BUDDHA, *DHAMMAPADA*

SIDDHARTH KANUNGO

DISCRETE ELEMENT METH

RYAN GOSSELIN, NICOLAS ABATZOGLOU

UNDER THE GUIDANCE OF RYAN GOSSELIN, NICOLAS ABATZOGLOU

MITACS GLOBALINK RESEARCH INTERNSHIP

Submitted November 2015

Contents

List of Figures

List of Tables

To my rational brain, for always being
there for me, despite all the shaky evidences
pointing otherwise.

Acknowledgement

This Documentation is a part of the project that I worked on during the summer of 2015 at the PAT-Pfizer Chair in Université dé Sherbrooke. This document wouldn't have been possible without the guidance of Prof. Ryan Gosselin and Prof. Nicolas Abatzoglou.

I would also like to also like to thank all the Lab Members of the Research Group, who helped me generously during my internship and graciously welcomed me into their part of the life.

A special thanks to Pedro Duraó, Philippe Kikongi, Himmat Dalvi, Francis Lavois, Ounaima and

I would also like to thank all the people I met during the internship, who helped me settle in and feel welcome at the same time. At last I would like to thanks MITACS for funding this incredible journey.

This document was submitted much late than it was supposed, even though it was almost completed very early. I would sincerely apologize for that.

Siddharth Kanungo

Working With Ubuntu

Introduction

It is very essential that one should learn about the operating system that one would be working with while using *LIGGGHTS*. Unlike Windows, which offers the least learning curve to adequately use the Machine, in Linux-based Operating systems, there is a quite a bit of learning curve and for someone who has never been exposed to Unix based system, this can seem overwhelming. However, in order to work with LIGGGHTS, we will use an operating System called *Ubuntu*, which was designed keeping in mind the ease of using Windows in the first place. Therefore, apart from a few basic commands, which will become intuitive with progressive use, there are not many commmands that one must learn to use *LIGGGHTS*. The following section will give introuction to the history of Ubuntu, Linux and basics of computing in general. The reader may skip and proceed directly to subsequent sections, without any loss of continuity.

Operating Systems

An operating system is a piece of software you can install on a computer whose purpose is to manage the hardware so that other programs you use don't have to do it. That means programmers of Microsoft Word do not have to worry about what kinds of graphics card you have in order to display letters on your screen since the operating system takes care of that for it. This makes it easier for programmers to make software that works together.[1] *Linux, Unix, and Ubuntu* are all operating systems, however there are subtle differences between them. For example Ubuntu is a kind of Linux, and Linux is basically UNIX rewritten from scratch.

[1] This has edited from the discussion present here

A kernel is the *core* part of an operating system. Microsoft's Windows has a kernel as well, albeit obviously a different one from UNIX and Linux. The kernel does all the very low-level dirty work of the operating system like decide how files are stored to your disk or providing an interface to other programs that allow them to output sound without having to write new code for each type of sound card out there. Aside from a kernel, other things are often included with an operating system and are considered part of the package.

For example, Windows comes with a graphics interface that is not really separable from it whereas Linux can run without a graphical interface just fine.

Unix & Linux

UNIX IS AN OPERATING SYSTEM that was originally created at Bell Labs in the late 1960s/early '70s. In the '70s and '80s it became very popular, and various derivative versions were made - some by commercial companies.

In the early 1990s, Linus Torvalds created *Linux* while he was at university in Helsinki. Linux isn't based on Unix, but is more like a clone or remake. So to the user it mostly behaves the same as Unix, although technically it isn't derived from Unix. Linus wanted there to be a free version of Unix that could be used on PCs (at the time most versions of Unix cost money and required expensive computers). It became very popular, other people contributed code to it and it is now one of the most-used operating systems in the world. *Linux* itself is just a kernel. As explained in the previous section, the kernel is the core of the operating system - it "sets the stage" on which everything else runs. The complete system is often referred to as "Linux", but it's actually Linux plus a whole load of different pieces of software from different projects. There are many different LINUX DISTRIBUTIONS (OR DISTROS). A DISTRO takes all of these different components and assembles them into a complete system that you can install on your computer. Although the end result is usually largely the same, different distros do things in slightly different ways. Ubuntu is one example of a distro. Others would be Debian, Fedora, Gentoo, etc.

The Ubuntu Story

UBUNTU IS AN ANCIENT AFRICAN WORD meaning HUMANITY TO OTHERS. It also means *I am what I am because of who we all are*. The Ubuntu operating system brings the spirit of Ubuntu to the world of computers.[2]

When did it all began? *Linux* was already established as an enterprise server platform in 2004, but free software was not a part of everyday life for most computer users. That is why *Mark Shuttleworth* gathered a small team of developers from one of the most established Linux projects Debian and set out to create an easy-to-use Linux desktop: **Ubuntu**.The vision for Ubuntu is part social and part economic: free software, available to everybody on the same terms, and

[2] For more information on Ubuntu, the reader can visit here.

funded through a portfolio of services provided by Canonical.

Open Source and Free Software

Open Source Movement and Free Software movement[3]both stem from the belief that software is a creative tool and it should be always available to each and every member of society to tinker with and expand on. Keeping in mind, Ubuntu meets the goal of free and open source software.

[3] For more information on philosophy of Ubuntu, the reader can read the philosophy of Ubuntu

There is a distinct design philosophy when someone from Windows [4] leaves its convienient shell to work in Linux-based environment starts to notice. These maybe, the subtle changes that one may notice in the working around the operating system . Although, these are primary essence of the user about the way he/she treats the work environment, sometimes it is worth the investment to understand why something is the way it has turned out be.

[4] There is an excellent write up on the philosophy of working in an Windows Environment to that of Linux Environment. It was written primarily for users who were switching from Windows to Linux for the first time. It is highly recommended to give it a read

I hope that this short introduction will help the user to understand the key points of emergence of Linux as an alternative and therefore make the learning, a different and fun experience altogether.

Terminal

The terminal is an interface in which you can type and execute text based commands.It can be much faster to complete some tasks using a Terminal than with graphical applications and menus. Another benefit is allowing access to many more commands and scripts. A common terminal task of installing an application can be achieved within a single command, compared to navigating through the Software Centre or Synaptic Manager. [5]

[5] This has been taken from discussion here

Getting into the Terminal

There are two ways to to get into terminal.

- Click on the Dash (Windows Key/ Super Key)
- Type terminal and wait for it find the application
- Click on Terminal to access the Terminal.

Alternatively, the terminal can also be accessed by using the shortcut

```
CTRL + ALT + t
```

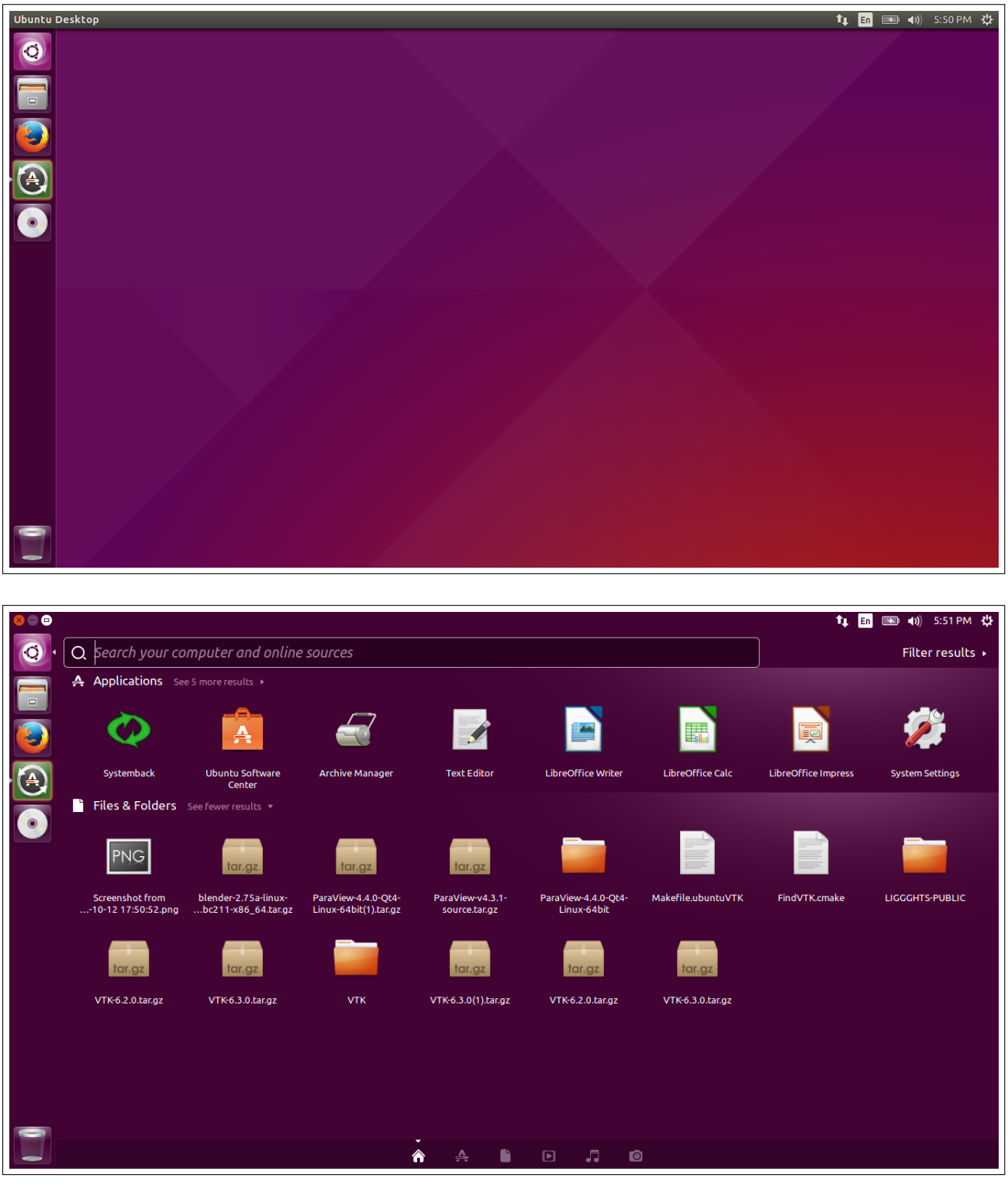
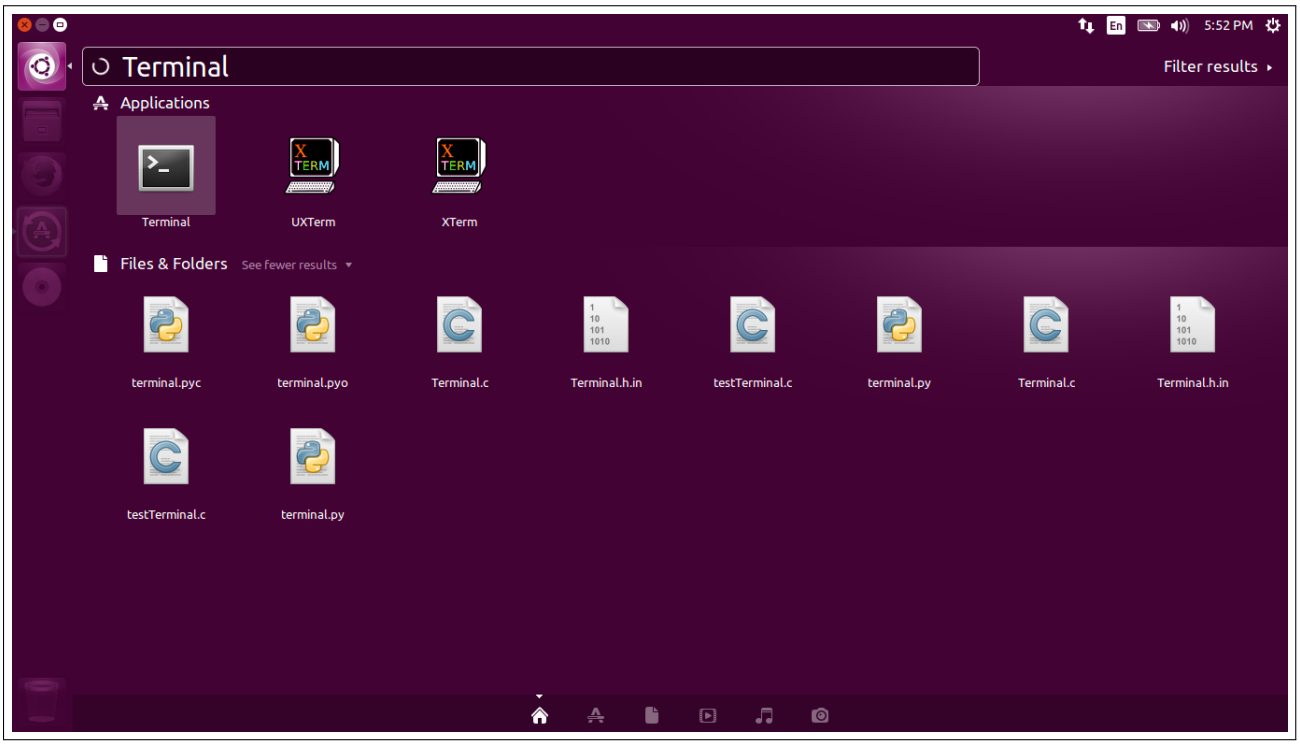


Figure 1: *Side-bar* is similar to start-menu of WINDOWS. *Dash* or start menu if you may call, gives access to the list of applications and settings installed the OS



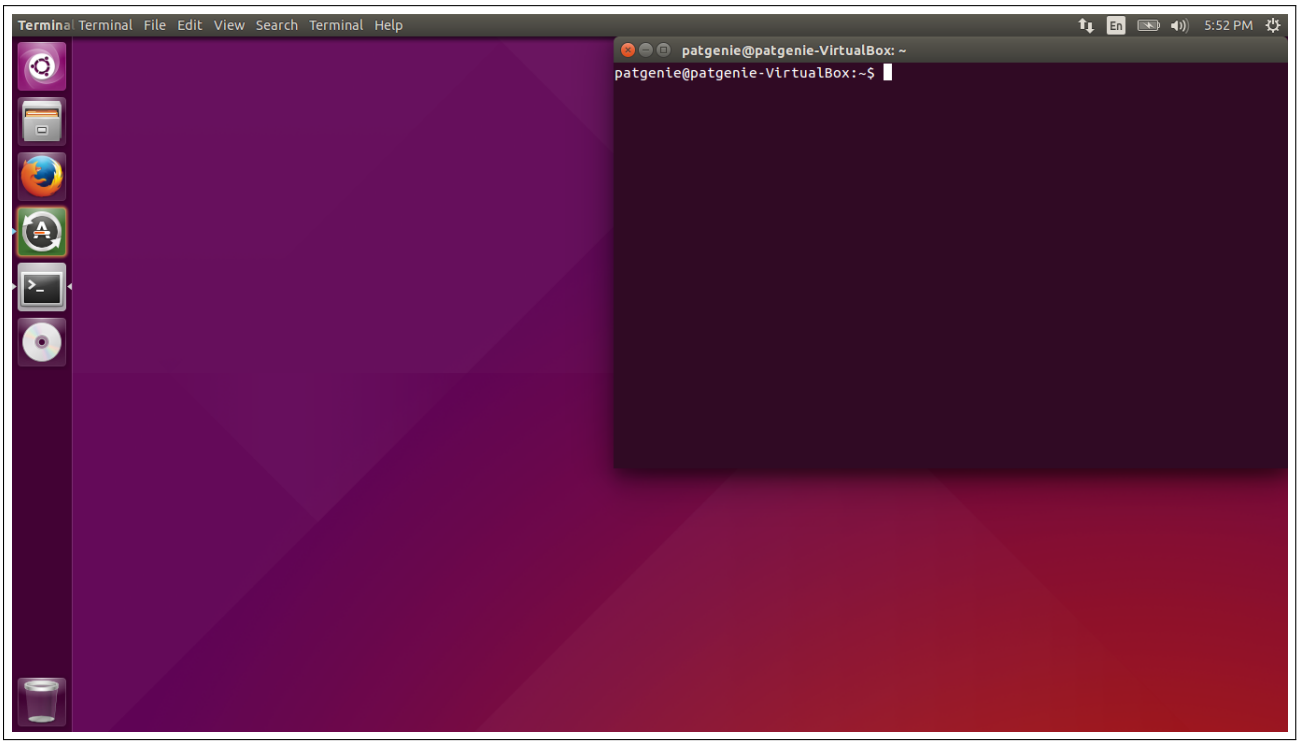


Figure 2: In the *Dash*, type *Terminal* and click on the Terminal Icon that appears. The terminal windows is shown in the above figure.

Command	Usage
cd ..	change directory and goes one folder up
cd <*foldername*>	change directory and go the specified folder
cd <*folderName1*>/<*folderName2*>	change directory and goto the specified folderName2 which is in folderName1
cd ../<*folderName*>	change directory and go one folder up and go to the specified folderName
ls	List all files and folders
ls *-a*	List all files and folders including the hidden folders
pwd	Find out the location of current directory

Table 1: Commonly Used Commands

Common Commands

The number of useful commands in UBUNTU while working with *LIGGGHTS*is very limited. Therefore it is advisable to learn all the commands for better productivity and speeder navigation around the OS. It is also advisable to try each command in the shell for better and quick understanding.

Understanding filestructure of Ubuntu

THE FILESYSTEM OF UBUNTU is somewhat different from that of Windows. Ubuntu (like all UNIX-like systems) organizes files in a hierarchical tree, where relationships are thought of in teams of children and parent.[6] Directories can contain other directories as well as regular files, which are the "leaves" of the tree. Any element of the tree can be references by a path name; an absolute path name starts with the character / (identifying the root directory, which contains all other directories and files), then every child directory that must be traversed to reach the element is listed, each separated by a / sign.[7]

[6] HowToGeek has also a brief article about understanding the filestructure.

[7] To understand more, visit this page

Copying, Paste, Delete

UNDERSTANDING FILESTRUCTURE is useful to learn about copying, pasting and deleting new files and folders.

Commands Usage	Description
cp *sourceFileName destinationFoler*	Copy files from one source to the Folder mention
rm *sourceFileName*	Delete the fileMentioned
rm -rf <*FolderName*>	Delete the folder recursively
mkdir <*folderName*>	Make a new directory
mkdir <*folderName*>*{1..n}*	Make *n* number of directory

Table 2: Copying and Moving

More Common Commands

After working on the terminal for some time, the terminal screen often gets cluttered. It is often good practice to clear the screen for better readability. Just as in windows, files and folders are often compressed and packaged in rar and zip. However, more common format include tar.gz. (See table 3)

Commands Usage	Description
clear	Clears the whole terminal screen
tar -xzvf <*file.tar.gz*>	Unzip a tar file
apt-get install <*softwareName*>	Install a software

Table 3: More common commands

There are many restricted areas where the user cannot do some of the above operations such as removing files or creating folders. These restrictions need administer level priviledges.[8]

Therefore the above commands can be prefixed with ś́udo ́to use the administer level functionality.

[8] *For e.g sudo apt-get update*

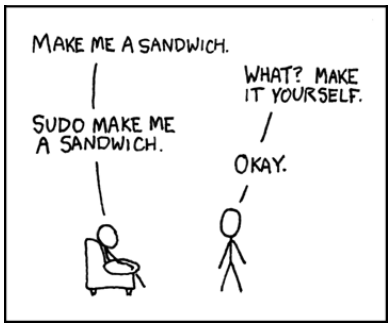


Figure 3: xkcd shows what-if linux was real world

Further Resources

As you can notice this is very rudimentary introduction to Ubuntu's command line. If you did not have enough time to go through the commands or you did not have access to linux machine, it is fine. As we will proceed further to use *LIGGGHTS*, we will revise the commands and what it does.[9]

Also there is available a very comprehensive cheatsheet or refcard where many important commands are written in tabular form. It is recommended to go through the list and test whichever looks interesting.

[9] There are many online resources available to acquaint oneself with basic ubuntu commandline. This is also a good discussion on how to learn the commands

Working with LIGGGHTS

Introduction

In this section, we will get to speed by directly diving deep into working with *LIGGGHTS*. As *LIGGGHTS*does not have a graphical user interface, the commands that we saw in the previous section (see TABLE 1, TABLE 2 and TABLE 3).

In this section, we will do the following:

- Run example scripts of *LIGGGHTS*
- Convert the results into *VTK Format* using *LPP.py* and understand the need for it
- Post Process or View the results in the *Paraview* Software

Example Scripts

THE EXAMPLES scripts are present default in the folder:
It can be accessed by
All the steps are shown in the Figure 9

- First going to home folder using cd ~
- Then using typing cd *LIGGGHTS*-PUBLIC. As mentioned earlier you use TAB Completion to save time while writing.
- ls to show the files and folders
- Again type cd examples to change directory to examples script
- ls to show the files and folders
- Type cd *LIGGGHTS* and cd tutorials to reach example scripts
- Type ls to show the list of examples

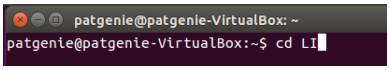


Figure 4: Typing in the terminal

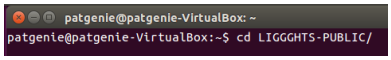


Figure 5: On pressing the TAB Command

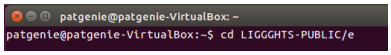


Figure 6: Using TAB Again

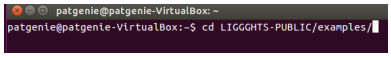


Figure 7: Demonstration of Auto completion command

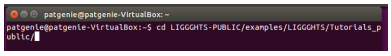


Figure 8: Getting into Examples Folder

Figure 9: Press enter

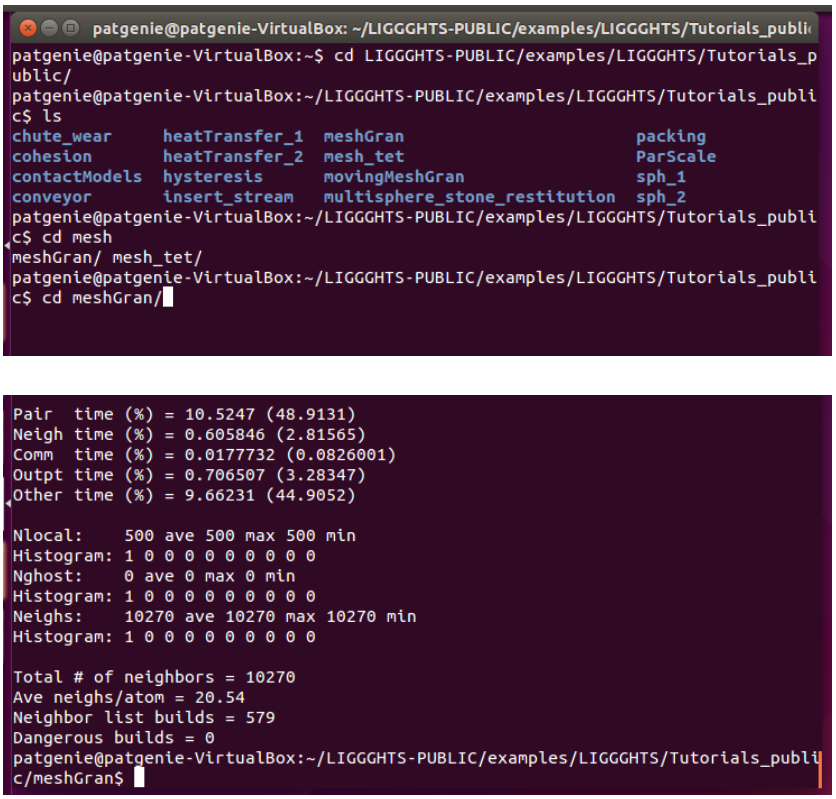


Figure 10: Successfully converted all the files

Running LIGGGHTSScript

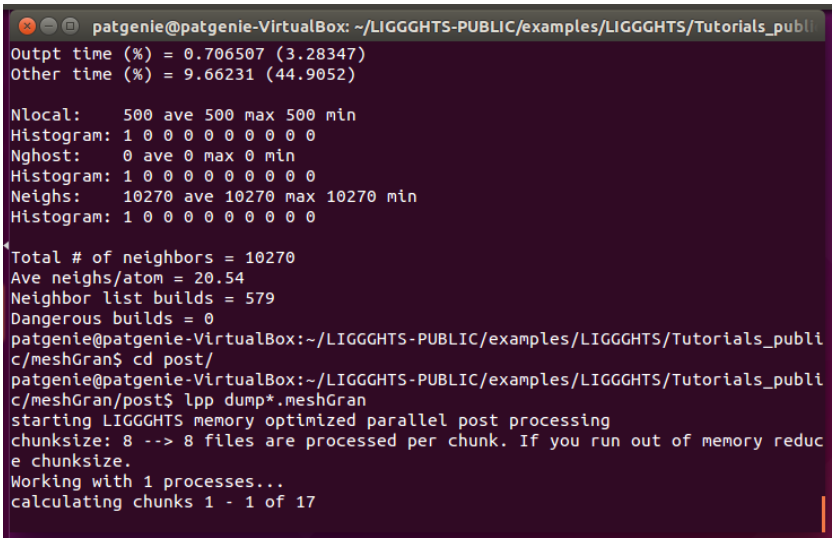
In this section we will run the script of MeshGran. A *LIGGGHTS*simulation file can be run in the following method:

- In the Examples folder, change directory to the MeshGran using cd meshgran
- Type ls to show the contents of the folder
- The in.meshGran file is the input script for *LIGGGHTS*simulation
- Run the script using liggghts < in.meshGran
- If there are any errors in the input script, the task would show the error and close out.
- If there are no errors, then you should successfully written command

See Figure 13, 14 & 15 for instructions.

IF EVERYTHING went smoothly, congratulations, you have run your first *LIGGGHTS*script file successfully.

Converting the files



The LPP command can be started with ***lpp dump*.meshGran****in the post folder of the LIGGGHTS*

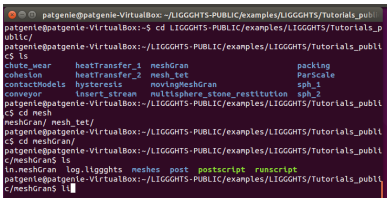


Figure 11: Type li into the shell prompt and hit

Figure 12: Type liggghts < in.meshGran

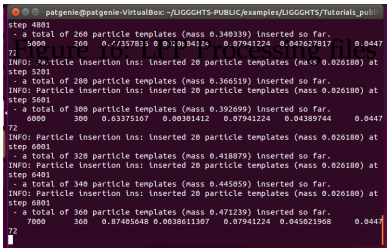


Figure 13: *LIGGGHTS*should begin executing the file

Figure 14: To view results, goto the post folder

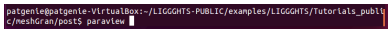


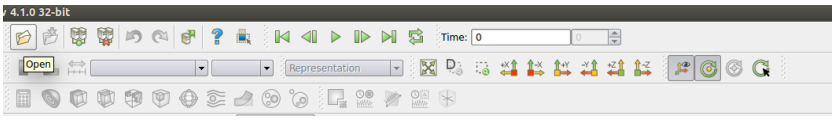
Figure 15: Type paraview, to enter paraview

ALTHOUGH we completed the running the input script file in the last section, we won't be able to see what we have done as the file generated by the *LIGGGHTS*input file is compatible with any software. To get around this, we first need to convert the files generated to a format that we can use to visualise the results that we got. In our case, the format we use will be VTK. VTK stands for *Visualisation Toolkit* Let us see, how we can convert the files into VTK file format

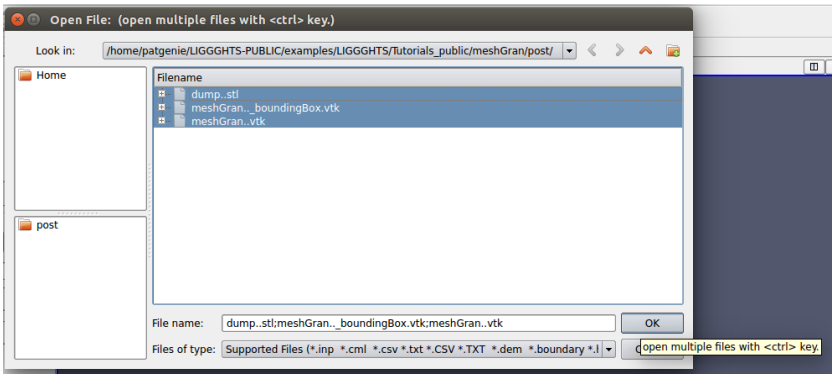
The following method should be used:

- Once the files are converted, the *LIGGGHTS*files are usually stored in the post folder. Therefore, change directory to Post folder using command cd post. *(See Figure 16)*
- In the post folder, type lpp dump*.meshGran to convert all the timesteps generated into VTK format.
- Once the operation is complete, you should see it has successfully completed the operation.

Using Paraview



Now that we have got the files and folders in the format that is compatible, we can view our results in the Paraview software.



We can enter Paraview and use it by the following method:

- Type paraview in the post folder
- Click on the Open Icon, and the select all.(See Figure 22)
- Click Apply on the property bar shown in the Figure 20.

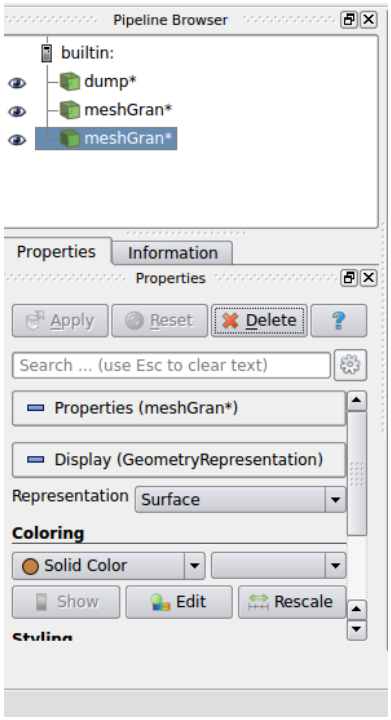


Figure 17: Properties Browser

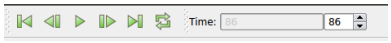


Figure 18: Playback Bar

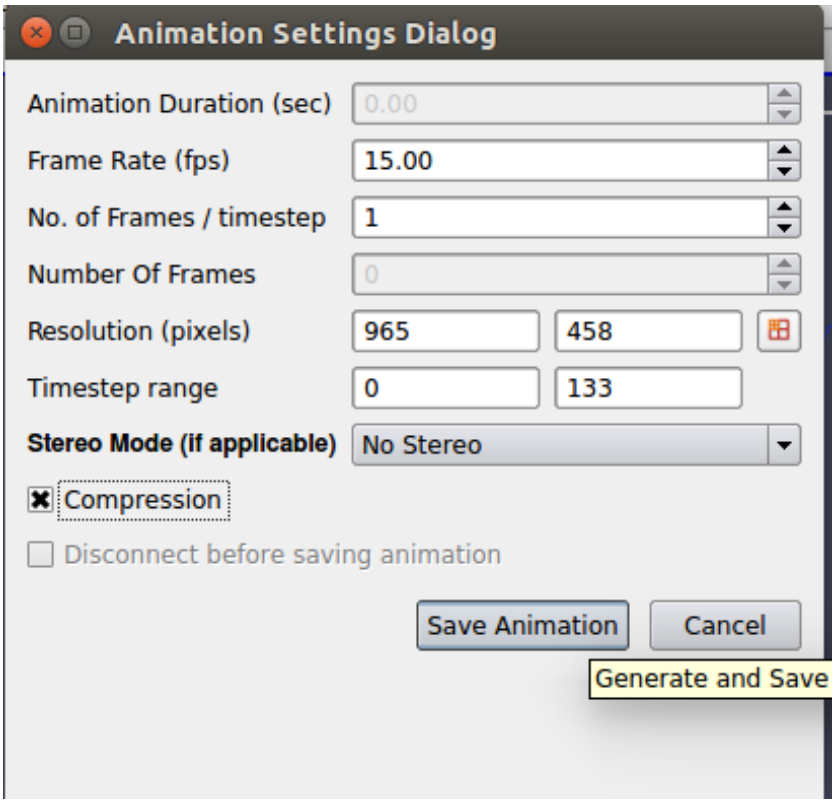


Figure 19: Saving Animation

- Once, the files and setting have imported into the region, you can begin by clicking on the play button on the playback bar.
- You can also save the animation in *File->Save Animation*
- You can choose the format, avi((if available)).

Opening the input file

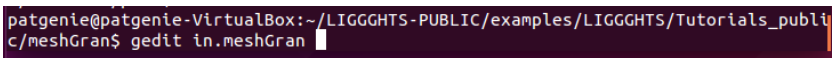


Figure 20: Opening the file using command prompt

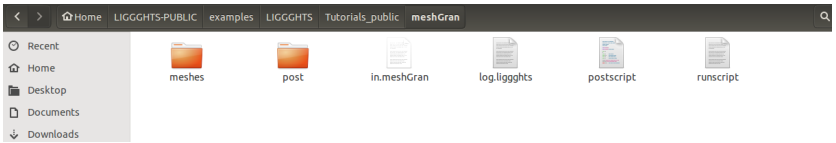


Figure 21: Opening the files using File Manager

You can open the input file using the shell prompt, as shown in the figure 24. Also you can open the file using a file manager shown

in the figure 25. The default application for text editing in ubuntu is an application of *Gedit*.

Different Files and format

Before delving into the file structure, there are few file formats that needs to be explained.

- **dump*.meshGran:** It is the dump file generated by *LIGGGHTS*at a given timestep. All the specification that you write to dump is dumped here. We will see ahead how you can generate. The stands for all them. There is a dumpfile for each timestep interval. Therefore while converting in LPP, we use the asterisk() as a placeholder for everyone of them. You can open and have a look what's inside to get a better idea about this file.
- **dump.vtk:** After running the LPP converter script, these are the output we get. These can read by Paraview.
- **meshGran.stl:** The stl/cad files that you import during the script are store in these scripts. unlike the .meshGran format which is unreadable by Paraview, these are perfectly readable by paraview. Like the dump files, sometimes these are also store on regular timestep interval, the idea behind being if there is motion of the stl file for e.g rotation of feedframe, the location and position should be stored somewhere to visualise it later.

In the meshGran folder

- **in.meshGran** is the main input script
- **mesh** is the folder where stl files are kept
- **post** is the folder where generated files are kept

Anatomy of input file

NOW THAT we have run an example script, we are in a good position to understand what is inside the *LIGGGHTS*input file. Let us first look into the meshgran input file, for better understanding.

```
# Wall import from CAD
atom_style        granular
boundary        m m m
newton               off
atom_modify       sort 0 0

communicate       single vel yes
units               si

region               reg block -0.01 0.51 -0.06 0.01 -1.22 0.01  units box
create_box       1 reg

neighbor        0.02 bin
```

```
neigh_modify        delay 0

#Material properties required for new pair styles

fix                 m1 all property/global youngsModulus peratomtype 5.e6
fix                 m2 all property/global poissonsRatio peratomtype 0.45
fix                 m3 all property/global coefficientRestitution peratomtypepair 1 0.7
fix                 m4 all property/global coefficientFriction peratomtypepair 1 0.05
fix                 m5 all property/global characteristicVelocity scalar 2.

#New pair style
pair_style gran model hooke tangential history #Hooke without cohesion
pair_coeff        * *

timestep        0.00005

fix                 1 all nve/sphere
fix                 2 all gravity 9.81 vector 0.0 0.0 -1.0

#import triangular mesh
fix                 cad all mesh/surface file meshes/mesh.stl type 1 &
                          scale 0.001 move 0. 0. 0. &
                         rotate axis 1. 0. 0. angle -90. #temperature 100.

#use the imported mesh as granular wall
fix                  granwalls all wall/gran model hooke tangential &
                   history mesh n_meshes 1 meshes cad

#definition of insertion face
fix                 inface all mesh/surface file meshes/insertion_face.stl type 1

#distributions for insertion
fix                 pts1 all particletemplate/sphere 1 atom_type &
                   1 density constant 2500 radius constant 0.005
fix                 pdd1 all particledistribution/discrete 1.  1 pts1 1.0

group                nve_group region reg

#particle insertion
fix                 ins nve_group insert/stream seed 5330 distributiontemplate pdd1 &
                         maxattempt 100 nparticles 10000 particlerate 1000 &
                          overlapcheck yes all_in no vel constant 0. 0. -1.0 &
                         insertion_face inface extrude_length 0.02
```

```
fix                ts all check/timestep/gran 1000 0.1 0.1
compute             1 all erotate/sphere
thermo_style       custom step atoms ke c_1 f_ts[1] f_ts[2] vol
thermo              1000
thermo_modify      lost ignore norm no
compute_modify      thermo_temp dynamic yes

#dump commands
dump               dmp all custom 300 post/dump*.meshGran id type &
                  type x y z ix iy iz vx vy vz &
                  fx fy fz omegax omegay omegaz radius
dump                dmpstl all mesh/stl 300 post/dump*.stl

#insert particles
run                       10000 upto
unfix              ins

#run
run                       40000 upto
```

THE INPUT SCRIPT can be divided into four parts.

- **Initialisation** Setting the parameters that are needed to defined before the particles can be created.
- **Properties** Defining the properties of materials, particles to be inserted, geometry, defining walls and particle generation.
- **Detailed Settings** Defining settings that correspond to output, the integreting method to be used etc
- **Execution** The actual run command that executes the simulation.

Initialisation

COMMANDS THAT YOU CAN ALTER: **boundary, units, region, create_box, neighbor and neigh_modify**

```
# Define the style of atom that you would like to simulate
atom_style        granular
# Set the boundary to movable i.e as atoms cross the boundary, the boundary expands
boundary        m m m
newton             off
atom_modify       sort 0 0
communicate       single vel yes
# Set the unit system to SI units
units              si
# Define the region in space where simulation will occur
region              reg block -0.01 0.51 -0.06 0.01 -1.22 0.01  units box
```

```
# Specify the number of atoms that will be involved and the region command
create_box        1 reg

neighbor        0.02 bin
neigh_modify        delay 0
```

COMMANDS THAT IS BEST LEFT AS IT IS UNLESS YOU KNOW WHAT YOU ARE DOING: **atom_style, newton, atom_modify**

Properties

```
  /*Material properties required for new pair styles. Notice the format is
  same for all the properties. It shows the important properties required
  for a basic granular simulation.*/

fix                m1 all property/global youngsModulus peratomtype 5.e6
fix                m2 all property/global poissonsRatio peratomtype 0.45
fix                m3 all property/global coefficientRestitution peratomtypepair 1 0.7
fix                m4 all property/global coefficientFriction peratomtypepair 1 0.05
fix                m5 all property/global characteristicVelocity scalar 2.

/* Import triangular mesh from a cad file. It can scaled, moved and rotated */
fix                cad all mesh/surface file meshes/mesh.stl type 1 scale 0.001 &
               move 0. 0. 0. rotate axis 1. 0. 0. angle -90.

/* We will use the previously imported mesh as a wall */
fix             granwalls all wall/gran model hooke tangential history mesh &
               n_meshes 1 meshes cad

/* We can insert particles by defining an insertion face.
This can be done by either importing a cad file and using
it as insertion face or using primitive shapes from \Li library*/
fix                inface all mesh/surface file meshes/insertion_face.stl type 1

#Define a basic template of a particle to be inserted
fix                pts1 all particletemplate/sphere 1 atom_type 1 density constant 2500 radius constant
# Using the template defined previously, create distribution
fix                pdd1 all particledistribution/discrete 1.  1 pts1 1.0
#Group the particles together
group               nve_group region reg

/*Now the insert the particles using distribution template and insertion face
defined earlier. We can also give addition info such as velocity.
We can insert the particles using mass, volume or number. */
fix                ins nve_group insert/stream seed 5330 distributiontemplate pdd1 &
               maxattempt 100 nparticles 10000 particlerate 1000 overlapcheck yes&
               all_in no vel constant 0. 0. -1.0 &
```

```
                insertion_face inface extrude_length 0.02

#New pair style
pair_style gran model hooke tangential history /*Hooke without cohesion*/
pair_coeff * *
```

Here the important thing to understand is you don't have to understand everything. Here is a very high level overview to understand the input file. However, while making your own script, you will mostly be tweaking these parameters/settings.

Very rarely you would need to change these settings

Detailed Settings

```
/* Define the timestep equal to 0.001 sec.
    The unit depends on the SI unit used.
    It is recommended that the simulation should
    be 20\% or lesser than a Rayleigh TimeStep*/
timestep        0.00005

/* Define the timestep equal to 0.001 sec.
    The unit depends on the SI unit used.
    It is recommended that the simulation should
    be 20\% or lesser than a Rayleigh TimeStep*/
    fix                1 all nve/sphere

/* For every particle, we define a property gravity
  acts in the direction of the vector specified */
 fix                2 all gravity 9.81 vector 0.0 0.0 -1.0

 /* Check the time step and compare it with Rayleigh.
  Give error if its greater than 0.01 \% greater.
  Initialize dump by running 1 timestep. Is recomended to do so.
  Unfix the checking after once. The result is written in log file*/
 fix                ts all check/timestep/gran 1000 0.1 0.1

/* Compute the rotational properties of sphere */
compute                1 all erotate/sphere

/*Describe quantities to be printed on logfile
  and the output screen. The number of interval to the
  write the thermodynamic quantities.
  Lost _Ignore ignore lost particle */
thermo_style        custom step atoms ke c_1 f_ts[1] f_ts[2] vol
thermo                1000
thermo_modify        lost ignore norm no
compute_modify        thermo_temp dynamic yes

/*Generates a set of dump files
  that contain information for imaging
```

```
  the system. Give the interval and file type to save.
  This generates huge files, therefore it is
  recommended to dump properties that are necessary.  */

dump                dmp all custom 300 post/dump*.meshGran id type type x y z  radius
dump                 dmpstl all mesh/stl 300 post/dump*.stl
```

Execution Commands

```
#insert particles
run                          10000 upto
/*Unfix the insertion i.e stop the insertion */
unfix                ins

#run
run                          40000 upto
```

These commands are final commands, which brings the input script into action. The 10000 number is the *number of Timesteps*. Considering we took timestep 0.00005 The number of seconds in the real world would be numberOfTimeSteps x TimeStep = 0.5 seconds

Summary

THIS WAS AN EXAMPLE SCRIPT that was provided in with *LIGGGHTS*. Similar to mesh Gran example, there are a lot of different examples currently in the the examples folder of the *LIGGGHTS*. It is recommended that the reader must at least familiarise with most of them, by simulating and reading the source code and understanding. Understandably there are many commands that one who is not familar with *LIGGGHTS*will be able to understand . In that case, the reader should visit and look up the reference manual.[10]

[10] The *LIGGGHTS*reference manual is an exhaustive manual with all the possible commands. It is not possible to read the entire manual and it is not recommended to attempt so. However, it is advisable to read the sections which where introduced in this particular chapter.

Exercises

The following exercise can be done after this particular chapter to become much familiar with the material introduced in the chapter.

1. Run all the example scripts accompanied with LIGGGHTS.
2. Tweak parameters and notice the effect in the simulation: **radius, run, disable the unfix, boundary, units**

Making your own simulation

The most important part of this documentation is how you can go on and design your own simulation using *LIGGGHTS*. In the chapter we will go over some of the steps that will enable you to do that.

Pre-requisites

There are few pre-requisite or preliminary data that needs to be collected before creating a *LIGGGHTS*Simulation. The following constitutes the

- **Number of Different Atoms** In *LIGGGHTS*, walls are basically composed of atoms/particles. The properties assigned to these atoms acts for the wall as well. Therefore if you will working with, lets say three powder, the number of different atoms will be *3 + 1 = 4*
- **Radius of each particle size** The size of particles. Either a constant value or a Gaussian Value with Standard Deviation and Mean size.
- **Density of particles** The true density of particles. Similar to the radius, either a constant value or a Gaussian Value with SD and Mean size.
- **Coefficient of Friction** The Coefficient of friction requires symmetric matrix, with combination of different particles with each other.

$$COF = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}$$

Figure 22: Symmetric Matrix

COF is a symmetric matrix as$P_{ab} = P_{ba}$ for the coefficient of friction between two particles.

- **Coefficient of Restitution** The Coefficient of Restitution also requires a symmetric matrix, with the combination of different particles with each other.
- **Poisson's Ratio** A constant value for each particle.
- **Young's Modulus** A constant value for each particle.

- **Cohesion or No Cohesion** In case using cohesion, then the value of Cohesion Energy Density is required.
- **The CAD File** and the **dimensions** and orientations are needed to properly set up.
- **The Domain box** It is the region where the simulation will occur. It is an imaginary control volume in space, in which all the particles and geometry is kept. If the particles go out of the domain box, then the simulation stops, unless specified to ignore lost particles. The ***boundary*** command sets up the dimension of the domain box. It should be noted that this should be minimum as possible and should completely engulf the geometry. If you have a system where the atoms will move past the boundary, then it is recommended to have the domain be movable.

Steps Involved

NOW THAT YOU have all the information, we can move along to create our own first simulation. In this particular section we will re-construct the feedframe and understand, how one might do so from scratch.

- Define your objective
- Setting up meshing and domain box
- Collecting Particle Properties information
- Write the Initialization Part
- Write the Properties Setup
- Write the Detailed Settings
- Write the Execution and further Settings

We will walkthrough all of that using the following two settings. Next Stop. ***The FeedFRAME Junction***

The Tale of FeedFrame

Feedframe is a part of tab-letting process in the drug-manufacturing process of pharmaceutical industries. Our goal here will be emulate the behaviour of feedframe with a binary mixtures (in our case beads), and make note of the segregation tendencies that results.

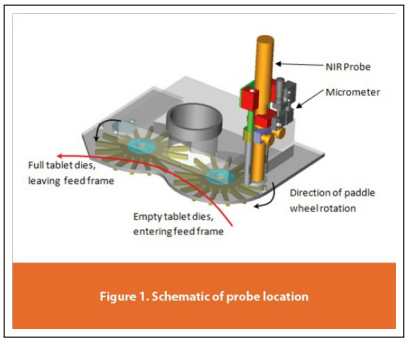


Figure 23: Graphical Representation of feedframe.Image taken from the paper

Defining our objective

We will have the following objectives.

- Filling the feedFrame with beads uniformly
- Rotating the feedframe with some angular velocity to check their effect

Setting up the meshing and domain box

Once we have defined our objective, we can now work towards where we will fill the particles. That requires to first examine our geometry that we have and construct an insertion face.

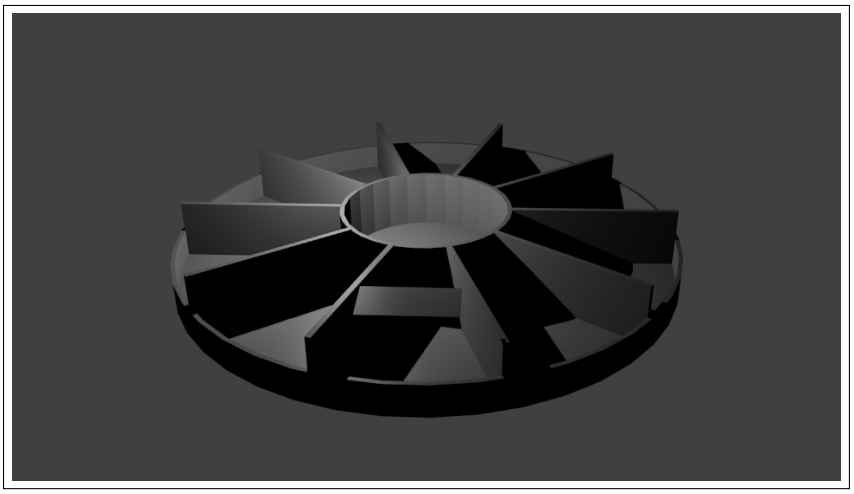


Figure 24: Main View

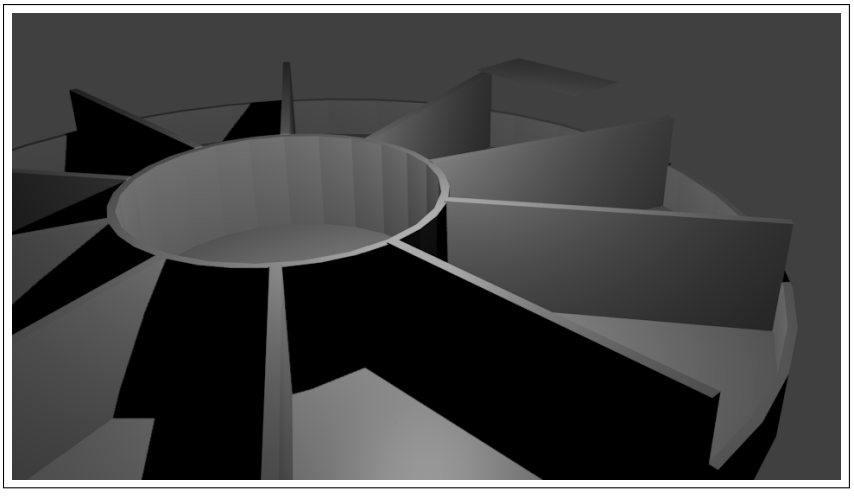


Figure 25: Side View

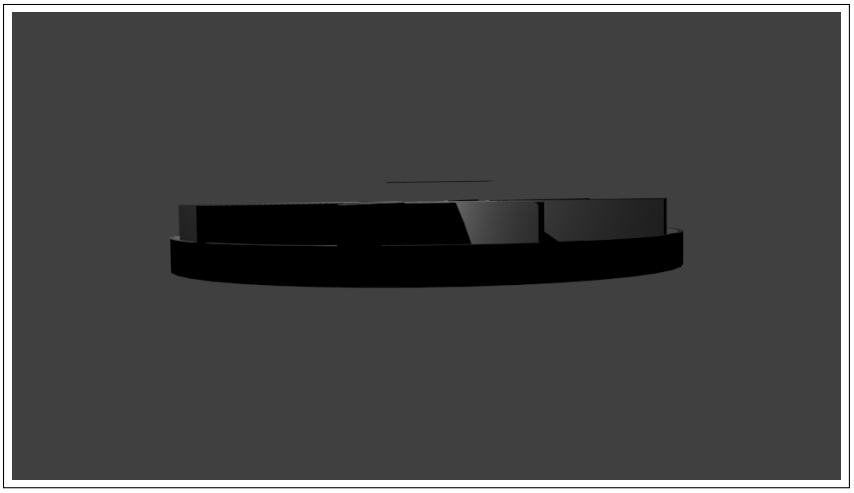


Figure 26: Another Side View

In the Figure 24, 25 and 26, we can see three objects. There is the **outer base, an inner paddle** and a **rectangle face** just right up top of a paddle.

The rectangle face will act as our insertion face i.e the region from where the particles will generated and dropped into the simulation box.

THE WHOLE SETUP was created in blender, which is an excellent open source free software for *Modelling, Meshing, Rendering, Visualising* and lots more. This documentation wouldn't cover CAD Modelling, however there are tons of resources online, which can you get you started working on it.

The important thing to remember is the following:

- The geometry should be at the origin
- The relative position of multiple geometries should be carefully noted.
- The dimensions of the geometry should be noted.

The geometry should be at origin because, *LIGGGHTS*is completely based on commandline. As there are no GUI available, it is often difficult to get an idea where our imported geometry is. Keeping the geometry to origin makes sure that our geometry land right at the origin.

The following are the dimensions obtained:

There are no dimension associated with an stl file. It is usually set by the *Units* command in *LIGGGHTS*. Therefore we will have *24 m* for an SI unit while *24 cm* for CGS

Collecting particle properties information

Direction	Highest Magnitude
X	24
Y	24
Z	3.5

Table 4: Dimension Info

	Properties of Particles		
Properties Name	P_1	P_2	Wall
Young's Modulus	$5X10^7$	$5X10^7$	$5X10^7$
Poissons Ratio	0.45	0.45	0.50
Radius	0.001	0.001	0.001
Density	7800	1190	2000

Table 5: Properties Table

$$COF = \begin{bmatrix} 0.7 & 0.65 & 0.68 \\ 0.65 & 0.60 & 0.68 \\ 0.68 & 0.68 & 0.3 \end{bmatrix}$$

$$COR = \begin{bmatrix} 0.25 & 0.24 & 0.23 \\ 0.24 & 0.35 & 0.23 \\ 0.23 & 0.23 & 0.22 \end{bmatrix}$$

Figure 27: Coefficient of Restitutions and Friction

Writing the Initialisation

```
/*Initialisation*/
atom_style   granular
atom_modify  map array
boundary     f f f
newton       off
/*processors  1 1 2 This is commented as the we are
not using parallel processing */

communicate  single vel yes

units        si

/* cylinder args = dim c1 c2 radius lo hi
    dim = x or y or z = axis of cylinder
    c1,c2 = coords of cylinder axis in other 2 dimensions (distance units)
    radius = cylinder radius (distance units)
    radius can be a variable (see below)
    lo,hi = bounds of cylinder in dim (distance units)

    region void cylinder y 2 3 5 -5.0 EDGE units box*/
region       reg cylinder z 0 0 0.125 -0.03 0.08 units box

create_box   3 reg

neighbor     0.001 bin

neigh_modify delay 0
```

For more information on how to setup the region, visit here

The following needs to written during the initialisation. These commands are not needed to be remembered. You can copy off from an example file and edit on your own.

- *Boundary* The rational behind making boundary fixed is so that we know if a geometry is getting out of the domain box or the particle is getting out of domain box.
- *Region* Since, we are using a cylindrical system, it is best to use a cylindrical shaped domain box. This would result is less empty space, hence saving in computation.
- *Create_Box* We will use 3 atoms and the domain is the cylinder we just specified
- *Neighbor* The neighbour list will be created at a distance which is approximately equal to the diameter of the atom.

Writing the Properties

The following are the properties that will needed to be written

- Material properties *for example young's modulus, friction etc*
- Importing Geometries to be used as wall and using the imported geometries as wall
- Insertion face
- Particle template
- Particle distribution template
- Particle insertion
- Pair style
- Gravity

```
# Properties of Material

fix                 m1 all property/global youngsModulus peratomtype 5.e7 5.e7 5.e7
fix                 m2 all property/global poissonsRatio peratomtype 0.45 0.45 0.50
fix                 m3 all property/global coefficientFriction peratomtypepair &
                3 0.7 0.65 0.68 0.65 0.60 0.68 0.68 0.68 0.3
fix                 m4 all property/global coefficientRestitution peratomtypepair &
                3 0.25 0.24 0.23 0.24 0.35 0.23 0.23 0.23 0.22
```

Properties Setting The setting of properties is pretty much self explanatory. For every peratomtype, you would need input values for each material type. For every peratompair properties, you would need to give the matrix in the form of Figure 27

```
    /* Importing Geometries */

fix                 out all mesh/surface file mesh/OuterPaddle.stl type 3 scale 0.01
fix                 incad all mesh/surface file mesh/InnerPaddle.stl type 3 scale 0.01&                 cu
# Make the imported cad as walls
fix                  wall all wall/gran model hertz tangential history mesh &
                n_meshes 2 meshes incad out
```

- **file** keyword refers to a location of file
- **Type** refers to the atom type. We will refer all the walls being consist of a atom of type 3
- **Scale** It can be seen that the dimensions if used directly will be enormous. For *example* 24 m. Therefore we will scale down our imported geometry down 100 times.
- **Wall/gran** This command converts the imported geometry to walls.

```
# use the walls to fix the domain box with primitive walls
fix                  zwalls1 all wall/gran model hertz tangential history &
                primitive type 3 zcylinder 0.125 0 0
#Use the wall at the time of filling
fix                  cylinder1 all wall/gran model hertz tangential history &
                primitive type 3 zcylinder 0.12 0. 0.
fix                  cylinder2 all wall/gran model hertz tangential history &
                primitive type 3 zcylinder 0.04 0. 0.

# region for insertion
fix                  inface all mesh/surface file mesh/insertionFace.stl type 3 scale 0.01
```

Apart from the traditional imported walls, we can also generate walls within *LIGGGHTS*itself. This is done by using the primitive type command. [11] The insertion face can be imported using the similar command as of geometry. Notice the type and scale keywords.

[11] For more information, visit here

```
  /* distributions for insertion */
  fix                 pts1 all particletemplate/sphere 1 atom_type 1 &
                 density constant 7800 radius constant 0.00100
  fix                 pts2 all particletemplate/sphere 1 atom_type 2 &
                 density constant 1190 radius constant 0.00100
  fix                 pdd1 all particledistribution/discrete 1.  2 &
                 pts1 0.50 pts2 0.50
```

In this particular step, we did two things

- Defined the basic template of particles i.e *radius, density and atom type*. It is always good practice to define different atom types with different type of atom. Currently you can define upto 32 atoms of atoms.
- Defined the distribution of the insertion. Here we are taking 50 % of particle 1 and % of particle 2 in the distribution. Thus it is a perfectly homogenous mixture distribution.

```
/* particle insertion */
fix         ins all insert/stream seed 1001 distributiontemplate pdd1 &
            nparticles 6000 vel constant 0. 0. -0.2. particlerate 20000 &
            overlapcheck yes insertion_face inface extrude_length 0.015
```

Particle Insert Stream The Latest version of *LIGGGHTS*, which as of now is 3.3.0, offers some really great optimisations in the insertion using insert/stream method.

- **Seed** This is a random number generator, to make the simulation for more non-deterministic.
- **Distribution Template** This is the Distribution we just defined with perfectly homogenous mixture.
- **nparticles** We will use the number of particles method.
- **vel constant** We can use the velocity at the time of insertion. The input here is a vector. This one points that it will have *0.2 metres/sec* in the negative z direction.
- **Insertion Face** The geometry we had imported earlier to be used as insertion face.
- **Extrude length** The commands extrudes the planar insertion face to create a region, from where particles will be generated.

```
  /* New pair style */
pair_style      gran model hertz tangential history # Hertzian without cohesion
pair_coeff        * *

  /*Timestep */
timestep        0.00001

/* Defining
fix               2 all gravity 9.81 vector 0.0 0.0 -1.0
```

Pair Style In general, *LIGGGHTS*provide two pair styling.

- Hooke

- Hertz

Hertz performs better for short range forces and therefore we will use that for our granular simulation. [12] The *pair style* * * commands refers that all atom type interaction interaction follows the same pairing defined by the hertz. We can also define separate pair for interaction of two different atoms here. If we need to add cohesion into our system, then we can write

[12] You can find out more about it here

```
pair_style gran model hertz tangential no_history cohesion sjkr
```

[13]

[13] For more information on cohesion, you can find here

Timestep It is generally advised to have a timestep about 20 % of the Rayleigh timestep. You can check the hertz time being

$$RayleighTimeStep = \frac{R \times \pi \times \sqrt{\rho/G}}{(0.1631 \times \nu + 0.8766)}$$

Here, ρ is density, G is Youngs Modulus, ν is poissons ratio and R is the radius

The first thing to determine the time step will be to calculate the Rayleigh timestep and take around 20% of its value. Other wise, you could use an inbuilt function of *LIGGGHTS*to use an inbuilt function to calculate the Rayleigh timestep and adust your timestep accordingly.

```
/* Detailed Settings */
#apply nve integration to all particles that are inserted as single particles
fix                integr all nve/sphere

/* output settings, include total thermal energy */
fix                ts all check/timestep/gran 1000 0.1 0.1
compute              1 all erotate/sphere
thermo_style        custom step atoms c_1 f_ts[1] f_ts[2]
thermo               1000
thermo_modify        lost ignore norm no
compute_modify        thermo_temp dynamic yes
```

Calculations and thermodynamic properties.

The thermostyle command is actually the log output and terminal that you see during the operation of *LIGGGHTS*. Here we can see that we will observe the number of steps, number of atoms, the rotational energies and the Rayleigh and Hertz Time.

```
/* insert the first particles so that dump is not empty */
run                        1
dump                dmp all custom 100 post/dump*.feedFrame id type type x y z ix iy iz vx vy vz radius
dump                 dmpstl all mesh/stl 100 post/dump*.stl
```

```
#insert particles
run                 40000 upto
unfix                 ins
fix                  move all move/mesh mesh incad rotate origin &
                0. 0. 0. axis  0. 0. 1. period 10
run                  100000 upto
```

It is highly recommended that we should run 1 simulation first, so that the first timeslot is not null. The dump commands generate dump files. As we had the insertion, running will insert until we run out of timesteps or particles to insert. Once we are satisfied that we have inserted all the particles or we want to stop the insertion in the midst, we can unfix the ins.

The second fix command, actually starts rotating the mesh around the origin and it rotates upto 100000 timesteps. In this case, the incad or the inner paddle is rotated. We can also rotate, the outer paddle and it will be the exact same thing.

Errors, Errors Everywhere

The first time you will run your own manufactured code, there is high chances that you will get an error. But do not worry, errors are handled pretty well by *LIGGGHTS*. If you have an error, then *LIGGGHTS*will give you most information to find and debug it. The subsequent section gives some overview how to what kind of errors, one might face.

Errors

- **ERROR on proc 0: Cannot open mesh file meshes/OuterPaddle.stl (../input_mesh_tri.cpp:94)** Check the path given to importing mesh file
- **ERROR: Fix wall/gran (id wall): could not find fix mesh id you provided (../fix_wall_gran.cpp:264)** While making the imported CAD as walls, make sure that you give the correct ID, you had given while importing.
- **ERROR: Could not find fix group ID (../modify.cpp:754)** You have given an invalid group ID.
- **ERROR: Fix insert/rate/region (id ins1): region ID does not exist (../fix_insert_pack.cpp:86) If you are using an imported mesh as insertion face then rather than *region*, you have to use *insertion_face* command. 11**

- **ERROR: Could not locate a fix/property storing value(s) for coefficientRestitution as requested by model hertz**. (../modify_liggghts.cpp:346)

  Debugging basics

  Which line prompted the error. In the process of debugging, this is the most vital question that one may ask. And *LIGGGHTS*reads the input the file, line by line. This is a good thing for us as it will tremendously useful for us to debug it. One of the important method you could use to debug, is introducing a foreign word or an unrecognised word into the script which is bound to make it fail. This would help us narrow down to the region where there is an error.

  For example consider this,

```
  /*continued from previous script... */
  #output settings, include total thermal energy
fix                ts all check/timestep/gran 1000 0.1 0.1
compute                1 all erotate/sphere
thermo_style        custom step atoms c_1 f_ts[1] f_ts[2]
thermo                1000
thermo_modify        lost ignore norm nope
compute_modify        thermo_temp dynamic ye
```

  Running the script will give us the following error

```
ERROR: Illegal thermo_modify command (../thermo.cpp:471)
```

In order to identify where is the error, we will introduce the word **stop** somewhere in the code.

```
  /*continued from previous script... */
  #output settings, include total thermal energy
fix                ts all check/timestep/gran 1000 0.1 0.1
compute                1 all erotate/sphere
thermo_style        custom step atoms c_1 f_ts[1] f_ts[2]
stop
thermo                1000
thermo_modify        lost ignore norm nope
compute_modify        thermo_temp dynamic ye
```

  Now running the code again, will result in the following error.

```
ERROR: Unknown command: stop (../input.cpp:256)
```

This will give us an idea that the error is probably after the line where have put the word *stop*.

```
  /*continued from previous script... */
  #output settings, include total thermal energy
fix                ts all check/timestep/gran 1000 0.1 0.1
compute             1 all erotate/sphere
thermo_style       custom step atoms c_1 f_ts[1] f_ts[2]
thermo              1000
thermo_modify       lost ignore norm nope
compute_modify       thermo_temp dynamic ye
stop
```

Running the code again results in

```
ERROR: Illegal thermo_modify command (../thermo.cpp:471)
```

So, we are back to where we started. But the idea is that we know the error is amongst the two lines. And I can very well see that the keyword for **thermo_modify** will *no* instead of *nope*. And that's how we catch an error.

However, you could have easily inferred that there was something wrong with thermo command from the error itself. This would not be the case everytime. Hope you get the idea.

How to get help

Okay, now we understand how to detect errors. The obvious next step would be debug and remove the error is to remove it. The way to do that is by refering more information on that particular command which is causing error in the documentation of *LIGGGHTS*. And the second place where you could get help is the forum. The thing to remember is before asking any questions, you should check out previous queries by searching. There is quite less activity however the community is friendly and someone always answers your question.

Summary

Hopefully this part will make you more confident in approaching towards making your own simulation.

Further Resources & Ending Note

Learning to work with *LIGGGHTS* will need much more than reading a documentation. This documentation is an attempt to initiate you with the basic skills to get you started with it. I hope that reading and working out the manual was a painless ordeal for you.

More functionality

1. **ave/spatial:** Divides the region into bins and calculates the number of particles and their properties in each bin. Used for averaging. It can be used to calculate the *Danckwert's Segregation Intensity*.
2. **property/atom/tracer:** Particles are marked if they are inside a region specified by the region_mark keyword. Can be combined with **compute nparticles/tracer/region** to calculate the **residence time distribution** in case of a hopper.
3. **multisphere:** Treat one or more sets of atoms as independent rigid bodies.
4. **heat/gran/conduction :** Calculates the heat conduction between particles in contact.
5. **humiidity model :** LIGGGHTS can be coupled with other systems to check the effect of Water(Humidity) on the system.
6. **cohesion :** There are two models of cohesion that can be added to LIGGGHTS to simulate cohesive behaviour of substances

Further Studying

Here is the list of further resources for you to study (*if interested*) on Discrete Element Method.

- LIGGGHTS User Manual It gives the most comprehensive functionalists associated with LIGGGHTS. I would recommend reading it atleast once, even though you might not understand everything at first.

The Feedframe simulation input and mesh files are accompanied with this document.

- DEM Primer on Yade Yade is another open source discrete element method, completely built with Python and possibly Cython. It has a good primer on DEM Formulation.
- Carles Bosch Padroś thesis on Discrete element simulations with *LIGGGHTS* This document specifies how *LIGGGHTS*implements Discrete Element Method and also has a short section on DEM Theory.
- Computational Granular Dynamics: Models and Algorithms An excellent book defining DEM and how to implement it. It contains codes in C++ implementing the same. A definite book if you really want to learn to implement your own DEM or develop *LIGGGHTS*or *LAMMPS*source code. The source code files are available in the website too.

Rationale behind working with LaTeX

Git is a powerful version control system. It stores snapshots of your current work as a set of files and the changes made to each file over time. It is a good workflow to add git in your current working directory, if your are working on a computational project. You can learn more about git in a Quora answer

DEM is an extremely handy tool for gaining a good insight on the flow of powders. Understandably simulation of physical phenomenon is as accurate to begin with, but still can provide several results that may increase our understanding and aid us in developing appropriate models. These are the fewer incentives why you should invest in learning DEM:

- The models which depict real world phenomenon will become more and more accurate in the future.
- The cheaper computational cost will enable to calculate and host a wide range of experiments and thus gain an admirable advantage
- Modelling flow of powders is quite difficult only through experimental calculations. Also the overhead cost and time, to model a new experimental data is quite large sometimes.

Learning & working with*LIGGGHTS*would be a continual process. This documentation could serve as a basis of recording progress in the continual improvement, where new methods and algorithm implementation in *LIGGGHTS*could be kept and added for future reference. Therefore, this documentation is available at Github repo. You are welcome fork, push changes or file issues.

Bibliography

Index