

LCC

1.0.0.

Generated by Doxygen 1.8.17

1 LCC	1
2 Input file choices	5
3 LCC	9
4 LCC DOCUMENTATION	13
5 Building regular shapes	15

Chapter 1

LCC

About

Los Alamos Crystal Cut (LCC) is simple crystal builder. It is an easy-to-use and easy-to-develop code to make crystal solid/shape and slabs from a crystal lattice. Provided you have a '.pdb' file containing your lattice basis you can create a solid or slab from command line. The core developer of this code is Christian Negre (cnegre@lanl.gov).

License

© 2022. Triad National Security, LLC. All rights reserved. This program was produced under U.S. Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the U.S. Department of Energy/National Nuclear Security Administration. All rights in the program are reserved by Triad National Security, LLC, and the U.S. Department of Energy/National Nuclear Security Administration. The Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this material to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

This program is open source under the BSD-3 License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Requirements

In order to follow this tutorial, we will assume that the reader have a `LINUX` or `MAC` operative system with the following packages properly installed:

- The `git` program for cloning the codes.
- A `C/C++` compiler (`gcc` and `g++` for example)
- A `Fortran` compiler (`gfortran` for example)
- The `LAPACK` and `BLAS` libraries (GNU `libblas` and `liblapack` for example)
- The `python` interpreter (not essential).
- The `pkgconfig` and `cmake` programs (not essential).

On an `x86_64` GNU/Linux Ubuntu 16.04 distribution the commands to be typed are the following:

```
$ sudo apt-get update
$ sudo apt-get --yes --force-yes install gfortran gcc g++
$ sudo apt-get --yes --force-yes install libblas-dev liblapack-dev
$ sudo apt-get --yes --force-yes install cmake pkg-config cmake-data
$ sudo apt-get --yes --force-yes install git python
```

NOTE: Through the course of this tutorial we will assume that the follower will work and install the programs in the home directory (`$HOME`).

Download and installation

We will need to clone the repository as follows:

```
$ cd; git@github.com:cnegre/ClusterGen.git
```

Compiling **PROGRESS** and **BML** libraries

The LCC code needs to be compiled with both `PROGRESS` and `BML` libraries. In this section we will explain how to install both of these libraries and link the code against them.

Scripts for quick installations can be found in the main folder. In principle one should be able to install everything by typing:

```
$ ./clone_libs.sh
$ ./build_bml.sh
$ ./build_progress.sh
$ ./build.sh
```

Which will also build LCC with its binary file in `./src/lcc_main`.

Step-by-step install

Clone the BML library (in your home directory) by doing^[^1]:

```
$ cd
$ git clone git@github.com:lanl/bml.git
```

Take a look at the `./scripts/example_build.sh` file which has a set of instructions for configuring. Configure the installation by copying the script into the main folder and run it:

```
$ cp ./scripts/example_build.sh .
$ sh example_build.sh
```

The `build.sh` script is called and the installation is configured by creating the `build` directory. Go into the build directory and type:

```
$ cd build
$ make -j
$ make install
```

To ensure bml is installed correctly type `$ make tests` or `$ make test ARGS="-V"` to see details of the output. Series of tests results should follow.

After BML is installed, return to your home folder and “clone” the PROGRESS repository. To do this type:

```
$ cd
$ git clone git@github.com:lanl/progress.git
```

Once the folder is cloned, cd into that folder and use the `example_build.sh` file to configure the installation by following the same steps as for the bml library.

```
$ sh example_build.sh
$ cd build
$ make; make install
```

You can test the installation by typing `$ make tests` in the same way as it is done for BML.

LCC

Open the `Makefile` file in the `lcc/src` folder make sure the path to both bml and progress libs are set correctly. NOTE: Sometimes, depending on the architecture the libraries are installed in `/lib64` instead of `/lib`. After the aforementioned changes are done to the `Makefile` file proceed compiling with the “make” command.

Contributors

Christian Negre, email: cnegre@lanl.gov

Andrew Alvarado, email: aalvarado@lanl.gov

^[^1]: In order to have access to the repository you should have a github account and make sure to add your public ssh key is added in the configuration windows of github account.

Contributing

Formally request to be added as a collaborator to the project by sending an email to cnegre@lanl.gov. After being added to the project do the followig:

- Create a new branch with a proper name that can identify the new feature (`git checkout -b "my_new_branch"`)
- Make the changes or add your contributions to the new branch (`git add newFile.F90 modifiedFile.F90`)
- Make sure the tests are passing (`cd tests ; ./run_test.sh`)
- Commit the changes with proper commit messages (`git commit -m "Adding a my new contribution"`)
- Push the new branch to the repository (`git push`)
- Go to repository on the github website and click on "create pull request"

Chapter 2

Input file choices

In this section we will describe the input file keywords. Every valid keyword will use "cammel" syntax and will have and = sign right next to it. For example, the following is a valid keyword syntax `JobName= MyJob`. Comments need to have a # (hash) sign right next to the phrase we want to comment. Example comment could be something like: `#My comment`.

JobName=

This variable will indicate the name of the job we are running. It is just a tag to distinguish different outputs. As we mentioned before and example use could be: `JobName= MyJob`

Verbose=

Controls the verbosity level of the output. If set to 0 no output is printed out. If set to 1, only basic messages of the current execution point of the code will be printed. If set to 2, information about basic quantities are also printed. If set to 3, all relevant possible info is printed.

CoordsOutFile=

This will store the name of the output coordinates files. Basically if `CoordsOutFile= coords` two output files will be created: `coords.xyz` and `coords.pdb`.

PrintCml=

By setting `PrintCml= T` will also print create `coords.cml` which can be read by avogadro. In order to have this option working one needs to install **openbabel**. In order to read a cml file one needs to have **avogadro** installed. On gnu linux:

```
sudo apt-get install avogadro
sudo apt-get install openbabel
```

ClusterType=

This variable will define the type of shape/cluster/slab we want to construct. There are many options including Bulk, Planes, Bravais and Spheroid. We will explain all these in the following section.

ClusterType= Bulk

This will just cut a "piece of bulk" by indicating how many lattice point we want. For example, the following will create a bulk/lattice with 50 points on each a,b,c direction.

```
ClusterType= Bulk
LatticePoints= 50
```

The following, instead, will create a bulk/lattice with 100 lattice points in the x direction and 50 on the rest.

```
LatticePointsX1= 1
LatticePointsX2= 100
LatticePointsY1= 1
LatticePointsY2= 50
LatticePointsZ1= 1
LatticePointsZ2= 50
```

ClusterType= Spheroid

This will produce a "spheroid" center at the origin. And example follows:

```
ClusterType= Spherid
LatticePoints= 50 #This is necessary to construct the initial bulk
AAxis= 1.0 #Radius in direction x
BAxis= 2.0 #Radius in direction y
CAxis= 2.0 #Radius in direction z
```

See section REGULAR to see another example.

ClusterType= Planes

This will cut a shape using Miller indice. This is an important tool to construct a slab to study a surface. The cut does not guarantee periodicity. In order to have a periodic structure different plane boundaries need to be tried and the structures needs to be checked using a molecular visualizer. An example is given as follows:

```
NumberOfPlanes= 6
Planes[
  0 1 1 2.5
  0 -1 -1 1.5
  0 -1 1 4.5
  0 1 -1 3.5
  1 0 0 4.5
  -1 0 0 3.5
]
```

Three first number on each row indicate the Miller indices. The fourth number indicates how many Miller planes from the origin will be cut out. If the number of planes is 6, then the system tries to get the slab periodicity vectors since if the Miller planes are orthogonal to each other, the shape will be a "Parallelepiped". If instead the number different than 6, then the periodicity vectors are given by the "Boundaries" of the minimal box that contains the shape.

CenterAtBox=

If set to T, the shape will be centered at the box (the periodicity vectors of the shape/cluster)

Reorient=

If set to `T` this, will reorient the shape, such that vector "a" will be aligned with the x direction. This is important when making slabs needed to study a surface.

AtomType=

This will set the atom symbol if the lattice basis is not read from file.

TypeOfLattice=

This will set the Lattice unit cell. If set to `SC` or `FCC` either a simple cubic or face centered cubic lattice is built provided we set `LatticeConstanta=` to the lattice constant value. For general unit cell we can set `TypeOfLattice= Triclinic`, and provide the lattice parameters as in the following example:

```
LatticeConstanta= 6.5329400000000000
LatticeConstantb= 11.0221000000000000
LatticeConstantc= 7.3568800000000003
LatticeAngleAlpha= 90.0000000000000000
LatticeAngleBeta= 102.6520000000000000
LatticeAngleGamma= 90.0000000000000000
```

RandomSeed=

To generate random positions in the lattice. This will need to be used in conjunction with `RCoeff=` which controls the degree of deviation from the lattice positions.

PrimitiveFormat=

This will indicate if the lattice needs to be constructed out of a,b,c and angle parameter or primitive lattice vectors. If `PrimitiveFormat= Angles` (default), then the lattice parameters will need to be passed as in the following example:

```
LatticeConstanta= 6.5329400000000000
LatticeConstantb= 11.0221000000000000
LatticeConstantc= 7.3568800000000003
LatticeAngleAlpha= 90.0000000000000000
LatticeAngleBeta= 102.6520000000000000
LatticeAngleGamma= 90.0000000000000000
```

If instead, `PrimitiveFormat= Vectors` then the primitive vectors will need to be passed as in the following example:

```
LatticeVectors[
  2.0 0 0      #First lattice vector
  0.0 2.0 0
  0.0 2.0 2.0
]
```

UseLatticeBase=

This is an important tool that allows us to "dress" every lattice point with a basis of choice. The basis is defined to be the minimal set of coordinates and atom types needed to define a crystal system lattice point. The basis here will be read from file by providing the latticebase `LatticeBaseFile=` which will contain our atom types and coordinates. If `ReadLatticeFromFile=` is set to T, then, the lattice parameters will be read from the lattice basis file. If it is set to F, the the lattice parameters will need to be passed as explained before. Another important keyword is the `BaseFormat=`. If this is set to `abc`, then the basis coordinates stored in the file are assumed to be given in fractional coordinates of the lattice parameters. If it is set to `xyz`, then it will be assumed to be given in cartesian coordinates.

SymmetryOperations=

If the basis needs to be constricted from symmetry operation, then one needs to pass all these operations to the code as follows:

```
SymmetryOperations= T
NumberOfOperations= 4
Translations[
  0 0 0 0.0
  1 1 1 0.5
  1 1 0 1.0
  -0.5 1.5 0.5 1.0
]
Symmetries[
  0 0 0
  -1 1 -1
  -1 -1 -1
  1 -1 1
]
```

The first block indicates the "translations" within the unit cell. The first three rows indicate the directions of the translation and the fourth indicates the intensity. The second block indicates the symmetry of operations. For example, if an operation is indicated as $(-x + 1/2, -y, -z)$ then there will be a translation `0.5 0 0 1.0` and a symmetry `-1 0 0`.

```
NumberOfOperations= 0 MaxCoordination= 1 NumberOfIterations= 1 Truncation= 1.0000000000000000E+040
RCut= 20.000000000000000
RTol= 1.0000000000000000E-002 CutAfterAddingBase=F
SeedFile=seed.pdb
```

Chapter 3

LCC

About

Los Alamos Crystal Cut (LCC) is simple crystal builder. It is an easy-to-use and easy-to-develop code to make crystal solid/shape and slabs from a crystal lattice. Provided you have a '.pdb' file containing your lattice basis you can create a solid or slab from command line. The core developer of this code is Christian Negre (cnegre@lanl.gov).

License

© 2022. Triad National Security, LLC. All rights reserved. This program was produced under U.S. Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the U.S. Department of Energy/National Nuclear Security Administration. All rights in the program are reserved by Triad National Security, LLC, and the U.S. Department of Energy/National Nuclear Security Administration. The Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this material to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

This program is open source under the BSD-3 License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Requirements

In order to follow this tutorial, we will assume that the reader have a `LINUX` or `MAC` operative system with the following packages properly installed:

- The `git` program for cloning the codes.
- A `C/C++` compiler (`gcc` and `g++` for example)
- A `Fortran` compiler (`gfortran` for example)
- The `LAPACK` and `BLAS` libraries (GNU `libblas` and `liblapack` for example)
- The `python` interpreter (not essential).
- The `pkgconfig` and `cmake` programs (not essential).

On an `x86_64` GNU/Linux Ubuntu 16.04 distribution the commands to be typed are the following:

```
$ sudo apt-get update
$ sudo apt-get --yes --force-yes install gfortran gcc g++
$ sudo apt-get --yes --force-yes install libblas-dev liblapack-dev
$ sudo apt-get --yes --force-yes install cmake pkg-config cmake-data
$ sudo apt-get --yes --force-yes install git python
```

NOTE: Through the course of this tutorial we will assume that the follower will work and install the programs in the home directory (`$HOME`).

Download and installation

We will need to clone the repository as follows:

```
$ cd; git@github.com:cnegre/ClusterGen.git
```

Compiling **PROGRESS** and **BML** libraries

The LCC code needs to be compiled with both `PROGRESS` and `BML` libraries. In this section we will explain how to install both of these libraries and link the code against them.

Scripts for quick installations can be found in the main folder. In principle one should be able to install everything by typing:

```
$ ./clone_libs.sh
$ ./build_bml.sh
$ ./build_progress.sh
$ ./build.sh
```

Which will also build LCC with its binary file in `./src/lcc_main`.

Step-by-step install

Clone the BML library (in your home directory) by doing^[^1]:

```
$ cd
$ git clone git@github.com:lanl/bml.git
```

Take a look at the `./scripts/example_build.sh` file which has a set of instructions for configuring. Configure the installation by copying the script into the main folder and run it:

```
$ cp ./scripts/example_build.sh .
$ sh example_build.sh
```

The `build.sh` script is called and the installation is configured by creating the `build` directory. Go into the build directory and type:

```
$ cd build
$ make -j
$ make install
```

To ensure bml is installed correctly type `$ make tests` or `$ make test ARGS="-V"` to see details of the output. Series of tests results should follow.

After BML is installed, return to your home folder and “clone” the PROGRESS repository. To do this type:

```
$ cd
$ git clone git@github.com:lanl/progress.git
```

Once the folder is cloned, cd into that folder and use the `example_build.sh` file to configure the installation by following the same steps as for the bml library.

```
$ sh example_build.sh
$ cd build
$ make; make install
```

You can test the installation by typing `$ make tests` in the same way as it is done for BML.

LCC

Open the `Makefile` file in the `lcc/src` folder make sure the path to both bml and progress libs are set correctly. NOTE: Sometimes, depending on the architecture the libraries are installed in `/lib64` instead of `/lib`. After the aforementioned changes are done to the `Makefile` file proceed compiling with the “make” command.

Contributors

Christian Negre, email: cnegre@lanl.gov

Andrew Alvarado, email: aalvarado@lanl.gov

^[^1]: In order to have access to the repository you should have a github account and make sure to add your public ssh key is added in the configuration windows of github account.

Contributing

Formally request to be added as a collaborator to the project by sending an email to cnegre@lanl.gov. After being added to the project do the followig:

- Create a new branch with a proper name that can identify the new feature (`git checkout -b "my_new_branch"`)
- Make the changes or add your contributions to the new branch (`git add newFile.F90 modifiedFile.F90`)
- Make sure the tests are passing (`cd tests ; ./run_test.sh`)
- Commit the changes with proper commit messages (`git commit -m "Adding a my new contribution"`)
- Push the new branch to the repository (`git push`)
- Go to repository on the github website and click on "create pull request"

Chapter 4

LCC DOCUMENTATION

The folder (`src/docs`) contains all the documentation relevant to both users and developers.

Prerequisites

- **pdflatex**
Latex GNU compiler. pdfTeX is an extension of TeX which can produce PDF directly from TeX source, as well as original DVI files. pdfTeX incorporates the e-TeX extensions.
- **doxygen**
Doxygen is a documentation system for C++, C, Java, Objective-C, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D.
- **sphinx**
Sphinx is a documentation generator or a tool that translates a set of plain text source files into various output formats, automatically producing cross-references, indices, etc. That is, if you have a directory containing a bunch of reStructuredText or Markdown documents, Sphinx can generate a series of HTML files, a PDF file (via LaTeX), man pages and much more.
- Any pdf viewer.
- Any web browser.

These programs can be installed as follows:

```
sudo apt-get install pdflatex
sudo apt-get install doxygen
sudo apt-get install dot2tex
sudo apt-get install dot2tex
sudo apt-get install python3-sphinx
pip3 install PSphinxTheme
pip3 install recommonmark
```

Build the full documentation

This will build all three types of docs (Sphinx, Doxygen, and latex)

```
make
```

The documentation that is build with Sphinx can be tested as follows:

```
firefox ccl.html
```

The file can be explored using any web browser.

One can also build any of the documentations separatly. For example, to build the Sphinx documentation, we can do:

```
make sphinx
```

Documenting

In order to add a documentation using Sphinx follow these steps: 1) make a file with a proper name under `./sphinx-src/source/`. For example: `MYPAGE.md`. 2) Add the documentation inside the file using "mark-down" syntax. 3) Modify the file in `./sphinx-src/source/index.txt` to include the documentation just as shown in the following example:

```
.. toctree::
    :maxdepth: 2
    :caption: Contents:
    ./README-main
    ./README
    ./MYPAGE
```

After modifying this file, recompile Sphinx by typing `make sphinx`.

Chapter 5

Building regular shapes

One can also build regular shapes, such as for example a "spheroid." The parameters to do this can be entered as follows:

```
#Lcc input file.
LCC{
  JobName=                AgSpheroid          #Or any other name
  TypeOfLattice=          FCC
  LatticePoints=          50                  #Number of point in each direction
  LatticeConstanta=       4.08
  AtomType=               Ag
  ClusterType=            Spheroid
  AAxis=                  10                  #Radius in Ang for x direction
  BAxis=                  10                  #Radius in Ang for y direction
  CAxis=                  5                   #Radius in Ang for z direction
}
```

This will produce the following "oblate":

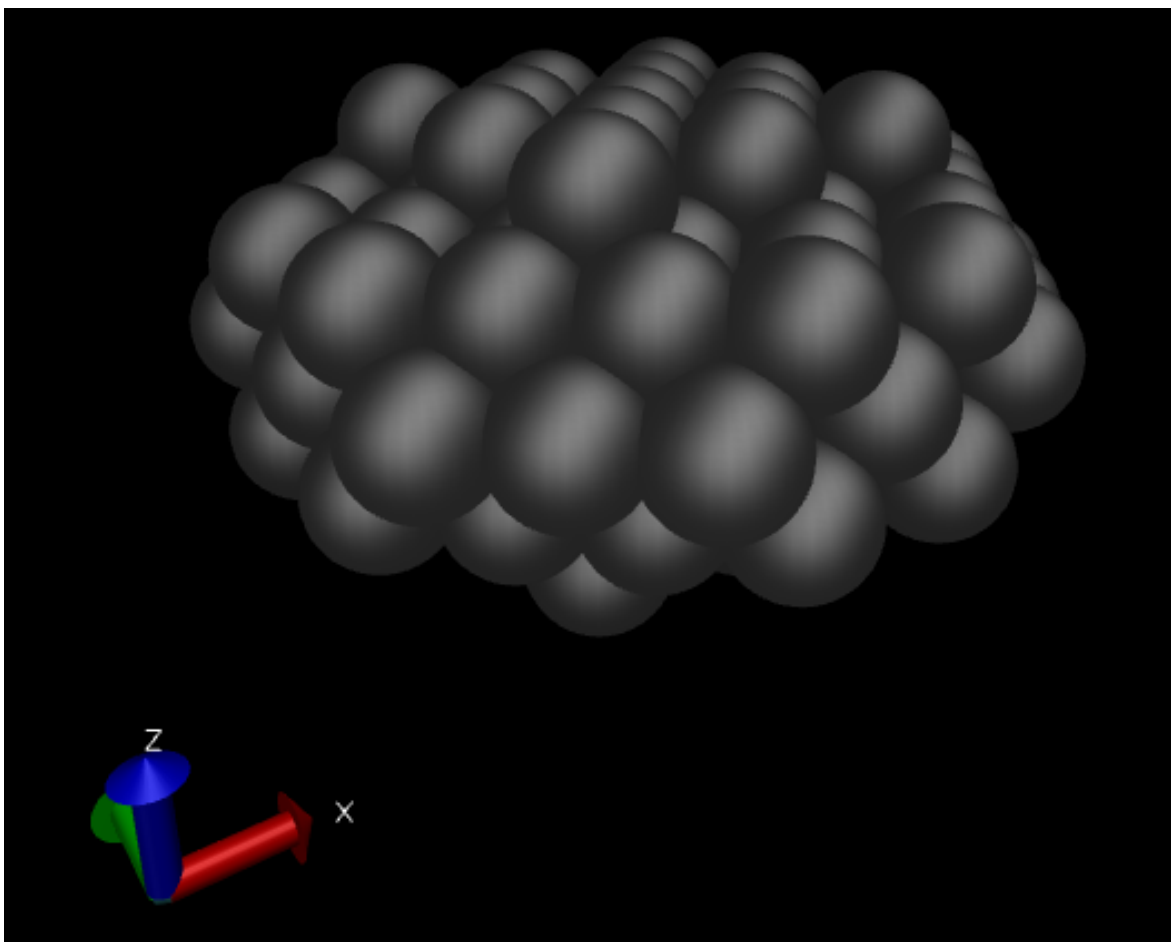


Figure 5.1 Oblate shape