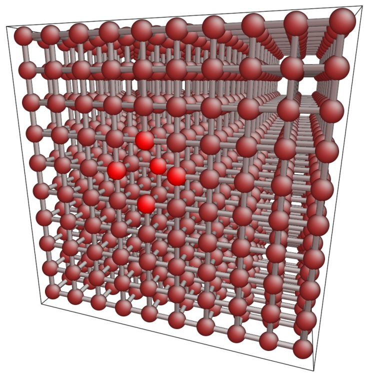
Now let’s tell AtomEye what is the nearest neighbor distance in this model. While in AtomEye, press “r.” Next, hold down “control” and repeatedly press the “home” button until the value of “rcut(A,A)” shown in the AtomEye text window exceeds 3.36, which is the interatomic distance in our model. Finally, press “r” again. Now that AtomEye knows the distance between nearest neighbors, it can show you the bonding between them. Press “b.” You should see this:



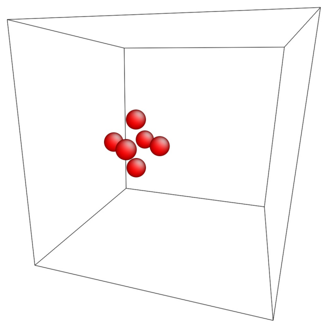
Notice how the bonds make cubic boxes, just as we would expect in a simple cubic structure. To convince yourself that this is indeed a quasi-infinite crystal under PBCs, move the atoms inside the simulation cell by dragging with the mouse while holding “shift.”

<u>Making a model of a vacancy</u>

Copy the file with your model of perfect crystal Po into a new file, which we will modify to create a vacancy. I named the new file “sc_Po-vac.cfg.” Open this new file, change the number of atoms from 1000 to 999. Delete the first row in part C. You now have a model of a vacancy in an otherwise perfect crystal of Po. Open the file with AtomEye. If the nearest neighbor distance in the program is set to a value greater than 3.36Å, then pressing "k" will give the same color to all atoms with a given number of nearest neighbors. The atoms next to a vacancy have one nearest neighbor less than all other atoms, so they will be colored differently from the other atoms, as shown below:



It's often easier to see defects if you suppress all the atoms with perfect crystalline environments. While holding control and shift, right click on one of the atoms with a perfect crystalline environment (red, in the picture above). This will make all perfect crystal atoms disappear and you'll only see the atoms neighboring the vacancy:



<u>Making a model of a free surface</u>

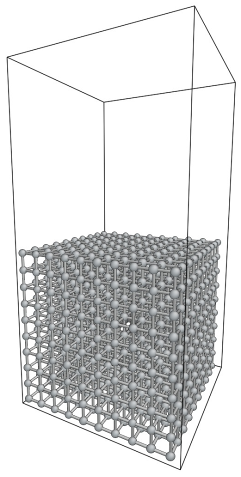
You can create a model of a free surface by increasing the size of the simulation cell in one direction while keeping all the real (not reduced!) atomic coordinates fixed. For example, suppose we double the simulation cell size in the z-direction. From the definition of how to compute reduced units, we get

$$\begin{bmatrix} sx \\ sy \\ sz \end{bmatrix} = \begin{bmatrix} 33.6 & 0 & 0 \\ 0 & 33.6 & 0 \\ 0 & 0 & 67.2 \end{bmatrix}^{-1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X/33.6 \\ Y/33.6 \\ Z/67.2 \end{bmatrix}$$

Thus, starting from a model of a perfect Po crystal, we have to do two things to create a free surface:

1. Double the value of H0(3,3) in part B_3
2. Halve the reduced z-coordinates of all the atoms in part C

You should get something like this (file "sc_Po-fs.cfg"):



Note that if you just did the first step and not the second step, this would stretch your crystal in the z-direction, but would not create a free surface. What would happen if you only did the second step and not the first step?

TUTORIAL 2 (by M. J. Demkowicz, all rights reserved)

COMPUTING ENERGIES OF ATOMIC STRUCTURES IN LAMMPS

We are often unable to make analytical calculations describing the behavior of complex atomic structures, such as defect clustering or vacancy-interstitial recombination. Atomistic modeling has been a very important source of information concerning such behaviors, especially since many of them are also very difficult to study experimentally. This tutorial will show how to make some basic calculations of defect properties using a freely available atomistic modeling code called LAMMPS.

Atomic interactions

The energy of a set of atoms $\{\vec{x}_i\}$ denoted by their locations, $\vec{x}_i$ (i=1…N, N=number of atoms), consists of the electrostatic interactions between the atomic nuclei, the interactions between the nuclei and the electrons, and the interaction between the electrons. The first of these terms can be computed simply by treating nuclei as classical point masses with charge. Quantum mechanics is required to treat the nucleus-electron and electron-electron interactions and this is a very challenging task in itself. Suppose, however, that we restrict ourselves purely to considering electrons in their ground state, i.e. we neglect any electronic excitations.

We know from quantum mechanics that for given nucleus locations (i.e., for given "external potential"), the electrons have a single, unique ground state with an associated energy, E_1. That means that in principle we can also write the electron ground state energy as a function of the nuclear coordinates: $E_1 = E_1\left(\{\vec{x}_i\}\right)$. In practice, writing down that dependence may be a very difficult thing to do, but the mere possibility of doing it provides the intellectual underpinning for a very popular method of modeling interatomic interactions: the classical potential method.

Since the nucleus-nucleus interaction energies can be written directly as a function of $\{\vec{x}_i\}$ and the nucleus-electron and electron-electron may be parameterized in terms of $\{\vec{x}_i\}$ (because the electronic ground state is unique for given $\{\vec{x}_i\}$), that means we can in fact write down the total ground state potential energy of an atomic system, V, as a function of the nuclear coordinates alone:

$$V = V\left(\{\vec{x}_i\}\right).$$

V behaves just like any other classical potential. For example, we may determine the force on any given atom j by taking the derivative of V with respect to the position of that atom:

$$\vec{f}_j = -\frac{\partial V\{\vec{x}_i\}}{\partial \vec{x}_j}.$$

Using classical potentials provides some advantages over first principles calculations because the computational cost of the former is much lower. For this reason, we will only conduct simulations using classical potentials in this section. Naturally, the quality of the calculations depends on the accuracy of the potentials. A cottage industry has grown up around the construction of classical potentials, which now range from the simplest Lennard-Jones pair potentials that describe noble gasses to elaborate potentials generated by genetic algorithms to describe chemically complex phenomena [see, e.g., ACT van Duin *et al.*, J. Phys. Chem. A **112**, 11414 (2008)].

Because there are no hard-and-fast rules for creating classical potentials or even for quantifying their uncertainty, these models have sometimes been referred to as "empirical" potentials. I find this description a little unfair since—if anything at all has become clear about potentials during the ~50 years that they have been in use—it is that the ones that stand the test of time were built directly upon some insight concerning the physics of bonding behavior in the system they intend to model. Prototypical examples are embedded atom method (EAM) potentials, which are probably the most widely used of all potentials and were motivated directly by density functional theory (DFT) [MS Daw, MI Baskes, Phys. Rev. B **29**, 6443 (1984)].

LAMMPS on HPRC

Please log into ada using a VNC session, just as we did before in the AtomEye tutorial. Once you're in, issue the following command:

- ml LAMMPS/24Oct2018-intel-2018b

This is all you need to do to have access to the LAMMPS executable in VNC. You may also be able to install LAMMPS on your own computer. See lammps.sandia.gov for details. I highly recommend perusing the LAMMPS website, whether or not you want to install it on your computer. LAMMPS is very well documented and there is a highly responsive user community. You can access both through the website.

Running an energy calculation

We will begin by computing the energy of a perfect fcc Cu structure. Create (anywhere you like) a new directory to hold the calculation (I used "fcc_Cu"). Copy into that directory the file "Cu_mishin1.eam.alloy," provided along with this tutorial. This file contains all the parameters of a popular EAM potential that describes the behavior of Cu. If you were to install LAMMPS on your own computer, then you would have at your disposal a directory called "potentials" that contains "Cu_mishin1.eam.alloy" as well as many other potentials for a variety of materials.

Also, please copy into your directory the files "in.lammps_energy" and "Cu_structure.data." The former contains a set of instructions that tells LAMMPS what to do and the latter is a model of perfect crystalline FCC Cu. Now give the following command to run LAMMPS:

```
lmp -in in.lammps_energy
```

If all goes well, then you should see several lines of output ending in "Dangerous builds = 0." The content of this output is also written to a file called "log.lammps." Have a look at what's inside this file and compare it to the output to the terminal to convince yourself that the two are identical.

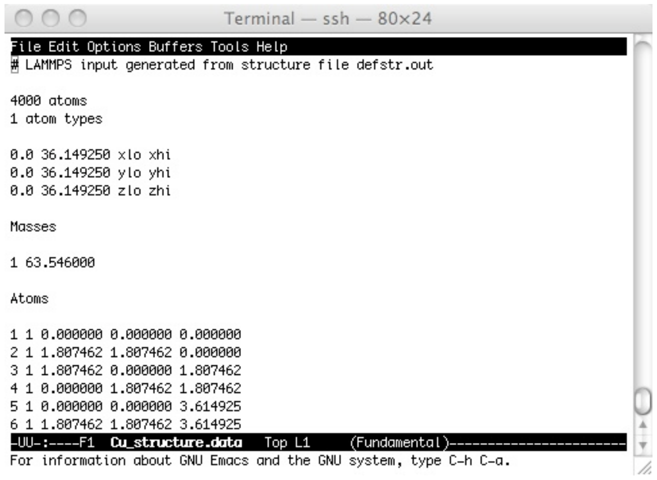
About midway through the output, you should see a line that reads

```
Step Atoms Temp Press Volume PotEng KinEng TotEng
```

Except for the first entry, "Step," each of these corresponds to a certain physical quantity characterizing the system that has been calculated by LAMMPS. The meanings should be fairly clear, e.g. "Temp" means temperature, "Press" is pressure, etc. "Step" tells you at which stage of the simulation the quantities were calculated (in a simple energy calculation, there is only one step, numbered zero).

The following line contains the actual numerical values computed by LAMMPS, given in the same order as the labels above. For example, under "Atoms" we have 4000, telling us that the atomic structure on which the calculation was done contains 4000 atoms. Under "PotEng" we find "-14160.873." The energy units in this calculation are eV. Thus, we can calculate that the energy per atom (the cohesive energy) of fcc Cu is E_{coh}=14160.873/4000eV=3.54eV.

Now, open the atomic structure file, "Cu_structure.data," using your favorite text editor. If I use emacs in the terminal window, here's what I see:



```
# LAMMPS input generated from structure file defstr.out

4000 atoms
1 atom types

0.0 36.149250 xlo xhi
0.0 36.149250 ylo yhi
0.0 36.149250 zlo zhi

Masses

1 63.546000

Atoms

1 1 0.000000 0.000000 0.000000
2 1 1.807462 1.807462 0.000000
3 1 1.807462 0.000000 1.807462
4 1 0.000000 1.807462 1.807462
5 1 0.000000 0.000000 3.614925
6 1 1.807462 1.807462 3.614925
-UU-:----F1  Cu_structure.data   Top L1     (Fundamental)----------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

As you can tell, this structure file is a little different from the one we made when using AtomEye, but it contains the same ingredients:

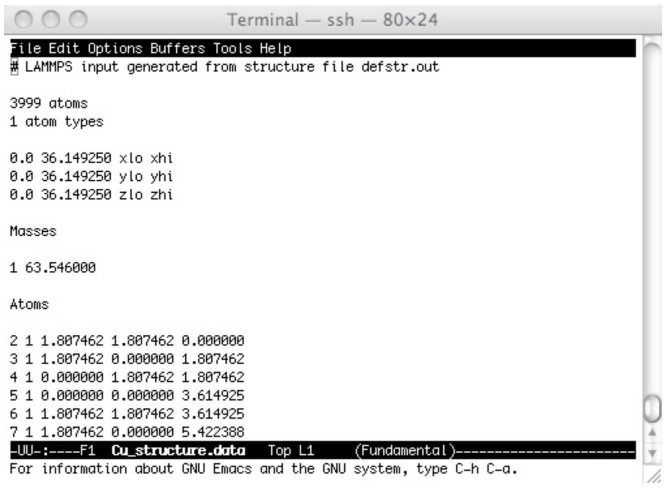
- The third line contains the number of atoms in the system (4000). This file also lists the number of atom types on line 4 (just one: Cu).
- Lines 6-8 contain the coordinates of the unit cube that defines the simulation cell. Notice that, unlike in the .cfg format, the simulation cell shape is not specified as a matrix of vectors. In fact, for this file format, LAMMPS assumes that the system is inside an orthorhombic cell, i.e. that the vectors that define

its edges are orthogonal to each other and aligned with the x-, y-, and z-axes. Thus, all you really need to specify is the length of each edge, which is what the lines here do. All lengths are in Å.

- Line 12 gives the atomic mass of Cu.
- Lines 16 and onward are quantities given for individual atoms. The first entry in each line is just the atom index (in this case, it goes from 1 to 4000 as you proceed down the column). The second is the atom type (here, all atoms are of the same type: 1, i.e. Cu). The third, fourth, and fifth entries are the x-, y-, and z-coordinates of the atom. Note that unlike in the .cfg format, all the coordinates are given in Å, not in reduced units!

Next, we will create some crystal defects and find their energies. I will initially work with two types of point defects: vacancies and self-interstitials atoms (SIAs). In case it's been a while since you've thought about them, I've included some slides along with this tutorial that will give you something of a crash course.

We learned in the last tutorial that you can create a vacancy by simply removing an atom. I do that by deleting row 16 (actually, it doesn't matter which row you delete, since the system is under periodic boundary conditions) and change the total number of atoms to 3999 in line three. Now my file looks like this:



```
# LAMMPS input generated from structure file defstr.out

3999 atoms
1 atom types

0.0 36.149250 xlo xhi
0.0 36.149250 ylo yhi
0.0 36.149250 zlo zhi

Masses

1 63.546000

Atoms

2 1 1.807462 1.807462 0.000000
3 1 1.807462 0.000000 1.807462
4 1 0.000000 1.807462 1.807462
5 1 0.000000 0.000000 3.614925
6 1 1.807462 1.807462 3.614925
7 1 1.807462 0.000000 5.422388
-UU-:----F1  Cu_structure.data   Top L1     (Fundamental)-----------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

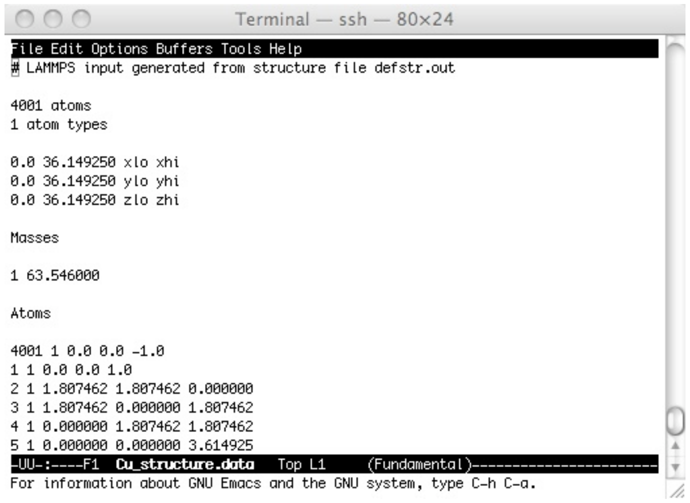
Keeping the name of the file unchanged (still "Cu_structure.data"), I recalculate the energy by running LAMMPS. Now I get an energy of -14156.024eV. Comparing this to the energy of perfect fcc Cu, I see that the energy has increased by ΔE=4.85eV. Is this the vacancy formation energy? No. The creation of a vacancy is the removal of an atom from a perfect crystal *and its placement on a free surface*. The latter part recovers some of the energy needed to break bonds during atom removal.

While we could explicitly model a free surface and put the extra atom on it, the effect of doing so is simply to recover an amount of energy equal to the cohesive energy of Cu. We already know from our first calculation that this is equal to 3.54 eV. Thus, the vacancy formation energy is:

$$\Delta E_v^f = \Delta E - 3.54eV = 1.31eV \; .$$

This compares well with the experimentally measured value, $\Delta E_v^f = 1.28eV$.

Similarly, we could introduce a SIA into the picture and calculate its energy. I created a [001] split dumbbell by adding two atoms to the vacant site from the last structure file. My file looks like this:



```
File Edit Options Buffers Tools Help
# LAMMPS input generated from structure file defstr.out

4001 atoms
1 atom types

0.0 36.149250 xlo xhi
0.0 36.149250 ylo yhi
0.0 36.149250 zlo zhi

Masses

1 63.546000

Atoms

4001 1 0.0 0.0 -1.0
1 1 0.0 0.0 1.0
2 1 1.807462 1.807462 0.000000
3 1 1.807462 0.000000 1.807462
4 1 0.000000 1.807462 1.807462
5 1 0.000000 0.000000 3.614925
-UU-:----F1  Cu_structure.data   Top L1     (Fundamental)-----------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

Note that my file now has 4001 atoms and I labeled the atom in the first row 4001. This is allowed: the atom indices do not have to be in any particular order. Also notice that I made the split dumbbell by having one atom displaced by -1Å in the z-direction from the location of the vacant size (which is at the origin) and the other displaced by 1Å in the z-direction. I have no way of knowing in advance that these are the right distances for the members of the dumbbell—I just do my best to guess. The next section will show how you can get the correct separation between these atoms.

I compute the energy of this system and compare it to the energy of a perfect crystal. Creating an interstitial means removing an atom from the surface, which costs energy. In fact, it costs exactly one cohesive energy. Thus, I find an interstitial formation energy of

$$\Delta E_i^f = \Delta E + 3.54eV = 9.11eV .$$

This value far exceeds the one determined by first principles calculations: $\Delta E_i^f = 3.23eV$. Could this be a flaw in the potential? Fortunately, in this case it is not. The high energy arises from the fact that the defect structure we created is quite far from being in mechanical equilibrium and is therefore in a high-energy state.

Running an energy minimization

The solution to the problem of a non-equilibrium atomic structure is to displace all the atoms such the energy of the system (as computed from the potential) reaches its nearest minimum. In this state, the first derivative of the potential with respect to any atomic coordinate is zero (this must be true because that's what defines being in a minimum!). Since the forces on atoms are proportional to the derivatives of the potential, an atom in a minimum energy state is also one where there is no net force acting on any atom. A popular way to minimize energies of atomic aggregates is the conjugate gradient method. You can learn more about it from DP Bertsekas, Nonlinear Programming, Athena Scientific, 1999.

To perform an energy minimization on the interstitial structure we constructed above, use the "in.lammps_minimize" provided with this tutorial and run

```
lmp -in in.lammps_minimize
```

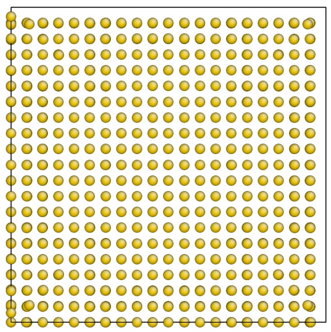
The simulation may now take a second or two to run and will generate several steps of output (probably about 50 or so). The last step gives you the final energy of the system. With this result, I compute the interstitial formation energy using the same formula as before and get

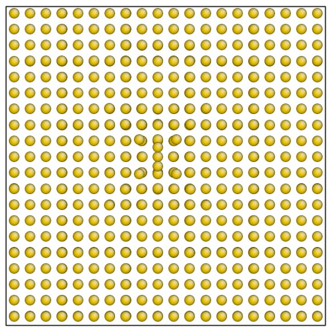
$$\Delta E_i^f = \Delta E + 3.54eV = 3.08eV,$$

which is much closer to the value determined by first principles (within acceptable error).

A similar calculation carried out on the vacancy causes its formation energy to be revised downward to $\Delta E_i^v = 1.27eV$, i.e. not by nearly as much as in the case of the interstitial. The reason is that the atoms around a vacancy relax inward only slightly (they are close to equilibrium to begin with). Interstitials, however, cause much greater elastic distortion in the crystal than do vacancies, so the one I created (which did not have any distortion associated with it) was actually initially much further from equilibrium. What do you think would happen if I ran energy minimization on a perfect (defect-free) fcc Cu crystal?

One of the things we get for free when we perform an energy minimization are the coordinates of the atoms in their lowest energy state. The LAMMPS minimization command I gave you outputs those coordinates in a file called "Cu_minimized.*.cfg," where "*" is replaced by the number of steps it took to minimize the energy of the system. The file is in the .cfg format that we already discussed. It may be visualized directly using AtomEye. Here are some AtomEye visualizations that I made of my relaxed [001] split dumbbell:





The picture of the left is what the system looks like when I start up the visualization. Since I added the interstitial close to the origin, it actually crosses over the periodic boundaries and is therefore hard to see clearly. If I move the system under periodic boundary conditions, I get the picture on the right, which shows the interstitial much more clearly.

<u>Other calculations using LAMMPS</u>

LAMMPS is capable of performing many other kinds of calculations besides energy minimization. The energy and energy minimization calculations, however, are good starting points, if you're new to LAMMPS. However, I certainly do encourage you to peruse the LAMMPS website, especially the "LAMMPS commands" section, which tells you about the content of the "in.lammps_*" files I gave you:

http://lammps.sandia.gov/doc/Section_commands.html#comm

You may also be interested in running some other LAMMPS sample problems. You can find a list at http://lammps.sandia.gov/doc/Section_example.html

The files you need to run them are in the “examples” directory, which comes with the LAMMPS package that you can download from the website.

TUTORIAL 3 (by M. J. Demkowicz, all rights reserved)

PERFORMING MOLECULAR DYNAMICS SIMULATIONS IN LAMMPS

Molecular dynamics (MD) is a type of atomistic simulation. In it, Newton's 2nd law (f=ma) is used to generate atomic trajectories as a function of time. To run MD simulations, you need to know the forces (f) acting on all the atoms. These can come from classical potentials or from first principles calculations, such as density functional theory (DFT).

Supposing you have all the atomic forces, you can get the atomic trajectories by solving the set of coupled ordinary differential equations (ODEs)

$$m_i \frac{\partial^2 \vec{x}_i}{\partial t^2} = \vec{f}_i$$

Here, $\vec{x}_i$ is a vector describing the position of atom i, $\vec{f}_i$ is the force on that atom, and m_i is the mass of the atom. If there are N atoms, then there are N equations of the type given above, all of which have to be solved simultaneously because they are coupled. The coupling between them comes from the fact that the force acting on any atom depend, in general, on the positions of all the other atoms.

In most cases of interest, the equations given above cannot be solved analytically. Thankfully, they are relatively easy to solve numerically. A description of some common methods used to solve these equations may be found in standard intro books on MD. My favorite if Allen and Tildesley's "Computer Simulation of Liquids" (Oxford, 2000). These techniques are not so difficult to learn. However, you don't have to become expert in them if you just want to get started on some simple MD simulations because they are already implemented in LAMMPS.

<u>MD simulation of thermal motion</u>

Let's run a simple MD simulation in LAMMPS where we just look at atoms moving around *via* thermal vibration at 300K. We will use the same initial structure and potential file as in the previous tutorial on energy minimization. To run the simulation, issue the following command:

```
lmp -in in.lammps_therm
```

This run should take a few seconds to complete. After it's done, you should have a bunch of *.cfg files in your directory. These are snapshots of the structure taken at increments of 10 time steps. The simulation is set up to run for 1000 time steps, so you should have 100 snapshots. By default, the timestep LAMMPS uses in metals is 1fs, so our simulation models 1ps of time for this atomic model.

Let's make a movie out of these files using AtomEye. To that end, we will use a file named "scr_anim" (provided). Create a new directory called "Str" and put all of your *.cfg files into it. Next, create another directory called "Pic." Now use AtomEye to open any of the *.cfg files in your Str directory. Orient it any way you like that helps you best see the crystal. Now press the button "y." AtomEye will run a movie where frames are the successive snapshots of our MD simulations. Simultaneously, AtomEye will also create *.jpg files of each frame in the Pic directory. You can convert these frames into a stand-alone movie file with MATLAB using the script "jpgmov.m," which I've also provided to you.

MD simulation of melting

Now let’s run an MD simulation of melting by issuing the following command:

```
lmp -in in.lammps_melt
```

This simulation will take even longer to run than the previous one because we are simulating a whopping 10ps of atomic trajectories. You can actually halve the amount of time it takes this simulation to run by running it in parallel mode using two processors:

```
mpirun –np 2 lmp -in in.lammps_melt
```

To use more processors, you would change the “2” in the above command to the number of processors you want to use. Note as you increase the number of processors, the speedup you get becomes smaller and smaller, so it doesn’t make sense to increase this number indefinitely. Rather, you want to find an optimal number that runs the code fast without using up too much computing resources (which are finite!). Also note that I’ve changed the snapshot frequency to once every 1000 time steps. You can use these to make a movie, but the movie would have only 10 frames and it would look very choppy because the structure is changing dramatically between frames.

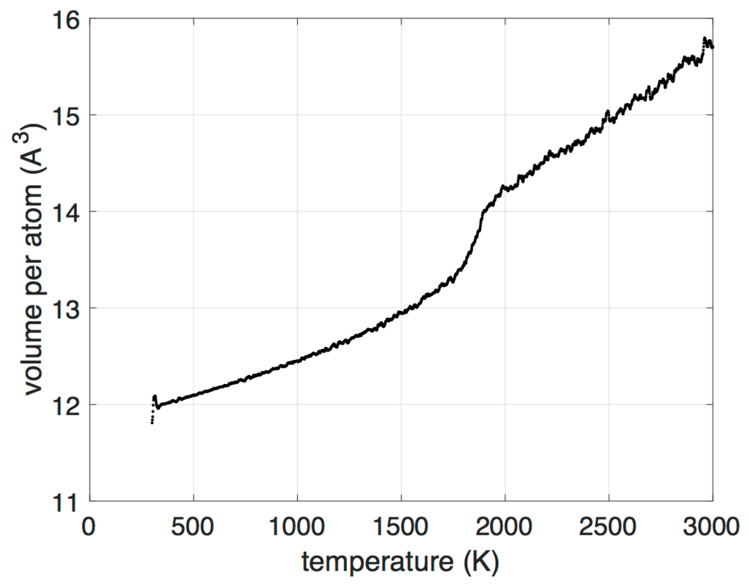
Let’s take a look at the log.lammps output file for this simulation. Look for the following line:

```
Step Atoms Temp Press Volume PotEng KinEng TotEng Pxx Pyy Pzz Pxy Pxz Pyz
```

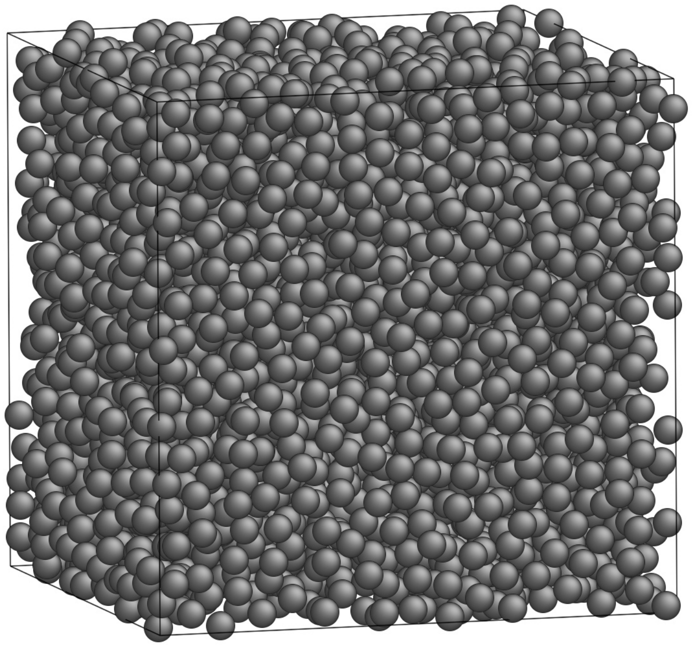
The entries below it give you all kinds of interesting physical info about the state of the system during the run:

- Step: number of timesteps so far
- Atoms: number of atom (constant, in our simulation)
- Temp: temperature (in K)
- Press: pressure (in bar)
- Volume: volume (in A^3)
- PotEng: total potential energy (in eV)
- KinEng: total kinetic energy (in eV)
- TotEng: total energy (sum of potential and kinetic):
- Pxx Pyy Pzz Pxy Pxz Pyz: the six independent components of the pressure tensor of the model

Let's plot these quantities. To that end, I suggest deleting all the text above and below output data and using the "dlmread" command in MATLAB to read in the file. You can try making lots of different plots, but the one I'd really like to draw your attention to is the plot of volume per atom vs. temperature, shown below. As you can see, the volume initially increases approximately linearly with temperature. This behavior is known as thermal expansion. Then, in a range of temperatures between about 1700K and 1900K, the volume expands rapidly. At even higher temperatures, the volume continues to expand, but now at a rate close to the one it had initially.



What happens during the temperature range where the volume expands rapidly? To find out, let's use AtomEye to visualize one of the higher temperature structures, say at 17000 time steps, as shown below.



This structure is clearly no longer crystalline. Indeed, it is liquid. The rapid rise in volume that we saw is just a consequence of the 1st-order phase change from solid to liquid (the liquid is less dense, so the volume expands). To be more rigorous in confirming that this is indeed Cu in its molten state, we should compute the atomic mobility and see that it is large (unlike in a solid, atoms in a liquid are not fixed to a lattice site and move around much more freely).

Challenge questions:

- The thermodynamic melting temperature of Cu is 1358K, so why do we see melting occurring in the much higher temperature range between 1700K and 1900K?
- Melting of crystalline Cu is a 1st-order phase transition, meaning that it is associated with a discontinuous change in intensive quantities, such as

density. Why, then does the change in volume between 1700K and 1900K appear to be gradual?

- How might you use this simulation to estimate the heat of fusion in Cu?