Title of the master thesis

by

Filip Henrik Larsen

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

May 2017

Abstract

This is an abstract text.

To someone

This is a dedication to my cat.

Acknowledgements

Flora Joelle Larsen
Anders Hafreager

Contents

Chapter 1

Introduction

Why is the subject of this thesis of any interest?
What is our take on the problem?
What do we hope to accomplish?
How will this be of any contribution to anything?
How is the thesis laid out?

Chapter 2

LAMMPS

LAMMPS stands for *Large-scale Atomic/Molecular Massively Parallel Simulator*. It is a classical molecular dynamics simulation code designed to run efficiently on parallel computers. It's development began in the mid 1990s at Sandia National Laboratories, with funding from the U.S. Department of Energy. It was a cooperative project between two DOE labs and three private companies. The development is still ongoing and contributions are revised thoroughly.

Today LAMMPS is an open-source code with extensive and user friendly documentation. This is one of the main reasons that we have chosen to use LAMMPS as opposed to other molecular dynamics software.

2.1 Installation

Installing LAMMPS is a fairly simple procedure if only the basic settings are needed.

2.1.1 Linux

Users with a Unix based OS may download the lammps distribution as a tarball from LAMMPS' download page[1] and then unpack it from the command line.

```
gunzip filename.tar.gz
tar xvf filename.tar
```

The user may then change directory into `/path/to/lammps/src/`, and execute the following commands in order to list available packages.

```
make package-status
```

Installing specific packages is accomplished as shown below.

[1] http://lammps.sandia.gov/download.html

```
make yes-molecule yes-manybody yes-python yes-rigid
```

The above example installs the packages *molecule, manybody, python and rigid.* Next, the user can build LAMMPS using either of the lines below. Assuming the user has MPI installed line 2 makes the resulting executable compatible with parallelization in MPI.

```
make serial
make mpi
```

At this point there should be an executable in the `/path/to/lammps/src/` directory named `lmp_serial` or `lmp_mpi`, depending on the previous choice. These are now ready to run. To use it one has to point to this file from the command line at every run. It may be practical to set up a symlink as follows shown below.

```
sudo ln -s /path/to/lammps/src/lmp_mpi
   /usr/local/bin/lmp_mpi
```

The executable is now available as `lmp_serial` or `lmp_mpi` from anywhere.

2.1.2 Mac OS X with Homebrew

Mac users can follow the procedure described above, however they may also install even easier using *Homebrew*[2].

```
brew tap homebrew/science
brew install lammps              # serial version
brew install lammps --with-mpi   # mpi support
```

Where the user obviously should choose either line 2 or line 3, depending on if the user wants MPI comparability. This will install an executable named "lammps", a python module named "lammps", and resources with standard packages. This is basically it. LAMMPS is now ready to run, however, not all packages are installed.

The location of the resources and available packages can be found using the following command.

```
brew info lammps
```

Specific packages are available as options, and may be installed with the following command.

```
brew install lammps --enable-manybody
```

In the example shown we installed the package manybody.

[2] http://brew.sh/

2.1.3 Windows

2.2 Running LAMMPS

Chapter 3

Setting up the system

We wish to construct a system consisting of two elements made out of silica: a slab and a sphere cap. In order to do this we need to generate the spacial position coordinates (x,y,z) of every single atom. Considering that we are making a system consisting of about 10^5 atoms, this is obviously not done manually. We have chosen to use a tool named *Moltemplate*[1], which is included in the LAMMPS distribution.

The main idea is to manually enter the coordinates of only the atoms in a unit cell of the material one wish to generate, and then simply copy this unit cell wherever desired. The software will shift the coordinates of the copied unit cell by the displacement from the original image. In addition it will generate files containing data such as which atoms they share bonds with, if any, and angles between such bonds.

3.1 Silica

Silica is a chemical compound also known as Silicon dioxide, having the chemical formula SiO_2. It has several polymorph structures, the most common being quartz, which is one of the most abundant minerals in the Earth's crust. Other polymorphs include cristobalite, tridymite, coesite and more.

For our purpose it is insignificant which one we choose. Once the material is melted, it is indifferent which configuration we started from, as long as the density is correct. In this project we will build the constituents of the system from a type of cristobalite named β-cristobalite. This is mainly because it has a simple structure and a cubical unit cell.

[1] http://www.moltemplate.org/index.html

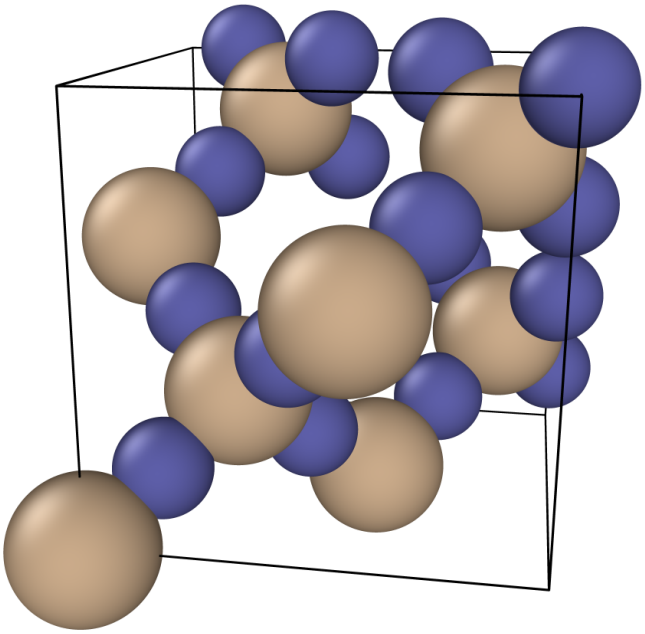

Figure 3.1: Unit cell of b-cristobalite. Tan and blue spheres represent silicon and oxygen atoms respectively.

3.1.1 Unit cell of β-cristobalite

In order to construct the unit cell of a material, one should look up the coordinates of the atoms in a crystallography database. We have used the unit cell of β-cristobalite found at *Crystallography Open Database*[2]. At this site one can download a `.cif`-file consisting of the spatial positions of each atom, the length of the unit cell edges and angles between faces of the cell. In the case of β-cristobalite the unit cell is cubical with edges of length 7.12Å. It contain 8 silicon atoms and 16 oxygen atoms. The density of the unit cell can easily be computed and is 2.2114 g/cm^3.

3.2 Building a crystal

The coordinates gotten from the `.cif`-file can now be implemented into *moltemplate* together with whatever bond and angle data required by the potential. In our simulations we will use the Vashishta potential, which does not require these.

Moltemplate has its own structure and syntax. The first step to build up

[2] http://www.crystallography.net/cod/1010944.html

a larger material is, as mentioned, to create the unit cell. Data concerning the unit cell are placed in a `.lt`-file, which is readable by Moltemplate. Such a file is shown in Listing 3.1.

For a more profound understanding of the structure and syntax of these files, the reader is advised to read the moltemplate manual

```
# file "beta-cristobalite.lt"

beta-cristobalite {
  write("Data Atoms") {
    $atom:Si1    @atom:Si    0.00    0.00    0.00
    $atom:Si2    @atom:Si    0.00    3.56    3.56
    $atom:Si3    @atom:Si    1.78    1.78    1.78
    $atom:Si4    @atom:Si    3.56    0.00    3.56
    $atom:Si5    @atom:Si    1.78    5.34    5.34
    $atom:Si6    @atom:Si    5.34    5.34    1.78
    $atom:Si7    @atom:Si    3.56    3.56    0.00
    $atom:Si8    @atom:Si    5.34    1.78    5.34
    $atom:O1     @atom:O     0.89    0.89    0.89
    $atom:O2     @atom:O     6.23    4.45    2.67
    $atom:O3     @atom:O     2.67    2.67    0.89
    $atom:O4     @atom:O     4.45    0.89    4.45
    $atom:O5     @atom:O     0.89    4.45    4.45
    $atom:O6     @atom:O     4.45    4.45    0.89
    $atom:O7     @atom:O     2.67    6.23    4.45
    $atom:O8     @atom:O     2.67    0.89    2.67
    $atom:O9     @atom:O     4.45    2.67    6.23
    $atom:O10    @atom:O     6.23    2.67    4.45
    $atom:O11    @atom:O     2.67    4.45    6.23
    $atom:O12    @atom:O     0.89    6.23    6.23
    $atom:O13    @atom:O     0.89    2.67    2.67
    $atom:O14    @atom:O     4.45    6.23    2.67
    $atom:O15    @atom:O     6.23    6.23    0.89
    $atom:O16    @atom:O     6.23    0.89    6.23
  }

  write_once("Data Masses") {
    @atom:Si 28.0855
    @atom:O  15.9994
  }

} # end definition of beta-cristobalite molecule type
```

Listing 3.1: Typical moltemplate file containing unit cell data. The columns of the "Data Atoms" section hold, from left to right, information of atom ID, atom type, x-, y- and z-position. The "Data Masses" section stores the weight of silicon and oxygen atoms in atomic mass units.

We use the unit cell as building blocks, placing them concurrently until we

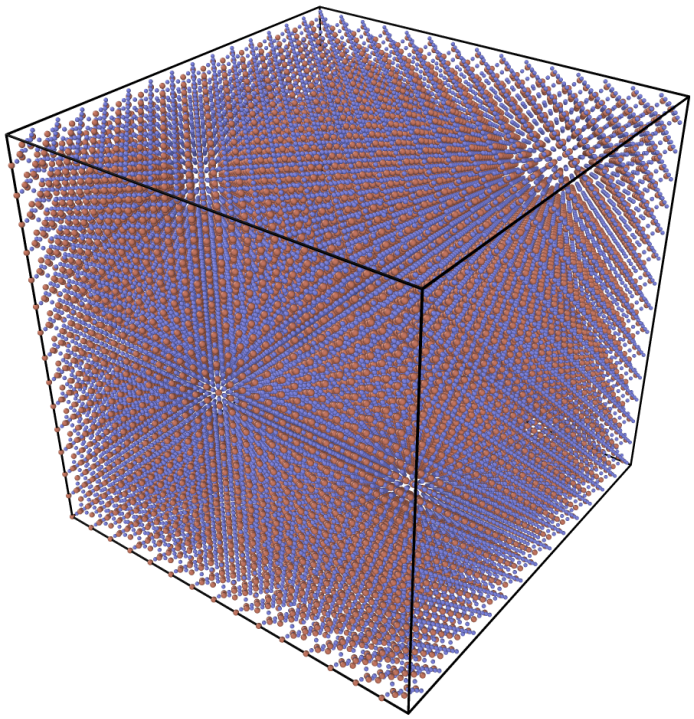

Figure 3.2: System built from $15 \times 15 \times 15$ unit cells of b-cristobalite.

have a crystal of the desired size. For our purpose, we generate a large cube of $15 \times 15 \times 15$ unit cells. This is done as follows.

3.3 Shaping the silica

The huge cube of silica can be carved however we like by defining regions from which we delete the containing atoms. In LAMMPS this is done by using the `region`, `union`, `intersect` and `delete atoms` commands. Our implementation is stated in Listing 3.2, which is very simple due to the way we are going to treat the boundary conditions.

```
region sphereRegion sphere 53.4 53.4 226.8 150
group  sphereGroup region sphereRegion

region slabRegion block 0 INF 0 INF 0 35.6
group  slabGroup region slabRegion

region bothRegion union 2 sphereRegion slabRegion side out
delete_atoms region bothRegion
```

Listing 3.2: Defining regions to keep or delete from a system of dimensions $106.8 \times 106.8 \times 106.8$ Å.

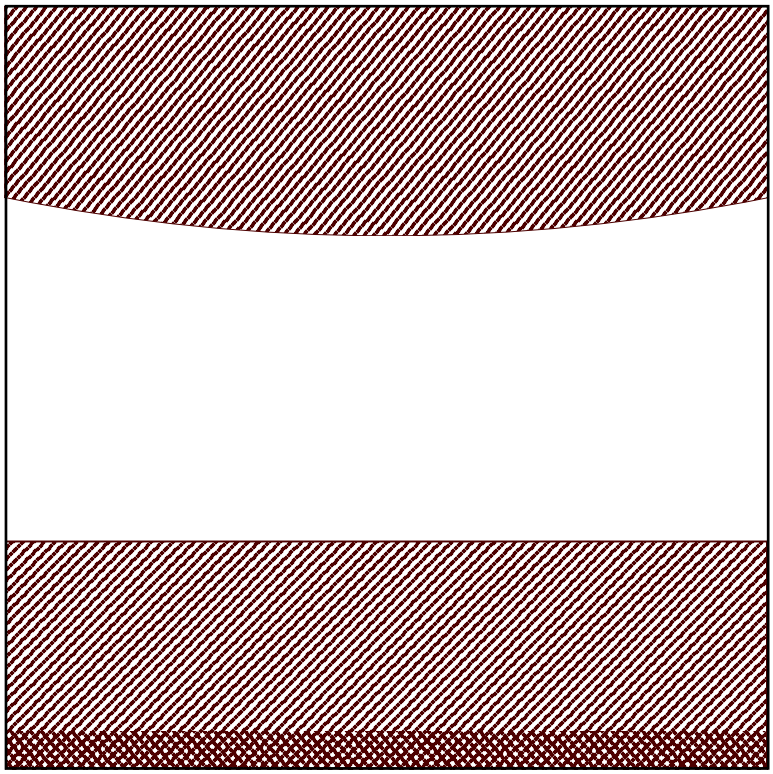

Figure 3.3: Illustrative drawing of what how the system should look. Red parallel stripes symbolize areas of silica. Red crossing stripes indicate areas of frozen silica. The boundaries are periodic in all dimensions, causing both the slab and the sphere to be connected to the frozen silica through the z-boundaries.

Chapter 4

This must be sorted in designated chapters

4.1 Radial distribution of normal force

In order to find a radial distribution of the normal force, F_N, we partition the system into a grid in the xy-plane. We then use the command

```
1   compute chunkID all chunk/atom bin/2d x 0 7.12 y 0 7.12
2   compute stressID all stress/atom NULL
3   fix fixChunkID all ave/chunk 1 1 10 chunkID
       c_stressID[3] file forcesInChunks.txt
```

to compute the stress of every chunk in the z-direction, σ_{zz} (sum of every individual atom stress in the chunk).

Line 1 establishes the grid, with bin width 7.12Å.

Line 2 creates a compute of the stress

Line 3 stores the sum of individual stresses in each chunk to the file `forcesInChunks.txt`. This is done every 10 time steps in order to reduce correlation effects.

The data in stored from each time step can easily be averaged to produce a result as shown in figure X.

We can then find the radial distribution simply by binning this matrix in radial bins, and average the normal forces of the chunks within the bins.

Bibliography

[1] Bin Luan and Mark O Robbins. The breakdown of continuum models for mechanical contacts. *Nature*, 435(7044):929–932, 2005.