lcc Documentation

Release 1.0.0

Christian F. A. Negre

Jul 13, 2023

CONTENTS:

1 LCC **3**
- 1.1 About . . . 3
- 1.2 License . . . 3
- 1.3 Requirements . . . 4
- 1.4 Quick installation using *spack* . . . 4
- 1.5 Download and installation . . . 5
- 1.6 Compiling PROGRESS and BML libraries . . . 5
- 1.7 Step-by-step installation . . . 5
- 1.8 Testing the code . . . 6
- 1.9 Quick example run . . . 6
- 1.10 Contributors . . . 7
- 1.11 Contributing . . . 7
- 1.12 Citing . . . 7

2 Building LCC documentation **9**
- 2.1 Prerequisites . . . 9
- 2.2 Build the full documentation . . . 9
- 2.3 Documenting . . . 10

3 Input file choices **11**
- 3.1 *JobName=* . . . 11
- 3.2 *Verbose=* . . . 11
- 3.3 *CoordsOutFile=* . . . 11
- 3.4 *PrintCml=* . . . 11
- 3.5 *ClusterType=* . . . 12
- 3.6 *ClusterType= Bulk* . . . 12
- 3.7 *ClusterType= Spheroid* . . . 12
- 3.8 *ClusterType= Planes* . . . 12
- 3.9 *CenterAtBox=* . . . 13
- 3.10 *Reorient=* . . . 13
- 3.11 *AtomType=* . . . 13
- 3.12 *TypeOfLattice=* . . . 13
- 3.13 *RandomSeed=* . . . 13
- 3.14 *PrimitiveFormat=* . . . 14
- 3.15 *UseLatticeBase=* . . . 14
- 3.16 *SymmetryOperations=* . . . 14

4 Building a Lattice **17**

5 Building/cutting a shape **21**

5.1 Growing shapes from a seed file 21
5.2 Cutting using planes 22

6 Building a slab **25**
6.1 Building a slab from three PBC vectors 29

7 Building regular shapes **31**

8 Computing Roughness **33**

9 Indices and tables **39**

Issues	Pull Requests	CI

CHAPTER

ONE

LCC

1.1 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).

1.2 License

1.3 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`).

1.4 Quick installation using *spack*

Clone and setup the spack code:

```
cd ~

git clone git@github.com:spack/spack.git

. spack/share/spack/setup-env.sh
```

Get info on the package:

```
spack info lcc
```

Install the package, this will take a while because it'll install everything from scratch:

```
spack install lcc
```

Load the lcc module:

```
spack load lcc
```

Try lcc:

```
cd tmp ; lcc_main

spack install ovito
```

(continues on next page)

(continued from previous page)

```
spack load ovito

cd /tmp

echo "LCC{ ClusterType= Spheroid TypeOfLattice= FCC AAxis= 10.0 BAxis= 10.0 CAxis= 10.0 }
↪"  | tee input.in  ; lcc_main input.in

ovito coords.xyz
```

1.5 Download and installation

We will need to clone the repository as follows::

```
cd; git@github.com:lanl/LCC.git
```

1.6 Compiling PROGRESS and BML libraries

The LCC code needs to be compiled with both [PROGRESS](https://github.com/lanl/qmd-progress) and [BML](https://github.com/lanl/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`.

1.7 Step-by-step installation

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

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

Take a loot 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
```

[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.

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 you home folder and "clone" the PROGRESS repository. To do this type:

```
cd
git clone git@github.com:lanl/qmd-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.

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.

1.8 Testing the code

A test script can be run as follows:

```
./run_test
```

1.9 Quick example run

Assuming the code is installed in the `$HOME` directory, we will run a simple example::

```
cd /tmp
echo "LCC{ ClusterType= Spheroid TypeOfLattice= FCC AAxis= 10.0 BAxis= 10.0 CAxis= 10.0 }
↪"  | tee input.in  ; $HOME/LCC/build/lcc_main input.in
```

This will generate a spherical structure with an FCC lattice using default parameters. One can quickly get an input file sample by running the code without giving any input file. The available keywords can be listed by running `lcc_main -h`

1.10 Contributors

Christian Negre, email: cnegre@lanl.gov

Andrew Alvarado, email: aalvarado@lanl.gov

1.11 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"

SUGGESTION: Please, avoid commiting a large number of changes since it is difficult to review. Instead, add the changes gradually.

1.12 Citing

If you find this code useful, we encourage you to cite us. Our project has a citable DOI (DOI:10.1088/1361-648X/acc294) with the following `bibtex` snipped:

```
@ARTICLE{lcc,
   title    = "A methodology to generate crystal-based molecular structures for
               atomistic simulations",
  author    = "Negre, Christian F A and Alvarado, Andrew and Singh, Himanshu and
               Finkelstein, Joshua and Martinez, Enrique and Perriot, Romain",
  abstract  = "We propose a systematic method to construct crystal-based
               molecular structures often needed as input for computational
               chemistry studies. These structures include crystal 'slabs' with
               periodic boundary conditions (PBCs) and non-periodic solids such
               as Wulff structures. We also introduce a method to build crystal
               slabs with orthogonal PBC vectors. These methods are integrated
               into our code,Los Alamos Crystal Cut(LCC), which is open source
               and thus fully available to the community. Examples showing the
               use of these methods are given throughout the manuscript.",
  journal   = "J. Phys. Condens. Matter",
  volume    =  35,
  number    =  22,
  month     =  mar,
  year      =  2023,
  keywords  = "crystal structures; extended structures; miller indices; quantum
               chemistry; unit cells",
```

(continues on next page)

(continued from previous page)

```
  language = "en"
}
```

CHAPTER

TWO

BUILDING LCC DOCUMENTATION

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

2.1 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 python3-sphinx
pip3 install PSphinxTheme
pip3 install recommonmark
```

2.2 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 lcc.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
```

2.3 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 "markdown" syntax.
3) Modify the file in `./sphinx-src/source/index.txt` to include the documentation.

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

CHAPTER

THREE

INPUT FILE CHOICES

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

3.1 *JobName=*

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

3.2 *Verbose=*

Controls the verbosity level of the output. If set to `0` no output is pronted out. If set to `1`, only basic meesages 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.

3.3 *CoordsOutFile=*

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

3.4 *PrintCml=*

By setiing `PrintCml= T` will also print create `coords.cml` which can be readed 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 avogadro
sudo apt-get obabel
```

3.5 *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 sention.

3.6 *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
```

3.7 *ClusterType= Spheroid*

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

```
ClusterType= Spherid
LatticePoints=  50   #This is necesary 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.

3.8 *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 gurantee periodicity. In order to have a periodic structure different plane boudaries need to be tried and the structures needs to be checked using a molecular sivualizer. 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
```

(continues on next page)

(continued from previous page)

```
-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 peridicity vectors since if the Miller planes are orthogonal to each other, the shape will be a "Parallelepiped". If instead the number diferent than 6, then the periodicity vectors are given by the "Boundaries" of the minimal box that contains the shape.

3.9 *CenterAtBox=*

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

3.10 *Reorient=*

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

3.11 *AtomType=*

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

3.12 *TypeOfLattice=*

This will set the Lattice unit cell. if set to `SC` or `FCC` either a simple cubic or face centered cubinc 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=   11.022100000000000
LatticeConstantc=   7.3568800000000003
LatticeAngleAlpha=   90.000000000000000
LatticeAngleBeta=   102.65200000000000
LatticeAngleGamma=   90.000000000000000
```

3.13 *RandomSeed=*

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

3.14 *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=   11.022100000000000
LatticeConstantc=   7.3568800000000003
LatticeAngleAlpha=   90.000000000000000
LatticeAngleBeta=   102.65200000000000
LatticeAngleGamma=   90.000000000000000
```

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
]
```

3.15 *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 corrdinates and atom types needed to define a crystal system lattice point. The basis here will be red from file by providing the latticebase `LatticeBaseFile=` wich 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 is set to `F`, the the lattice parameters will need to be passed as explaines 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 is set to `xyz`, the it will be assumed to be given in catesian coordinates.

3.16 *SymmetryOperations=*

If the basis needs to be constricted from symetry operation, then one needs to pass all these operation 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 indictes the "translations" within the unit cell. The first three rows indicating the directions of the translation and the fourth indicating 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 traslation `0.5 0 0 1.0` and a summetry `-1 0 0`:

```
NumberOfOperations=           0
MaxCoordination=           1
NumberOfIterations=           1
Truncation=   1.000000000000000E+040
RCut=   20.000000000000000
RTol=   1.000000000000000E-002
CutAfterAddingBase=F
SeedFile=seed.pdb
```

CHAPTER

FOUR

BUILDING A LATTICE

In this section we briefly explain how to build a lattice using LCC. The finite set of points obtained in this ways has the shape that is bound by crystal faces which are paralell to the "canonical Miller planes" (100), (010), and (001) We will first execute lcc without any input file to create a sample input. Syntax follows:

```
./lcc_main
```

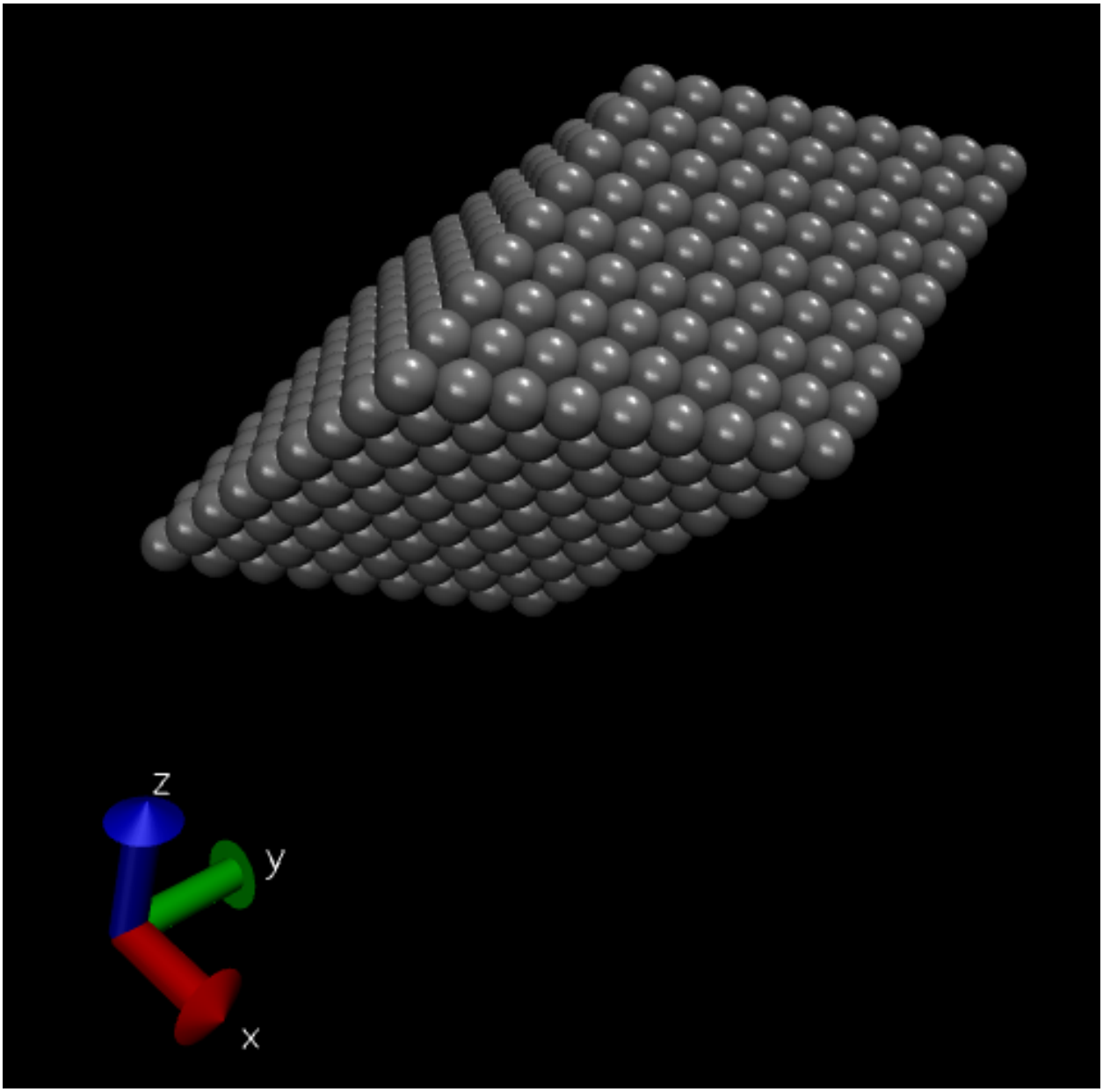
This will generate a sample input file called `sample_input.in`. You can either edit this file or make a new one having the following:

```
#Lcc input file.
LCC{
 JobName=                AgBulk        #Or any other name
 ClusterType=            Bulk
 TypeOfLattice=          FCC
 LatticePoints=          8             #Number of total lattice points in one direction
 LatticeConstanta=       4.08
 AtomType=               Ag
}
```

In order to run the code, just type:

```
./lcc_main sample_input.in
```

The run will produce two coordinate files `*_coords.xyz` and `*_coords.pdb`. If we visualize this with VMD we get the following "piece of bulk" for Silver



We can recover the same lattice by entering the Angles and edges of the unit cell as follows:

```
#Lcc input file.
LCC{
 JobName=                AgBulk          #Or any other name
 ClusterType=            Bulk
 TypeOfLattice=          Triclinic
 LatticePoints=          8               #Number of total lattice points in each direction
 AtomType=               Ag
 PrimitiveFormat=        Angles          #Will use angles and edges
 LatticeConstanta=       2.885
 LatticeConstantb=       4.08
 LatticeConstantc=       2.885
 LatticeAngleAlpha=      45
```

(continues on next page)

(continued from previous page)

```
 LatticeAngleBeta=        45
 LatticeAngleGamma=       60
}
```

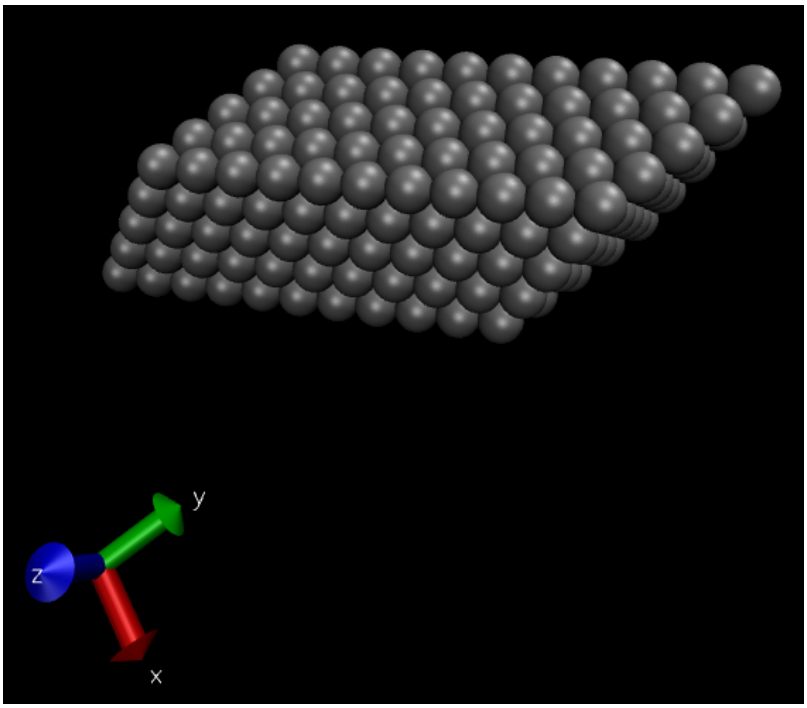
Yet another way of constructing an fcc lattice is by providing the lattice vectors directly which can be done by doing:

```
#Lcc input file.
LCC{
 JobName=                 AgBulk        #Or any other name
 ClusterType=             Bulk
 TypeOfLattice=           Triclinic
 LatticePoints=           8             #Number of total lattice points in each direction
 AtomType=                Ag
 PrimitiveFormat=         Vectors       #Will use primitive vectors
 LatticeVectors[
        2.885 2.885 0.000
        0.000 4.080 0.000
        0.000 2.885 2.885
        ]
}
```

If we want a bulk with a particular number of lattice points on each direction we can use the following input parameters:

```
#Lcc input file.
LCC{
 JobName=                 AgBulk        #Or any other name
 ClusterType=             Bulk
 TypeOfLattice=           Triclinic
 LatticePointsX1=        -2             #Number of point in the direction of the first␣
↪Lattice Vector
 LatticePointsX2=         8
 LatticePointsY1=        -2
 LatticePointsY2=         2
 LatticePointsZ1=        -2
 LatticePointsZ2=         2
 AtomType=                Ag
 PrimitiveFormat=         Angles        #Will use angles and edges
 LatticeConstanta=        2.885
 LatticeConstantb=        4.08
 LatticeConstantc=        2.885
 LatticeAngleAlpha=       45
 LatticeAngleBeta=        45
 LatticeAngleGamma=       60
}
```

The latter will produce a “bulk” enlarged in the direction of the first lattice vector.


y
z
x

CHAPTER

FIVE

BUILDING/CUTTING A SHAPE

5.1 Growing shapes from a seed file

The Bravais theory says that a crystal face will grow faster is the atom/unit cell that is added to the face finds a higher coordination. In this way, faces that have a high reticular density will grow slower since the adatom will potentially find only a "top" position.

Here we give an example of how to grow a shape from a seed using only geometrical parameters which are: the `MinCoordination` and the `RCut`. `RCut` is used as a criterion to search for the coordination. If the adatom (possible atom to be included in the shape) has 3 atoms that are within `RCut`, the coordination of such an adatom will be 3. If `MinCoordination = 2`, the the adatom with coordination = 3 will be included in the shape.

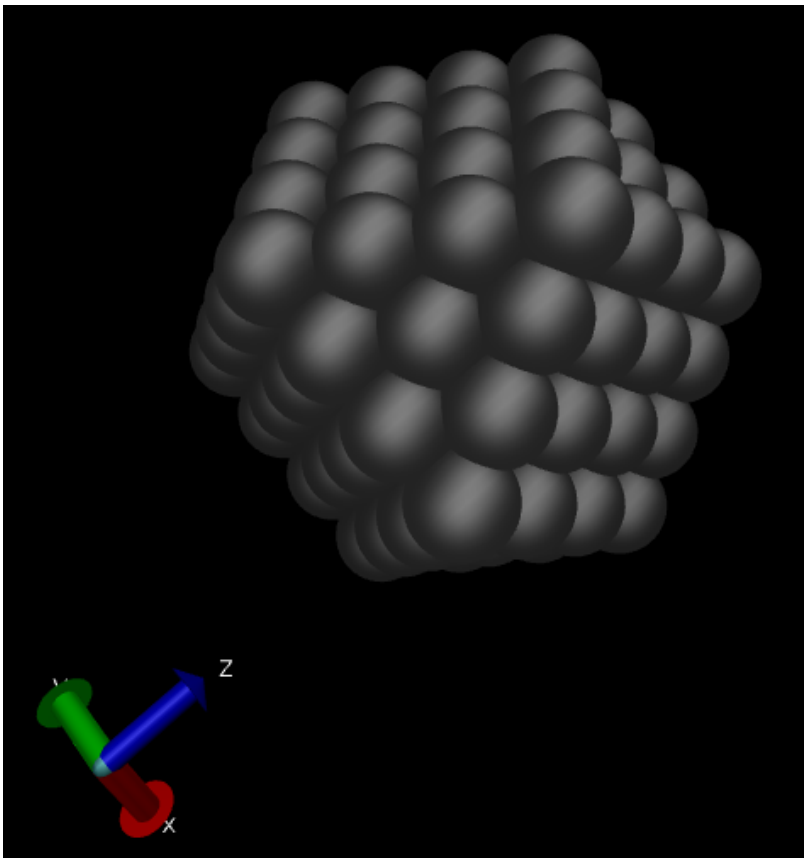
An exaple input file is given as follows:

```
#Lcc input file.
LCC{
 JobName=                AgBulk        #Or any other name

 ClusterType=            BravaisGrowth
 NumberOfIterations=     3
 RTol=                   0.01
 MaxCoordination=        1
 RCut=                   3.5
 SeedFile=               "seed.pdb"

 TypeOfLattice=          FCC
 LatticePointsX1=        -8            #Number of point in the direction of the first␣
↪Lattice Vector
 LatticePointsX2=        8
 LatticePointsY1=        -8
 LatticePointsY2=        8
 LatticePointsZ1=        -8
 LatticePointsZ2=        8
 AtomType=               Ag
 PrimitiveFormat=        Angles        #Will use angles and edges
 LatticeConstanta=       4.08
}
```

The `NumberOfIterations` parameter controls the cycles of gorwing that we want. The `SeedFile` paramter is the name of the file containg the "seed" from where the shape will grow. For this particular exaple we will used a seed (`seed.pdb`) file with the following content"

```
REMARK                        Seed File
TITLE coords.pdb
CRYST1  137.192  231.464  154.494  90.00 102.65  90.00 P 1           1
MODEL                     1
ATOM      1  Ag  M       1       0.000   0.000   0.000  0.00  0.00          Ag
TER
END
```

This means that we will be growing from "only one" Ag atom center at the origin. The result is the following shape:



5.2 Cutting using planes

A crystal shape can also be cut using planes. This could be usefull to comput a Wulff type of crystal shape by listing the planes and the surface energies or just for creating a "slab" to study a particular surface. An example of cutting by planes is provided as follows:

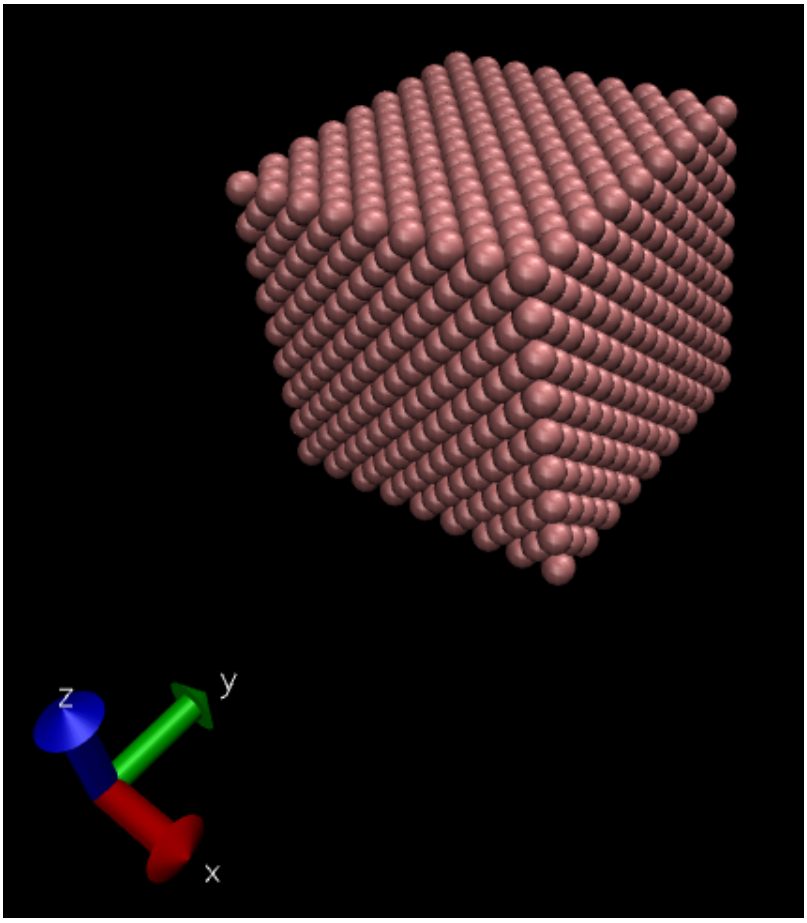
```
#Lcc input file.
LCC{
 JobName=                AgPlanes        #Or any other name
 Verbose=                3

 TypeOfLattice=          FCC
 LatticePoints=          50              #Number of point in each direction
 LatticeConstanta=       4.08
```

(continues on next page)

(continued from previous page)

```
 AtomType=                Ag

 ClusterType=             Planes

 NumberOfPlanes=   6
 Planes[
  1 0 0 4.1
  -1 0 0 4.1
  0 1 0 4.1
  0 -1 0 4.1
  0 0 -1 4.1
  0 0 1 4.1
]
}
```

This creates the following cubic shape:



CHAPTER

SIX

BUILDING A SLAB

In this tutorial we will explain the steps to construct a crystal "slab" that could be used to study a particular surface.

We will hereby use sucrose as an example. We will build the (0 1 1) surface and create a periodic slab. The data (CIF file) on sucrose was downloaded from *svn://www.crystallography.net/cod/cif/3/50/00/3500015.cif*

To this end we will use the following input file:

```
LCC{
  ClusterType=           Planes
  ClusterNumber=         2
  Verbose=  3
  LatticeBaseFile= "lattice_basis.xyz"
  WriteCml= F
  CheckPeriodicity= T

  ReadLatticeFromFile=  F
  TypeOfLattice=        Triclinic
  LatticePoints=        30
  CheckLattice=         T

  PrimitiveFormat=      Angles
  AtomType=             At
  UseLatticeBase=       T
  BaseFormat=           abc
  CutAfterAddingBase=   F

  LatticeConstanta=     7.789
  LatticeConstantb=     8.743
  LatticeConstantc=     10.883

  LatticeAngleAlpha=     90
  LatticeAngleBeta=      102.760
  LatticeAngleGamma=     90
  RCoeff= 0.0

  CenterAtBox=       T
  Reorient=          T

  #+X,+Y,+Z
  #-X,1/2+Y,-Z
  SymmetryOperations= T
```

(continues on next page)

(continued from previous page)

```
  NumberOfOperations= 2
  OptimalTranslations= T
  Translations[
    0  0   0   0
    0  1   0   0.5
  ]
  Symmetries[
    1  1  1
    -1  1 -1
  ]

 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   2.5
 -1  0   0   1.5
]
```

The "basis" needs to be provided via the *lattice_basis.xyz* file. The content of such file is provided below:

```
45
#Sucrose basis
O  0.63189     0.34908    0.62279
O  0.7136      0.2018     0.41867
O  0.6440      -0.0665    0.6512
O  0.2978      -0.0008    0.69117
O  0.2529      0.3114     0.77094
O  0.60891     0.40061    0.82857
O  0.68400     0.65323    0.78776
O  0.3785      0.5127     0.97000
O  0.9607      0.5091     0.67341
O  1.0893      0.6500     1.02195
O  0.7957      0.42950    1.07412
C  0.7053      0.1955     0.64075
C  0.5578      0.0769     0.6265
C  0.4362      0.1116     0.71451
C  0.3651      0.2728     0.6871
C  0.5149      0.3897     0.70028
C  0.8157      0.1767     0.5431
C  0.6306      0.5556     0.87572
C  0.8718      0.6862     0.82381
C  0.9441      0.5804     0.93500
C  0.7861      0.5573     0.99233
C  0.4569      0.6161     0.8967
C  0.9532      0.6662     0.7110
H  0.7813      0.1873     0.7252
H  0.4894      0.0781     0.5393
H  0.5018      0.1046     0.8022
H  0.2953      0.2763     0.6004
```

(continues on next page)

(continued from previous page)

```
H  0.4639      0.4900     0.6734
H  0.9127      0.2488     0.5604
H  0.8647      0.0743     0.5487
H  0.733       0.298      0.402
H  0.2287      0.0165     0.7364
H  0.2152      0.3986     0.7560
H  0.8878      0.7925     0.8526
H  0.9806      0.4827     0.9048
H  0.7738      0.6491     1.0414
H  0.4764      0.7140     0.9395
H  0.3769      0.6323     0.8158
H  0.3772      0.4263     0.9405
H  0.8853      0.7242     0.6409
H  1.0716      0.7077     0.7308
H  0.860       0.480      0.654
H  1.185       0.604      1.009
H  0.8058      0.3509     1.0352
H  0.553      -0.128      0.642
```

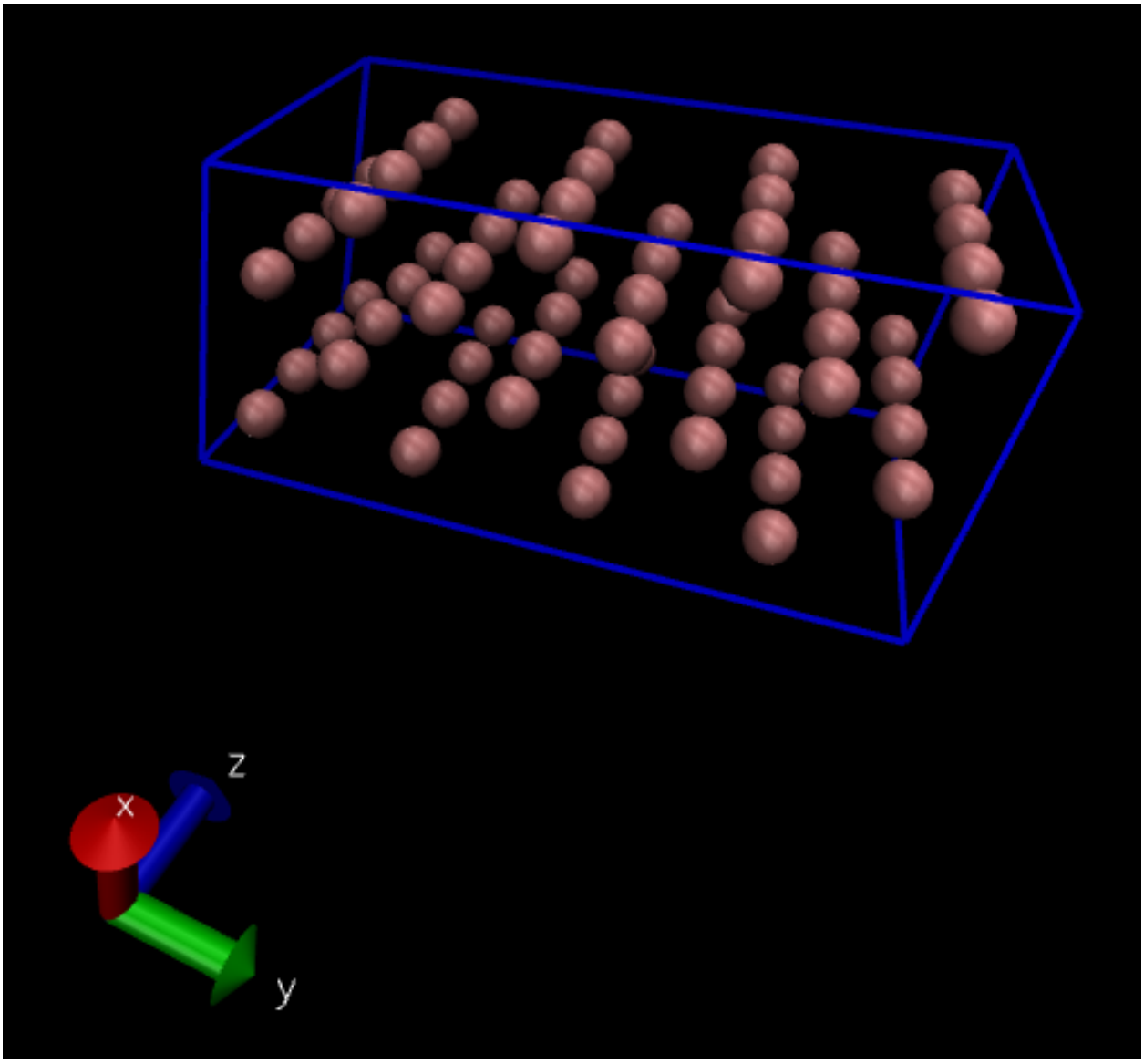
The first run we will do needs to have *UseLatticeBase= F*. In this way we will be able to inspect the lattice points and make sure that we get a periodic slab. The code automatically checks the periodicity. If the Miler planes are not ensuring periodicity, the code will raise an error. The lattice could be visualized with avogadro or vmd.:

```
avogadro coords.cml
```

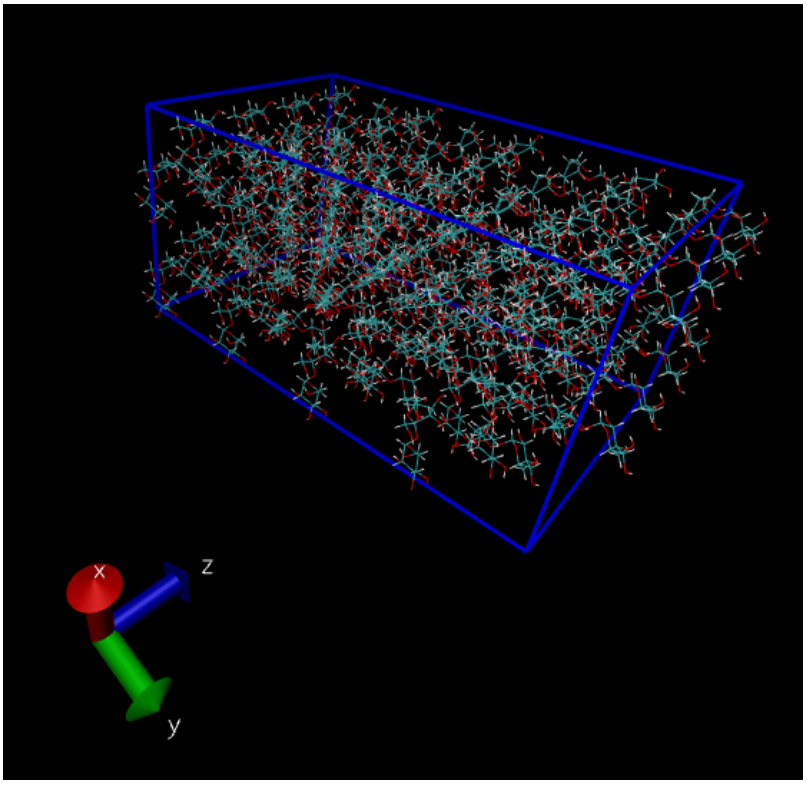
or:

```
vmd -f coords.pdb
```

This will show the following structure:



The next step is to run the code with *UseLatticeBase= T* to generate the final structure. Note that the input file contains the symetry operation to "complete" the unit cell. After running the code we will get the following structure:



6.1 Building a slab from three PBC vectors

Another method we have to build a crystal slab is to give the program the PBC vectors. For this, we will set *ClusterType=* to *ClusterType= Slab*. We will also need to give the PBC vectors and their lenghts as follows:

```
Slab[
    1.0 0.0 0.0 10.0
    0.0 1.0 1.0 10.0
    0.0 0.0 1.0 10.0
    ]
```

The latter input block means that the first vector will be the (1,0,0) with lenght 10.0. Note that three general vectors cannot guarantee that the slab will be congruent with the lattice. If we give three random vectors and have *CheckPeriodicity= T*, the code will most likely give an error. An example of construction of this type of slab can be find in *examples/build_from_vectors/*.

CHAPTER

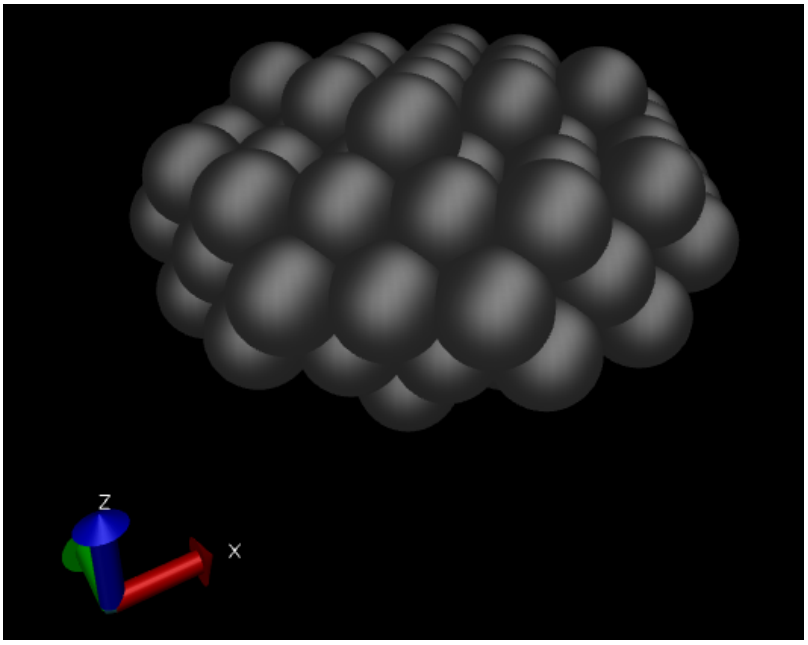
SEVEN

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":



CHAPTER

EIGHT

COMPUTING ROUGHNESS

In this tutorial we will explain the steps to compute a crystal surface Roughness defined as the ration between the "effective" surface and the "flat" surface.

We will hereby use sucrose as an example. We will analyse (100) and (10-1) crystal faces. Lattice information on sucrose was downloaded from *svn://www.crystallography.net/cod/cif/3/50/00/3500015.cif*

To this end we will use the following input file which can be also found under `/examples/Roughness`:

```
LCC{
  ClusterType=           Planes
  ClusterNumber=         2
  Verbose=  3
  LatticeBaseFile= "lattice_basis.xyz"
  WriteCml= F
  CheckPeriodicity= T

  ReadLatticeFromFile= F
  TypeOfLattice=         Triclinic
  LatticePoints=         30

  CheckLattice=          T

  PrimitiveFormat=       Angles
  AtomType=              At
  UseLatticeBase=        T
  BaseFormat=            abc
  CutAfterAddingBase=    F

  LatticeConstanta=      7.789
  LatticeConstantb=      8.743
  LatticeConstantc=      10.883

  LatticeAngleAlpha=      90
  LatticeAngleBeta=       102.760
  LatticeAngleGamma=      90
  RCoeff= 0.0

  CenterAtBox=        T
  Reorient=           T

  #+X,+Y,+Z
```

(continues on next page)

(continued from previous page)

```
  #-X,1/2+Y,-Z
  SymmetryOperations= T
  NumberOfOperations= 2
  OptimalTranslations= F
  Translations[
   1  0   1   -1
   0  1   2   0.5
  ]
  Symmetries[
     1  1  1
    -1  1 -1
  ]

  NumberOfPlanes=   6

  ComputeRoughness= T
  RoughnessParameters[
    50.0 1.0 40 80 80
  ]

  Planes[
    1  0   0   2.5
    -1  0  0   1.5
    0  1   0   4.5
    0  -1  0   3.5
    0  0   1   2.5
    0  0   -1   1.5
  ]

  #Planes[
    1  0   -1   2.5
    -1  0  1   1.5
    0 -1   0   4.5
    0  1  0   1.5
    1  0   1   4.5
    -1  0   -1   1.5
  ]

}
```

The "basis" needs to be provided via the `lattice_basis.xyz` file. The content of such file is provided below and can also be found under `/examples/Roughness`:

```
45
#Sucrose basis
O  0.63189     0.34908    0.62279
O  0.7136      0.2018     0.41867
O  0.6440      -0.0665    0.6512
O  0.2978      -0.0008    0.69117
O  0.2529      0.3114     0.77094
O  0.60891     0.40061    0.82857
O  0.68400     0.65323    0.78776
```

(continues on next page)

(continued from previous page)

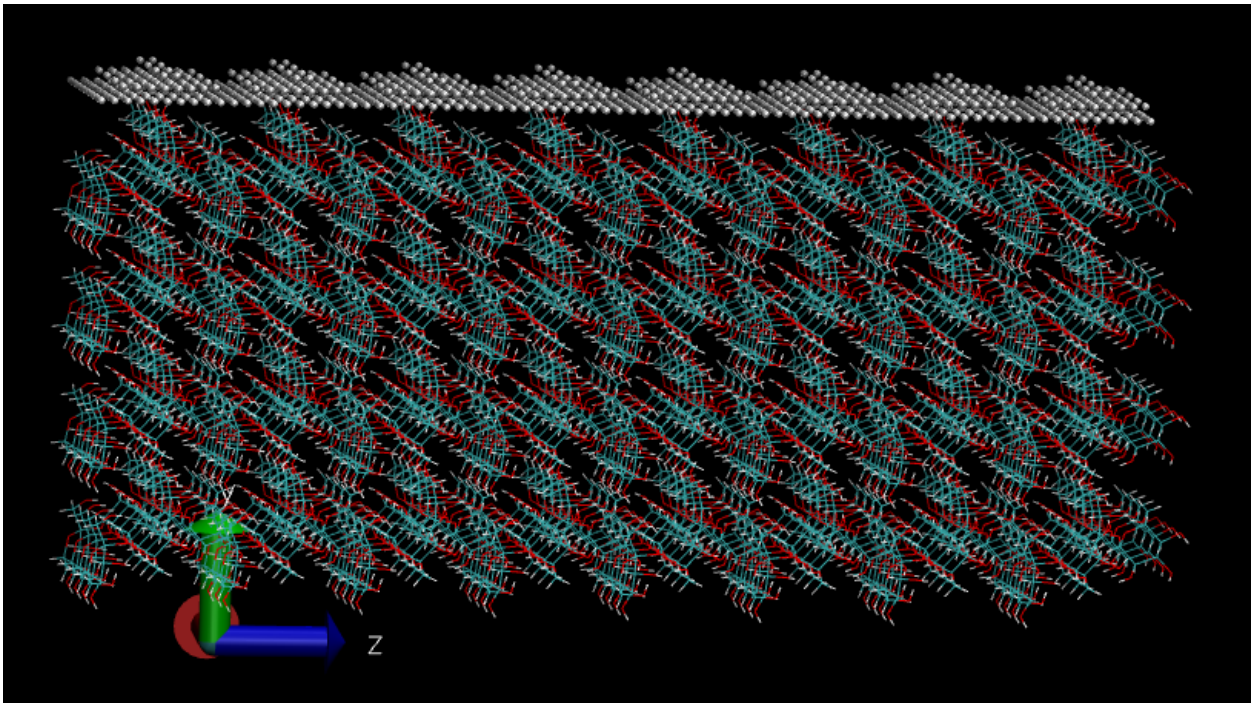
```
O  0.3785      0.5127     0.97000
O  0.9607      0.5091     0.67341
O  1.0893      0.6500     1.02195
O  0.7957      0.42950    1.07412
C  0.7053      0.1955     0.64075
C  0.5578      0.0769     0.6265
C  0.4362      0.1116     0.71451
C  0.3651      0.2728     0.6871
C  0.5149      0.3897     0.70028
C  0.8157      0.1767     0.5431
C  0.6306      0.5556     0.87572
C  0.8718      0.6862     0.82381
C  0.9441      0.5804     0.93500
C  0.7861      0.5573     0.99233
C  0.4569      0.6161     0.8967
C  0.9532      0.6662     0.7110
H  0.7813      0.1873     0.7252
H  0.4894      0.0781     0.5393
H  0.5018      0.1046     0.8022
H  0.2953      0.2763     0.6004
H  0.4639      0.4900     0.6734
H  0.9127      0.2488     0.5604
H  0.8647      0.0743     0.5487
H  0.733       0.298      0.402
H  0.2287      0.0165     0.7364
H  0.2152      0.3986     0.7560
H  0.8878      0.7925     0.8526
H  0.9806      0.4827     0.9048
H  0.7738      0.6491     1.0414
H  0.4764      0.7140     0.9395
H  0.3769      0.6323     0.8158
H  0.3772      0.4263     0.9405
H  0.8853      0.7242     0.6409
H  1.0716      0.7077     0.7308
H  0.860       0.480      0.654
H  1.185       0.604      1.009
H  0.8058      0.3509     1.0352
H  0.553      -0.128      0.642
```

The first run we will do needs to have the first plane listed uncommented. Note that the first plane listed will be the one used to compute the roughness. After executing lcc as follows:

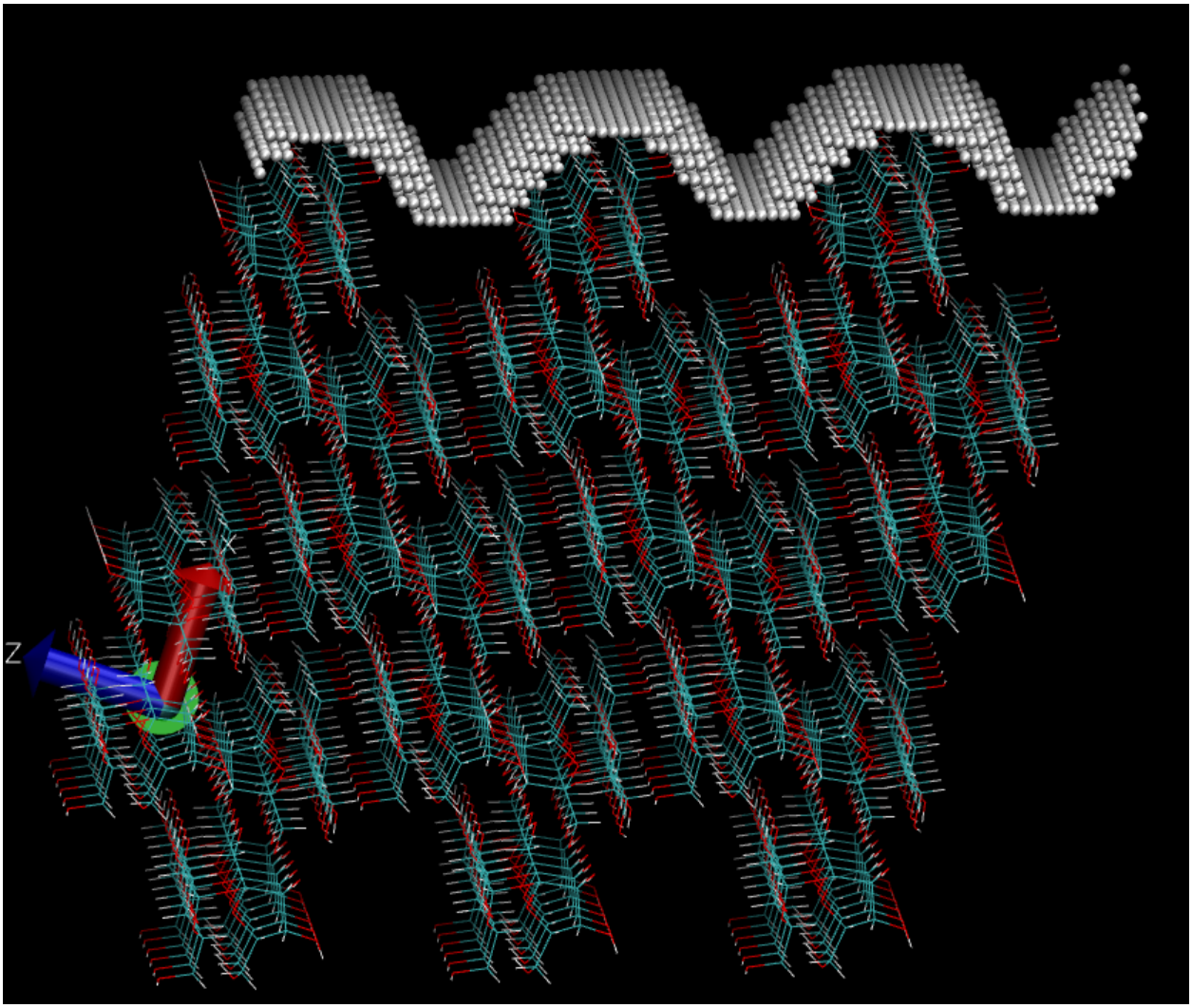
```
lcc_main sucrose.in
```

We will get information about the surfa areas S1 (effective) and S0 (flat) surface areas, together with their ratios.

The run will also produce a file called `mask.xyz` which contains a set of coordinates showing the surface pattern of the crystal face (`vmd -e mystate.vmd`).



If we now uncomment the second plane (10-1) and rerun we will see the following surface:



The parameters controlling these computations in the input file are the following:

```
ComputeRoughness= T
RoughnessParameters[
50.0 1.0 40 80 80
]
```

The first value controls the isovalue to compute the surface, the second value controls the radius of the "probe sphere" used to construct the surface, the third, fourth and fifth values control the discretization along the a1, a2, and a3 axis respectivelly.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search