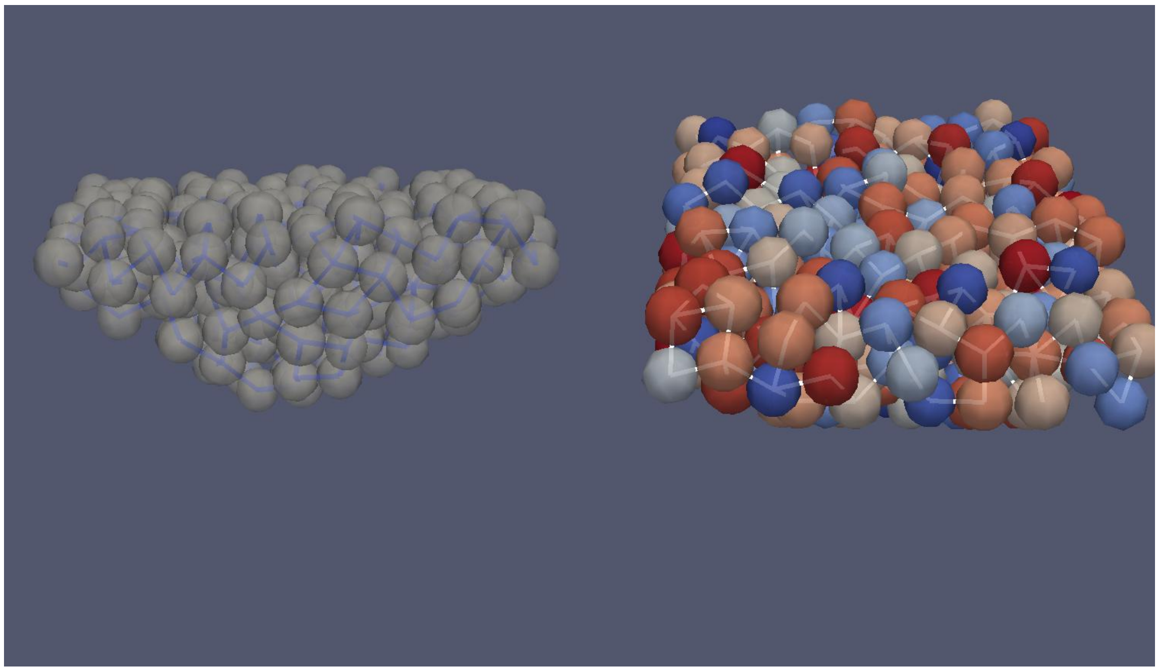
Developing Discrete Element Method Simulations for Ice Mechanics Using Yet Another Dynamic Engine and ParaView



Prepared By: Colin Power

Date: August 22nd, 2014

Summary:

The Numerical Modeling group performed multiple simulations of ice mechanics using Discrete Element Method with the software Yade. A methodology and standardized set of tools for designing the simulations has been developed. Going from a paper design specification, it is now possible to model ice mechanics with bonded spheres using the software programs Gmsh for geometrical drawings, Yade for generating datasets via Python scripts, and ParaView for post-processing of the datasets.

The need to standardize the codebase and mange files better is apparent. The team should adopt a Python standard such as PEP8 and use a source control system. GitHub would be an ideal source control system since the Yade developers themselves use it. As the scope of the work increases, it is increasingly important to answer these questions before the current deficiencies in work tools start to impede the research progress.

An analysis of the impact of changing the densities of the triangular shapes that create boundaries and register forces within the Yade simulations shows that there are very minor changes in accuracy when altering said densities. Furthermore, tests were done to benchmark the Yade performance on three separate computer configurations. The results showed that Yade worked best when running on Ubuntu 14.04 LTS and that the second newer workstation has a moderate performance increase over the first one.

The report also outlines some key areas of interest related to ParaView such as a small project to allow ParaView to accurately visualize particle bonds using Yade datasets. Finally, a short tutorial on using Gmsh has been included in an appendix. It walks the reader through creation of both a box and hollow sphere mesh that can then be used in Yade.

TABLE OF CONTENTS **PAGE**

1. Introduction	**1**
1. Yet Another Dynamic Engine	**2**
1.1. Facets	**3**
1.2. Materials	**3**
1.3. Volumes and Packings	**5**
1.4. Support from Developers and Community	**8**
1.5. Current Workflow	**8**
2. Simulation Descriptions	**9**
2.1. Wall and Ramp	**9**
2.2. 2D and 3D Indenter	**10**
2.3. Ice Keel	**12**
2.4. Ice Floes	**12**
2.5. Facet Test	**14**
2.5. Iceberg Net Proposal Test	**18**
3. Other Yade Features	**18**
3.1. What .mesh, .stl, .geo (GenGeo and Gmsh), .gts, and .vtu are	**18**
3.2 Remote Desktop	**20**
3.3 Snapshots in Yade	**20**
3.4 Using Batch mode in Yade	**21**
4. Hardware Tests	**22**
5. File Management	**23**
5.1. Implementing Source Control	**24**
5.2. Improving Codebase	**24**
5.3. Modularization of Repeated Code	**25**
6. Visualization of Data Using ParaView	**26**

6.1. Sphere Glyphs 27

6.2. Time Source Filter 27

6.3. Extract Selection 28

6.4. Visualizing Bonds 28

6.5. Animation Problems 29

6.6. Camera Interpolation 30

6.7. Macros and Python Scripting 30

7. Conclusions 30

8. Recommendations 31

APPENDICES

A Gmsh Tutorial

B Hardware Performance Test Results

Introduction:

The Centre for Cold Ocean Resource Engineering (CORE) has founded the Centre for Arctic Resource Engineering (CARD) using $12.5 million in funding over five years from the Hibernia and Terra Nova oil project stakeholders to pursue research that will reduce the costs and risks associated with Arctic Ocean resource development. CARD has selected the study of ice mechanics and loading as part of its five-year research plan that focuses its research on problems relevant to the general research community and various sectors of the oil and gas industry. The study of ice using the theory of Discrete Element Method (DEM) and numerical modelling has been chosen as one of CARD's research projects.

The main feature of DEM is the use of numerous perfect spheres as particles to simulate real world problems on powerful computers. In contrast, Finite Element Method uses meshed surface areas to explore similar topics. DEM has its roots in geological research such as soil mechanics and rock fracturing. C-CORE wants to adapt this method to model ice as a brittle solid in DEM. This is a fresh and exciting research topic that has never been explored by anyone else except for tangentially related 2D ice simulations.

As the numerical modeling group at CARD is very new and expected to expand in the next 6 months, it would be useful to create a report describing the design process and purpose of the ongoing simulations so that new team members can become quickly informed. As of now, only small scale and limited scope reports exist for this project. Furthermore, analysis is done on deficiencies on the software methodologies caused by the early stage status of the project and various solutions are proposed.

1. Yet Another Dynamic Engine:

Also known as Yade, is a physics simulation software that applies DEM to recreate real world phenomenon from various physical contact models. Simulations are scripted in Python and then Yade writes data into text files for analysis. ParaView has been selected by CARD as its data visualization software of choice due to its very high ease of use and popularity within the general DEM community.

Yade was adopted as the DEM software after struggling with ESyS-Particle due to the latter's lack of documentation and features. The Yade developers have provided superior documentation and community support and thus ESyS-Particle was abandoned in favour of Yade. Choosing Yade has greatly improved the rate of progress for this project.

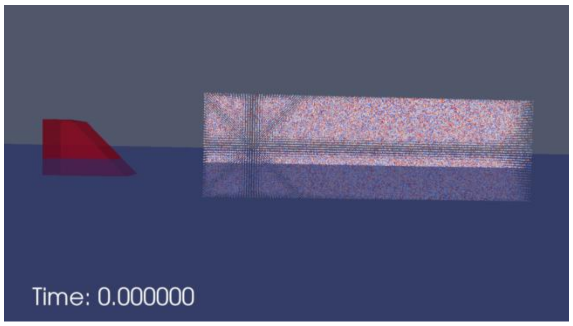


Figure 1.1: A Yade simulation visualized in ParaView. A block of spheres floats in water and a solid ramp runs through the spheres.

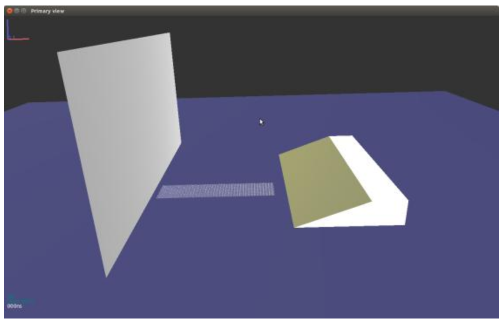


Figure 1.2: A simulation being viewed in Yade's real time 3D viewer.

1.1. Facets:

Facets are essentially 2D or 3D shapes that cannot be altered by sphere collisions but can interact with the spheres. Facets cannot interact with each other and are comprised of several interconnected triangles. They can be used as simulation boundaries for the particles, or as mechanical objects to collide with and apply force to the spheres. The function ***TranslationEngine()*** causes the facet to move in a given velocity vector while ***RotationEngine()*** gives the facet an angular velocity set about an axis point. Furthermore, facets can be generated descriptively using Yade functions and mesh generation via Python modules. Also, facets can be created using computer aided design (CAD) programs such as Gmsh to create a .mesh and import it as outlined in appendix A.

1.2. Materials:

Every particle and facet has to be assigned user defined materials that are derived from several different DEM contact models. The various materials contain typical parameters such as Poisson's ratio, density, and their frictional angles. Every material also has its own special set of properties and behaviours. For example, there are frictional materials and there are cohesive materials that allow particles and facets to form strong bonds such as concrete would.

In the following line of code:

steel=O.materials.append(FrictMat(young=50e6, poisson=0.3, density=rho_s, frictionAngle=angle))

A generic steel material is added to the simulation and is set to be FrictMat with a Young's modulus of 50×10^6, Poisson ratio of 0.3, variable density, and variable frictional angle. Another example is a cohesive ice material:

ice=O.materials.append(CpmMat(young=youngModulus, poisson=poissonRatio, density=rho_p, frictionAngle=angle, sigmaT=5.5e3, relDuctility=30, epsCrackOnset=1e10, isoPrestress=0, damLaw=0))

Here, several complex ***CpmMat*** specific parameters are passed to the material. These parameters are detailed in the Yade documentation, but their purpose and mathematical validity are still undergoing investigation by CARD.

Every simulation requires that the various contract models be defined in the interationLoop. If the steel and ice materials were assigned to facets and particles respectively, the simulation would require the following interaction loop to succeed:

```
InteractionLoop(
        [Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom()],
        [Ip2_CpmMat_CpmMat_CpmPhys(), Ip2_FrictMat_CpmMat_FrictPhys()],
        [Law2_ScGeom_FrictPhys_CundallStrack(), Law2_ScGeom_CpmPhys_Cpm()]
        ),
```

The code shows that CpmMat on CpmMat and FrictMat on CpmMat interactions are now defined. Since facets do not interact with other facets, a FrictMat on FrictMat definition is not needed in this example. If these definitions are not supplied to the Yade engine then the simulation would return an error or crash when the specific interaction type was required.

1.3. Volumes and Packings:

Volumes, also called predicates, can be defined in Yade through module functions or by importing closed volume mesh shapes that are created in a CAD package.

The GenGeo Python module can be used independently of Yade to generate 2D and 3D packings that are randomly distributed within a defined volume. GenGeo can create extremely high particle packings that have no directionality and smoothed edges. Once a GenGeo script is done running, a .geo file is saved containing every single particle position and radius. GenGeo can be used in both ESyS-Particle and Yade as the file format is supported by both programs and is ideal for intricate packings containing thousands of particles.

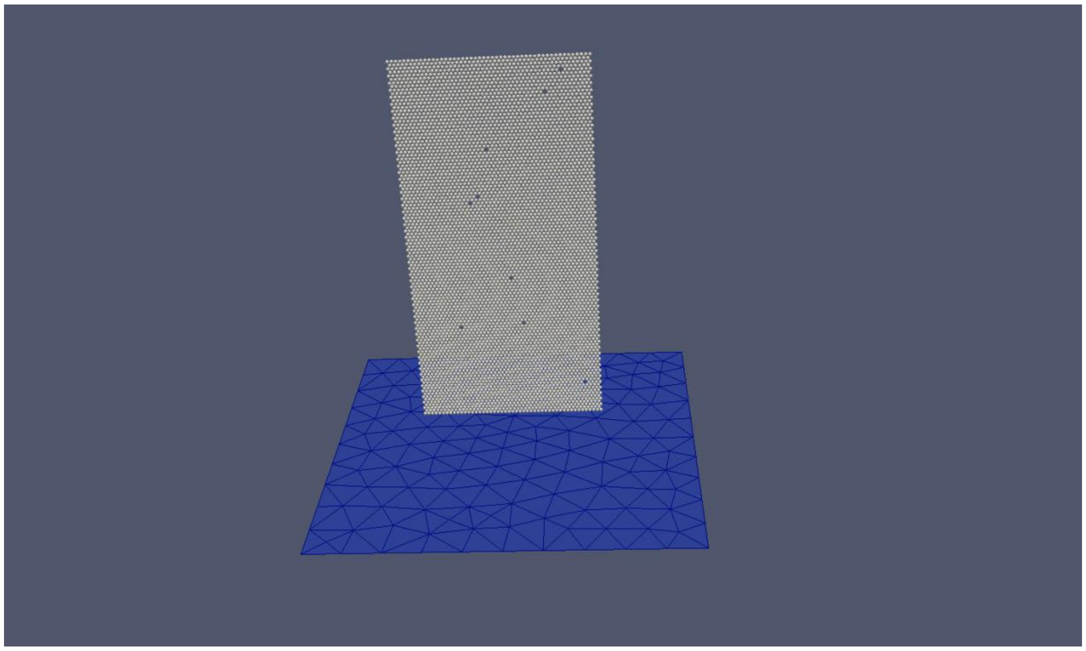


Figure 1.3.1: This 2D rectangle was generated using GenGeo. The small holes are caused by the random insertion of spheres into the defined volume until no more spheres can fit into the volume. The visualization is done in ParaView.

Similarly, the Yade function, ***randomDensePack()*** is able to generate a random packing that contains spheres distributed about a mean radius . The Yade developers have poorly documented this function so it will have to be clarified here.

Figure 1.3.2 was generated using the following command:

> *O.bodies.append(pack.randomDensePack(predicate=volume, radius=radius, material=Mat, cropLayers=0, rRelFuzz=0.6666, spheresInCell=50000, memoizeDb=None, useOBB=False, memoDbg=False, color=None, returnSpherePack=False))'''*

rRelFuzz restricts the spheres to be radius ± radius*rRelFuzz while ***spheresInCell*** is the target number of spheres for the algorithm to pack in the volume. ***MemoizeDB***, when set to True, makes the function search the active directory for an SQL file containing previous packings generated using similar parameters. If such a packing is found, then that packing will be loaded instead of generating new one and thus saving some computational time.

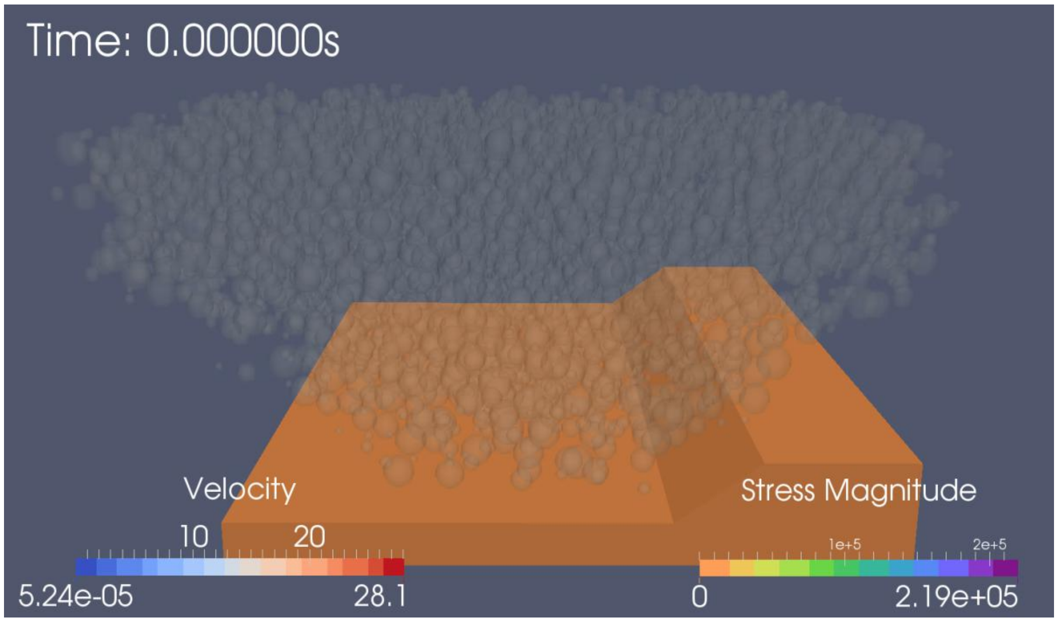


Figure 1.3.2: A 3D packing created by importing a mesh file of a keel shape and then using **randomDensePack()** to pack the volume.

A packing can also be made by manually generating the spheres such as in this example script that generates a box using an iterative loop in Python:

```
#Generate spheres in a box
print "Creating %d spheres..." %(nbSpheres[0]*nbSpheres[1]*nbSpheres[2]),
for i in xrange(nbSpheres[0]):
    for j in xrange(nbSpheres[1]):
                    for k in xrange(nbSpheres[2]):
                            x = (i*2 - nbSpheres[0])*sphereRadius+sphereRadius*0.1
                            y = -j*sphereRadius*2.0
                            z = (k*2 - nbSpheres[2])*sphereRadius+sphereRadius*0.1
                            r = random.uniform(sphereRadius, sphereRadius*1)
                            fixed=False
                            color=[0.6,0.6,1.0]
                            O.bodies.append(sphere([x,y+30,z+10], r, material=steel,
color=color, fixed=fixed))
```

(Šmilauer, 2013).

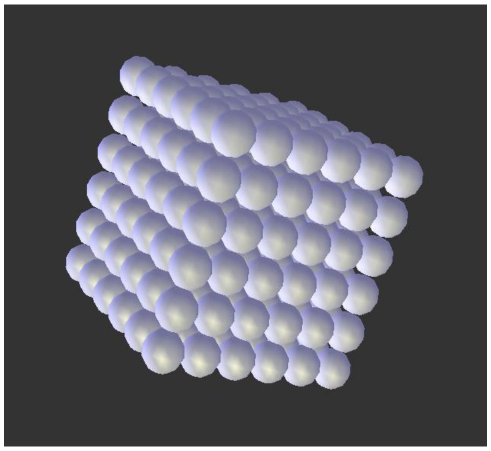


Figure 1.3.3: A 6x6x6 box created in Yade using the example code.

1.4. Support from Developers and Community:

Questions made on the Yade community Launchpad have been met with quick and insightful responses. Generally, questions are answered by the primary Yade developers and sometimes from the general community. Feedback is often received on the same day as the question is asked. In addition, the developers have been willing to add new features to Yade for free at the request of the community. C-CORE was able get the Yade team to add the ability to record Torque and Forces on facets and spheres for visualization in ParaView. The feature was added within a few days of requesting it.

The documentation written by the Yade developers is extensive and well done, despite some minor problems with clarity and depreciated functions. The project is support by a 150 page plus manual that includes tutorials, examples, and DEM scientific theory. Those interested in adding to the C++ source code can read the ***Programmer's manual*** for insight into the design and algorithms used by Yade. Furthermore, there are many example scripts hosted on the Yade GitHub repository that are of great use when learning how to write a Yade script.

1.5. Current Workflow:

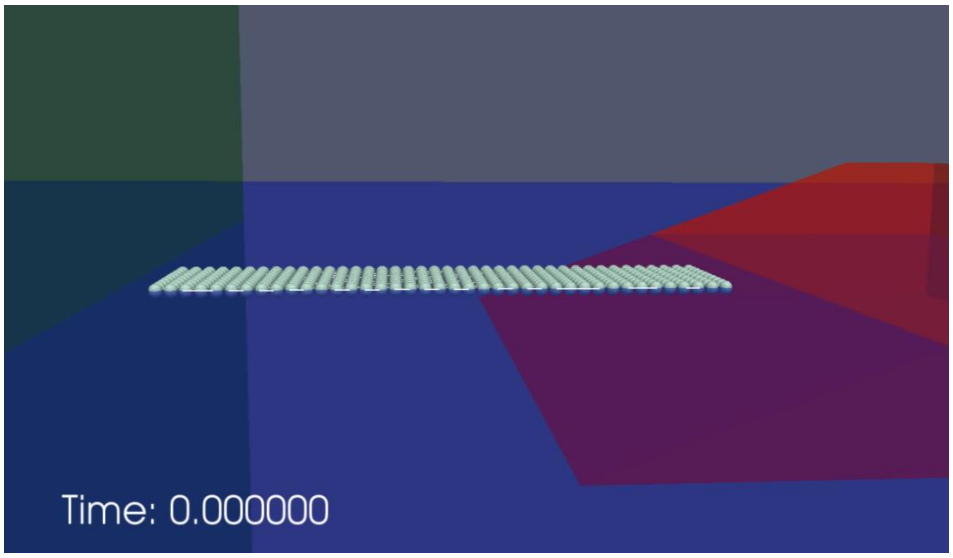
To efficiently create a simulation requires the knowledge to take it from paper design up to the post-processing stage. First, a CAD package such as Gmsh is used to generate meshes that are then imported into Yade. Next, the sphere packing is provided either through importing volumetric shapes, GenGeo, or by function definitions in Yade. Every step must take into account that all programs cooperate on the same Cartesian coordinate systems. So placing a point at (5, 5, 10) in Gmsh also means that it will be at

(5, 5, 10) in ParaView or Yade unless the user shifts, rotates, or scales up the imported object. Once the simulation is running, it will save data in text based formats which can then be opened for post-processing in ParaView.

2. Simulation Descriptions:

2.1. Wall and Ramp:

A simulation has been designed to model a horizontal sheet of ice floating in water being pushed by a vertical rigid wall of ice towards a steel ramp. This simulation is the called Wall and Ramp or Ice Rubbling experiment. Both the bonded spheres and vertical wall are given ***CpmMat*** material so there is some stickiness between the particles and the wall.



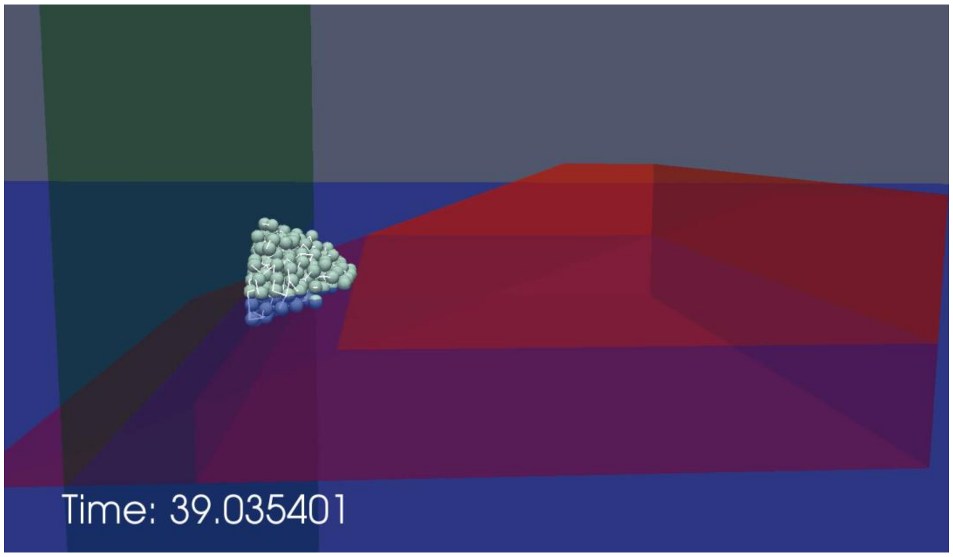


Figure 2.1.1: The simulation data is shown in ParaView. The particles clump up as the wall pushes them up the ramp. The facets are set to be transparent and coloured by their material type. The blue facet is the water level where the buoyancy interaction stops being applied.

2.2. 2D and 3D Indenter:

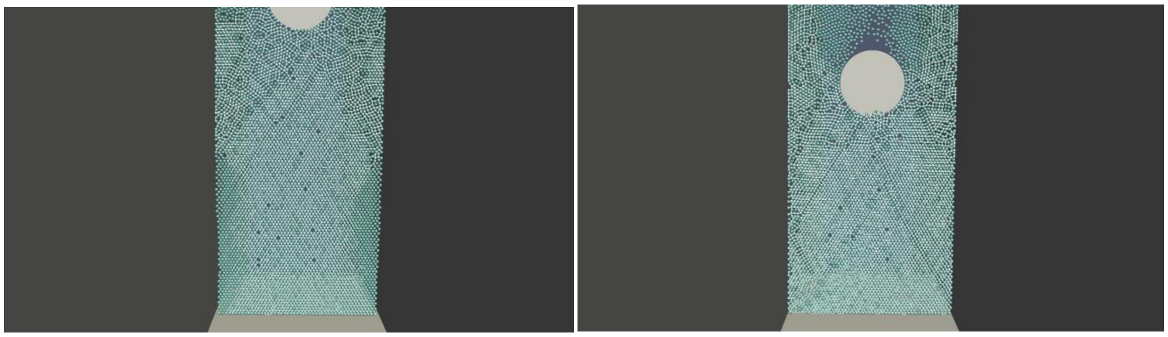


Figure 2.2.1: The 2D indenter disc pushes the spheres as it moves downwards. The white lines show the bonds of the ice.

The numerical modelling group has prepared a script based upon the physical experiments detailed in Habib et al (2014). The simulation seeks to reproduce the qualitative attributes of ice when a steel cylinder crushes a cubic specimen. A 2D simulation has been prepared and is waiting on fine tuning while a basic 3D simulation has been ran as a proof of concept and performance test.

A steel 2D disc is translated negatively along the Z axis at a constantly velocity until the simulation is stopped. The rectangular sphere packing is made pseudo two dimensional by restraining the spheres from moving on the Y axis or rotating on the X and Z axes. This is done in Yade by using: for b in O.bodies: *b.state.blockedDOFs = 'Zxy'*, *w*hich iterates over all sphere objects and blocks specific degrees of freedom on the Cartesian planes to produce a 2D result in the 3D simulation.

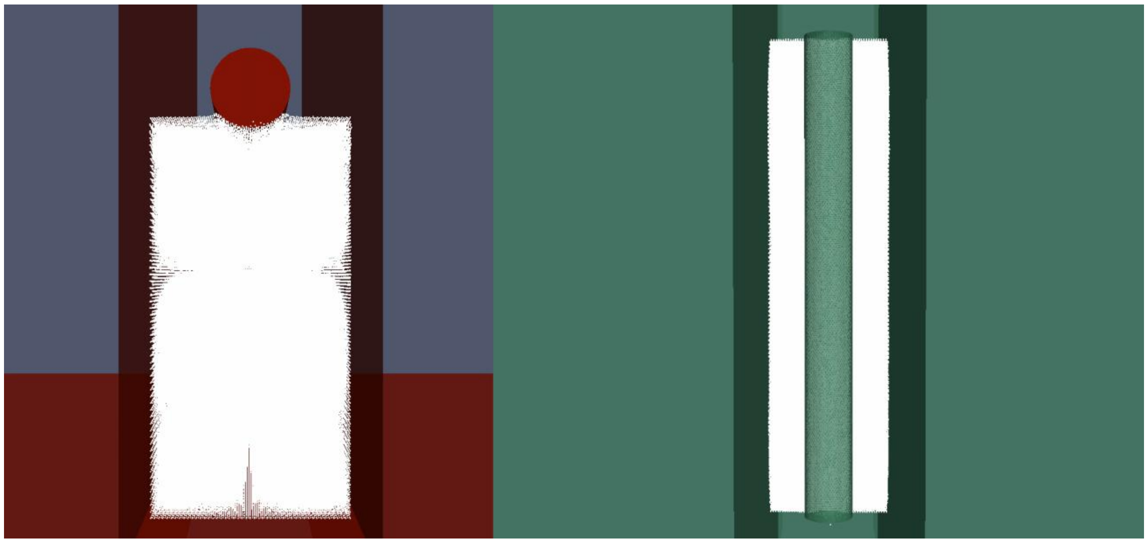


Figure 2.2.2: A cylinder is moving downwards to indent 1,268,947 individual particles over the span of 70 hours to reach the final position shown in the 3D cylinder version of the indenter.

2.3. Ice Keel:

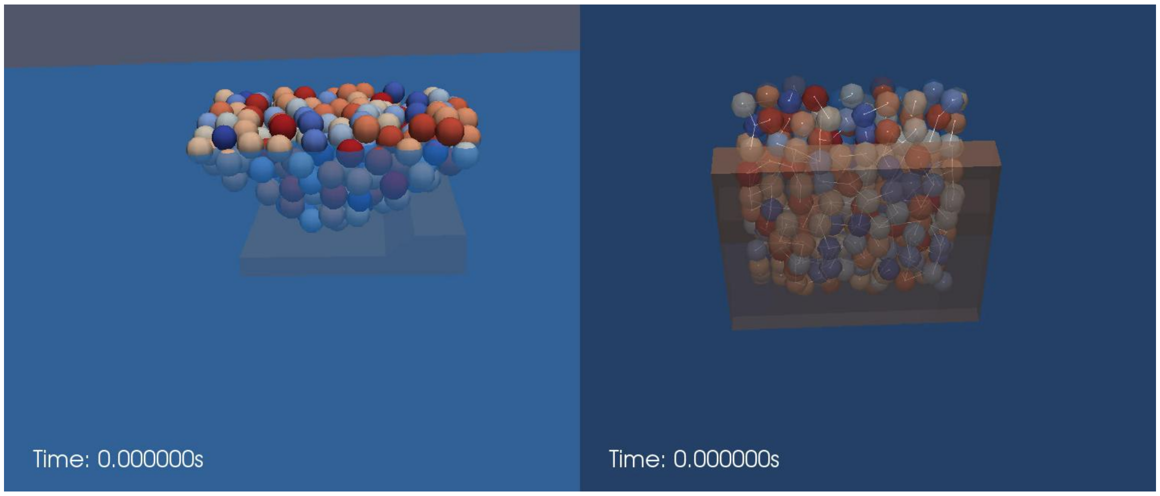


Figure 2.3.1: About 300 spheres are bonded together to form a keel shape.

This simulation models the Development of Ice Ridge Keel Strength project at C-CORE where the impact of multi-year ice keels on the seabed in the Arctic Ocean are replicated in a laboratory with manufactured ice. During the simulation, the seabed made from rocks scraps against the bottom of the bonded ice keel at a rate of 0.5m/s until the simulation ends. The results are recorded and the datasets of forces and stresses acting on the seabed are reviewed. At this moment, the simulation is being limited to approximately 300 spheres for testing purposes but this can easily scaled up to thousands of spheres once the script is ready for full simulation.

2.4. Ice Floes:

The ice floe simulation has twenty separate clumped sphere packings moving at a constant and constrained velocity N along the X-axis to simulate the movement of a boat into an ice field. The purpose of the simulation is to build upon the numerous 2D ice flow models that have been created by C-CORE in the past. After so many iterations, more ice blocks will spawn at the left side and flow towards the right side. When clumps exit from the square blue water indicators, they are deleted using ***DomainLimiter()***.

There are 1000 uniquely randomized cubic packings that the simulation imports to build the various clump shapes. When they are brought into the simulation, they are spawned in a row made of twenty blocks and then are randomly shifted in the X and Y planes enough to create a non-uniform distribution. The next step in developing the simulation will be adding knock back on the ice floes once they collide with the boat. The aim is to alter the X-direction speed of the blocks only when they crash.

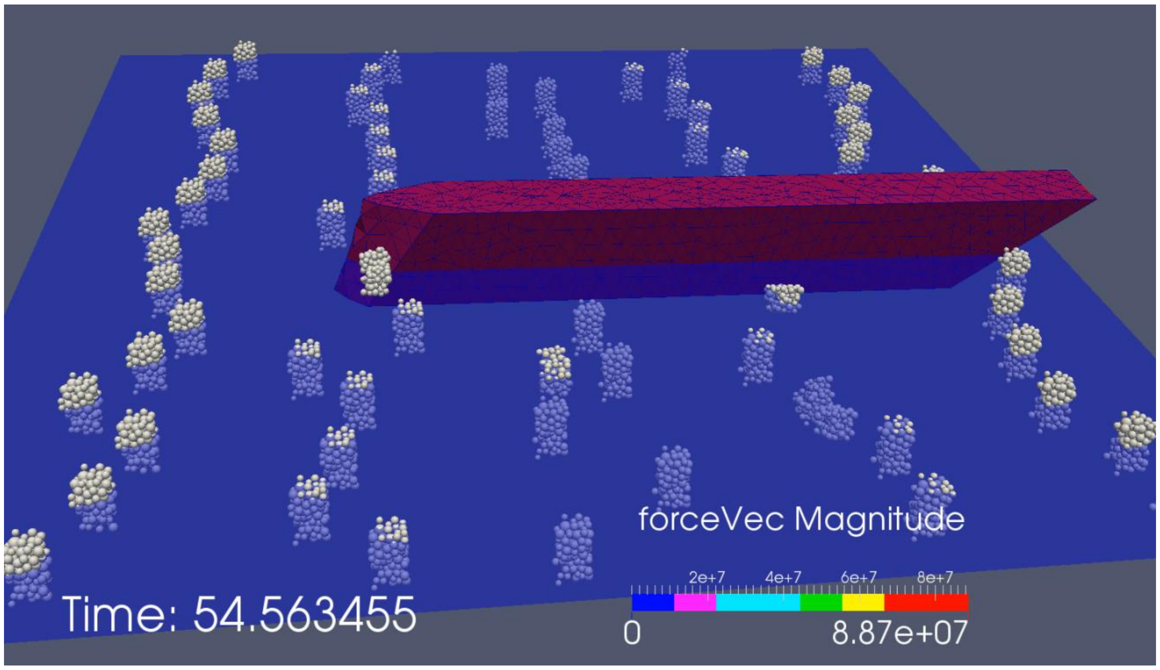


Figure 2.4.1: The boat remains static while the small blocks move from left to right until they leave the blue boundary. Each individual sphere is grouped with its neighbours so that they can never separate.

Another version of the simulation is in devlopment. In this case, the boat is now dynamic while the blocks are all spawned at the beginning to the left of it to form an ice field. The blocks are free to drift in any direction and bounce off the boat naturall since they are no longer contrainedi n the X-axis. This simulation seems superior but is more complicated for Yade to run.

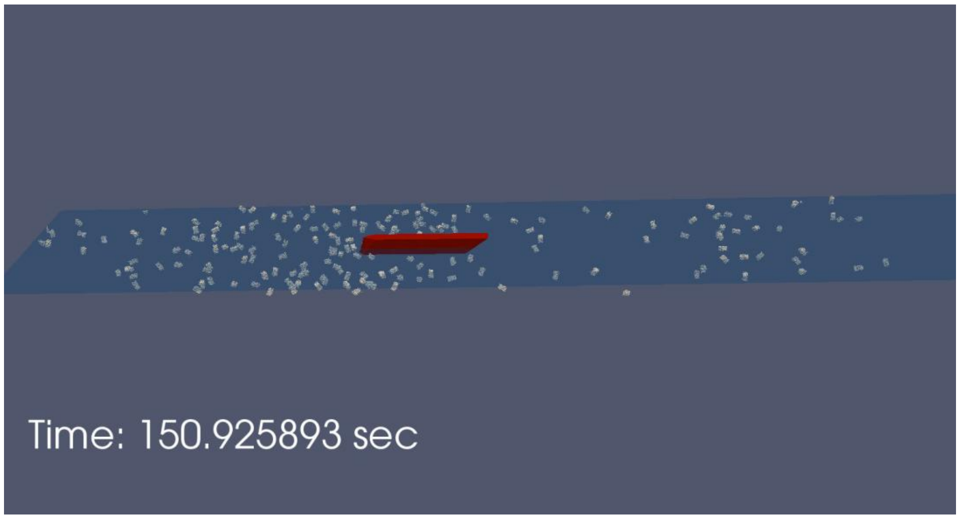


Figure 2.4.2: The red boat moves from left to right banging into the various ice blocks.

2.5. Facet Test:

C-CORE is investigating the consequences of varying the number of mesh elements in a geometrical shape made in Gmsh. It is very easy to change the density of triangular elements by magnitudes such as four times or one tenth. Tests using Yade to check the performance impact of greatly increasing the number of triangles in a rectangle have shown that there is a considerable impact on performance. Additionally, it was originally thought that adding more triangles for the spheres to collide with would lead to more accurate results but results seem to disagree.

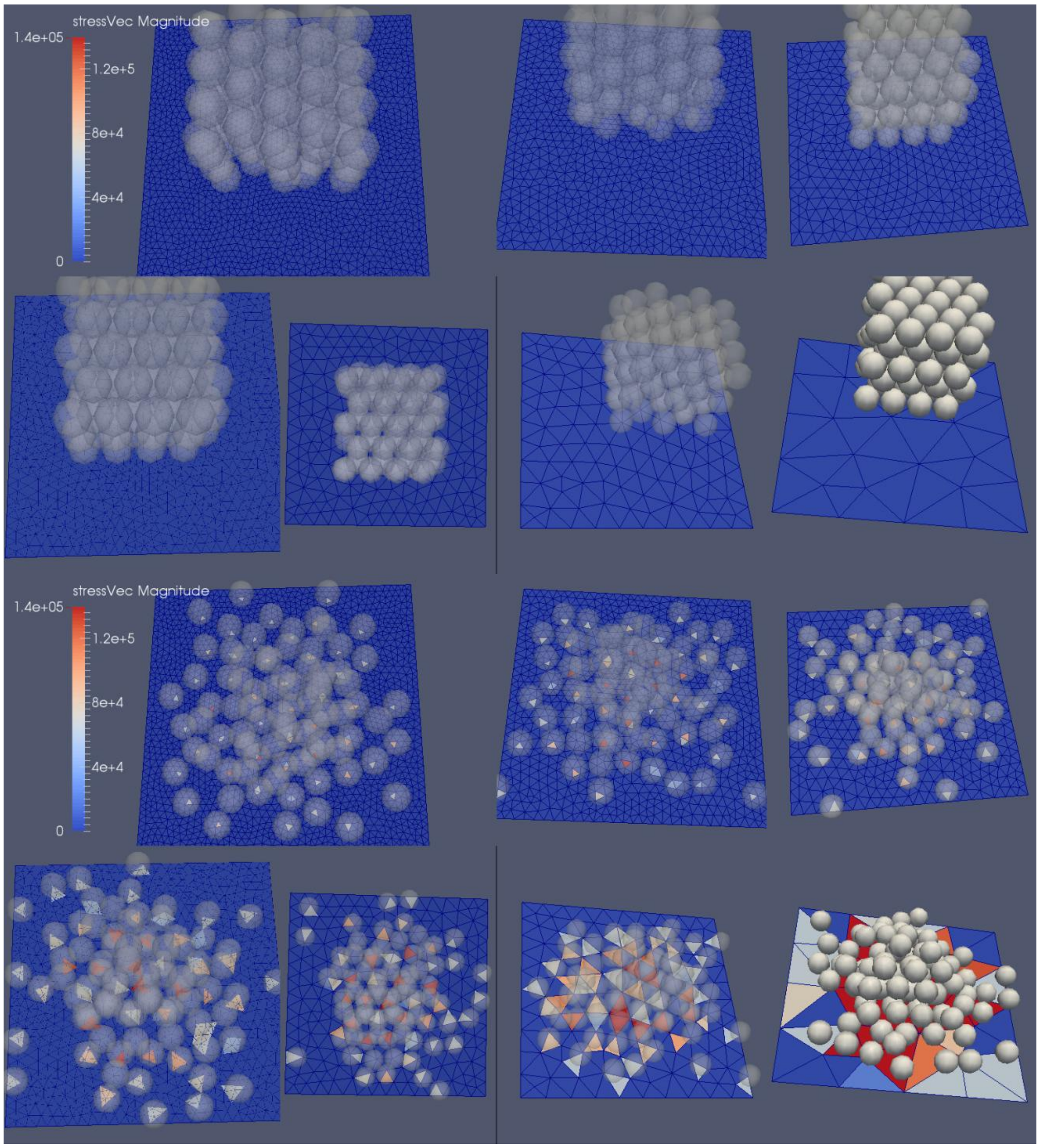
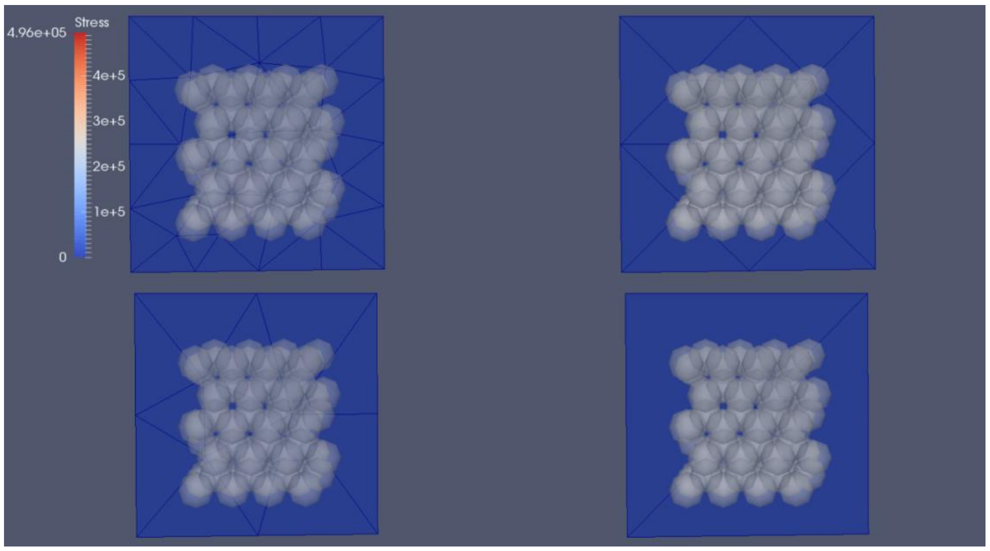


Figure 2.5.1: Fourteen different densities were tested and the stress on the surface areas was colour mapped for comparison.

In this experiment, a small cube was dropped several times onto a flat square facet, with each simulation using a different triangular element density. The excepted result was that the highest density square would have many different coloured elements and provide a higher degree of accuracy than the lower resolution ones. However, this was not the case. It appears that the high density squares only show sphere interactions on single elements and not on their neighbouring triangles.

After seeing the results of the simulations, C-CORE has now hypothesized that there is a sweet spot in relation to the average sphere radius for getting the best data. Therefore, another experiment was done has shown in the figure below.



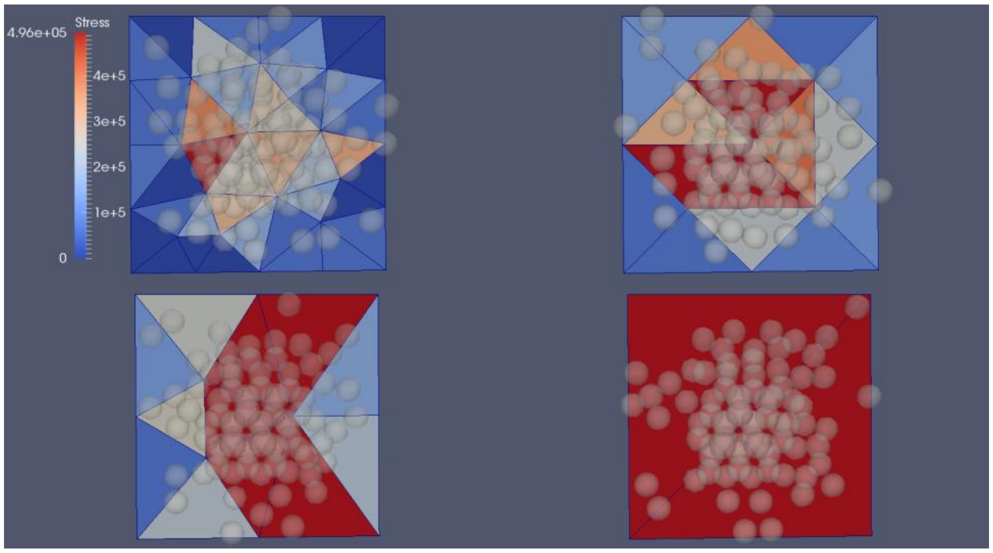


Figure 2.5.2: In these simulations, the triangle densities have much less variation and are generally close to the radius of the spheres. The completely red rectangle is made of only two triangles, the lowest density possible.

Finally, four simulations were done to gather more datasets so that C-CORE can determine if the total forces imparted on the triangles during the entire simulation add up to the same value for different element densities and to for comparison with theoretical calculations. Calculations show that there was between 0.3-0.7% error from the simulation. This shows that the higher the triangle densities the greater the error, but that the overall error is very small.

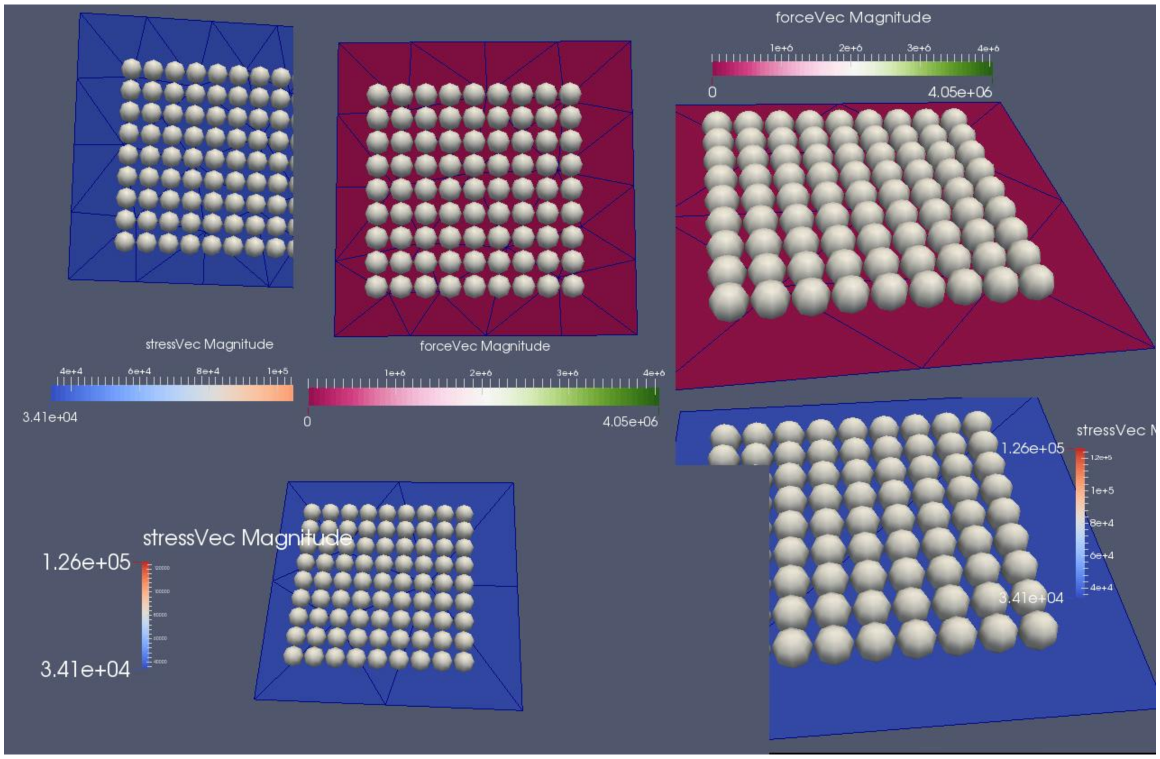


Figure 2.5.3: Four datasets are compared in ParaView. The two top left squares are from the same dataset and there is a graphical error created when the video was rendered.

2.6. Iceberg Net Proposal:

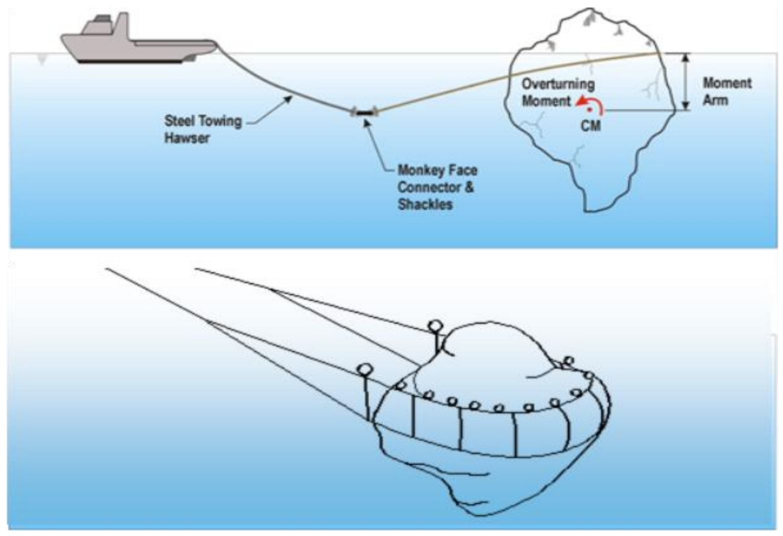


Figure 2.6.1: Courtesy of https://www.c-core.ca/ice

Yade has the ability to model meshed nets made with discrete nodes and wires. Given that C-CORE has expertise in the process of towing ice bergs with large nets, this would be a possible opportunity to expand the research on DEM by simulating the process in Yade.

3. Other Yade Features:

3.1. What .mesh, .stl, .geo (GenGeo and Gmsh), .gts, and .vtu are:

Burns (1989) defines the stl file format as:

> "An StL ("StereoLithography") file is a triangular representation of a 3-dimensional surface geometry. The surface is tessellated or broken down logically into a series of small triangles (facets). Each facet is described by a perpendicular direction and three points representing the vertices (corners) of the triangle..."

Yade is able to directly import an stl file as a facet using the function ***ymport.stl()***.

The main usage for stl in C-CORE's work is that it can be converted into gts format using the terminal; an action not possible with .mesh. The terminal command ***gmsh -2 filename.mesh -o filename.stl*** opens the named mesh file in Gmsh and then creates a new version as an stl file. Afterwards, the command ***stl2gts -r < volumeName.stl > volumeNamegts*** is used to convert the stl file into gts format. Stl essentially acts as an intermediary for converting mesh into gts. mesh files contain plaintext that describe how the 2D or 3D mesh shape is made up of small little triangles elements and are very similar in nature to stl files.

"GTS stands for the GNU Triangulated Surface Library. It is an Open Source Free Software Library intended to provide a set of useful functions to deal with 3D surfaces meshed with interconnected triangles." (GNU Triangulated Surface Library, 2011)

This format is important because it is currently the only way in which a packing volume can be generated using a CAD program and not using the Yade packing functions. This is the best way to achieve particle packings in shapes that are not basic geometries. Yade allows the user to treat an imported gts file as a volume to be filled with spheres as long as the geometry yields a fully closed volume with finite boundaries.

Both GenGeo and Gmsh produce files in their own proprietary .geo formats. GenGeo's files contain coordinates and radi for every sphere written to a generated packing. Differently, Gmsh loads and saves to .geo files to produce geometries when using the software. Every command that the user does is saved to the active .geo file and the user can also edit the file to alter the 3D shape that Gmsh renders on screen. The file formats have nothing in common except for their extensions.

Using the ***VTKRecorder***() function, Yade is able to periodically write the simulation data to .vtu and .vtp files. These file formats hold the facet and sphere movements, alongside physical data such as velocity, stress, and force that each object in the simulation possesses.

3.2. Remote Desktop:

It is possible to setup any of the Yade machines for remote desktoping. This would allow the user to alter simulations or check data while out of the office. However, there are security concerns with enabling remote access. As it is always possible for unwanted people to hack into a remotely accessible system there is always the danger of research being stolen. Considering that Chinese hackers compromised the National Research Council, one of C-CORE primary collaborators, in 2014 it is logical to assume that C-CORE would be a good target too (Barton, 2014). Unless a computer security expert reviews the system's security, it is not recommended that remote desktoping be used at this time.

3.3. Snapshots in Yade:

Yade is capable of saving screenshots of the current simulation view at user-defined periods. The ***qt.SnapshotEngine()*** function allows Yade to periodically write images of a chosen format to a folder. JPEGs are preferred over the PNG format due to the file sizes. PNGs are usually between 1-2MB whereas JPEGs are just 100-200kB in size when generated by Yade. Furthermore, ***utils.makeVideo()*** is able to take all the screenshots created by ***qt.SnapshotEngine()*** and render them into a video made by compositing all the images. Yade actually invokes the software ***mencoder*** when creating the video file. But this feature has generally given lackluster results. The software is prone to rendering errors and hard to debug. Overall, ParaView's data visualization and animation capabilities far exceed the usefulness of these functions.

3.4. Using Batch mode in Yade:

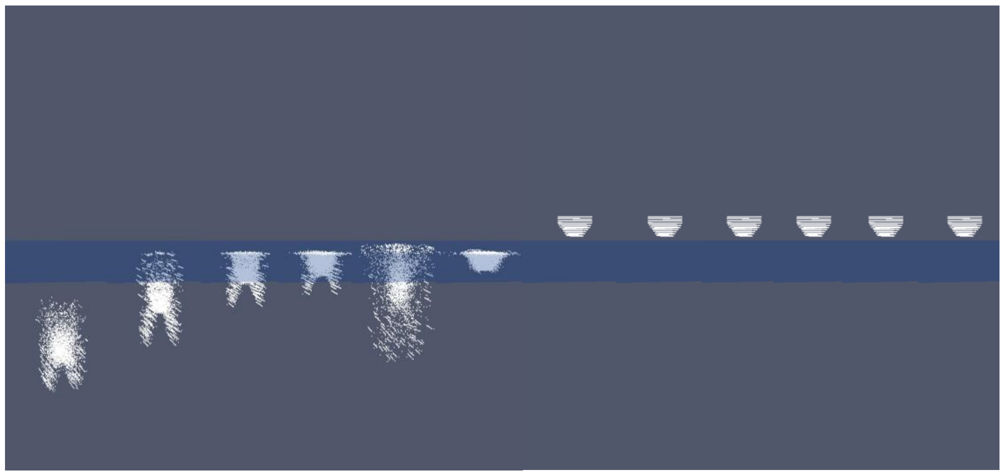


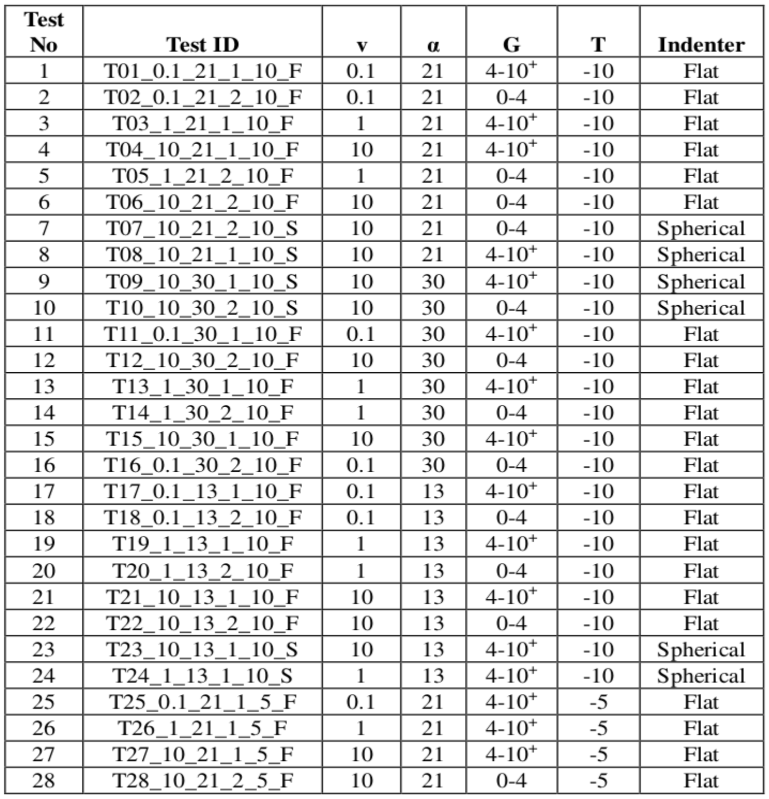
Figure 3.4.1: Batch simulations of a keel falling in water are compared in ParaView

Six unique simulations were run on six CPU cores simultaneously using the Batch mode in Yade. The feature allows the user to set a variation of any chosen parameters in a plain text file and run the scripts automatically. This saves the user the tedium of manually altering the desired variables in the Yade script and invoking Yade to run the individual scripts. In addition, batch mode allows the user to monitor the simulation progress by entering http://localhost:9080 into their web browser address bar.

In figure 3.4.1, the damping and gravity variables were arbitrarily changed to test the capabilities of batch mode. Then the datasets were loaded into six separate 3D views in ParaView so that a comparison of behaviour could easily be seen.

Table 3.4.1 lists the various parameters used by Habib et al. Given that the indenter simulation is derived from this paper, it might be useful to leverage batch mode to emulate the varying parameters in the table.

Table 1: Test Matrix



Test No	Test ID	v	α	G	T	Indenter
1	T01_0.1_21_1_10_F	0.1	21	$4\text{-}10^+$	-10	Flat
2	T02_0.1_21_2_10_F	0.1	21	0-4	-10	Flat
3	T03_1_21_1_10_F	1	21	$4\text{-}10^+$	-10	Flat
4	T04_10_21_1_10_F	10	21	$4\text{-}10^+$	-10	Flat
5	T05_1_21_2_10_F	1	21	0-4	-10	Flat
6	T06_10_21_2_10_F	10	21	0-4	-10	Flat
7	T07_10_21_2_10_S	10	21	0-4	-10	Spherical
8	T08_10_21_1_10_S	10	21	$4\text{-}10^+$	-10	Spherical
9	T09_10_30_1_10_S	10	30	$4\text{-}10^+$	-10	Spherical
10	T10_10_30_2_10_S	10	30	0-4	-10	Spherical
11	T11_0.1_30_1_10_F	0.1	30	$4\text{-}10^+$	-10	Flat
12	T12_10_30_2_10_F	10	30	0-4	-10	Flat
13	T13_1_30_1_10_F	1	30	$4\text{-}10^+$	-10	Flat
14	T14_1_30_2_10_F	1	30	0-4	-10	Flat
15	T15_10_30_1_10_F	10	30	$4\text{-}10^+$	-10	Flat
16	T16_0.1_30_2_10_F	0.1	30	0-4	-10	Flat
17	T17_0.1_13_1_10_F	0.1	13	$4\text{-}10^+$	-10	Flat
18	T18_0.1_13_2_10_F	0.1	13	0-4	-10	Flat
19	T19_1_13_1_10_F	1	13	$4\text{-}10^+$	-10	Flat
20	T20_1_13_2_10_F	1	13	0-4	-10	Flat
21	T21_10_13_1_10_F	10	13	$4\text{-}10^+$	-10	Flat
22	T22_10_13_2_10_F	10	13	0-4	-10	Flat
23	T23_10_13_1_10_S	10	13	$4\text{-}10^+$	-10	Spherical
24	T24_1_13_1_10_S	1	13	$4\text{-}10^+$	-10	Spherical
25	T25_0.1_21_1_5_F	0.1	21	$4\text{-}10^+$	-5	Flat
26	T26_1_21_1_5_F	1	21	$4\text{-}10^+$	-5	Flat
27	T27_10_21_1_5_F	10	21	$4\text{-}10^+$	-5	Flat
28	T28_10_21_2_5_F	10	21	0-4	-5	Flat

Table 3.4.1: Variation of indentation experiment parameters from (Habib et al, 2014).

4. Hardware Tests:

Yade has severe limitations with its multicore processor support. Yade's batch feature can be used to run an individual simulation on each thread of the processor but single simulations gain relatively little from each extra thread on the CPU.

To test the primary Yade workstations, C-CORE developed a simple script to run the Yade performance command for every possible number of threads on the CPU. Generally, the script tested the Yade performance for 1 to 8 active cores and wrote the results to individual text files. The results are summarized in appendix B.

Originally, the computer ***rsarracino*** was running Yade on Ubuntu 12.04 LTS and received stagnant results when adding more threads to the performance test. These results seemed counterintuitive especially when compared to the substantial gains made by ***CARDLN1***. It was decided that ***rsarracino*** should be upgraded to 14.04 LTS and retested. After the machine was retested, its performance followed the same trend as ***CARDLN1***. It can be extrapolated that the computer's operating system should be updated to coincide with the Yade developers' most current Ubuntu package distribution.

The Yade team currently supports releases on several Ubuntu versions as listed on http://yade-dem.org/packages/. It is currently recommended that C-CORE update any machines used for Yade to match the newest Ubuntu distribution that the Yade developers are supporting. It is likely that Yade will mainly be supported for LTS distributions of Ubuntu. Given that LTS versions are supported for 2 years, the next version that C-CORE will have to upgrade to will be 16.

5. File Management:

The Numerical Modeling group is currently not running any routine backups of its computers or storing files on a network drive. It would be disastrous if any hard drives failed as all of its data would be lost. It is very likely that the IT department has implemented data redundancy somewhere else in C-CORE and this should be easy for them to implement. Moreover, adding a shared network drive to all the Numerical Modeling group's computers would make collaboration and file sharing much easier. Instead of emailing or using USB drives to transfer files, employees could then access the data on the network drive from their own computers. This too is should be a routine task for the IT department.

5.1. Implementing Source Control:

Source control is a method of tracking and backing up changes made to computer files. Common systems for version control include Git, Subversion, and Mercurial. Interfaces such as GitHub and Tortoise SVN are built upon these systems to provide multiple users with the ability to work on the same files and projects simultaneously without overwriting one another's work.

The Yade developers use GitHub, which is based on Git, to maintain the Yade source code and any changes made to it. Anyone willing can a make an open source contribution to Yade and since C-CORE wants to modify the source code it is very likely someone within in the group will eventually have to learn how to use GitHub to make commits and create branches of the Yade code. Furthermore, C-CORE could subscribe to GitHub to receive their own private code repository wherein the staff could keep track of changes and share internal scripts. CCORE has subversion repositories setup for software development and project management but GitHub is specifically required if C-CORE wants to properly contribute to the Yade project. At the end of the day, no serious software development team can avoid source control if they want to a smooth, efficient workflow.

5.2. Improving Codebase:

The Numerical Modeling group is going to be expanded to a total of three numerical modellers, one PhD Student, one Masters Student, and two work term students.

As the scope of the project is being expanded, it is important to consider the work tools and practices the group will be using. It is necessary to plan ahead to mitigate any complications.

Adopting a style guide for Python scripts would help standardize work practices and provide a more coherent codebase. Python Enhancement Proposal 8 (PEP8) contains in depth and easy to understand guidelines for writing easily readable Python code. The group should review the style suggestions and adopt style practices to maintain consistency. In addition, PEP8 should be reviewed so that the reader can take note about typically is considered bad code formatting.

PEP8 suggests that the programmer uses four spaces per indentation level instead of tabs to create indentation. Since indentation is a defining feature of the language, C-CARD must come to a conclusion of whether to standardize Python scripts with tabs or spaces (Rossum et al, 2014). This report recommends that the group follows PEP8 and use four spaces per indentation. But ultimately consistency is more important than the actual style used.

5.3. Modularization of Repeated Code:

Currently, there is no modularization to reduce repeated code stubs. Several of the scripts contain the exact same ***ApplyBuoyancy()*** definition and also the same code to generate the water level facets. It was attempted to covert these functions into an importable module but there were errors due to the global namespace feature of Python. Yade had problems generating facets when the command was invoked from an imported module.

So far, a small module named ***ccore_yade.py*** has been written to contain a few helper functions whose purposes are mainly to manage file writing, record the timestep, and convert .mesh files into .gts format. Currently, the module has to be located in the same directory as the Yade scripts and thus not very modular. In the future, it would be desirable for someone to properly install the module as a real Python module using Linux environment variables, so that changes in the module can be done system wide and shared more easily between workstations.

6. Visualization of Data Using ParaView:

This section does not seek to supersede or replace the various ParaView tutorials and documentation available but to illuminate specific topics that are of special interest to C-CORE.

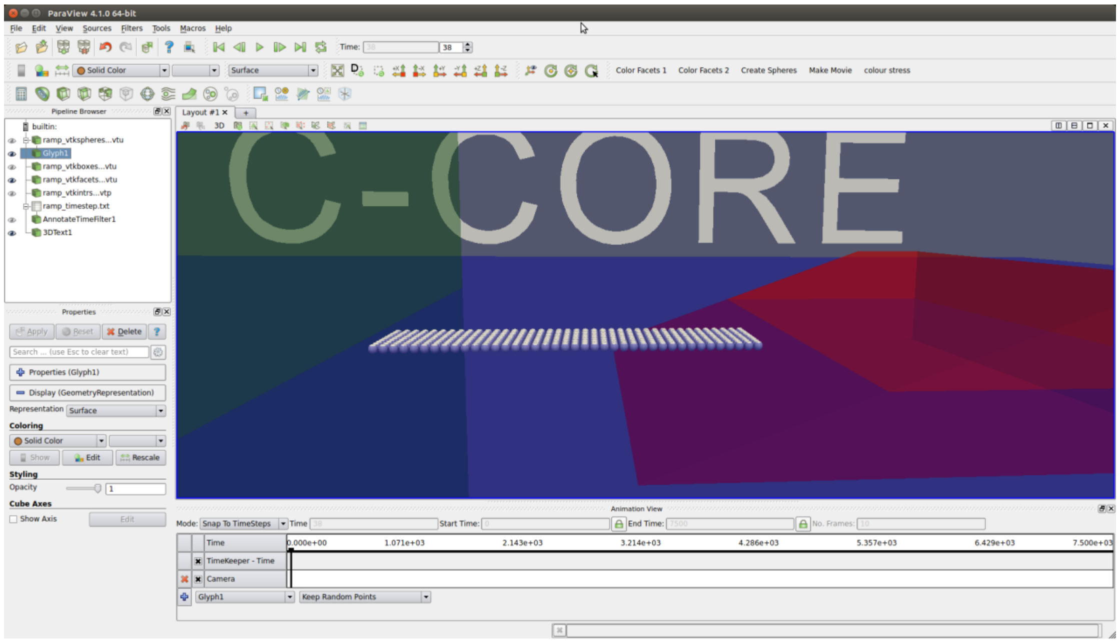


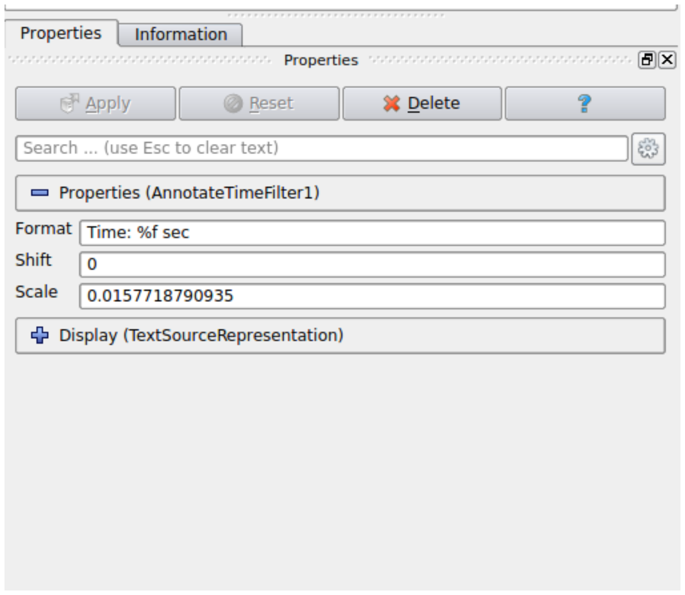
Figure 6.1: The wall and ramp simulation being edited in ParaView.

6.1. Sphere Glyphs:

To get an accurate representation of the spheres the ***Glyph*** tool must be used. Weatherly (2011) suggests that ***sphere radius*** be set to 1.0, the scale mode set to ***scalar*** and that the ***Scale Factor*** is set to 1.0. Additionally, unchecking ***Mask Points*** will allow ParaView to display all possible spheres in the case of very large simulations. By default ParaView will hide spheres past a certain threshold to increase performance.

6.2. Time Source Filter:

When ParaView is loaded with a multiple file dataset, it interprets each file as a timestep. It is up to the user to assign meaning to the time. Using ***Time Source Filter*** allows the user to create a time indicator on the 3D view. To receive an accurate representation of time, the ***scale*** parameter must be set to the timestep*VTKRecorder period. Essentially, it is the number of timestep in a period multiplied by the quantity of time that each iteration represents.


Properties
Information
Properties
Apply
Reset
Delete
?
Search ... (use Esc to clear text)
Display (TextSourceRepresentation)

Properties (AnnotateTimeFilter1)

Format	Time: %f sec
Shift	0
Scale	0.0157718790935

Figure 6.2.1: The Time Filter properties window in ParaView

6.3. Extract Selection:

Using the ***Find Data*** tool the user can perform filtering of the Yade spheres and facets. Since Yade saves all facets in a simulation to one file, it makes it harder to isolate and observe the behaviour of each facet when they are rolled into one manipulatable piece of data. Fortunately the facets can be filtered by their ***MaterialID.*** As long as every facet is assigned a unique ***MaterialID*** in Yade then they can be filtered into a separate file and examined independently.

6.4. Visualizing Bonds:

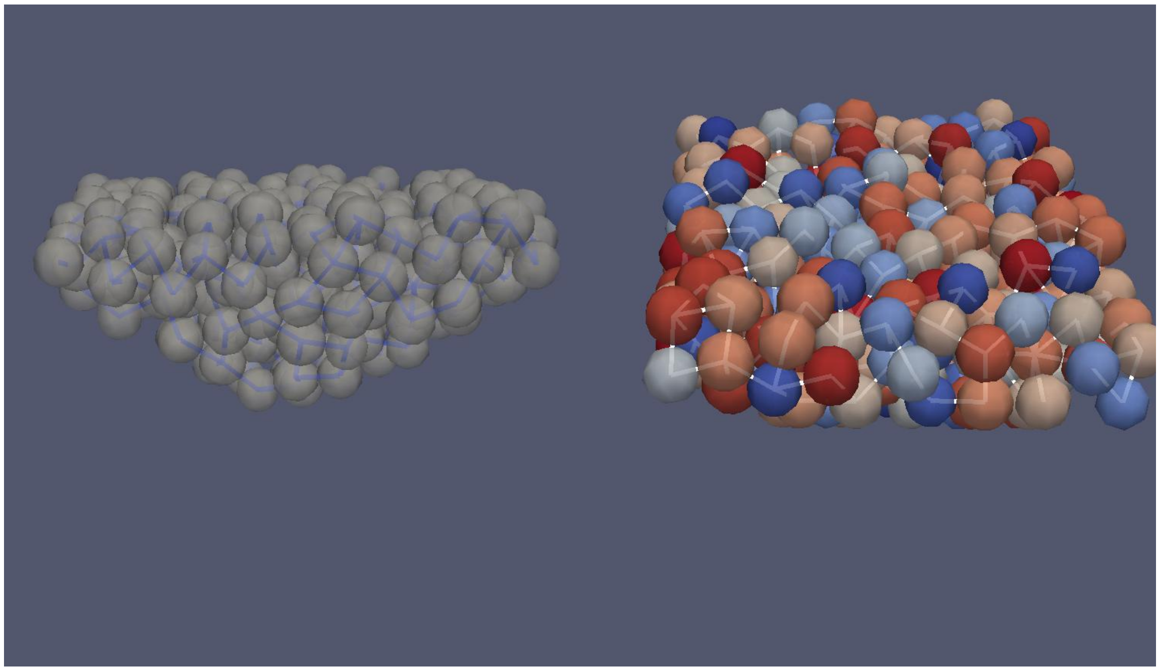
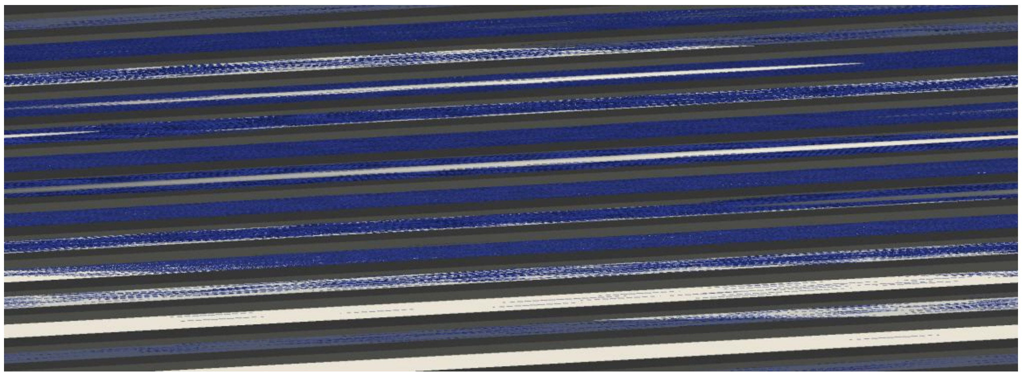


Figure 6.4.1: All interactions, including bonds, are being shown for the keel packing.

Visualization of cohesive bonds can be done in ParaView with one caveat. The intr (interactions) files generated by Yade record all interactions at the specific timestep. Interactions besides bonds may be unknowingly added to the visualization and in effect ruin the accuracy of the bonding visual. Duriez (2014) has suggested that the code used to visualize bonds for the JCFpm model can be adapted for the Cpm model.

6.5. Animation Problems:

At present there is an unsolved problem with video rendering in ParaView. For reasons unknown, rendered videos are heavily distorted or the views are skewed from their intended viewing point. The figure below shows two ways in which the videos can end up:



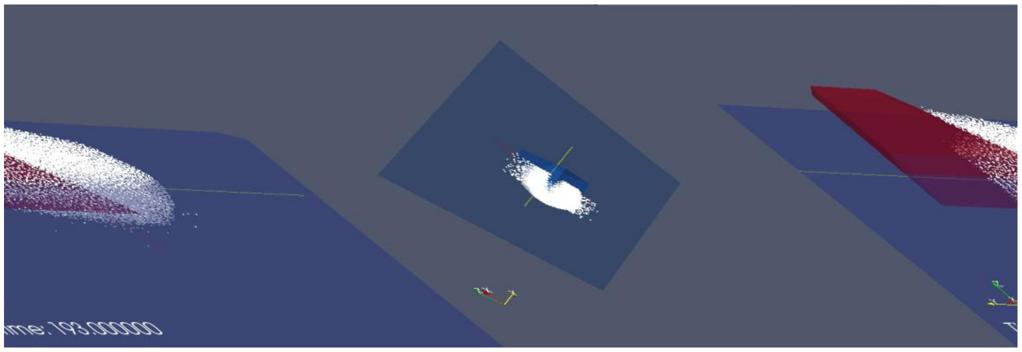


Figure 6.5.1: Screenshots of improperly rendered videos made in ParaView.

One potential cause for this may be resizing the view while the animation is being rendered. The resizing may be caused unintentionally be the user when they minimize ParaView or when using the ***Workplace Switcher*** in Ubuntu. Anecdotally, it seems that this problem occurs on more computationally intensive renderings in which ParaView lags and locks up while generating each successive frame of the animation. Some simple experimentation has indicated that rendering the video as an .ogv or .avi does not fix this problem and neither does rendering the video as uncompressed.

6.6. Camera Interpolation:

In the ***animation toolbar*** the user can make the camera, during an animation, follow a set path. By setting a starting camera point position at the first time step and an ending position in the last timestep, any animation generated in ParaView will interpolate a path from the beginning to the end position. Adding intermediate points can refine the path in case the original path is insufficient at tracking the data.

6.7. Macros and Python Scripting:

Tedious and repeated actions can be reduced by writing Macros and Python scripts for ParaView. The software has a Python interface and allows the user to program Python macros. Simple macros can be created using the trace tool. For example, the steps used to create an accurate glyph for a sphere can be traced and recorded as a macro. At the click of a button, the user will be able to create a glyph of the sphere and colour them if they desire.

7. Conclusions:

- Yade is currently better suited to CORE's needs than ESyS-Particle
- Several particle packing methods were presented for Yade
- Yade's documentation and community support are good
- A framework for developing Yade simulations was described
- The current ongoing simulations were explained
- Stress on facet triangle mesh elements can be accurately simulated in Yade even with high density meshes
- Simulation of iceberg nets was proposed

- Various file formats used for simulations were explained and compared
- Remote desktop capabilities of Yade were analyzed
- Explanation of why snapshots feature is not used
- Use of batch simulation was suggested
- Hardware test results were presented
- Several suggestions to improve codebase and software developments
- C-CORE specific ParaView concerns were mentioned

8. Recommendations:

The current pace of results in DEM simulations indicate that using Yade instead of ESyS-Particle is the correct course of action. Although it is less powerful in performance, the documentation and support make creating new scripts much easier and reliable. The current framework presented allows one to go from paper design to post processing in a predictable and routine workflow. There are no unambiguous steps to take. The best way to improve it would review other CAD packages against Gmsh and see which ones can produce better or faster results.

There exists an opportunity to leverage pre-existing knowledge at C-CORE and development another simulation. The iceberg proposal should be reviewed for usefulness as a research topic and developing it would enhance the Yade knowledge base at C-CORE. It would be nice short term project for a work term student to implement the more exotic Yade models such as meshed nets. Another worthy project would be to utilize batch mode to implement the various versions of the indenter simulation.

In addition, simulation data indicates that using low and high density facets produces accurate results. For those concerned with performance, the amount of triangles on a facet should be limited.

Due to security concerns, the potentially useful ability to remote desktop should not be used until an expert weighs in on this topic. Additionally, backups of computer data are needed and more files should be placed on network drives for easy sharing.

There are several things that can be done to improve the quality of the code and development environment. First of all, a private GitHub repository needs to be created for hosting DEM scripts and for modification to the Yade source control. The Python style guide PEP8 should be adopted or modified to suit the needs of the team and modules need to be created to make code changes more global. Also, as the hardware tests indicates, all workstations used for Yade simulations need to be updated to at least Ubuntu 14.04LTS due to the performance increases and Yade developer support. Lastly, if proper visualization of bonds in ParaView is wanted, someone will have to adapt the jcfpm code for the Cpm model.

References

Barton, R. (2014, July 29). Chinese cyberattack hits Canada's National Research Council. *Canadian Broadcasting Company*. Retrieved from http://www.cbc.ca/news/politics/chinese-cyberattack-hits-canada-s-national-research-council-1.2721241

Duriez, J (2014, Aug 18). *How to visualize bonds/cohesion using intr recorder and Paraview?*. Retrieved from https://answers.launchpad.net/yade/+question/253105

GNU Triangulated Surface Library. (2011, Oct 25). *The GTS Library*. Retrieved from http://gts.sourceforge.net

Habib, K., Taylor, R., Jordaan, I., and Bruneau, S. (June 2014). *EXPERIMENTAL INVESTIGATION OF COMPRESSIVE FAILURE OF TRUNCATED CONICAL ICE SPECIMENS*. Paper presented at ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering, San Francisco, CA. St. John's NL: Centre for Cold Ocean Resources.

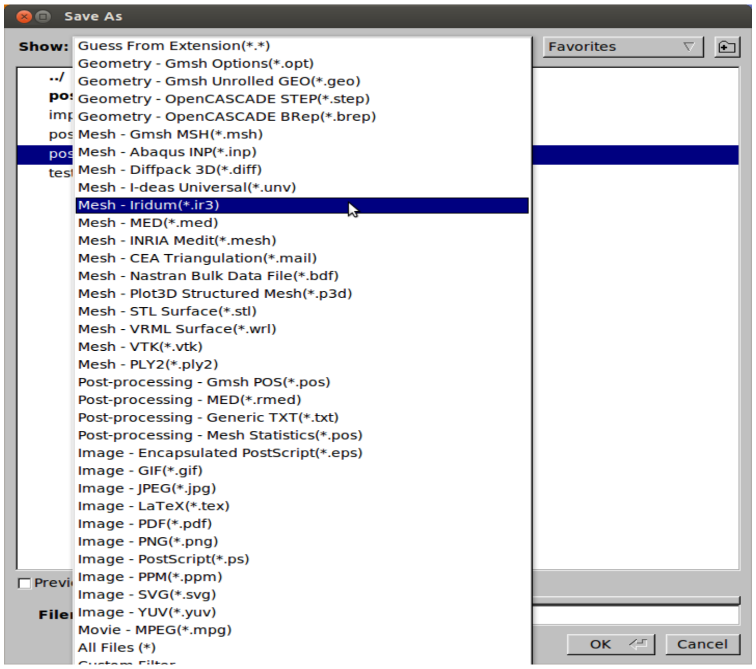
Rossum, G., Warsaw, B., and Coghlan, N. (2014, Aug 17). *Style Guide for Python Code*. Retrieved from http://legacy.python.org/dev/peps/pep-0008/

Šmilauer, V. (2010, Nov 28). *model.py*. Retrieved from https://github.com/yade/trunk/blob/588dc7d3b241373a087f66 7b0a3bf3d12b071fe8/examples/rod-penetration/model.py

Appendix A

Gmsh Tutorial:

GMSH is a free and open source CAD software package. It is used over commercial software such as SolidWorks or AutoCAD because it is capable of converting many of the various file formats that Yade can use. Gmsh's specialty is in handling triangular mesh designs. It supports the following list of file formats:



Guess From Extension(*.*)
Geometry - Gmsh Options(*.opt)
Geometry - Gmsh Unrolled GEO(*.geo)
Geometry - OpenCASCADE STEP(*.step)
Geometry - OpenCASCADE BRep(*.brep)
Mesh - Gmsh MSH(*.msh)
Mesh - Abaqus INP(*.inp)
Mesh - Diffpack 3D(*.diff)
Mesh - I-deas Universal(*.unv)
Mesh - Iridum(*.ir3)
Mesh - MED(*.med)
Mesh - INRIA Medit(*.mesh)
Mesh - CEA Triangulation(*.mail)
Mesh - Nastran Bulk Data File(*.bdf)
Mesh - Plot3D Structured Mesh(*.p3d)
Mesh - STL Surface(*.stl)
Mesh - VRML Surface(*.wrl)
Mesh - VTK(*.vtk)
Mesh - PLY2(*.ply2)
Post-processing - Gmsh POS(*.pos)
Post-processing - MED(*.rmed)
Post-processing - Generic TXT(*.txt)
Post-processing - Mesh Statistics(*.pos)
Image - Encapsulated PostScript(*.eps)
Image - GIF(*.gif)
Image - JPEG(*.jpg)
Image - LaTeX(*.tex)
Image - PDF(*.pdf)
Image - PNG(*.png)
Image - PostScript(*.ps)
Image - PPM(*.ppm)
Image - SVG(*.svg)
Image - YUV(*.yuv)
Movie - MPEG(*.mpg)
All Files (*)

Figure A.1: Saving file window in Gmsh.

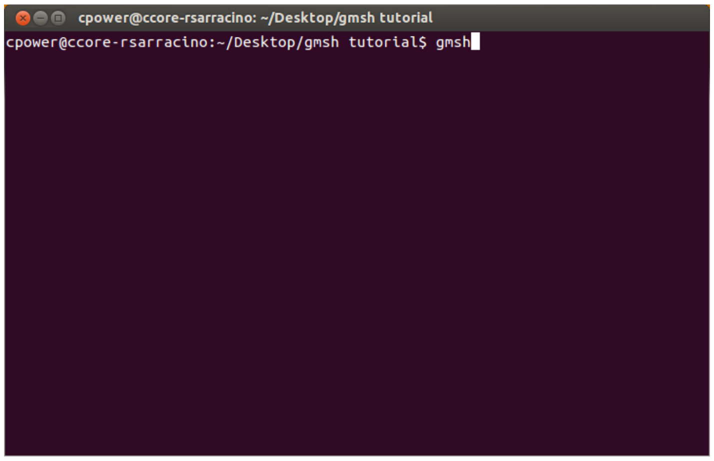
Gmsh is suitable for creating uncomplicated geometries but lacks ease of use when attempting to create complex designs. It is simple to use, but inelegant and cumbersome. The program is prone to crashing when using several of its features but the file changes are always remembered as they are stored as simple scripting commands in Gmsh's native .geo files.

In addition, Gmsh operations can be invoked via terminal commands, this functionality is used in the ***convertMeshToGts()*** function from the ***ccore_yade*** module. C-CORE uses Gmsh to design the Yade facets by specifying the object's dimensions and Cartesian coordinate points. In the event that more complicated shapes need to be simulated, it is feasible to generate the object's design in another CAD program as long as it can save into a format that Gmsh can open and convert into a .mesh file.

Overall, this approach is faster, easier, and more flexible than generating facets via the Yade functions. Mesh files can be scaled, rotated, or translated using both Gmsh and Yade. Plus, the amount of triangular elements comprising the mesh can be fine-tuned in Gmsh. This allows the user to increase the accuracy of the simulation at the cost of performance.

Generating Simple Geometries in Gmsh:

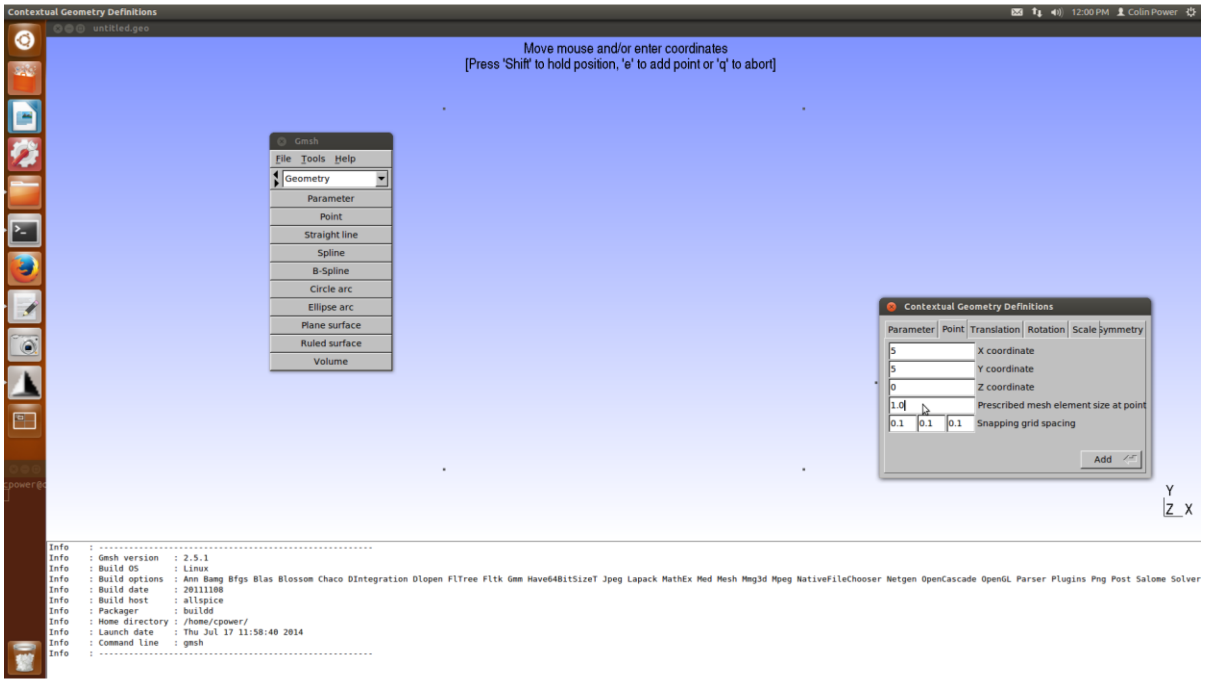
Gmsh is an adequate enough tool to quickly create basic 2D and 3D geometries from a combination of coordinate points, lines, and defined surface areas. Creating a 3D cube is a straightforward process for anyone who has used CAD software before. First, launch Gmsh through the terminal as follows:



```
cpower@ccore-rsarracino:~/Desktop/gmsh tutorial$ gmsh
```

Figure A.2: Launching Gmsh with the Linux terminal.

Next, add points by navigating to ***Geometry->Elementary entities->Add->New->Point***. The ***Contextual Geometry Definitions*** window allows the user to precisely define where they will place their next point. The user chooses the Cartesian coordinates of the point and then they choose a value for ***Prescribed mesh element size at point*** to change the default triangle mesh size of the object. Pressing ***e*** or clicking the ***Add*** button adds the defined point to the .geo file that Gmsh has generated. It will be named ***untitled.geo*** by default and every action will be automatically saved and documented in this file. If the program crashes, the very last command will be saved by default. For a simple square shape, four individual points have been added to define the vertices of the square.



Move mouse and/or enter coordinates
[Press 'Shift' to hold position, 'e' to add point or 'q' to abort]

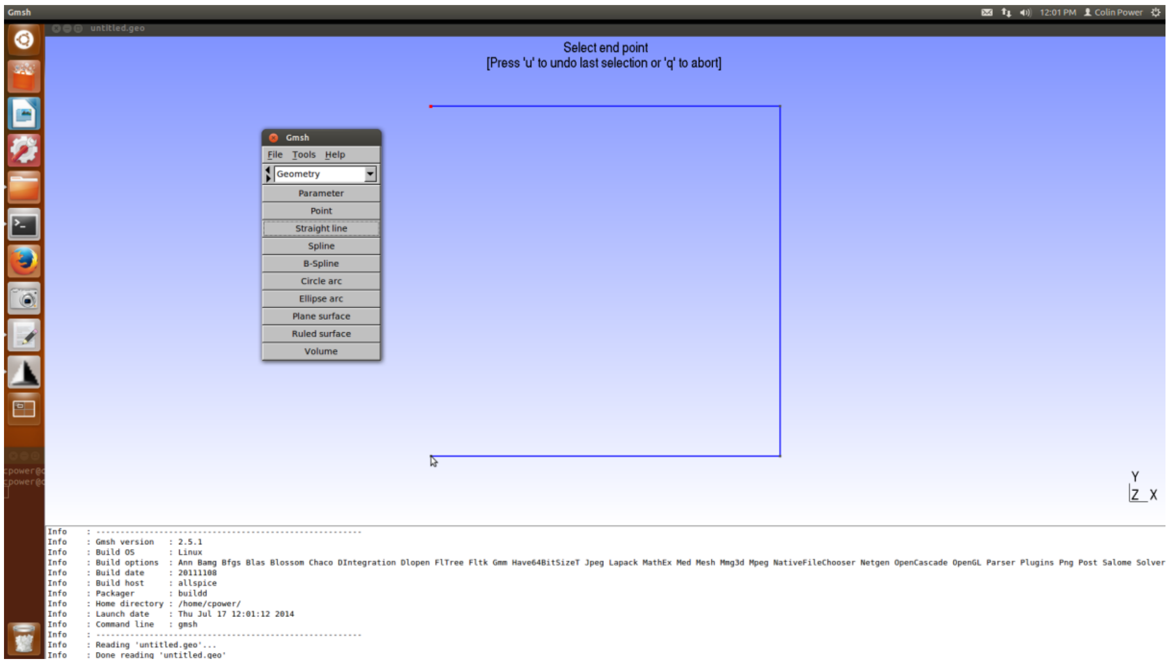
Contextual Geometry Definitions

5	X coordinate
5	Y coordinate
0	Z coordinate
1.0	Prescribed mesh element size at point
0.1 0.1 0.1	Snapping grid spacing

```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 11:58:40 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
```

Figure A.3: Adding points to create an object

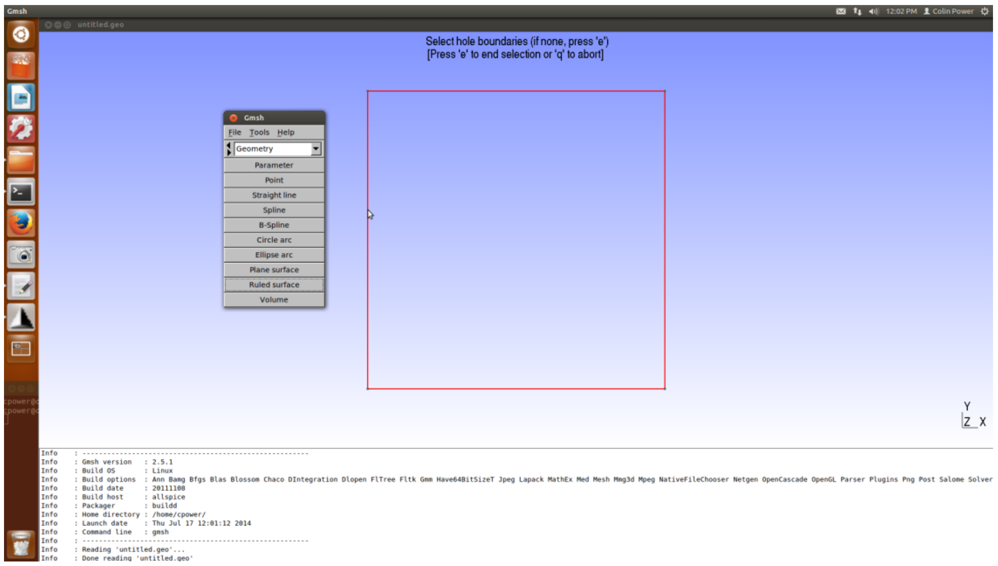
Now, the vertices will be connected using ***Geometry->Elementary entities->Add->New->Straight Line***. Two points will be connected by left clicking on the starting point and then left clicking on the ending point.



```
Info    : --------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 12:01:12 2014
Info    : Command line   : gmsh
Info    : --------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

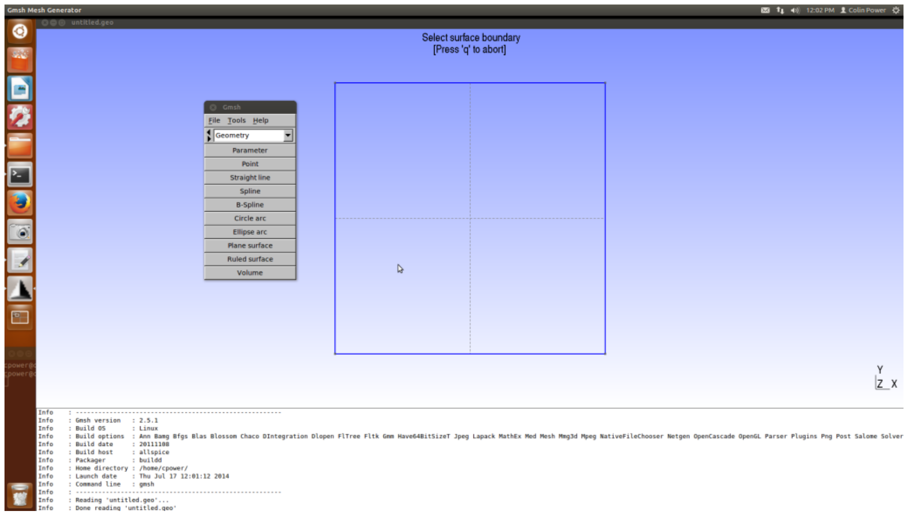
Figure A.4: Connecting the points with straight lines to form a square.

After the lines are connected, using ***Geometry->Elementary entities->Add->New->Ruled Surface*** will define a surface area in which a triangular mesh can be generated. The user clicks the lines until a closed loop in formed and then presses ***e*** to generate a ruled surface.



```
Info    : --------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 12:01:12 2014
Info    : Command line   : gmsh
Info    : --------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

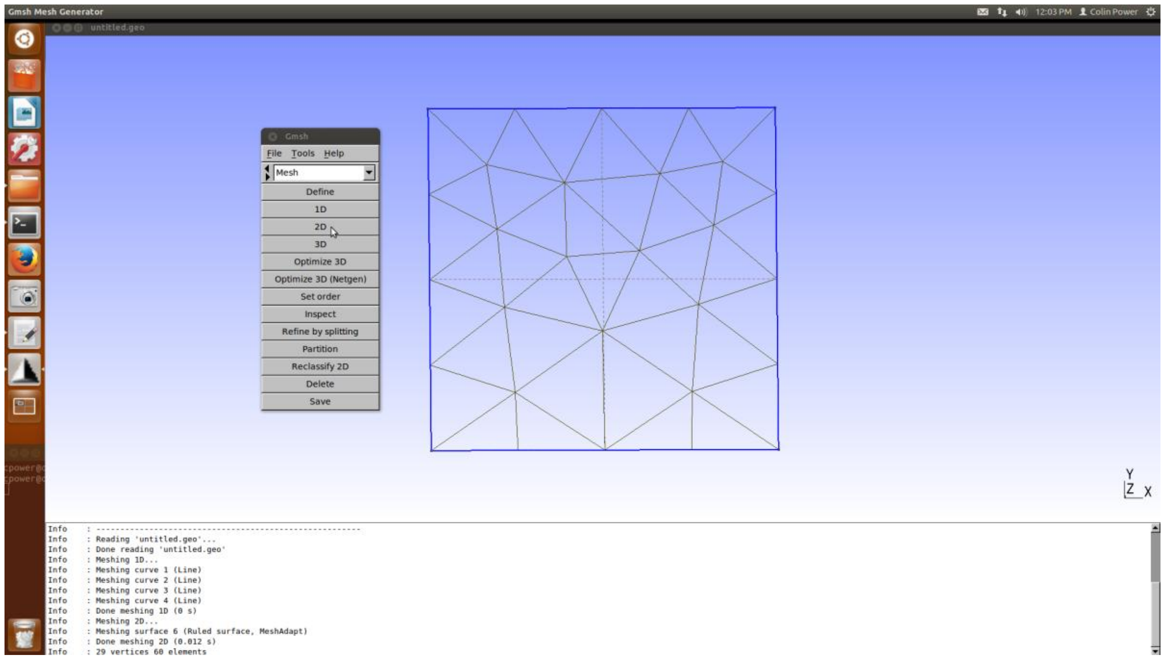
Figure A.5: Lines are selected to create a ruled surface.



```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 12:01:12 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

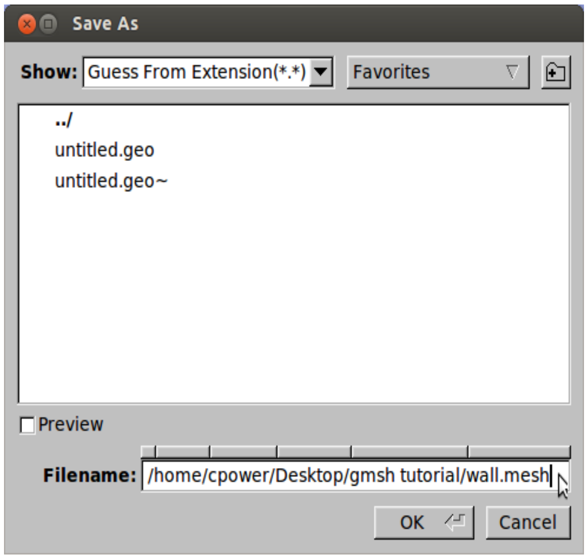
Figure A.6: The ruled surface is created.

Afterwards use ***Mesh->2D*** to generate the interlinked triangles that make up a .mesh file on the ruled surfaces.



```
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
Info    : Meshing 1D...
Info    : Meshing curve 1 (Line)
Info    : Meshing curve 2 (Line)
Info    : Meshing curve 3 (Line)
Info    : Meshing curve 4 (Line)
Info    : Done meshing 1D (0 s)
Info    : Meshing 2D...
Info    : Meshing surface 6 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.012 s)
Info    : 29 vertices 60 elements
```

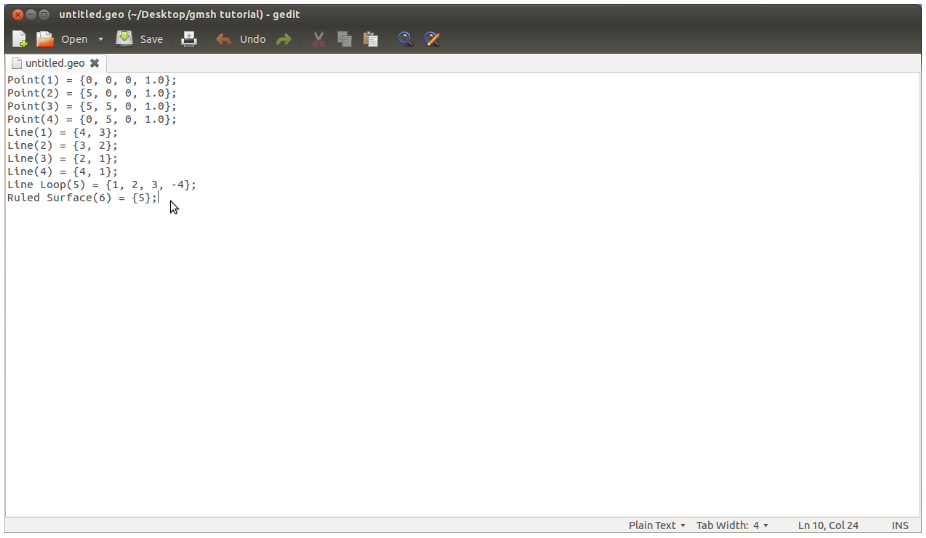
Figure A.7: The mesh pattern is created. These connected triangles can now be imported into Yade.



Filename: /home/cpower/Desktop/gmsh tutorial/wall.mesh

Figure A.8: The mesh will be saved as ***wall.mesh*** using Save as tool. Gmsh will convert the file according to the given extension.

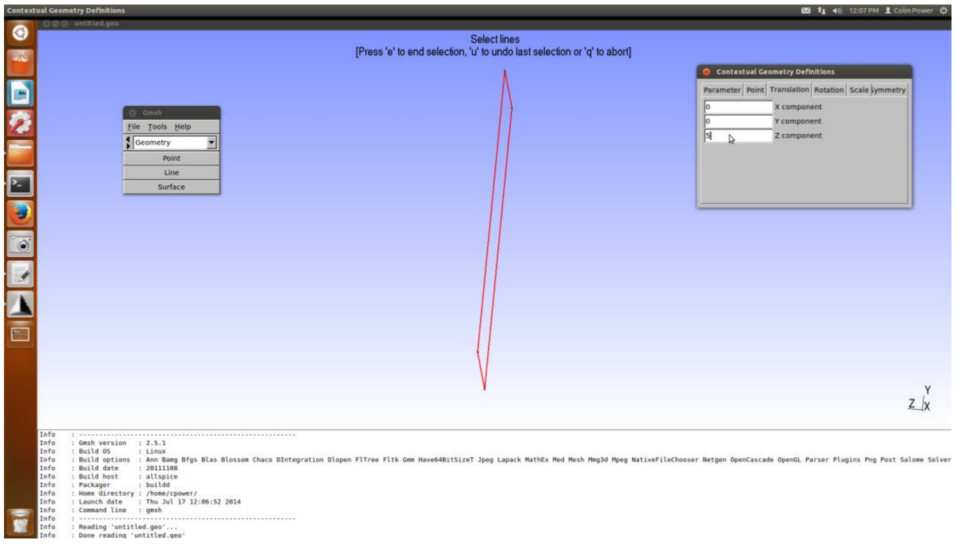
Opening up the .geo file reveals the commands that Gmsh uses to reconstruct the object when it is reloaded. The user can manually alter this file to change the geometry and this method can sometimes be much quicker. Changes to the text file can be made while Gmsh is still editing the file. Pressing ***Geometry->Reload*** will update the Gmsh view to reflect the most recent changes in the .geo file.



```
Point(1) = {0, 0, 0, 1.0};
Point(2) = {5, 0, 0, 1.0};
Point(3) = {5, 5, 0, 1.0};
Point(4) = {0, 5, 0, 1.0};
Line(1) = {4, 3};
Line(2) = {3, 2};
Line(3) = {2, 1};
Line(4) = {4, 1};
Line Loop(5) = {1, 2, 3, -4};
Ruled Surface(6) = {5};
```

Figure A.9: The square is actually created by these series of text command in the geo file

To turn this 2D square into a 3D cube the ***extrusion tool*** can be used instead of defining additional lines and points**.** ***Geometry->Elementary entities->Extrude->Translate->Line*** brings up the ***Contextual Geometry Definitions*** where the magnitude of the extrusion in each axis is defined. In this example, all four lines have been selected and will be translated by five units in the Z-axis.



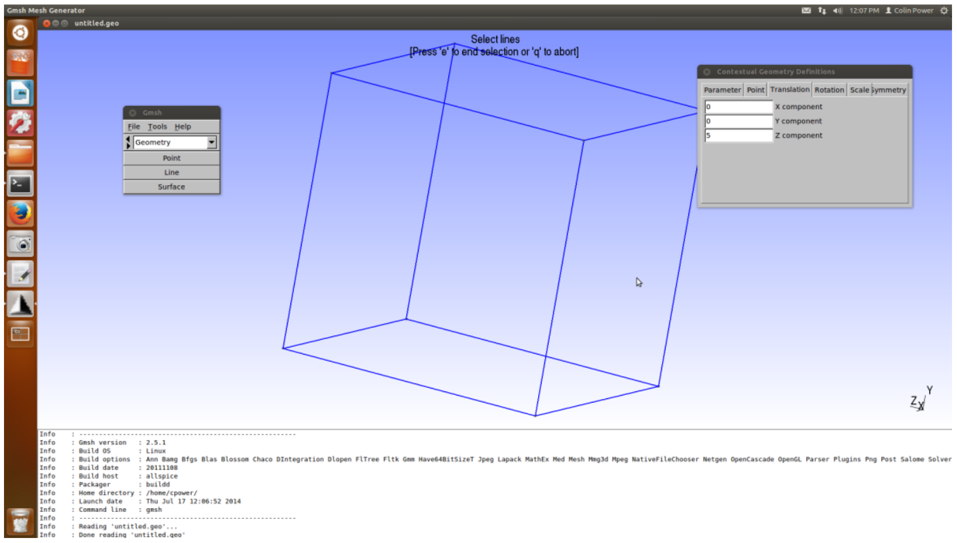
Select lines
[Press 'e' to end selection, 'u' to undo last selection or 'q' to abort]

Contextual Geometry Definitions

0 X component
0 Y component
5 Z component

```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 12:06:52 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

Figure A.9: The square has been selected for extrusion.



Select lines
[Press 'e' to end selection or 'q' to abort]

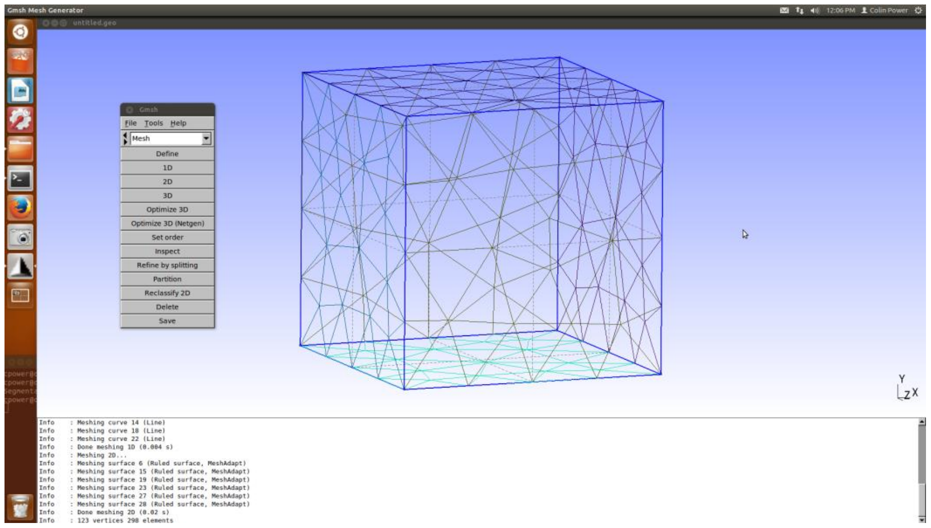
Contextual Geometry Definitions

0 X component
0 Y component
5 Z component

```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Thu Jul 17 12:06:52 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

Figure A.10: The resulting cube made from the extrusion tool. Rotations about a specified axis and coordinate can also be done in Gmsh.

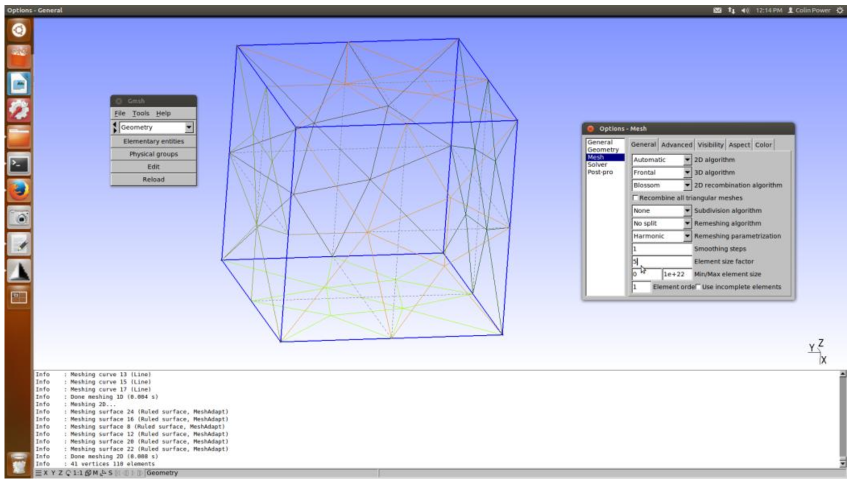
The extrusion process will also extend the ruled surfaces that were previously defined and allow the user to quickly mesh the new shape.



```
Info    : Meshing curve 14 (Line)
Info    : Meshing curve 18 (Line)
Info    : Meshing curve 22 (Line)
Info    : Done meshing 1D (0.004 s)
Info    : Meshing 2D...
Info    : Meshing surface 6 (Ruled surface, MeshAdapt)
Info    : Meshing surface 15 (Ruled surface, MeshAdapt)
Info    : Meshing surface 19 (Ruled surface, MeshAdapt)
Info    : Meshing surface 23 (Ruled surface, MeshAdapt)
Info    : Meshing surface 27 (Ruled surface, MeshAdapt)
Info    : Meshing surface 28 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.02 s)
Info    : 123 vertices 298 elements
```

Figure A.11: The ruled surfaces were also extruded and so a mesh was easily created.

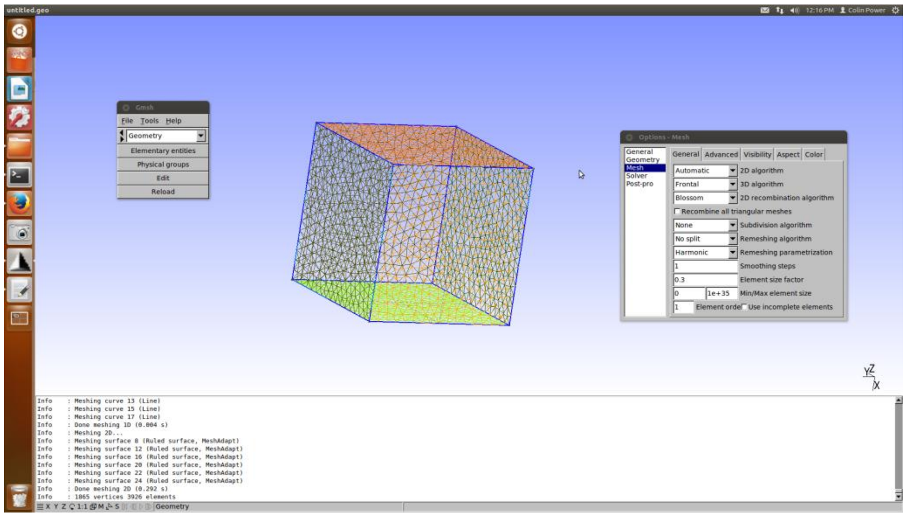
Sometimes it is wanted to change the number of triangles that make up the mesh. ***Tools->Options->Mesh->General->Element size factor*** scales the triangle sizes by the given factor. After this parameter is altered, the user must reload the file and remesh the ruled surfaces to obtain the newly redefined mesh pattern.



```
Automatic    2D algorithm
Frontal      3D algorithm
Blossom      2D recombination algorithm
☐ Recombine all triangular meshes
None         Subdivision algorithm
No split     Remeshing algorithm
Harmonic     Remeshing parametrization
1            Smoothing steps
5            Element size factor
0   1e+22    Min/Max element size
1   Element orde☐ Use incomplete elements
```

```
Info    : Meshing curve 13 (Line)
Info    : Meshing curve 15 (Line)
Info    : Meshing curve 17 (Line)
Info    : Done meshing 1D (0.004 s)
Info    : Meshing 2D...
Info    : Meshing surface 24 (Ruled surface, MeshAdapt)
Info    : Meshing surface 16 (Ruled surface, MeshAdapt)
Info    : Meshing surface 8 (Ruled surface, MeshAdapt)
Info    : Meshing surface 12 (Ruled surface, MeshAdapt)
Info    : Meshing surface 20 (Ruled surface, MeshAdapt)
Info    : Meshing surface 22 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.008 s)
Info    : 41 vertices 110 elements
```

Figure A.12: The number of triangle elements in the mesh was decreased by setting the ***Element size factor*** to 5.

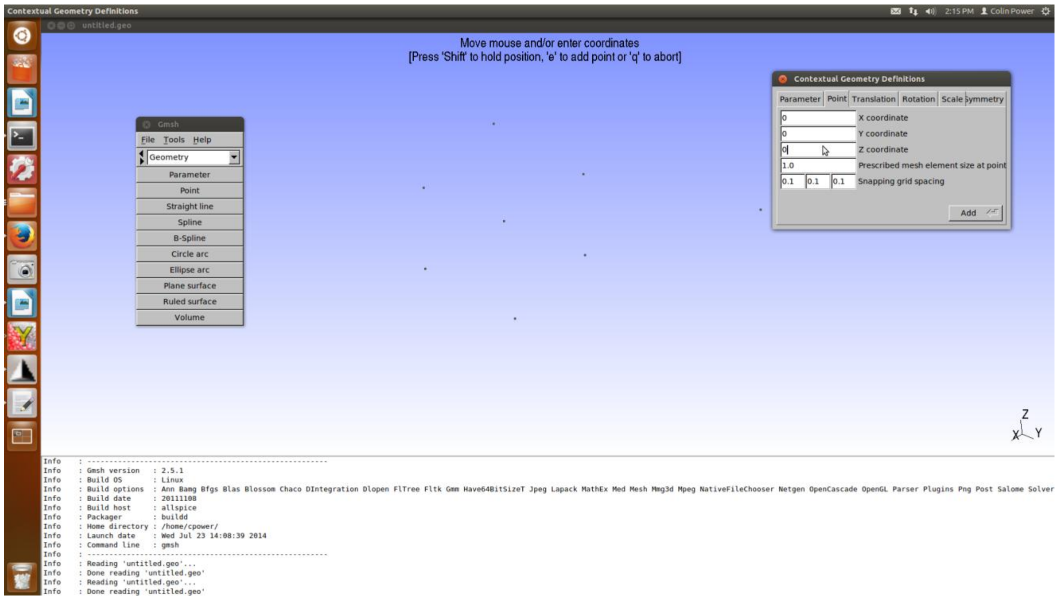


```
Automatic   2D algorithm
Frontal     3D algorithm
Blossom     2D recombination algorithm
☐ Recombine all triangular meshes
None        Subdivision algorithm
No split    Remeshing algorithm
Harmonic    Remeshing parametrization
1           Smoothing steps
0.3         Element size factor
0   1e+35   Min/Max element size
1   Element orde☐ Use incomplete elements
```

```
Info    : Meshing curve 13 (Line)
Info    : Meshing curve 15 (Line)
Info    : Meshing curve 17 (Line)
Info    : Done meshing 1D (0.004 s)
Info    : Meshing 2D...
Info    : Meshing surface 8 (Ruled surface, MeshAdapt)
Info    : Meshing surface 12 (Ruled surface, MeshAdapt)
Info    : Meshing surface 16 (Ruled surface, MeshAdapt)
Info    : Meshing surface 20 (Ruled surface, MeshAdapt)
Info    : Meshing surface 22 (Ruled surface, MeshAdapt)
Info    : Meshing surface 24 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.292 s)
Info    : 1865 vertices 3926 elements
```

Figure A.13: The number of triangle elements in the mesh was increased by setting the ***Element size factor*** to 0.3.

Creating a Sphere in Gmsh:

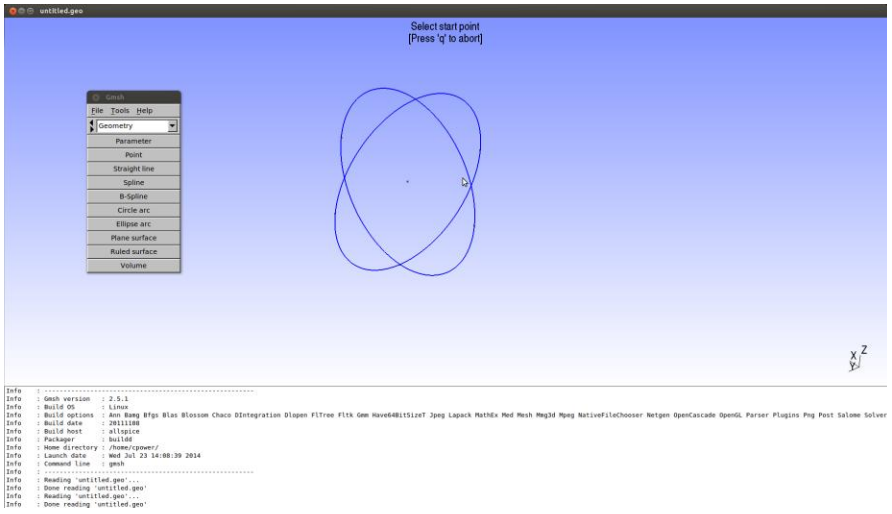

Contextual Geometry Definitions
2:15 PM Colin Power
untitled.geo
Move mouse and/or enter coordinates
[Press 'Shift' to hold position, 'e' to add point or 'q' to abort]
Gmsh
File Tools Help
Geometry
Parameter
Point
Straight line
Spline
B-Spline
Circle arc
Ellipse arc
Plane surface
Ruled surface
Volume
Contextual Geometry Definitions
Parameter Point Translation Rotation Scale Symmetry
Add
Z
X Y

0 X coordinate
0 Y coordinate
0 Z coordinate
1.0 Prescribed mesh element size at point
0.1 0.1 0.1 Snapping grid spacing

```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Wed Jul 23 14:08:39 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

Figure A.14: Seven points created to build a sphere.

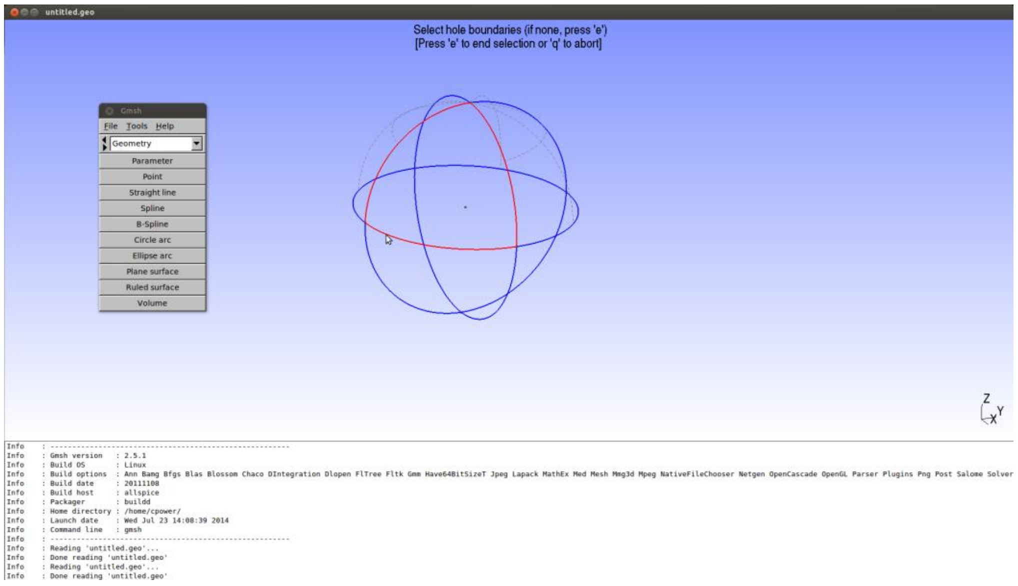
First, the centre point is added and then six points with the same radius displacement are added on each axis separately in the positive and negative directions. Using ***Geometry->Add->Circle arc***, the outer points are connected to each other using the middle point as the centre of the arc.



```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bamg Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Wed Jul 23 14:08:39 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

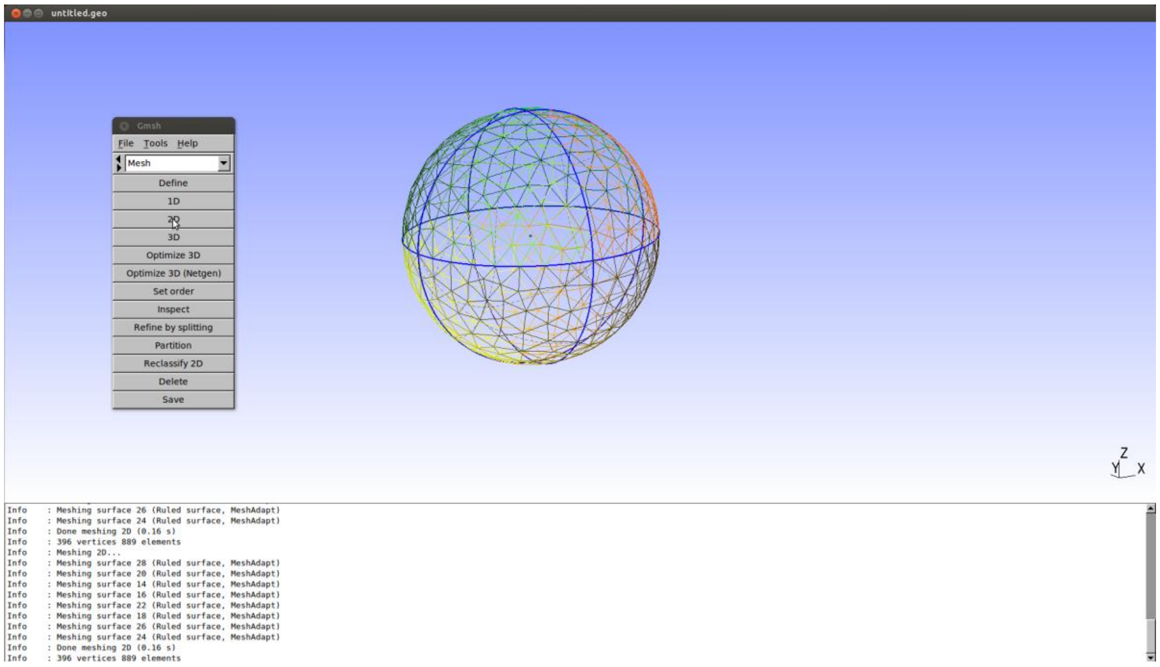
Figure A.14: Eight circle arcs are formed to create two circles using the centre point.

Afterwards, using the ***ruled surface*** tool, the eight surface areas closed off by the drawn lines are ruled to create the sphere surface.



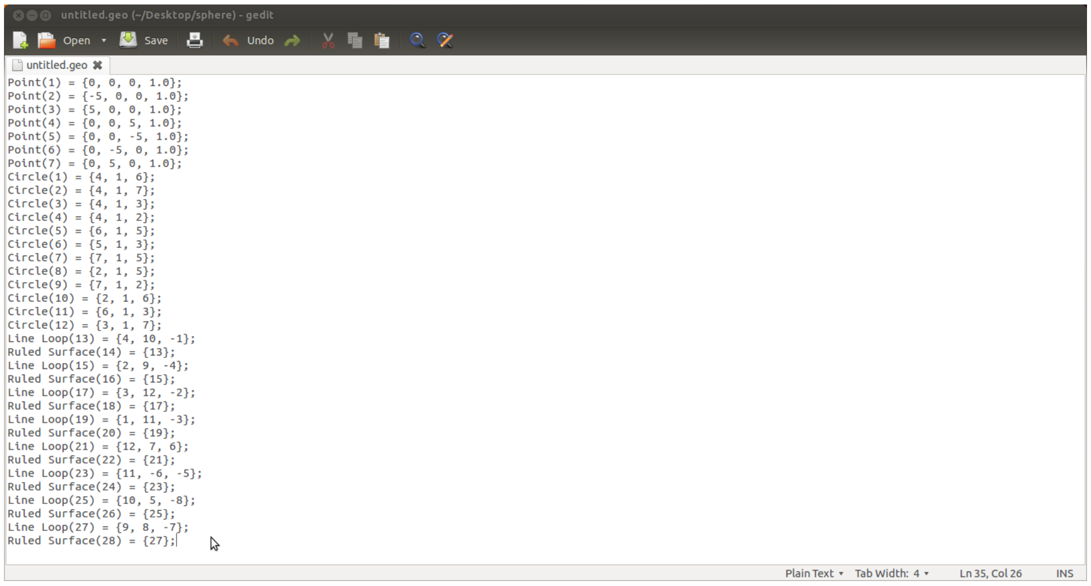
```
Info    : -------------------------------------------------------
Info    : Gmsh version   : 2.5.1
Info    : Build OS       : Linux
Info    : Build options  : Ann Bang Bfgs Blas Blossom Chaco DIntegration Dlopen FlTree Fltk Gmm Have64BitSizeT Jpeg Lapack MathEx Med Mesh Mmg3d Mpeg NativeFileChooser Netgen OpenCascade OpenGL Parser Plugins Png Post Salome Solver
Info    : Build date     : 20111108
Info    : Build host     : allspice
Info    : Packager       : buildd
Info    : Home directory : /home/cpower/
Info    : Launch date    : Wed Jul 23 14:08:39 2014
Info    : Command line   : gmsh
Info    : -------------------------------------------------------
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
Info    : Reading 'untitled.geo'...
Info    : Done reading 'untitled.geo'
```

Figure A.15: Three ruled surfaces have been formed from the circle arcs.



```
Info    : Meshing surface 26 (Ruled surface, MeshAdapt)
Info    : Meshing surface 24 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.16 s)
Info    : 396 vertices 889 elements
Info    : Meshing 2D...
Info    : Meshing surface 28 (Ruled surface, MeshAdapt)
Info    : Meshing surface 20 (Ruled surface, MeshAdapt)
Info    : Meshing surface 14 (Ruled surface, MeshAdapt)
Info    : Meshing surface 16 (Ruled surface, MeshAdapt)
Info    : Meshing surface 22 (Ruled surface, MeshAdapt)
Info    : Meshing surface 18 (Ruled surface, MeshAdapt)
Info    : Meshing surface 26 (Ruled surface, MeshAdapt)
Info    : Meshing surface 24 (Ruled surface, MeshAdapt)
Info    : Done meshing 2D (0.16 s)
Info    : 396 vertices 889 elements
```

Figure A.16: After drawing all ruled surfaces, the geometry is then 2D meshed and a hollow sphere is formed.



```
Point(1) = {0, 0, 0, 1.0};
Point(2) = {-5, 0, 0, 1.0};
Point(3) = {5, 0, 0, 1.0};
Point(4) = {0, 0, 5, 1.0};
Point(5) = {0, 0, -5, 1.0};
Point(6) = {0, -5, 0, 1.0};
Point(7) = {0, 5, 0, 1.0};
Circle(1) = {4, 1, 6};
Circle(2) = {4, 1, 7};
Circle(3) = {4, 1, 3};
Circle(4) = {4, 1, 2};
Circle(5) = {6, 1, 5};
Circle(6) = {5, 1, 3};
Circle(7) = {7, 1, 5};
Circle(8) = {2, 1, 5};
Circle(9) = {7, 1, 2};
Circle(10) = {2, 1, 6};
Circle(11) = {6, 1, 3};
Circle(12) = {3, 1, 7};
Line Loop(13) = {4, 10, -1};
Ruled Surface(14) = {13};
Line Loop(15) = {2, 9, -4};
Ruled Surface(16) = {15};
Line Loop(17) = {3, 12, -2};
Ruled Surface(18) = {17};
Line Loop(19) = {1, 11, -3};
Ruled Surface(20) = {19};
Line Loop(21) = {12, 7, 6};
Ruled Surface(22) = {21};
Line Loop(23) = {11, -6, -5};
Ruled Surface(24) = {23};
Line Loop(25) = {10, 5, -8};
Ruled Surface(26) = {25};
Line Loop(27) = {9, 8, -7};
Ruled Surface(28) = {27};
```

Figure A.17: The contents of the sphere.geo file.

Appendix B: Hardware Performance Test Results

Yade Performance Results for Specific Number of Active CPU Cores								
			Time (s)			Score		
Cores	**rsarracino Ubuntu 12.04**	**rsarracino Ubuntu 14.04**	**rsarracino Ubuntu 14.04 (no hyper threading)**	**CARDLN1 Ubuntu 14.04**	**rsarracino Ubuntu 12.04**	**rsarracino Ubuntu 14.04**	**rsarracino Ubuntu 14.04 (no hyper threading)**	**CARDLN1 Ubuntu 14.04**
1	910	867	858	774	8830	9335	9473	10313
2	928	537	526	474	8790	15762	16058	17325
3	935	472	448	400	8724	17896	18909	20255
4	960	458	423	367	8482	18537	20433	22761
5	954	425	-	343	8590	20236	-	24230
6	962	420	-	330	8474	20571	-	25338
7	970	411	-	334	8423	21080	-	25417
8*	972	430	-	331	8382	20279	-	25519

**** When running for eight cores the test produced the following warning: only 7 thread(s) used. The number of bodies is probably too small for allowing more thread***