LCC

1.0.0.

Generated by Doxygen 1.8.17

Chapter 1

LCC DOCUMENTATION

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

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

Chapter 2

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

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 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
```

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/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 afforemention 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 3

Building/cutting a shape

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 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 a 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:

Cutting using planes.

A crystal shape can also be cut using planes. This could be usefull to comput a Wolff 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
   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 4

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`.

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`

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.

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`.

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
```

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.

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 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.

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
-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.

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 dierction. This is important when making slabs needed to study a surface.

AtomType=

This will sed 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 cubinc lattice is built provided we set `LatticeConstanta=` to the lattice constant value. For general unit cell we can set `TypeOf↩ Lattice= 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
```

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.

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
]
```

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.

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= 1.0000000000000000E+040
RCut= 20.000000000000000
RTol= 1.0000000000000000E-002 CutAfterAddingBase=F
SeedFile=seed.pdb

Chapter 5

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" (1,0,0), (0,1,0), and (0,0,1) 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

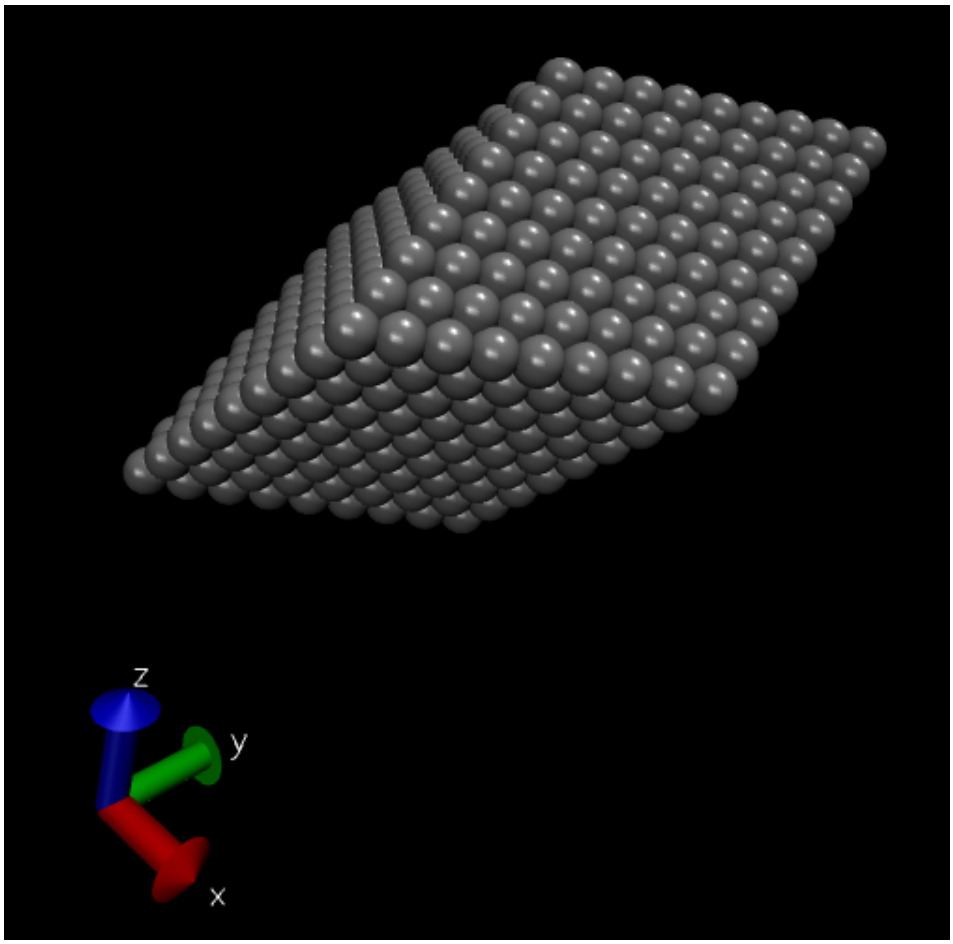

Figure 5.1 Ag bulk lattice chunk

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
   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.

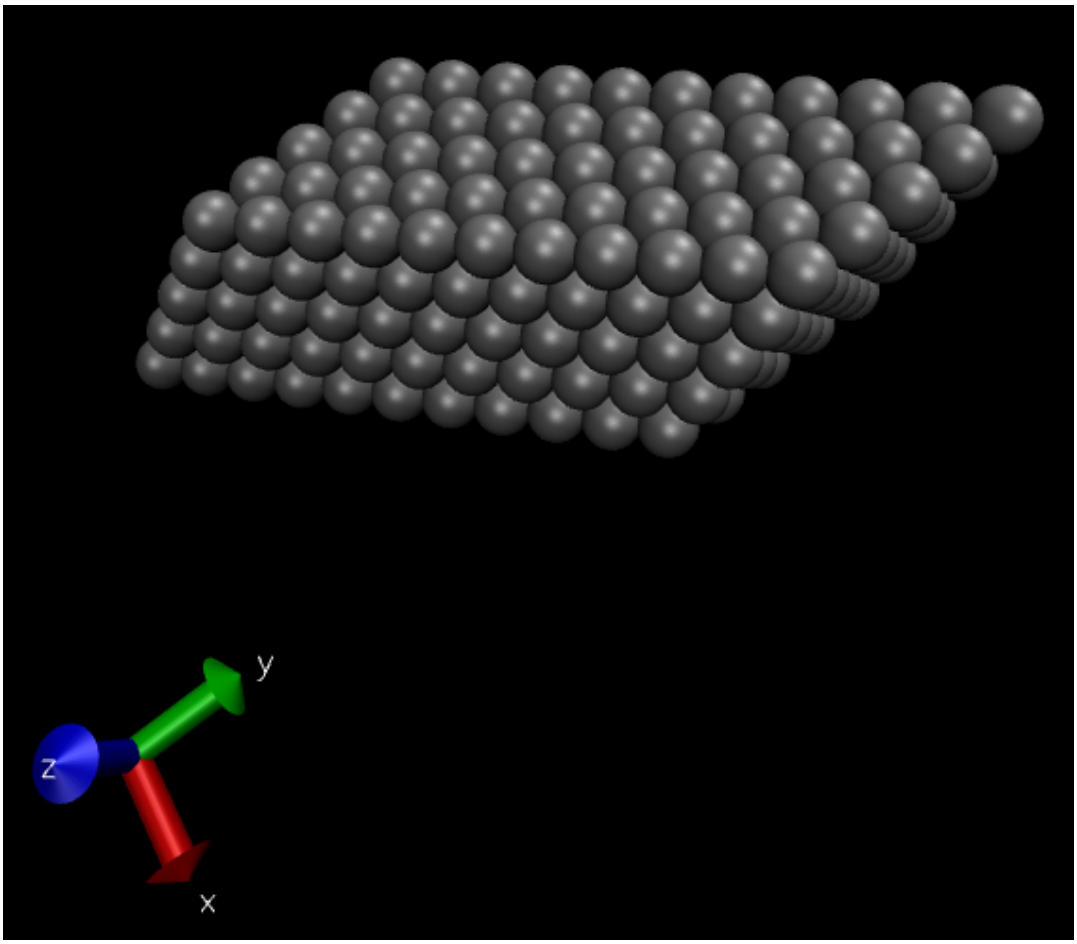

Figure 5.2 Ag bulk lattice enlarged on x direction

Chapter 6

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

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 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
```

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/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 afforemention 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 7

LCC DOCUMENTATION

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

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

Chapter 8

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":

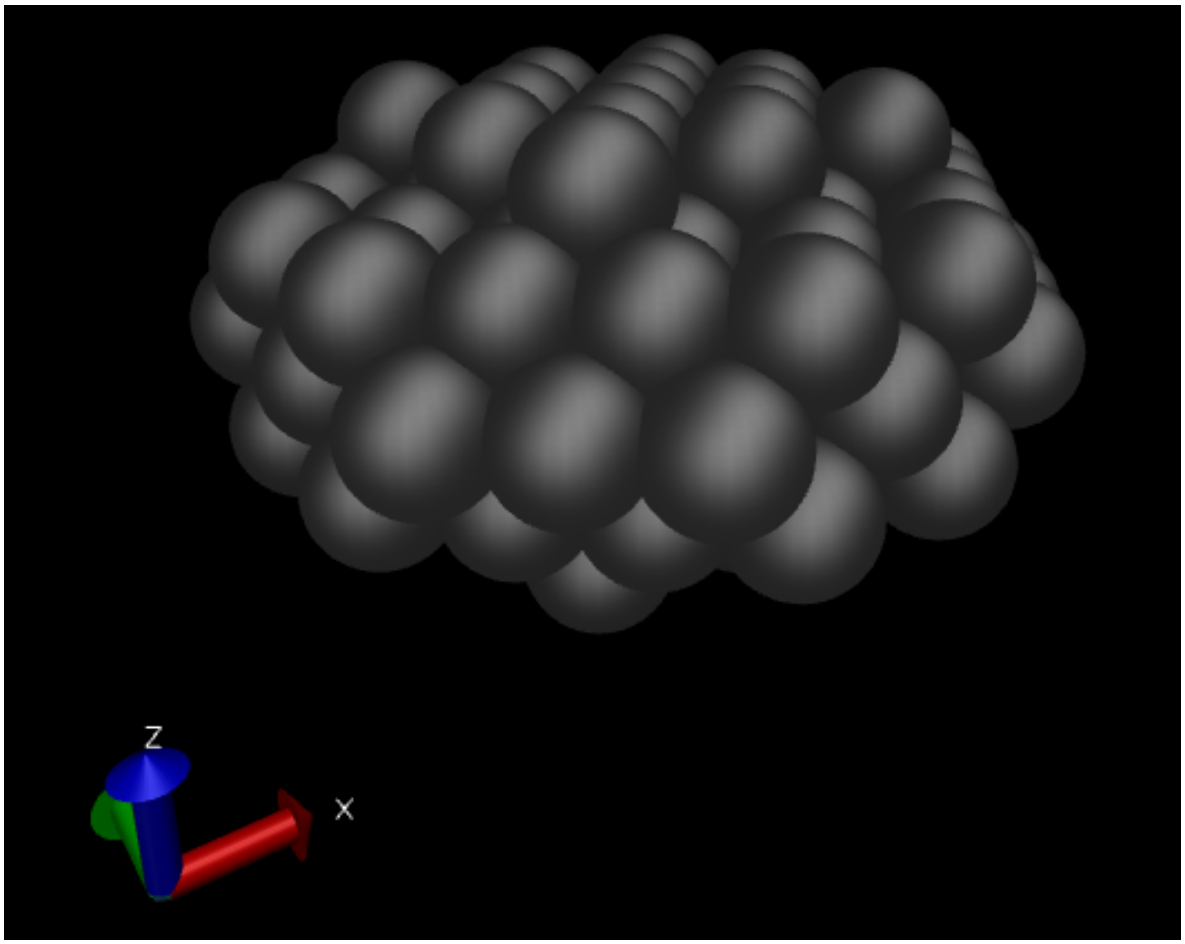

Figure 8.1 Oblate shape

Chapter 9

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
  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
 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:

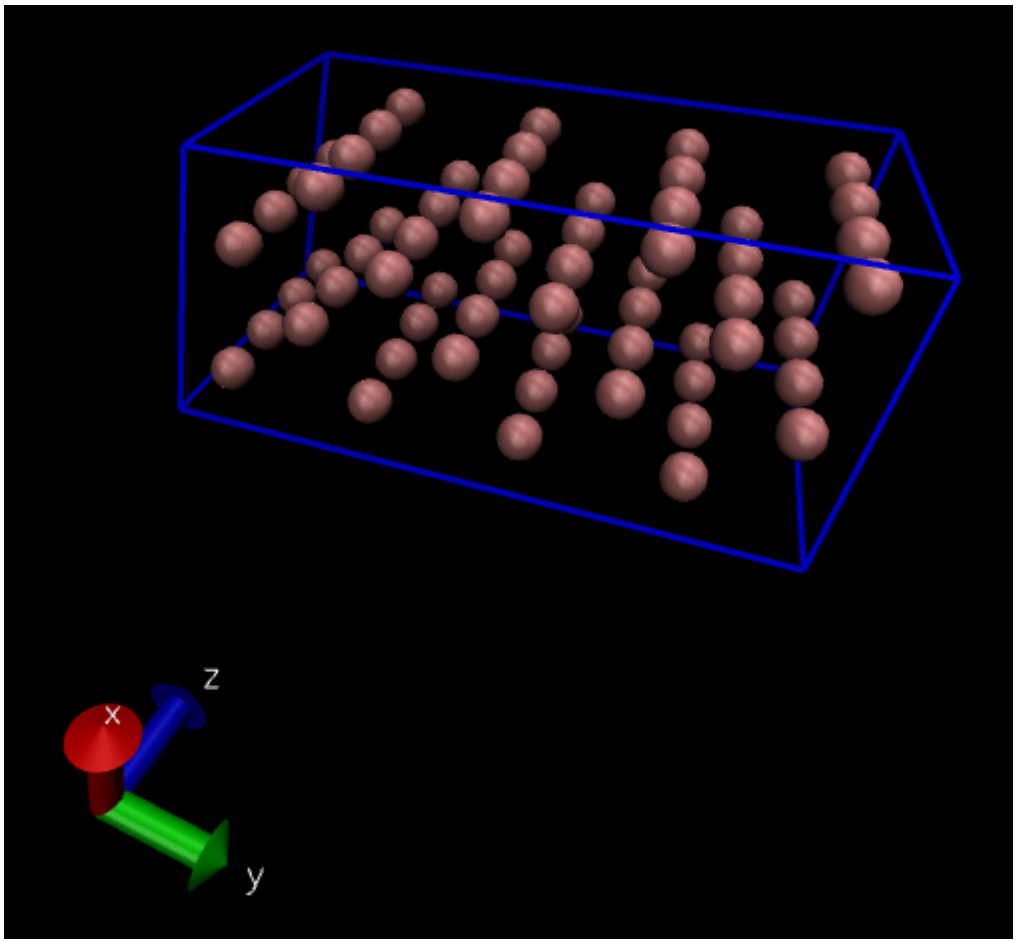

Figure 9.1 Slab generated from planes

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:

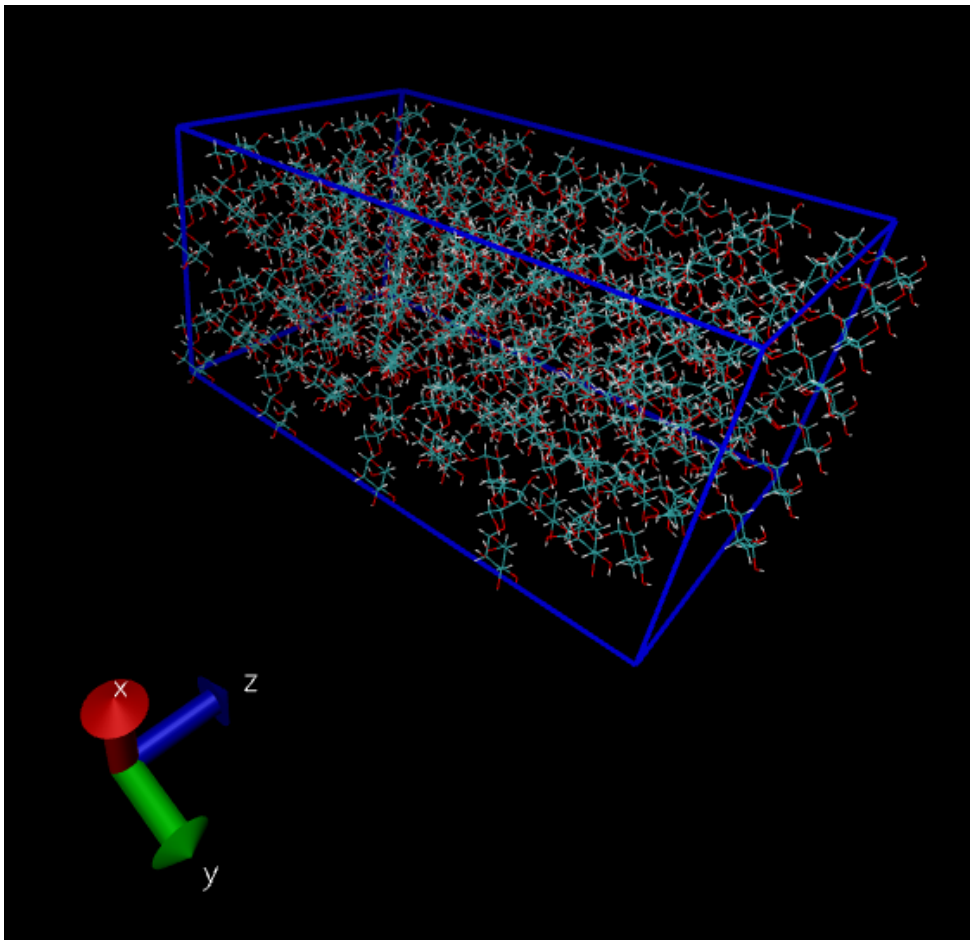

Figure 9.2 Slab generated from planes

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 `Check↩ Periodicity= 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 10

Testing the code

A test script ca be run as follows:

./run_test

Chapter 11

Todo List

Subprogram lcc_build_mod::lcc_bravais_growth (nCycles, dTol, dTo, tCoordination, seed_file, r_inout)

Optimize the routine.

Subprogram lcc_lattice_mod::lcc_triclinic (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, lattice_vectors, supra_lattice_↩ vectors, r_sy, verbose)

A angles_to_vectors transformation will be available.

Chapter 12

Namespace Index

12.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 13

Class Index

13.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 14

Namespace Documentation

14.1 lcc_allocation_mod Module Reference

Module for allocation operations.

Functions/Subroutines

- subroutine, public lcc_reallocate_realvect (vect, ndim)

  To reallocate a real vector.
- subroutine, public lcc_reallocate_realmat (mat, mdim, ndim)

  To reallocate a real mxn matrix.
- subroutine, public lcc_reallocate_intvect (vect, ndim)

  To reallocate a real vector.
- subroutine, public lcc_reallocate_intmat (mat, mdim, ndim)

  To reallocate an integer mxn matrix.
- subroutine, public lcc_reallocate_char2vect (vect, ndim)

  To reallocate a character vector.
- subroutine, public lcc_reallocate_char3vect (vect, ndim)

  To reallocate a character vector.

14.1.1 Detailed Description

Module for allocation operations.

14.1.2 Function/Subroutine Documentation

14.1.2.1 lcc_reallocate_char2vect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_char2vect (
             character(2), dimension(:), intent(inout), allocatable vect,
             integer, intent(in) ndim )
```

To reallocate a character vector.

This will reallocate a character len=2 vector If it is already allocated, a deallocation will first happen.

Parameters

vect	Character(2) 1D array.
ndim	Dimension to reallocate the vector to.

14.1.2.2 lcc_reallocate_char3vect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_char3vect (
             character(3), dimension(:), intent(inout), allocatable vect,
             integer, intent(in) ndim )
```

To reallocate a character vector.

This will reallocate a character len=3 vector. If it is already allocated, a deallocation will first happen.

Parameters

vect	Character(3) 1D array.
ndim	Dimension to reallocate the vector to.

14.1.2.3 lcc_reallocate_intmat()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_intmat (
             integer, dimension(:,:), intent(inout), allocatable mat,
             integer, intent(in) mdim,
             integer, intent(in) ndim )
```

To reallocate an integer mxn matrix.

This will reallocate a matrix. If it is already allocated, a deallocation will first happen.

Parameters

mat	Integer 2D array.
mnim	First dimension to realocate the matrix to.
ndim	Second dimension to reallocate the matrix to.

14.1.2.4 lcc_reallocate_intvect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_intvect (
             integer, dimension(:), intent(inout), allocatable vect,
             integer, intent(in) ndim )
```

To reallocate a real vector.

This will reallocate a vector If it is already allocated, a deallocation will first happen.

Parameters

vect	Integer 1D array.
ndim	Dimension to reallocate the vector to.

14.1.2.5 lcc_reallocate_realmat()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_realmat (
             real(dp), dimension(:,:), intent(inout), allocatable mat,
             integer, intent(in) mdim,
             integer, intent(in) ndim )
```

To reallocate a real mxn matrix.

This will reallocate a matrix If it is already allocated, a deallocation will first happen.

Parameters

mat	Real 2D array.
mnim	First dimension to realocate the matrix to.
ndim	Second dimension to reallocate the matrix to.

14.1.2.6 lcc_reallocate_realvect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_realvect (
             real(dp), dimension(:), intent(inout), allocatable vect,
             integer, intent(in) ndim )
```

To reallocate a real vector.

This will reallocate a vector If it is already allocated, a deallocation will first happen.

Parameters

vect	Real 1D array.
ndim	Dimension to reallocate the vector to.

14.2 lcc_aux_mod Module Reference

Module for auxiliary operations routines.

Functions/Subroutines

- subroutine, public lcc_vectors_to_parameters (lattice_vector, abc_angles, verbose)

  Transforms the lattice vectors into lattice parameters.
- subroutine, public lcc_parameters_to_vectors (abc_angles, lattice_vector, verbose)

  Transforms the lattice parameters into lattice vectors.
- subroutine, public lcc_get_coordination (r_at, r_env, thresh, cnum)

  Get the coordination of an atom.
- subroutine, public lcc_canonical_basis (lattice_vectors, r_inout, verbose)

  To "canonical base" transformation.
- subroutine, public lcc_center_at_box (lattice_vectors, r_inout, verbose)

  Cetering the system inside the lattice box.
- subroutine, public lcc_center_at_origin (r_inout, verbose)

  Cetering the system at the origin.
- real(dp) function, dimension(:,:), allocatable inv (A)

  Computes the inverse of a matrix using an LU decomposition.
- subroutine, public lcc_get_reticular_density (lattice_vectors, hkl_in, density)

  Get the reticular density of a particular hkl face: This soubroutine computes:
- real(dp) function, dimension(:), allocatable **crossprod** (r1, r2)

14.2.1 Detailed Description

Module for auxiliary operations routines.

14.2.2 Function/Subroutine Documentation

14.2.2.1 inv()

```
real(dp) function, dimension(:,:), allocatable lcc_aux_mod::inv (
             real(dp), dimension(:,:), intent(in) A )
```

Computes the inverse of a matrix using an LU decomposition.

Parameters

A	nxn Matrix to be inverted.
Ainv	Inverse of matrix A

14.2.2.2 lcc_canonical_basis()

```
subroutine, public lcc_aux_mod::lcc_canonical_basis (
             real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
```

```
            real(dp), dimension(:,:), intent(inout), allocatable r_inout,
            integer, intent(in) verbose )
```

To "canonical base" transformation.

This will reorient the shape/slab so that the first translation vector is alligned with x.

Parameters

lattice_vectors	Translation vectors for the shape/slab.
r_inout	Coordinates to be transformed.
verbose	Verbosity level.

14.2.2.3 lcc_center_at_box()

```
subroutine, public lcc_aux_mod::lcc_center_at_box (
            real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
            real(dp), dimension(:,:), intent(inout), allocatable r_inout,
            integer, intent(in) verbose )
```

Cetering the system inside the lattice box.

This will move the coordinates so that the geometric center of the system is at the center of the box.

Parameters

lattice_vectors	Translation vectors for the shape/slab.
r_inout	Coordinates to be transform.
verbose	Verbosity level.

14.2.2.4 lcc_center_at_origin()

```
subroutine, public lcc_aux_mod::lcc_center_at_origin (
            real(dp), dimension(:,:), allocatable r_inout,
            integer, intent(in) verbose )
```

Cetering the system at the origin.

This will move the coordinates so that the geometric center of the system is at (0,0,0).

Parameters

r_inout	Coordinates to be transform.
verbose	Verbosity level.

14.2.2.5 lcc_get_coordination()

```
subroutine, public lcc_aux_mod::lcc_get_coordination (
             real(dp), dimension(3), intent(in) r_at,
             real(dp), dimension(:,:), intent(in), allocatable r_env,
             real(dp), intent(in) thresh,
             integer, intent(inout) cnum )
```

Get the coordination of an atom.

Will count how many atoms are around a particular atom (coordination number) given a set radius.

Parameters

r_at	Coodinates of the atom for which we need the coordination.
r_env	Coordinated of the environment sorounding atom at r_at.
thres	Threshod distance to find coordinations.
cnum	Coordination number (output).

14.2.2.6 lcc_get_reticular_density()

```
subroutine, public lcc_aux_mod::lcc_get_reticular_density (
             real(dp), dimension(:,:), intent(in) lattice_vectors,
             real(dp), dimension(:), intent(in) hkl_in,
             real(dp), intent(out) density )
```

Get the reticular density of a particular hkl face: This soubroutine computes:

Parameters

lattice_vectors	Lattice vectors for the system.
hkl_in	Vector containing h, k, and l.
density	Reticular density.

14.2.2.7 lcc_parameters_to_vectors()

```
subroutine, public lcc_aux_mod::lcc_parameters_to_vectors (
             real(dp), dimension(2,3), intent(in) abc_angles,
             real(dp), dimension(3,3), intent(out) lattice_vector,
             integer, intent(in) verbose )
```

Transforms the lattice parameters into lattice vectors.

Parameters

abc_angles	2x3 array containing the lattice parameters. abc_angles(1,1) = a, abc_angles(1,2) = b, and abc_angles(1,3) = c abc_angles(2,1) = α , abc_angles(2,2) = β and abc_angles(2,3) = γ
lattice_vector	3x3 array containing the lattice vectors. lattice_vector(1,:) = $\overrightarrow{a}$
verbose	Verbosity level.

14.2.2.8 lcc_vectors_to_parameters()

```
subroutine, public lcc_aux_mod::lcc_vectors_to_parameters (
             real(dp), dimension(3,3), intent(in) lattice_vector,
             real(dp), dimension(2,3), intent(out) abc_angles,
             integer, intent(in) verbose )
```

Transforms the lattice vectors into lattice parameters.

Parameters

lattice_vector	3x3 array containing the lattice vectors. lattice_vector(1,:) = $\overrightarrow{a}$
abc_angles	2x3 array containing the lattice parameters. abc_angles(1,1) = a, abc_angles(1,2) = b and abc_angles(1,3) = c abc_angles(2,1) = α, abc_angles(2,2) = β, and abc_angles(2,3) = γ.
verbose	Verbosity level.

14.3 lcc_build_mod Module Reference

Module for generating the shapes after lattice is constructed.

Functions/Subroutines

- subroutine, public lcc_bravais_growth (nCycles, dTol, dTo, tCoordination, seed_file, r_inout)

  For "growing" a crystal shape using Bravias type of growth teory.
- subroutine, public lcc_plane_cut (planes, ploads, interPlanarDistances, lattice_vectors, cluster_lattice_↩ vectors, resindex, r_inout, verbose)

  Cutting a shape based on Miller planes.
- subroutine lcc_build_slab (slab, sloads, lattice_vectors, cluster_lattice_vectors, resindex, r_inout, verbose)

  Cutting a shape based on PBC vectors.
- subroutine, public lcc_add_randomness_to_coordinates (r_inout, seed, rcoeff)

  Will add randomness to the system.

14.3.1 Detailed Description

Module for generating the shapes after lattice is constructed.

14.3.2 Function/Subroutine Documentation

14.3.2.1 lcc_add_randomness_to_coordinates()

```
subroutine, public lcc_build_mod::lcc_add_randomness_to_coordinates (
             real(dp), dimension(:,:), intent(inout), allocatable r_inout,
             integer, intent(in) seed,
             real(dp), intent(in) rcoeff )
```

Will add randomness to the system.

Parameters

r_inout	System coordinates.
lattice_vectors	Lattice vectors.
seed	Random seed. rcoeff Coefficient for randomness.

14.3.2.2 lcc_bravais_growth()

```
subroutine, public lcc_build_mod::lcc_bravais_growth (
             integer, intent(in) nCycles,
             real(dp), intent(in) dTol,
             real(dp), intent(in) dTo,
             integer, intent(in) tCoordination,
             character(len=*), intent(in) seed_file,
             real(dp), dimension(:,:), intent(inout), allocatable r_inout )
```

For "growing" a crystal shape using Bravias type of growth teory.

Parameters

nCycles	Number of shells to add.
dTol	Tolerance for distinguising the coordinates from the seed to the coodinates from the bulk.
dTo	Parameter to determine the coordination the incoming atom.
tCoordination	Target coordination. If coodination is larger than the target, the atom will be picked.
seed_file	Name of the file containing the seed.
r_inout	Input: Bulk lattice, Output: Crystal shape.

Todo Optimize the routine.

14.3.2.3 lcc_build_slab()

```
subroutine lcc_build_mod::lcc_build_slab (
             real(dp), dimension(:,:), intent(in), allocatable slab,
             real(dp), dimension(:), intent(in), allocatable sloads,
             real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
             real(dp), dimension(:,:), intent(inout), allocatable cluster_lattice_vectors,
             integer, dimension(:), allocatable resindex,
             real(dp), dimension(:,:), intent(inout), allocatable r_inout,
             integer, intent(in) verbose )
```

Cutting a shape based on PBC vectors.

A set of PBC vectors and distances is provided.

Parameters

planes	List of planes to cut the shape with.
ploads	Distance from the origin to locate the plane.
interPlanarDistance	Use "interplanar distances" as measure for the cut.
lattice_vectors	Lattice vectors.
cluster_lattice_vectors	Lattice vectors of the shape. Note: this only makes sense if the planes make a parellelepiped.
r_inout	Coordinates in and out.
verbose	Verbosity level.

14.3.2.4 lcc_plane_cut()

```
subroutine, public lcc_build_mod::lcc_plane_cut (
             real(dp), dimension(:,:), intent(in), allocatable planes,
             real(dp), dimension(:), intent(in), allocatable ploads,
             logical, intent(in) interPlanarDistances,
             real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
             real(dp), dimension(:,:), intent(inout), allocatable cluster_lattice_vectors,
             integer, dimension(:), allocatable resindex,
             real(dp), dimension(:,:), intent(inout), allocatable r_inout,
             integer, intent(in) verbose )
```

Cutting a shape based on Miller planes.

A set of panes and distances is provided.

Parameters

planes	List of planes to cut the shape with.
ploads	Distance from the origin to locate the plane.
interPlanarDistance	Use "interplanar distances" as measure for the cut.
lattice_vectors	Lattice vectors.
cluster_lattice_vectors	Lattice vectors of the shape. Note: this only makes sense if the planes make a parellelepiped.
r_inout	Coordinates in and out.
verbose	Verbosity level.

14.4 lcc_check_mod Module Reference

Module for checking operations routines.

Functions/Subroutines

- subroutine, public lcc_check_periodicity (r_in, lattice_vectors, r_ref, tol, verbose)

  Check the periodicity.

14.4.1 Detailed Description

Module for checking operations routines.

14.4.2 Function/Subroutine Documentation

14.4.2.1 lcc_check_periodicity()

```
subroutine, public lcc_check_mod::lcc_check_periodicity (
             real(dp), dimension(:,:), intent(in), allocatable r_in,
             real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
             real(dp), dimension(:,:), intent(in), allocatable r_ref,
             real(dp), intent(in) tol,
             integer, intent(in) verbose )
```

Check the periodicity.

Will use a "brute force" approach to check periodidity.

Parameters

r_in	Input coordinates.
lattice_vectors	Translation vectors for the slab.
r_ref	Reference or "bulk structure from where the shape was cut.
verbose	Verbosity level.

14.5 lcc_constants_mod Module Reference

A module to handle the constants needed by the code.

Variables

- integer, parameter, public dp = kind(1.0d0)

Precision used troughout the code.

- real(dp), parameter pi = 3.14159265358979323846264338327950_dp

  Pi number.

14.5.1 Detailed Description

A module to handle the constants needed by the code.

This module will be used to store the constants needed in the code

14.6 lcc_lattice_mod Module Reference

Module to hold routines for handling the lattice and lattice base.

Functions/Subroutines

- subroutine, public lcc_make_lattice (bld, ltt, check, sy)

  Make a lattice depending on the input parameter.
- subroutine lcc_read_base (bld, ltt, check, verbose)

  Reading the basis from an input file.
- subroutine lcc_check_basis (base_format, r_base, lattice_vectors, verbose)

  Routine to check for atom repetitions in basis \bnrief It will do all possible translations searching for atoms that could be repeated.
- subroutine lcc_add_base_to_cluster (ltt, sy)

  Add a basis to the lattice.
- subroutine lcc_sc (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, h_lattice_a, supra_lattice_vectors, r_sy)

  Simple cubic (SC) lattice construction.
- subroutine lcc_fcc (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, h_lattice_a, supra_lattice_vectors, r_sy, verbose)

  Face center cubic (FCC) lattice construction.
- subroutine lcc_triclinic (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, lattice_vectors, supra_lattice_vectors, r_sy, verbose)

  Triclinic lattice construction.
- subroutine, public lcc_set_atom_type (a_type, atom_symbol, atom_name, nats)

  Sets the atom type.
- subroutine lcc_add_randomness (r_inout, lattice_vectors, seed, rcoeff)

  Will add randomness to the system.
- subroutine lcc_minimize_from (xVar, i, ai, nats, trs, verbose)

  To get the best translation that minimizes the distance to any previous fragment.

14.6.1 Detailed Description

Module to hold routines for handling the lattice and lattice base.

14.6.2 Function/Subroutine Documentation

14.6.2.1 lcc_add_base_to_cluster()

```
subroutine lcc_lattice_mod::lcc_add_base_to_cluster (
             type(lattice_type), intent(in) ltt,
             type(system_type), intent(inout) sy )
```

Add a basis to the lattice.

This routine will add the basis to the system points previously cut from the lattice. This is the last step of the solid/shape/slab creation.

Parameters

ltt	lattice_type See lcc_structs_mod
sy	system_type See progress library

14.6.2.2 lcc_add_randomness()

```
subroutine lcc_lattice_mod::lcc_add_randomness (
             real(dp), dimension(:,:), intent(inout), allocatable r_inout,
             real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
             integer, intent(in) seed,
             real(dp), intent(in) rcoeff )
```

Will add randomness to the system.

Parameters

r_inout	System coordinates.
lattice_vectors	Lattice vectors.
seed	Random seed. rcoeff Coefficient for randomness.

14.6.2.3 lcc_check_basis()

```
subroutine lcc_lattice_mod::lcc_check_basis (
             character(len=*), intent(in) base_format,
             real(dp), dimension(:,:), intent(in), allocatable r_base,
             real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
             integer, intent(in) verbose )
```

Routine to check for atom repetitions in basis \bnrief It will do all possible translations searching for atoms that could be repeated.

Parameters

base_format	Basis format, if xyz of abc
r_base	Coordinates of the basis. r_base(1,7) means coordinate x of atom 7
lattice_vectors	Lattice vectors. WARNING, in this case lattice_vector(1,3) means the coordinate 3=z of vector 1.

14.6.2.4 lcc_fcc()

```
subroutine lcc_lattice_mod::lcc_fcc (
             integer, intent(in) Nx1,
             integer, intent(in) Nx2,
             integer, intent(in) Ny1,
             integer, intent(in) Ny2,
             integer, intent(in) Nz1,
             integer, intent(in) Nz2,
             real(dp), intent(in) h_lattice_a,
             real(dp), dimension(:,:), intent(inout), allocatable supra_lattice_vectors,
             real(dp), dimension(:,:), intent(inout), allocatable r_sy,
             integer, intent(in) verbose )
```

Face center cubic (FCC) lattice construction.

Constructs a "bulk" of Face center cubic lattice.

Parameters

Nx1	Initial x lattice point.
Nx2	Final x lattice point.
Ny1	Initial y lattice point.
Ny2	Final y lattice point.
Nz1	Initial z lattice point.
Nz2	Final z lattice point.
h_lattice_a	Lattice parameter.
supra_lattice_vectors	Lattice unit vectors of the resulting slab.
r_sy	Output system coordinates.

14.6.2.5 lcc_make_lattice()

```
subroutine, public lcc_lattice_mod::lcc_make_lattice (
             type(build_type), intent(inout) bld,
             type(lattice_type), intent(inout) ltt,
             logical, intent(in) check,
             type(system_type), intent(inout) sy )
```

Make a lattice depending on the input parameter.

This will make one of the following latices: SC: Simple cubic, FCC: Face center cubic, or Triclinic.

Parameters

bld	Building structure (see lcc_structures_mod)
ltt	Lattice structure (see lcc_scturctures_mod)
check	If we want to check the basis for atom repetition. Note that checks can be expensive.

14.6.2.6 lcc_minimize_from()

```
subroutine lcc_lattice_mod::lcc_minimize_from (
             real(dp), dimension(:,:), intent(in) xVar,
             integer, intent(in) i,
             integer, intent(in) ai,
             integer, intent(in) nats,
             real(dp), dimension(3), intent(inout) trs,
             integer, intent(in) verbose )
```

To get the best translation that minimizes the distance to any previous fragment.

Parameters

xVar	Coordinates of the full basis (including symmetry operations).
i	Fragmet being added at the "i" operation.
ai	Atom index to translate and get the optimal translation.
nats	Number of atoms in the fragment.
trs	Optimal translation.

14.6.2.7 lcc_read_base()

```
subroutine lcc_lattice_mod::lcc_read_base (
             type(build_type), intent(inout) bld,
             type(lattice_type), intent(inout) ltt,
             logical, intent(in) check,
             integer, intent(in) verbose )
```

Reading the basis from an input file.

This will read the coordinates for the basis from an input file If information about the lattice is contained, it will also be read.

Parameters

bld	Building structure (see lcc_structures_mod).
ltt	Lattice structure (see lcc_scturctures_mod).
check	If we want to check the basis for atom repetition.
verbose	Verbose level. Note that checks can be expensive.

14.6.2.8 lcc_sc()

```
subroutine lcc_lattice_mod::lcc_sc (
             integer, intent(in) Nx1,
```

```
            integer, intent(in) Nx2,
            integer, intent(in) Ny1,
            integer, intent(in) Ny2,
            integer, intent(in) Nz1,
            integer, intent(in) Nz2,
            real(dp), intent(in) h_lattice_a,
            real(dp), dimension(:,:), intent(inout), allocatable supra_lattice_vectors,
            real(dp), dimension(:,:), intent(inout), allocatable r_sy )
```

Simple cubic (SC) lattice construction.

Constructs a "bulk" of Simple Cubic lattice.

Parameters

Nx1	Initial x lattice point.
Nx2	Final x lattice point.
Ny1	Initial y lattice point.
Ny2	Final y lattice point.
Nz1	Initial z lattice point.
Nz2	Final z lattice point.
h_lattice_a	Lattice parameter.
supra_lattice_vectors	Lattice unit vectors of the resulting slab.
r_sy	Output system coordinates.

14.6.2.9 lcc_set_atom_type()

```
subroutine, public lcc_lattice_mod::lcc_set_atom_type (
            character(len=2), intent(in) a_type,
            character(len=2), dimension(:), intent(inout), allocatable atom_symbol,
            character(len=3), dimension(:), intent(inout), allocatable atom_name,
            integer, intent(in) nats )
```

Sets the atom type.

Sets the atom "symbol/type/name."

Parameters

a_type	Atom symbol character.
atom_symbol	Atom symbols.
atom_name	Atom name. Note: Atom name is a tag that can distinguish atoms with same symbol.

14.6.2.10 lcc_triclinic()

```
subroutine lcc_lattice_mod::lcc_triclinic (
            integer, intent(in) Nx1,
```

```
integer, intent(in) Nx2,
integer, intent(in) Ny1,
integer, intent(in) Ny2,
integer, intent(in) Nz1,
integer, intent(in) Nz2,
real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
real(dp), dimension(:,:), intent(inout), allocatable supra_lattice_vectors,
real(dp), dimension(:,:), intent(inout), allocatable r_sy,
integer, intent(in) verbose )
```

Triclinic lattice construction.

Constructs a "bulk" of triclinic lattice.

Parameters

Nx1	Initial x lattice point.
Nx2	Final x lattice point.
Ny1	Initial y lattice point.
Ny2	Final y lattice point.
Nz1	Initial z lattice point.
Nz2	Final z lattice point.
lattice_vectors	Lattice vectors.
supra_lattice_vectors	Lattice unit vectors of the resulting slab.
r_sy	Output system coordinates. Note: Unit cell representation has to be transformed from edges and angles to vetors before calling this routine.

Todo A angles_to_vectors transformation will be available.

14.7 lcc_lib Module Reference

Library module.

Functions/Subroutines

- subroutine, public **lcc** (readInputFile, inputFileName, syOut, writeOut, clType, planeIn)

14.7.1 Detailed Description

Library module.

14.8 lcc_mc_mod Module Reference

Module for Monte Carlo related routines.

Functions/Subroutines

- subroutine lcc_check_system (r, iter, temp, cost, cost0)
  Maximize: This checks the acceptance.

14.8.1 Detailed Description

Module for Monte Carlo related routines.

14.9 lcc_message_mod Module Reference

Module for printing through the code.

Functions/Subroutines

- subroutine, public lcc_print_ussage ()
  For printing the instructions on how to execute the code.
- subroutine, public lcc_print_message (message, verbose)
  Print a simple message.
- subroutine, public lcc_print_warning (at, message, verbose)
  Print a Warning (will not stop execution).
- subroutine, public lcc_print_error (at, message)
  Print error (will stop execution).
- subroutine, public lcc_print_intval (name, value, units, verbose)
  Print integer magnitude.
- subroutine, public lcc_print_realval (name, value, units, verbose)
  Print real magnitude.
- subroutine, public lcc_print_realvect (name, vect, units, verbose)
  Print real vector.
- subroutine lcc_print_realmat (name, mat, units, verbose)
  Print real vector.
- subroutine **lcc_help** ()

14.9.1 Detailed Description

Module for printing through the code.

14.9.2 Function/Subroutine Documentation

14.9.2.1 lcc_print_error()

```
subroutine, public lcc_message_mod::lcc_print_error (
            character(len=*), intent(in) at,
            character(len=*), intent(in) message )
```

Print error (will stop execution).

Parameters

at	Name of the routine.
message	Message to print.

14.9.2.2 lcc_print_intval()

```
subroutine, public lcc_message_mod::lcc_print_intval (
             character(len=*), intent(in) name,
             integer, intent(in) value,
             character(len=*), intent(in) units,
             integer, intent(in) verbose )
```

Print integer magnitude.

Parameters

name	Name of the magnitude.
value	Value to print.
units	Units of the magnitude.

14.9.2.3 lcc_print_message()

```
subroutine, public lcc_message_mod::lcc_print_message (
             character(len=*), intent(in) message,
             integer, intent(in) verbose )
```

Print a simple message.

Parameters

message	Message to print.
verbose	Verbosity level.

14.9.2.4 lcc_print_realmat()

```
subroutine lcc_message_mod::lcc_print_realmat (
             character(len=*), intent(in) name,
             real(dp), dimension(:,:), intent(in), allocatable mat,
             character(len=*), intent(in) units,
             integer, intent(in) verbose )
```

Print real vector.

Parameters

name	Name of the quantities.
mat	Matrix to print.
units	Units of the quantities.
verbose	Verbosity level.

14.9.2.5 lcc_print_realval()

```
subroutine, public lcc_message_mod::lcc_print_realval (
             character(len=*), intent(in) name,
             real(dp), intent(in) value,
             character(len=*), intent(in) units,
             integer, intent(in) verbose )
```

Print real magnitude.

Parameters

name	Name of the magnitude.
value	Value to print.
units	Units of the magnitude.

14.9.2.6 lcc_print_realvect()

```
subroutine, public lcc_message_mod::lcc_print_realvect (
             character(len=*), intent(in) name,
             real(dp), dimension(:), intent(in), allocatable vect,
             character(len=*), intent(in) units,
             integer, intent(in) verbose )
```

Print real vector.

Parameters

name	Name of the quantities.
vect	Vector to print.
units	Units of the quantities.
verbose	Verbosity level.

14.9.2.7 lcc_print_warning()

```
subroutine, public lcc_message_mod::lcc_print_warning (
```

```
            character(len=*), intent(in) at,
            character(len=*), intent(in) message,
            integer, intent(in) verbose )
```

Print a Warning (will not stop execution).

Parameters

at	Name of the routine.
message	Message to print.
verbose	Verbosity level.

14.10 lcc_parser_mod Module Reference

This module controls the initialization of the variables.

Functions/Subroutines

- subroutine, public lcc_parse (filename, bld, ltt)

  Clustergen parser.
- subroutine, public lcc_make_sample_input ()

  Make a sample inputfile sample_input.in.
- subroutine, public lcc_write_coords (sy, bld, coordsout_file, verbose)

  Writes the coordinates to a file (coordsandbase.pdb)

14.10.1 Detailed Description

This module controls the initialization of the variables.

14.10.2 Function/Subroutine Documentation

14.10.2.1 lcc_parse()

```
subroutine, public lcc_parser_mod::lcc_parse (
            character(len=*), intent(in) filename,
            type(build_type), intent(inout) bld,
            type(lattice_type), intent(inout) ltt )
```

Clustergen parser.

This module is used to parse all the input variables for this program. Adding a new input keyword to the parser:

- If the variable is real, we have to increase nkey_re.
- Add the keyword (character type) in the keyvector_re vector.
- Add a default value (real type) in the valvector_re.
- Define a new variable and pass the value through valvector_re(num) where num is the position of the new keyword in the vector.

Parameters

filename	File name for the input.
bld	Build type.
ltt	Lattice type.

14.10.2.2 lcc_write_coords()

```
subroutine, public lcc_parser_mod::lcc_write_coords (
             type(system_type) sy,
             type(build_type) bld,
             character(len=*) coordsout_file,
             integer, intent(in) verbose )
```

Writes the coordinates to a file (coordsandbase.pdb)

Parameters

sy	System type.
bld	Build type.
coordsout_file	File name to write the coordinates to.
verbose	Verbosity level.

14.11 lcc_regular_mod Module Reference

Module for generating regular shapes after lattice is constructed.

Functions/Subroutines

- subroutine, public lcc_spheroid (a_axis, b_axis, c_axis, r_inout)

  For building spheroidal shapes out of a bulk lattice.

14.11.1 Detailed Description

Module for generating regular shapes after lattice is constructed.

14.11.2 Function/Subroutine Documentation

14.11.2.1 lcc_spheroid()

```
subroutine, public lcc_regular_mod::lcc_spheroid (
             real(dp) a_axis,
             real(dp) b_axis,
             real(dp) c_axis,
             real(dp), dimension(:,:), allocatable r_inout )
```

For building spheroidal shapes out of a bulk lattice.

Parameters

a_axis	Lenght in the x direction.
b_axis	Lenght in the y direction.
c_axis	Lenght in the z direction.
r_inout	Input and output coordinates.

14.12 lcc_string_mod Module Reference

Module for manipulating strings.

Functions/Subroutines

- subroutine, public lcc_get_word (string, posh, post, word)

  Cut a word from string.
- subroutine, public lcc_split_string (string, delimit, head, tail)

  Split a string in two words uning a delimiter.

14.12.1 Detailed Description

Module for manipulating strings.

14.12.2 Function/Subroutine Documentation

14.12.2.1 lcc_get_word()

```
subroutine, public lcc_string_mod::lcc_get_word (
             character(len=*), intent(in) string,
             integer, intent(in) posh,
             integer, intent(in) post,
             character(20), intent(inout) word )
```

Cut a word from string.

Parameters

string	Full string.
posh	Cut from position.
post	Cut to position.
word	Extracted word.

14.12.2.2 lcc_split_string()

```
subroutine, public lcc_string_mod::lcc_split_string (
             character(len=*), intent(in) string,
             character(1), intent(in) delimit,
             character(20), intent(inout) head,
             character(20), intent(inout) tail )
```

Split a string in two words uning a delimiter.

Parameters

string	Full string.
delimit	Delimiter.
head	First word.
tail	Last word.

14.13 lcc_structs_mod Module Reference

A module to handle the structures needed by the code.

Data Types

- type build_type

  Build type.
- type lattice_type

  Lattice type to be read and extended.

14.13.1 Detailed Description

A module to handle the structures needed by the code.

This module will be used to build and handle structures in the code.

Chapter 15

Class Documentation

15.1 lcc_structs_mod::build_type Type Reference

Build type.

Public Attributes

- character(len=20) job_name
  Job name.
- character(len=20) output_file_name
  Output file name.
- character(len=60), public coordsout_file
  Output file name for coordinates.
- character(len=60), public latticebase_file
  Lattice base file name.
- character(len=1) cut_by_planes
  Cut lattice using planes.
- character(len=1) cut_with_base
  Cut lattice after base is added.
- character(len=1) read_lattice_from_file
  Read lattice from file.
- character(len=1) use_lattice_base
  Use lattice base.
- character(len=60) cl_type
  Cluster (or solid) shape to be constructed.
- character(len=60) planes_type
  Type of planes used for the cut.
- character(len=60) seed_file
  File name for the seed used to grow a cluster.
- integer n
  Number of atoms.
- integer nplanes
  Number of planes to use in the cut.
- integer nx1
  Number of lattice points in +-(x, y, and z) directions.

- integer **nx2**
- integer **ny1**
- integer **ny2**
- integer **nz1**
- integer **nz2**
- integer seed

  Random seed.
- integer cl_number

  Cluster number (if it is a solid with "magic" numbers)
- real(dp) a_axis

  Axis length if cluster is a spheroid.
- real(dp) **b_axis**
- real(dp) **c_axis**
- real(dp) rcoeff

  Coefficient used with random seed to create noise in coordinates.
- real(dp) r_cut

  Cutoff radius to build spheroids.
- real(dp) trunc

  Truncation for solids.
- character(2) a_type

  Atom type (if specified on the input file)
- real(dp), dimension(:,:), allocatable planes

  Planes for the cut.
- real(dp), dimension(:), allocatable ploads

  Plenes weight factors.
- type(system_type) syseed

  System seed to be grow on top.
- integer ncluster

  Number of atoms in cluster/slab.
- character(2), dimension(:), allocatable atom_in

  Atoms in the cluster/slab.
- character(2), dimension(:), allocatable **atomname_in**
- integer, dimension(:), allocatable **resindex_in**
- character(2), dimension(:), allocatable **resname_in**
- real(dp), dimension(:,:), allocatable r_cluster

  Coordinates of the resulting cluster/slab.
- integer maxcoordination

  Max coordination number.
- real(dp) rtol

  Distance tolerance for distinguishing coordinates.
- integer niter

  Number of iterations.
- integer verbose

  Verbose level.
- logical center

  Center at box.
- logical reorient

  Reorient first lattice vector toward x direction.
- logical writecml

  Reorient first lattice vector toward x direction.
- logical checkperiod

To check periodicity.

- character(5) rdfpair

  To compute RDFs.
- logical writelmp

  Write LAMMPS input coordinates.
- logical interplanardistances

  Use "number of interplanar distances" as unit of measurement for plane cut.
- real(dp), dimension(:,:), allocatable slab

  To build a slab out of regular vectors.
- real(dp), dimension(:), allocatable **sloads**
- logical randomcoordinates

  To add randomness to coordinates.

15.1.1 Detailed Description

Build type.

The documentation for this type was generated from the following file:

- /home/cnegre/LCC/src/lcc_structs_mod.F90

15.2 lcc_structs_mod::lattice_type Type Reference

Lattice type to be read and extended.

Public Attributes

- character(len=3) base_format

  Lattice basis.
- character(len=60) primitive_format

  The lattice primitive format (Angles of Vectors)
- character(len=60) type_of_lattice

  Type of lattice (sc, bcc, fcc, and triclinic)
- real(dp) angle_alpha

  Angles for triclinic lattice.
- real(dp) **angle_beta**
- real(dp) **angle_gamma**
- real(dp) h_lattice_a

  abc parameters for lattice
- real(dp) **h_lattice_b**
- real(dp) **h_lattice_c**
- real(dp), dimension(:,:), allocatable lattice_vectors

  Lattice vectors.
- real(dp) volr

  Volume of the cell.
- real(dp), dimension(:,:), allocatable recip_vectors

  Lattice reciprocal vectors.

- real(dp) volk

  Volume of the reciprocal cell.
- integer nbase

  Number of atoms in the basis.
- character(2), dimension(:), allocatable base_atom

  Basis atoms.
- real(dp), dimension(:,:), allocatable r_base

  Basis coordinates.
- type(system_type) sybase

  System for the basis.
- logical bsopl

  If there are symmetry operations to be performed.
- integer nop

  Number of Symmetry operations.
- real(dp), dimension(:,:), allocatable bstr

  Translations to be performed.
- real(dp), dimension(:), allocatable bsopload

  Scaling factos (load) for the translation.
- real(dp), dimension(:,:), allocatable bssym

  Symmetry operation (diagonal)
- integer, dimension(:), allocatable spindex

  Spicies index.
- real(dp), dimension(:), allocatable base_mass

  System basis masses.
- integer, dimension(:), allocatable resindex

  Residue index.
- real(dp), dimension(:,:), allocatable bulk

  To save the "bulk" positions.
- logical check

  Check lattice.
- logical getopttrs

  Get optimal translations at symmetry operations.
- logical randomlattice

  To add randomness to each lattice position.

15.2.1 Detailed Description

Lattice type to be read and extended.

The type of lattice read from input.

The documentation for this type was generated from the following file:

- /home/cnegre/LCC/src/lcc_structs_mod.F90

Index