FACULTY OF SCIENCE

UNIVERSITY COLLEGE CORK

Numerical Model of a Granular Media

Development, Investigation of Granular Aspects & Extension to a Periglacial Model

A minor thesis submitted in part fulfillment of the requirements for the Master of Science in Applied Science (Mathematical Modeling & Scientific Computing)

Name : Bryan Foley

I.D : 109221484

Supervisor : Dr. Kieran Mulchrone

Head of Department : Prof Alexei Pokrovskii

Department : Department of Applied Mathematics

Date : October 2010

Acknowledgements

Its been a roller coaster of a year with many highs and lows and I would have had difficulty enjoying the ride only for the support and guidance of my family, friends and the faculty of the U.C.C Applied Mathematics Department. I wish to thank my supervisor, Dr. Kieran Mulchrone, for his guidance and support at our weekly meetings over the past 17 weeks and also for his wonderful job as course director during the academic year. I also must mention the advice of Dr. Sabin Tabirca and John Horan. Although we did not have enough time to create a version of the code which can be run on parallel processors, important groundwork was completed and appears on the website - thanks to their advice.

To Mum, Dad, T, J, Mags & Wheelo

Abstract

A granular media is a large conglomeration of discrete macroscopic particles. Using an efficient force summation technique, we numerically model a $2D$ granular aggregate in different geometries .Energy is added to the system as gravitational potential energy or vertical vibration. The model is also extended to create a basic representation of a Periglacial process known as frost heave. Results were compared to published experimental work to validate the models findings. The benefits/drawbacks of the Molecular Dynamics approach to simulating granular media are also discussed.

Videos of the simulations, source code and updated work is available at:

https://sites.google.com/site/compgransim/

Contents

List of Figures

Chapter 1

Introduction

1.1 Granular Media

Have you ever walked along a beach, cupped your hands then scooped up some dry sand? Undoubtedly, some of the sand would begin to seep its way through the narrow gaps between your fingers. Eventually you will uncup your hands and watch the sand flow back to the ground like a liquid or if it is a breezy day you may watch as it billows along the shore briefly. This is a common and relaxing experience, but did it ever occur to you that the sand you just let flow through your fingers and blow away is now supporting your entire weight like a solid would? How can the same substance act like three differing states of matter? Why does the sand at your feet not flow away and cause you to sink downwards uncontrollably when you apply pressure to it? At most, you may only be able to sink a few centimeters below the top level before coming to a complete halt irregardless of your weight.

Sand, like many other omnipresent materials, falls into a class known as Granular Materials. Granular materials differ from the established states of matter but can mimic the behaviour of each of them under appropriate conditions. We rely heavily on these materials for our basic needs: most of our staple foodstuffs (rice, corn etc.) are processed from grains, the construction industry works exclusively with raw materials of a granular nature and the ever expanding and innovative field of pharmaceuticals requires fine powders to be mixed and transported to produce the desired end products. Despite this strong dependence on granular materials in industry, inefficient processing and transport techniques lead to high losses - estimated to be up to 40% [Jae96a] in some cases.

Granular media are composed of a large number of *macroscopic* particles. If the particles are dry or *non-cohesive*, then the only forces experienced by individual particles are repulsive forces. If on the other hand the particles immersed in an

interstitial fluid, say water, they are termed *cohesive* with attractive and repulsive forces occurring between particles. Due to the fact that granular materials cannot be easily classified as being entirely in one state, attempts to describe these materials via kinetic gas theory or continuum models fail to grasp the overall properties of the system. To understand why, we need to describe the physics of granular materials.

1.2 The Physics of Granular Media

Our current knowledge of Granular media is confined to two extremes - the compact solid-like state as described by soil mechanics and the high shear rate flow as described by kinetic theory and our understanding of what happens between these two extremes is limited. The difficulty with Granular media is that they display a variety of behaviours that are in many ways different from those of other substances. Granular materials cannot be easily classified as solids, liquids or gases as they do not strictly act as one state of matter. In assuming that granular media are akin to a dense gas we expect that the thermal energy of the particles $k_\beta T$ where k_β is Boltzmann's Constant and T is the temperature in Kelvin, will be enough to describe the motion of the grains. This is absolutely false. The value of $k_\beta T$ is dwarfed by the gravitational potential energy of the grain $m\vec{\mathbf{g}}d$ where m is the mass, $\vec{\mathbf{g}}$ is the acceleration due to gravity and d is the diameter of the particle. $m\vec{\mathbf{g}}d$ is 10^{12} times $k\beta T$, so it is reasonable to assume that $k_\beta T \sim 0$. Thermal energy is what allows a system to explore its phase space and move from one phase to another, for example the transition of ice to liquid water is due to thermal energy. If $k_\beta T \sim 0$ then entropy effects are smothered by $m\vec{\mathbf{g}}d$, or dynamic effects. A consequence of $k_\beta T \sim 0$ is that granular particles will stay in metastable configurations until dynamic energy is introduced into the system either by shearing, compacting, vibrating or pouring. Each dynamic process will cause the granular media to behave differently. A lack of entropy effects means that the system will not average out over different configurations, metastable configurations with abrupt changes or discontinuities in mechanical properties will form in the system whilst it is unperturbed. Consequently continuum models cannot be applied as they require systems with gradual changes in mechanical properties.

The network of contact forces in a granular media exhibits the inhomogeneities in mechanical properties. Forces will not be distributed evenly like in a crystalline system, but are distributed inhomogeneously. These contact networks are known as *Force Chains*. An example is shown below. The brighter the line, the stronger the

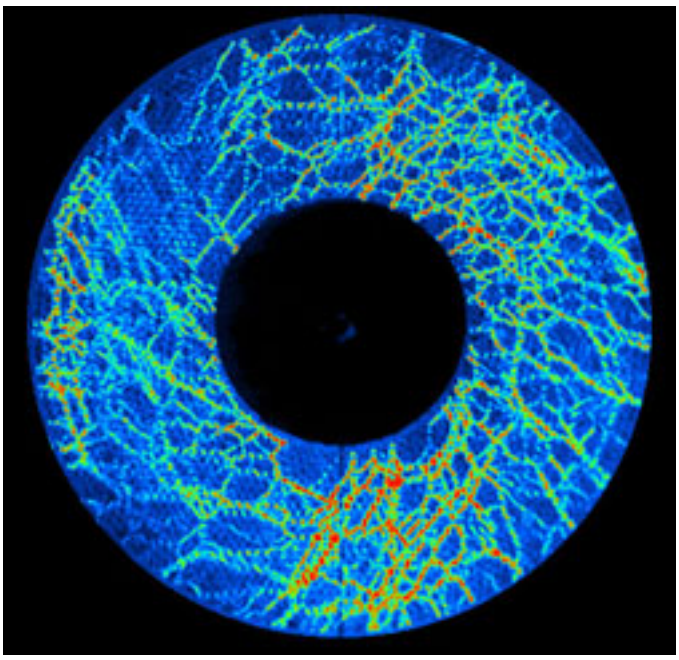


Figure 1.1: Force Chains

contact force. The force chains are stress bearing networks which can be used to explain some everyday phenomena of granular materials, such as sand in an hour glass. In an hour glass sand flows like a liquid at the center and acts like a solid near the edges of the glass. Force chains span the entire width of the hour glass and distribute the force from gravity to the glass walls. A convenient analogy is to think of the spanning force chain as the arch of a bridge, and imagine the space under the archway is occupied with particles that are experiencing weaker contact forces. These particles are relatively free to move around or flow downwards due to the supportive networks of successive force chains in the media. It is fair to say that forces in a granular media are bi-modally distributed, with some particles experiencing huge contact forces and others experiencing much weaker contact forces. A height independence is observed in an hour glass. Particles that experience weak contact forces will flow downwards at a constant velocity and is the basis of the hour glass being such a staple timepiece during centuries past. This height independence contrasts sharply with an incompressible liquid for which the pressure increases linearly with depth. Arching does not occur in a liquid or gas as the continuous thermal

motion of the particles prevents static configurations from forming. Since a granular medium is a disordered discrete packing, pressure fluctuations are significant. This can lead to the catastrophic failure of Silos.

Figure 1.2: Catastrophic failure of a Silo due to inhomogeneous force distribution.

Another interesting behaviour that is observed is size segregation. When a box filled with particles of different sizes is shaken vertically, the larger particles end up on the top surface. This is called the 'Brazil nut effect' as, after transport, Brazil nuts will always be found on top of the smaller nuts in Brazilian fruit trucks. This effect is linked to convection. When rough particles in a container with rough walls are vibrated, particles move up through the middle of the container and down at the walls of the container. It is observed that the larger particles can move upwards through the middle of the container but cannot move downwards at the walls since the zone in which downward-directed flow occurs is too narrow, [Jae96b].

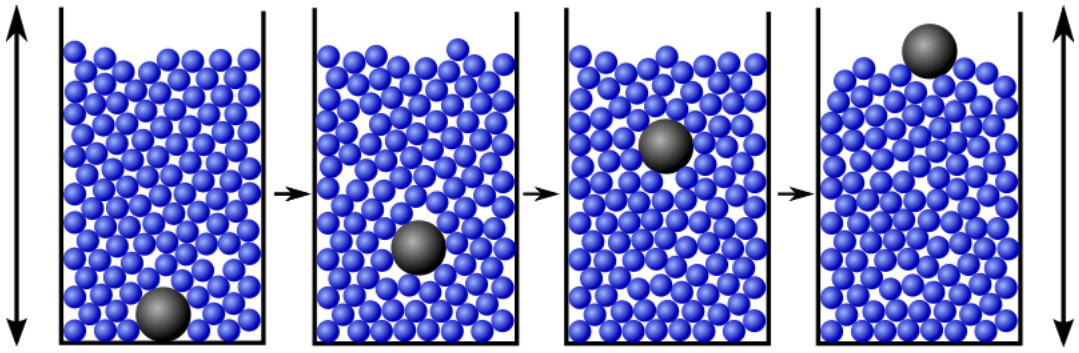
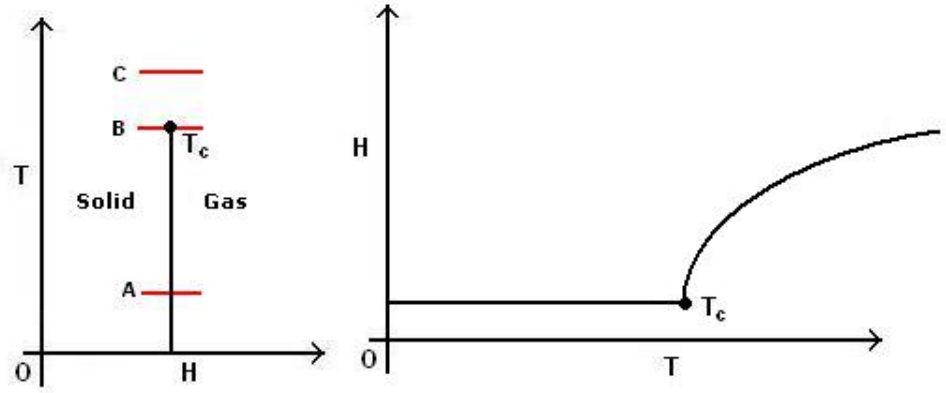


Figure 1.3: A simulation of Granular size segregation



(a) 1^{st} Order Phase transition (b) 2^{nd} Order Phase transition

Figure 1.4: Examples of 1^{st}&2^{nd} Order Phase Transitions

1.2.1 Criticality and Critical Behaviour in Granular Media

It is possible to draw loose parallels between the thermodynamic behaviour [Aha99] observed in fluids and gases, and the behaviour of granular systems. The concept of criticality will be introduced with a generic example from equilibrium thermodynamics.

A first order phase transition occurs at temperature A, as in Fig.1.4a, where the transition between solid and gas is distinct. At temperature C, there is no longer any distinction between the solid and the gas, and so there is no phase transition. The critical point is located at temperature B, which marks a second order phase transition between the two media. Fig. 1.4b contains a general example of a second order phase transition for free energy against temperature. The transition is continuous, but contains a discontinuity in the first derivative.

The critical point marks the point at which two individual phases of a material cease to exist. The most familiar example of a critical point in phase space is the critical point at which water vapour becomes indistinguishable from liquid water, which occurs at $\sim 647K$ and $\sim 22.064MPa$. The critical point is marked by critical opalescence, [Mic], a phenomenon where the water appears milky due to large fluctuations at all wavelengths. While in the solid phase (i.e. at low free energy), noise has a finite propagation length in the system. The state of a particular element of the material will influence the state of its neighbours. To a certain degree the states of neighbouring elements are predictable if the state of a single element is known. The temporal power spectrum of such a system is greater than one. The correlation length between elements in the gaseous phase is zero. The state of one element will have no effect on the states of its neighbours. The best estimate, for the

velocity of any molecule is zero, even if the velocities of its neighbours are known. The temporal power spectrum of such a system is less than one, which corresponds to white (uncorrelated) noise. At the critical point, the correlation length of the system is infinite. The critical point of a system is characterised by scale invariant size statistics. A system which is at a critical point will evolve over time to a strange attractor in its phase space. Aharanov and Sparks have identified a second order phase transition and a critical point when measuring shear modulus against packing fraction, Fig. 1.5, static granular packings, [Aha99]. The phase transition occurred at $\nu \sim 0.83$, which is also marked the transition from a state where grains are not in contact, to a state where grains were in contact, as in Fig. 1.6.

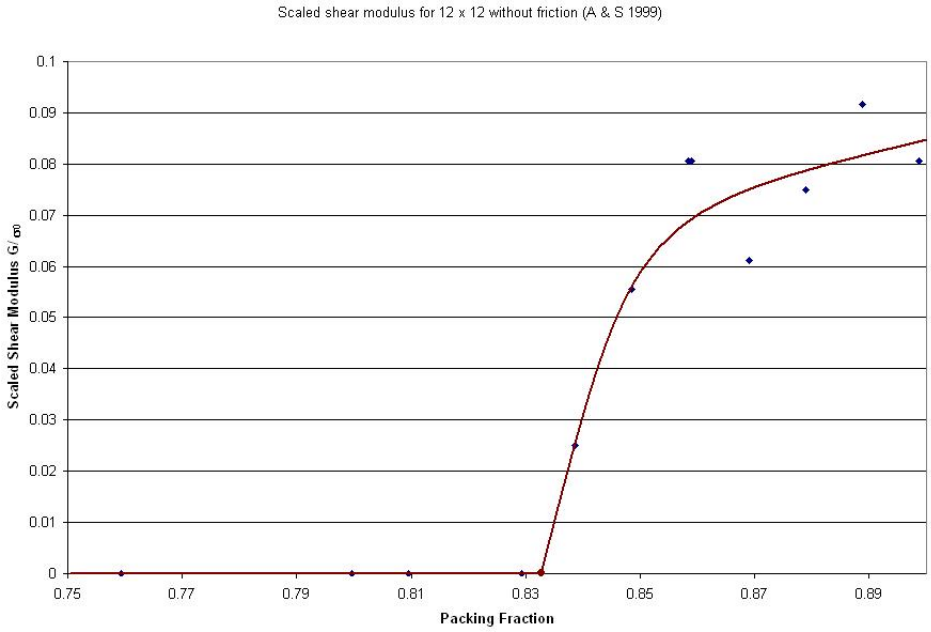


Figure 1.5: Scaled Shear Modulus against Packing Fraction for a frictionless granular system.

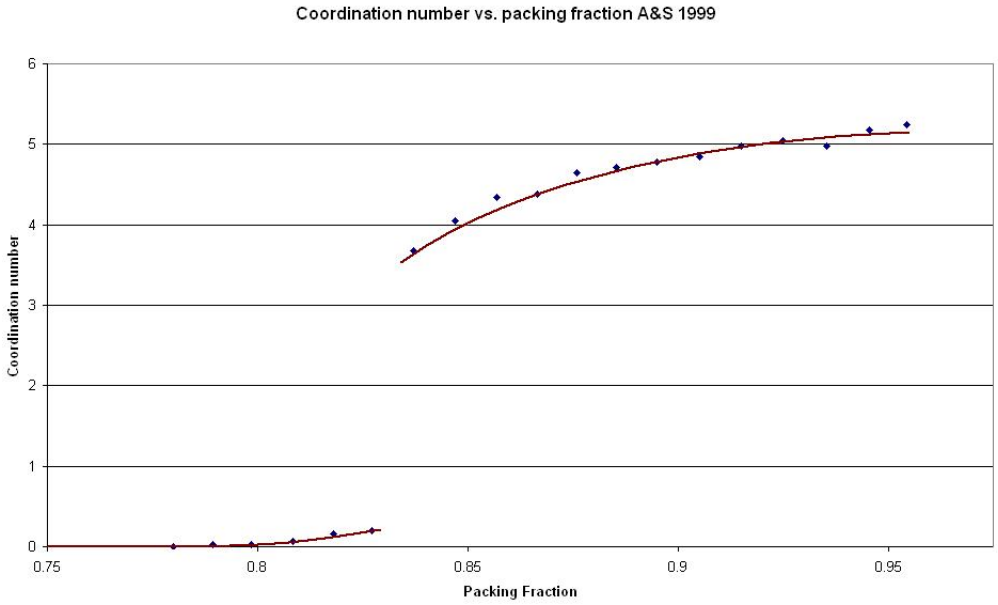


Figure 1.6: Coordination Number against Packing Fraction for a frictionless granular system.

1.3 Why numerically model Granular Media?

Although there is a long history of interest in these materials, with research dating back to Coulomb [Jae96a], as yet, there is no comprehensive theory of Granular Media from first principles [PÖ5a]. Given a set of initial conditions such as the number of grains, the distribution of the grain radii and the geometry of the system, we cannot accurately predict the dynamics of the system or monitor in detail the properties of each and every grain during a physical experiment. The need for numerical simulations of such Granular systems is two-fold. The first is to make predictions about such a system of grains with given initial conditions and use the resultant data to better engineer machinery to handle Granular aggregates. The second is to add to the expanding knowledge bank of Granular studies.

Chapter 2

The Model And Complimentary Software

2.1 Molecular Dynamics

Molecular Dynamics, **MD**, was first introduced by Alder & Wainright [Ald57; Ald59; Ald60] in the late 1950's to study molecular gases and simple liquids. Over the decades it has been improved and adapted to a plethora of problems, notably the study of Granular Dynamics [Cun79; Ris94; PÖ5b]. Below is a description of why **MD** was chosen to simulate our Granular System over other tools such as Event Driven Molecular Dynamics, Direct Simulation Monte Carlo Methods, and Cellular Automata Models but *not* a description of these alternatives. The reader is referred to [PÖ5b] for a more in-depth treatment of these methods.

Granular media consist of large and small conglomerates of grains whose radii range from micrometers to meters [Jae96a]. These grains do *not* interact via long range forces such as Electrostatic Potential or Gravity, rather they interact through *Mechanical Contact* of surfaces. The dynamics of a Granular Media is governed by Newtons Equation of Motion for the center of mass coordinates and the angular orientation of the grains which constitute the conglomerate:

$$\frac{\delta^2 \vec{\mathbf{r}}_i}{\delta t^2} = \frac{1}{m_i}\vec{\mathbf{F}}_i(\vec{\mathbf{r}}_j, \vec{\mathbf{v}}_j, \vec{\varphi}_j, \vec{\omega}_j) \tag{2.1}$$

$$\frac{\delta^2 \vec{\varphi}_i}{\delta t^2} = \frac{1}{J_i}\vec{\mathbf{M}}_i(\vec{\mathbf{r}}_j, \vec{\mathbf{v}}_j, \vec{\varphi}_j, \vec{\omega}_j) \tag{2.2}$$

where,

$i = (1, ..., N)$,
$j = (1, ..., N)$,
N = The Number of grains present,
$\vec{\mathbf{r}}_i$ = The position of grain i,
m_i = The mass of grain i,
$\vec{\mathbf{F}}_i$ = The Force acting on grain i,
$\vec{\mathbf{M}}_i$ = The Torque acting in grain i,
J_i = The Moment of Inertia of grain i,
$\vec{\mathbf{v}}_j$ = The velocity of grain j,
$\vec{\varphi}_j$ = The angular orientation of grain j,
$\vec{\omega}_j$ = The angular velocity of grain j,
$\frac{\delta^2}{\delta t^2}$ = The second derivative with respect to time.

Equations such as (2.1) & (2.2) are difficult to solve analytically. The approximate numerical solution to these equations is acheived using **MD**. The calculation of the forces and torques on each grain is central to the **MD** simulation. These are calculated by summing the pairwise interactions of grain i with all other grains in the simulation:

$$\vec{\mathbf{F}}_i = \sum_{j=1, j\neq i}^{N} \vec{\mathbf{F}}_{ij} \tag{2.3}$$

$$\vec{\mathbf{M}}_i = \sum_{j=1, j\neq i}^{N} \vec{\mathbf{M}}_{ij} \tag{2.4}$$

In this work, vectors are denoted as boldface leters and characters with arrows on top, $\vec{\mathbf{F}}_{ij}^{n}$, and scalars as regular letters and characters, m_i.

2.1.1 Contact of Particles

The **MD** method employs an 'a posteriori' collision detection algorithm, i.e. collisions between particles are detected after the initial contact. Particles are treated as inelastic discs with translational and rotational degrees of freedom. The particles i and j interact when the distance between their centers $\vec{R}_{ij}$ is less than the sum of their radii, $R_i + R_j$, so that there is a small overlap, ξ_{ij}, of the grains. This overlap is given by:

$$\xi_{ij} = R_i + R_j - \mid \vec{\mathbf{r}_i} - \vec{\mathbf{r}_j} \mid > 0 \tag{2.5}$$

where,

R_i = The Radius of particle i,
$\mid \vec{\mathbf{r}_i} - \vec{\mathbf{r}_j} \mid$ = The Position Vector from the center of particle i to the center of j.

The particles are *non-cohesive* so that when the overlap is zero, the contact force on each particle is zero. The overlap is physically interpreted as the elastic deformation of the colliding particles. The particles are therefore said to be *soft*.
During an interaction, i.e. when the overlap is greater than zero, the contact force on particle i due to its interaction with particle j, $\vec{\mathbf{F}}_{ij}$, has both **Normal**, $\vec{\mathbf{F}}^n_{ij}$, and **Tangential**, $\vec{\mathbf{F}}^t_{ij}$, components and can be summarised as:

$$\vec{\mathbf{F}}_{ij} = \begin{cases} \vec{\mathbf{F}}^n_{ij} + \vec{\mathbf{F}}^t_{ij} & \text{if } \xi_{ij} > 0 \\ 0 & \text{if } \xi_{ij} \leq 0 \end{cases} \tag{2.6}$$

This simulation is two dimensional, as such, the Normal and Tangential components can be written as:

$$\vec{\mathbf{F}}^n_{ij} = F^n_{ij}\vec{\mathbf{e}}^n_{ij} \tag{2.7}$$

$$\vec{\mathbf{F}}^t_{ij} = F^t_{ij}\vec{\mathbf{e}}^t_{ij} \tag{2.8}$$

where,

$\vec{\mathbf{e}}^n_{ij}$ = The Unit Vector in the Normal Direction,
$\vec{\mathbf{e}}^t_{ij}$ = The Unit Vector in the Tangential Direction.

2.2 The Force Laws

Two force laws are utilised in the **MD** simulation, which the user can choose between at the data input stage of a run (see section 2.8.1). Each force law treats the particles as two-dimensional disks and both are functions of the relative positions and velocities of a pairwise interaction.

2.2.1 The Normal Force

Viscoelastic Dampening Force, Force Law 0

The Viscoelastic Dampening Force, [PÖ5b], gives exceptionally better agreement with experimental results compared to force law 1. The normal force of a pairwise interaction between Viscoelastic particles i & j is given by:

$$\vec{\mathbf{F}}_{ij}^{n} = [\sqrt{\xi_{ij}} * Y_{ij}\sqrt{R_{ij}} * (\xi_{ij} + A_{ij}\dot{\xi}_{ij})]\vec{\mathbf{e}}_{ij}^{n} \tag{2.9}$$

where,

Y_{ij} = Young's Modulus,
R_{ij} = The effective Radius of a pairwise interaction, $\frac{R_i * R_j}{R_i + R_j}$
A_{ij} = Material Viscosity,
$\dot{\xi}_{ij}$ = Rate of change of the Overlap.

Given a set of material parameters, any granular material may be accurately simulated using this force law 0. Also, combinations of different materials in the same conglomerate may also be simulated by adapting equation (2.9) to the form given in [PÖ5c].

Linear Dashpot Force, Force Law 1

Force Law 1 [Aha99] is a more simple calculation of the normal force of a pairwise interaction. The normal force was modeled as a damped soft spring, with a displacement from its rest position equal to the overlap between the two grains in contact. The normal force for a pairwise interaction between two particles i & j is given by:

$$\vec{\mathbf{F}}_{ij}^{n} = [k_n\xi_{ij} - \gamma_{ij}m_{ij}\vec{\mathbf{v}}_{ij}]\vec{\mathbf{e}}_{ij}^{n} \tag{2.10}$$

where,

k_n = The Normal dissipation factor,

m_{ij} = The Harmonic mass of particles i & j, $\frac{m_i * m_j}{m_i + m_j}$
$\vec{\mathbf{v}}_{ij}$ = The relative Velocity between particles i & j.

2.2.2 The Tangential Force

Friction was incorporated into the model by calculating the tangential component of the contact force between particles. Instead of acsribing surface defects to each particle, the calculation was simplified by using the Relative Tangential Velocity, $\vec{\mathbf{v}}_{ij}^{t}$, of the particles at contact which was calculated by:

$$\vec{\mathbf{v}}_{ij}^{t} = (\vec{\mathbf{v}}_j - \vec{\mathbf{v}}_j) \cdot \vec{\mathbf{e}}_{ij}^{t} + R_i\vec{\omega}_i + R_j\vec{\omega}_j \tag{2.11}$$

and then by calculating the tangential contact force from:

$$\vec{\mathbf{F}}_{ij}^{t} = -sign(\vec{\mathbf{v}}_{ij}^{t}) \cdot min(\gamma^t \mid \vec{\mathbf{v}}_{ij}^{t} \mid, \mu \mid \vec{\mathbf{F}}_{ij}^{n} \mid) \tag{2.12}$$

where,

γ^t = The Tangential Dissipation factor,
μ = The Coulomb Friction Parameter.

The tangential force is limited by Coulomb's friction law $\vec{\mathbf{F}}^t \leq \mu \mid \vec{\mathbf{F}}^n \mid$. For a large relative velocity or a small normal force $\vec{\mathbf{F}}^t = \mu \mid \vec{\mathbf{F}}^n \mid$ is chosen. For simulations of static granular media, the Tangential Force model by Cundall & Strack [Cun79] is more appropriate, however since simulations run here focus more on the properties of *Dynamic* Granular media, we did not feel it was necessary to incorporate *Static* friction.

2.3 Gear's Integration Scheme

The Gears's Integration Scheme described in [PÖ5b] was chosen to numerically integrate Newton's equations of motion. Although many other integration schemes exist, many of which may be found in work such as [Pre07], the Gear's Algorithm is particularly suited to the application of Granular dynamics. The main advantage of the algorithm is that at each timestep, Δt, only *one evaluation of the contact force* is required. Other algorithms such as the Velocity Verlet algorithm, [Gio96], require two evaluations per timestep. The Gear's algorithm is also incredibly stable with an error of just $\sim (\Delta t)^5$ per timestep.

The particular flavour of the scheme we chose is the 5^{th} order algorithm. This implies that all of the time derivatives of the i^{th} particle's centre of mass coordinates and angular orientation are required:

$$\frac{\delta \vec{\mathbf{r}}_i}{\delta t^n}, \frac{\delta \vec{\varphi}_i}{\delta t^n} \tag{2.13}$$

where,

$n = 0, 1, 2, 3, 4.$

This is achieved by using a Taylor expansion of the current values. The Gear algorithm is comprised of two parts and is a form of a Predictor-Corrector scheme.

2.3.1 Predictor

The predictor computes the position, velocity, acceleration and remaining time derivatives of the i^{th} particle at time $t + \Delta t$:

$$\vec{r}_i^{pre}(t + \Delta t) = \vec{r}_i(t) + \Delta t \dot{\vec{r}}_i(t) + \frac{1}{2}\Delta t^2 \ddot{\vec{r}}_i(t) + \frac{1}{6}\Delta t^3 \dddot{\vec{\mathbf{r}}}_i(t) + \dots \tag{2.14a}$$

$$\dot{\vec{r}}_i^{pre}(t + \Delta t) = \dot{\vec{r}}_i(t) + \Delta t\, \ddot{\vec{r}}_i(t) + \frac{1}{2}\Delta t^2 \dddot{\vec{\mathbf{r}}}_i(t) + \dots \tag{2.14b}$$

$$\ddot{\vec{r}}_i^{pre}(t + \Delta t) = \ddot{\vec{r}}_i(t) + \Delta t \dddot{\vec{\mathbf{r}}}_i(t) + \dots \tag{2.14c}$$

$$\vdots$$

Up to the 4^{th} order. The predicted values of position and velocity are then used to calculate the forces and torques acting on the i^{th} particle using Equation (2.1) &

Equation (2.2).

2.3.2 Corrector

From the forces and torques the corrected linear and angular accelerations, $\ddot{\vec{r}}_i^{corr}$ & $\ddot{\vec{\varphi}}_i^{corr}$ are computed.
If $\ddot{\vec{r}}_i^{corr}$ & $\ddot{\vec{\varphi}}_i^{corr}$ differ from the predicted accelerations, then it is necessary to perform a correction to the predicted terms.

$\Delta\ddot{\vec{r}}_i = \ddot{\vec{r}}_i^{corr} - \ddot{\vec{r}}_i^{pre}$ is a measure of the deviation of the predicted acceleration. The corrector adds a predetermined weighting, c_n, proportional to the deviation to each predicted term, to arrive at the final corrected value:

$$\begin{pmatrix} \vec{r}_i^{corr}(t+\Delta t) \\ \dot{\vec{r}}_i^{corr}(t+\Delta t) \\ \ddot{\vec{r}}_i^{corr}(t+\Delta t) \\ \dddot{\vec{\mathbf{r}}}_i^{corr}(t+\Delta t) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vec{r}_i^{corr}(t+\Delta t) \\ \dot{\vec{r}}_i^{pre}(t+\Delta t) \\ \ddot{\vec{r}}_i^{pre}(t+\Delta t) \\ \dddot{\vec{\mathbf{r}}}_i^{pre}(t+\Delta t) \\ \vdots \end{pmatrix} + \begin{pmatrix} c_0 \\ c_1 \frac{1}{\Delta t} \\ c_2 \frac{2}{(\Delta t)^2} \\ c_3 \frac{6}{(\Delta t)^3} \\ \vdots \end{pmatrix} \frac{\Delta t^2}{2} \Delta\ddot{\vec{r}}_i \tag{2.4}$$

where,

$$c_0 = \tfrac{19}{90}, c_1 = \tfrac{3}{4}, c_2 = 1, c_3 = \tfrac{1}{2}, c_4 = \tfrac{1}{12}$$

These terms were then passed back to the predictor for the next timestep.

2.4 Flowchart

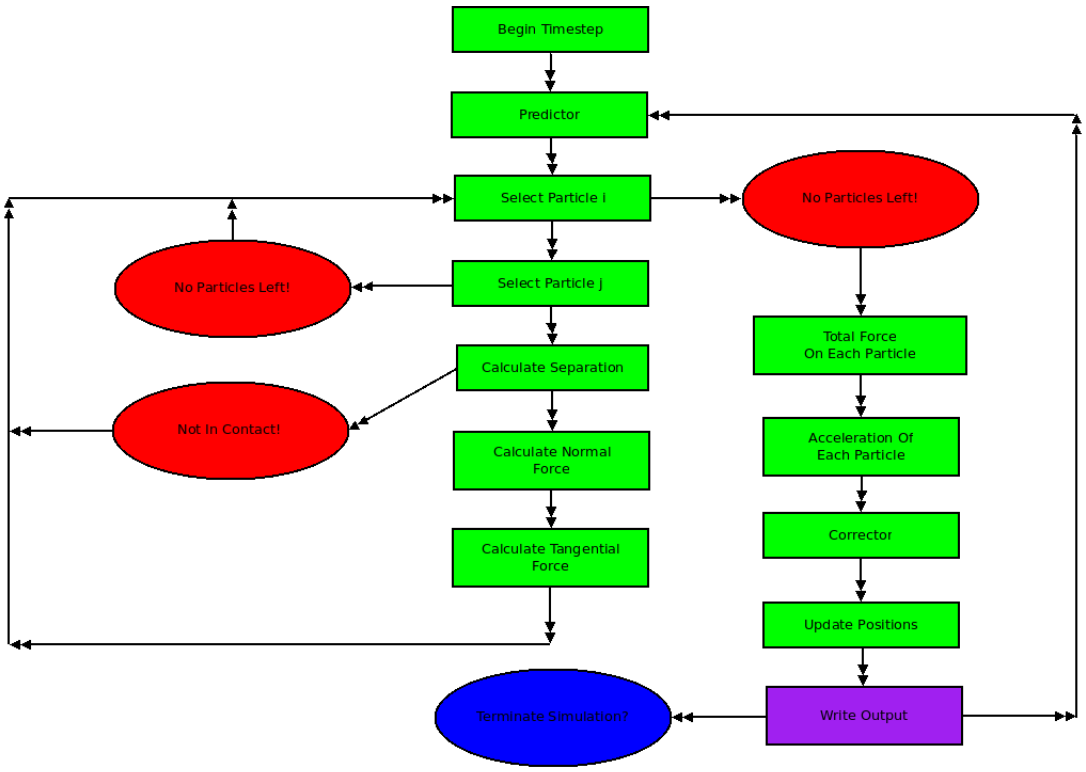


Figure 2.1: Flow Diagram for a single timestep.

At the beginning of a time step, a Taylor series prediction of the position, velocity etc of each particle was made, based on the present positions and higher order derivatives. Each particle was then examined for contact with other particles. To remove unecessary force calculations, only the immediate neighbours of a particle were examined for contact. This was achieved by using a *Verlet List*(see section 2.7.1) of neighbouring particles. If the particles were not in contact, they were neglected, and the next pair was examined. If in contact, the x, y & ϕ components of the normal and tangential forces were calculated as outlined above. The forces were converted to accelerations and the components were added algebraically to a subtotalled acceleration which was stored for each particle. When all particles had been examined for contacts, the corrector was used to update the predicted values of position and higher derivatives. Measurements of energy, momentum, contact number, average velocity and packing fraction were made and the data was written to an output file at a predetermined timestep.

2.5 Disk Radii & Disk Masses

2.5.1 Disk Radii

Polydispersivity of particle radii was introduced to discourage crystallisation effects. The values for the upper and lower radii limits were selected from 9 **Mean Particle Diameters**, **MPD's**, based on the Wentworth Scale of Soil and Particle sizes, [Gra09]. The **MPD's** are arranged as such:

Table 2.1: Wentworth Scale of Soil & Particle Sizes

Soil Type	**MPD** (mm)	Classification
Pebbles - Boulders	> 4.00	Gravels
Granules	4.00	Gravels
Very Coarse Sand	2.00	Sandy
Coarse Sand	1.00	Sandy
Medium Sand	0.5	Sandy
Fine Sand	0.25	Sandy
Very Fine Sand	0.125	Sandy
Silt	0.0625	Fines
Clay	< 0.0039	Fines

Once a particular soil type is selected, the radii of the particles are assigned a random value between the upper and lower radii limits and are calculated by the equation:

$$R_i = R_{min} + z(R_{max} - R_{min}) \tag{2.5}$$

where,

z = A random number between $[0, 1]$,
R_{max} = The Upper radii limit,
R_{min} = The Lower radii limit.

By editing the $*.in$ file, the user can generate a granular system of any desired geometry and populate the system with particles whose dimensions mimic those of real World soils and particles.[1]

[1] Caution must be exercised when generating a particle system though, as the simulation software does not perform 'a priori' checks on particle positions until the construction of the Verlet List 2.7.1. Any errors such as unrealistic overlaps or impossible configurations will not become obvious until run time. The software developed is a powerful aid in simulating the dynamics but is still far from completion.

2.5.2 Disk Masses

Once the radii are assigned, the assignation of the particle masses is trivial, particles with a diameter equal to the **MPD** have a mass of 1. All other particles are assigned their mass from the equation:

$$m = \frac{R_i}{\bar{R}} \tag{2.6}$$

where,

$\bar{R} = \frac{\mathbf{MPD}}{2}$.

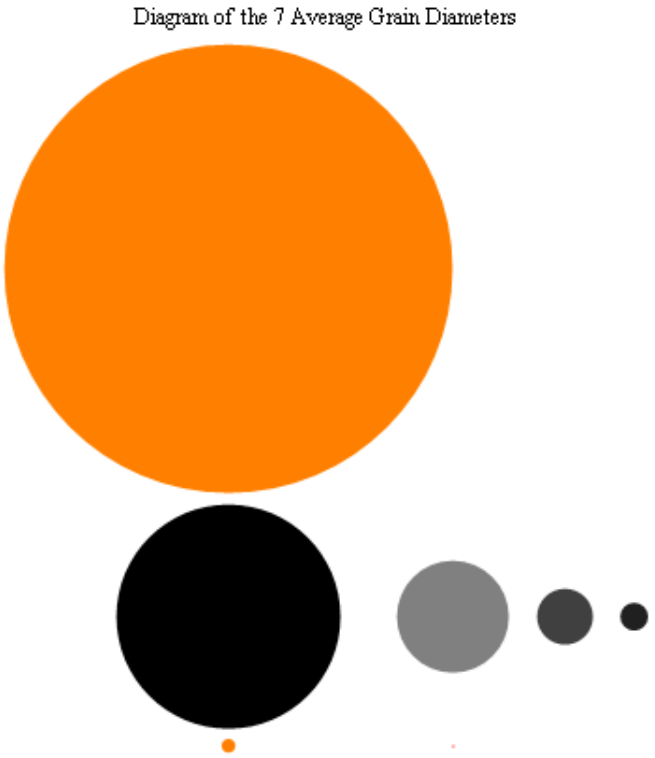


Figure 2.2: Diagram showing some of the Particle diameters, relative to each other. Boulders and Clays are not show. To the bottom right, a silt particle is *just* visible.

2.6 Particle Types and Boundary Conditions

2.6.1 Particle Types

Different particle *types* is not a synonym for **MPD**. In the context of this report, particle type refers to the behaviour exhibited during a simulation. Although numerous basic types may be created, all results were obtained using just 4 basic types. The 4 basic types are:

Type 0 A basic representation of a granular particle, known henceforth as a free particle. Its motion is governed by its interaction with other particles in the system that it comes into mechanical contact with and its acceleration due to gravity , $\vec{\mathbf{g}}$, all according to Newton's equations of motion.

Type 1 These particles, known as wall particles, are used to construct static boundaries. Free particles that come into contact with these wall particles experience a repulsive force and a change in their trajectory, yet these wall particles do not experience an equal but opposing force. Their position is static throughout an entire simulation.

Type 2 These particles represent the walls of a vertically vibrating structure which encompasses a granular media. Think of a bowl of peanuts. Particles of *type*2 would make up the bowl. The frequency, ω, at which these particles vibrate can be set by the user as can the Amplitude of the Vibration and the point about which each particle vibrates. The trajectory of these particles follows a Sinusoidal path.

Type 3 These particles are used to compress a system of particles vertically. Similar to static wall particles, they feel no force when in mechanical contact with other free particles. Their trajectory is downward at a velocity the user may set.

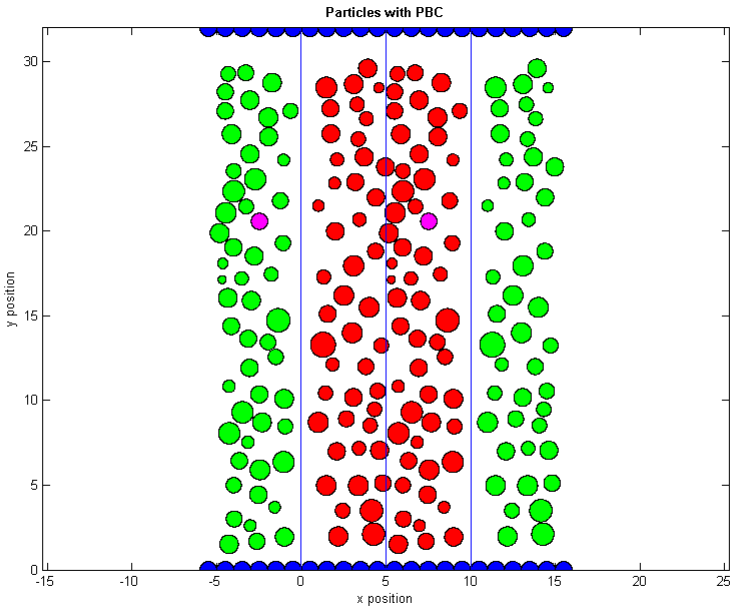


Figure 2.3: A small system with **PBC**'s

2.6.2 Boundary Conditions

This simulation software is primarily used to investigate the motion of small particle ensembles within well defined static boundaries. It has been extended to include vertical vibration and compression of the system. This implies that to simulate a very large system of particles, $\mathbf{N} \sim 20,000$ particles, and investigate some simple properties of such a large collection of particles - all $20,000$ particles must be generated and have their trajectories computed at every time step. To investigate the properties of large systems in a feasible timeframe, **Periodic Boundary Conditions**, **PBC's**, would allow the user to mimic a very large system with a much smaller number of particles. It was not felt that **PBC's** should be incorporated into the model primarily as the intention was to focus on the dynamics of particles contained within rough static boundaries.

2.7 Efficient Force Summation

The most computationally expensive part of a Granular Dynamics simulation is the calculation of the contact force between particles, Equation (2.9) & Equation (2.10). Since only short-range forces are present, paring down the number of calculations per timestep can dramatically improve the performance of a simulation, with the most notable effects being faster run times and larger system sizes.

For example, if there are $\mathbf{N} = 1000$ particles in a system, then by using a brute force method, where every particle i is checked for contact with the other particle of a higher index, $j; j > i$, there will be

$$\frac{\mathbf{N}(\mathbf{N}-1)}{2} = 500,000$$

calculations per timestep. Particles of average radii have no than six other particles in contact. So, in essence, only 3000 force calculations are necessary per timestep. Using the brute force method, excessive computation time is spent and hundreds of thousands of pointless force calculations are completed. In order to run simulations in acceptable times and increase the system size, a better method of determining which particles are in contact at each timestep is required.The method chosen was the **Verlet List** method, [PÖ5d].

2.7.1 The Verlet List

The Verlet List makes use of a property of granular systems. The neighbouring particles of the i^{th} grain rarely change. This assumption is valid in systems that are *not* violently agitated, (shaken, tipped, impacted, sheared, etc.), although the Verlet algorithm was successfuly used to simulate agitated systems. Over a few timesteps, the particles will only move minute distances and more than likely stay in contact.

Particles are deemed to be neighbours if the distance between their centers is less than a pre-defined constant known as the *verlet_distance*. Particles within the *verlet_distance* are those which are in contact, or probable contact with the i^{th} particle, defined as:

$$\mid \vec{\mathbf{r}}_i - \vec{\mathbf{r}}_j \mid -R_i + R_j < verlet\_distance \tag{2.7}$$

To make the algorithm even more efficient at locating neighbours, the system is

divided into a grid. Only particles in the grid of the i^{th} particle or adjacent grid cells of the i^{th} particle are checked using Equation (2.7). The force calculations for the i^{th} particle are carried out on the particles in the Verlet list of particle i for each timestep until the Verlet list needs to be updated or rebuilt. The first case, the update, occurs if the displacement of a neighbour particle of any particle in the system is greater than or equal too $\frac{verlet\_distance}{2}$. By using this criteria, static packings will rarely need an update, whilst more mobile systems will seldom experience a case where two particles pass right through each other or detect the collision too late.
If this does occur, given a very specific set of circumstances take place, [PÖ5e], and a collision is not detected when it should have, the Verlet list of every particle is rebuilt and the simulation time is brought back to the time when the safe version of the Verlet list was constructed.
The efficiency of the Verlet algorithm is governed by three constants. *verlet_distance*, *verlet_grid* and *verlet_increase*.
verlet_distance can be set to any positive value and usually a small value is preferable if the average velocity of the particles is small or if the timestep is very small. If this value is too small, then collisions will be detected late and the Verlet lists will constantly have to be rebuilt. If it is too large, then more particles will be added as neighbours to each Verlet list even though they are not really in contact or even close proximity with the i^{th} particle. *verlet_grid* should be set to the largest particle diameter in the system at a minimum. It can be set to a larger value but again this will increase the size of every Verlet list. *verlet_increase* is a number that *verlet_distance* is multiplied by if the Verlet lists must be rebuilt. In a way, the Verlet algorithm self-corrects until an optimal *verlet_distance* is found and all collisions are detected when they should be. For simulations run here, the values chosen were:

$verlet\_distance = \frac{\bar{R}}{2}$
$verlet\_grid = (1.1)R_{max}$
$verlet\_increase = 1.1$

2.7.2 Profile of a Non-Verlet Simulation versus a Verlet Simulation

On this page is a table of source data and a graph which illustrates the incredible efficiency improvment that the Verlet List algorithm acheived. The test simulation was run for 5 different system sizes at a timestep of $\Delta t = 10^{-6}$, for 1000 timesteps and was carried out using the **MD**software.

Table 2.2: Time taken for 1000 steps.

Particles	Brute Force (mins)	Verlet List (mins)
382	0.41667	0.0166667
685	1.8	0.0833333
1285	6.81667	0.183333
2485	25.8833	0.4
3885	63.3833	0.733333

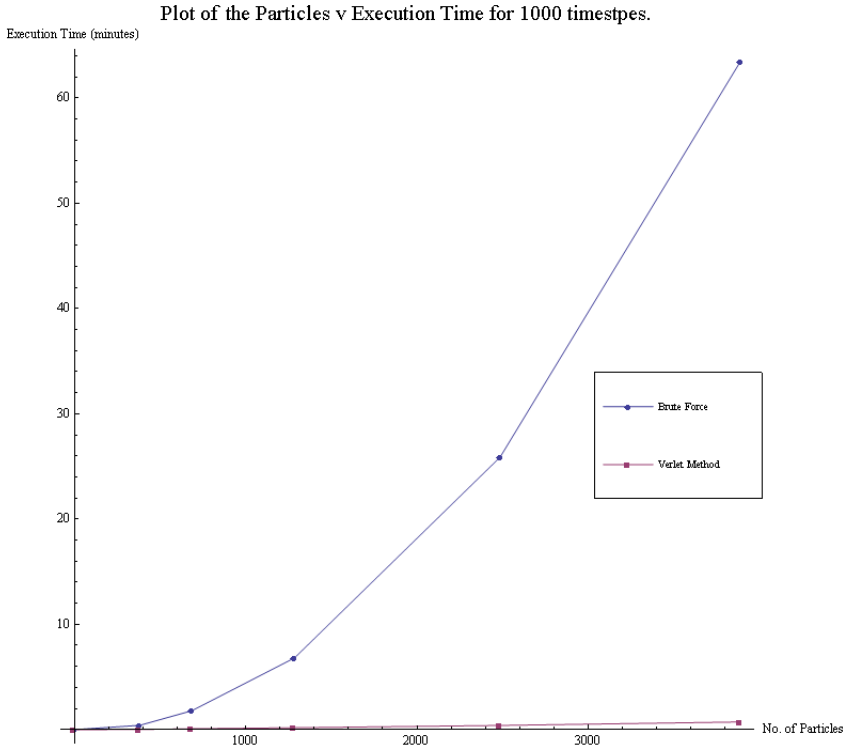


Figure 2.4: The speed-up achieved by the Verlet Algorithm.

2.8 Data Input

2.8.1 Generating an $*.in$ File

To run a simulation, first a $*.in$ file had to be generated. The $*.in$ file contained simulation specific information in addition to the properties of the particles and their initial positions, velocities etc. The first 5 lines of the $*.in$ file set the simulation parameters of the total number of particles, $\mathbf{N}$, the force law to be used, $0 or 1$, the maximum amount of time to run the simulation, t_{max}, the timestep, Δt and the x, y & z components of the gravity vector. A typical $*.in$ file header looks like this:

```
1711
0
5
0.000001
0 -9.81 0
```

Followed by the properties of each particle in the system. Every particle is represented by a line, where each line contains 13 properties:

- x-position,
- y-position,
- Euler angle,
- x-velocity,
- y-velocity,
- angular velocity,
- mass,
- radius,
- Particle type,
- Coulomb friction parameter,
- Tangential dampening factor,
- Young's Modulus,
- Dampening constant.

To generate an $*.in$ file the following procedure was executed:

1. The simulation parameters were written to the file first.

2. Next, the geometry of the simulation was set by choosing the positions of the wall particles.

3. The average radii was selected and the max and min values were calculated.

4. The free particles positions were computed. The particles initial x & y positions formed a lattice slightly above the the container defined by the wall particles. The particles initial velocities were set to zero or whatever initial value was desired.

5. The **MD** program then calculated the trajectories of each free particle as they fell under the influence of gravity, collided with the wall particles and each other until they came to rest in a stable configuration. The stable configuration was reached when the system Kinetic Energy, $E_k < \epsilon$, where ϵ was a user defined tolerance.

6. Finally, the resting positions, velocities and material properties of each free particle were written to the $*.in$ file for future use.

The generation of an $*.in$ file was a time consuming process - even with the Verlet algorithm implemented. The sedimentation procedure was used to find the final resting positions of the particles, which although computationally prolix, did guarantee very realistic packings of the particles within the defined geometry. Other techniques, such as the Bagi method, [Bag05], are faster at creating packings but are not exploited during this project due to their unrealistic results.

2.8.2 Basic Types

There are 4 basic types of $*.in$ files used during the simulations.

The Hopper [2] is the $2D$ analog of a $3D$ grain hopper. It was used to demonstrate the liquid like behaviour of Granular materials. A second inverted hopper geometry was be affixed to the bottom of one hopper to create a $2D$ hour-glass configuration.

The Silo is a rectangle with a height greater than its width. In simulations to investigate vibration of a granular material the Silo was used to simulate size segregation of a vibrated system.

The Window Box is a rectangle whose width is greater than its height. The window box was used to create soil samples which were later used in a very basic frost-heave simulation.

The Grain Press is similar in dimension to the window box, only it has an upper enclosed wall of particles that are translated downward at a slow velocity to compress the free particles within the press. It was used in elementary simulations to show phase transition from gas to solid in granular packings without gravity present.

2.8.3 Loading a User Generated $*.in$ File

Essentially, any Granular system with non-periodic boundaries and no long-range forces is well within the capabilities of this **MD** simulation software, provided a user generated $*.in$ file has a correct system geometry i.e, no unwanted gaps in the wall particles, or placement of the free particle lattice in the incorrect position relative to the container described by the wall particles and if proper boundary conditions are added to deal with new particle types.

[2] Please refer to Appendix D which contains the source code to generate a hopper geometry with 400 free particles in a lattice configuration above it.

2.9 Data Output

2.9.1 Generation of an $*.out$ File

In order to visualise a simulation run, it was necessary to periodically write the particle positions and properties to file during the run. The **MD** simulation generates an $*.out$ file which contains the same header properties but, which also prints and appends the updated positions, velocities etc., to the file. The $*.out$ file contains *snapshots* of the system throughout the run. The frequency that the snapshots are taken is set in the program. Its value was the inverse of the time step which guaranteed that there was a snapshot for every 1 second of simulated time. For example, at a time step of $\Delta t = 10^{-6}$, the $*$.out file was written to every 10^6 time steps. For an 8 second simulation, there were 8 snapshots. The frequency may be adjusted in the source code but in doing so, the size of of the $*$.out file increased or decreased to a point where too much, or not enough information about the simulation history was present.

2.10 Data Visualisation

To view the contents of an $*.out$ file a Mathematica notebook[3] was created which read in the data and created a movie of the simulation from the snapshots and then exported the movie to Flash format, $*.swf$ for later viewing. The visualisation notebook was not only limited to creating simulation movies. Any data written to a file type supported by Mathematica could be read by the notebook., this was used for the granular compaction simulation where a movie was not necessary but information about the coordination number was.

[3]Please refer to Appendix C to see the code contained in this notebook.

Chapter 3

Model Validation

3.1 Why validate the model?

Testing was carried out on the model to ensure that meaningful results would be obtained. To give meaningful results, the model had to obey three basic principles of a closed system:

The Conservation of Total Energy, E_T

The Conservation of Linear Momentum, $\vec{\mathbf{p}}_T$

The Conservation of Angular Momentum, $\vec{\mathbf{L}}_T$

Two different collision types had to be tested, Head on collisions (HoC), where particles have the same $y-$position but different $x-$positions, and Oblique collisions (ObC), where particles have different x and y positions and collide with each other at glancing angles. Then the dissipation factor, γ, also had to be verified.

After the tests have been completed, the outcome of each is analysed and returns adequate results, the model can be used as a competent simulation of a granular ensemble.

3.2 Conservation of Total Energy

The conservation of total energy states that for a closed system, i.e. a system where collisions are perfectly elastic and no dissipation is present, the total energy, E_T, *must* be equal to the sum of the Elastic Potential Energy, E_p, and the Kinetic Energy, E_k, and that this value of E_T is constant at all values of time, t:

$$E_T(t) = E_k(t) + E_p(t) \tag{3.1}$$

$$E_k = \frac{1}{2} m_i \vec{\mathbf{v}}_i^2 \tag{3.2}$$

$$E_p = \frac{1}{2} k_n \xi_{ij}^2 \tag{3.3}$$

where,

$\vec{\mathbf{v}}_i^2$ = Velocity of particle i
m_i = Mass of particle i
k_n = Spring constant
ξ_{ij} = The overlap between colliding particles i and j

In systems where dissipation is present, either through friction or inelastic collisions, then (Equation 3.1) is not satisfied.

3.2.1 Without Dissipation

Two free particles, i and j, were placed in a HoC with initial velocities:

$$\vec{\mathbf{v}}_i = 0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = -0.01_x + 0.00_y$$

The particles collided and both experienced repulsive forces in the x-direction and no forces in the y-direction.
After the collision, the particles final velocities are inverted from their initial values:

$$\vec{\mathbf{v}}_i = -0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = 0.01_x + 0.00_y$$

Since (Equation 3.2) takes the square of the velocity, the initial and final values of $\mathbf{E}_k$ are identical.

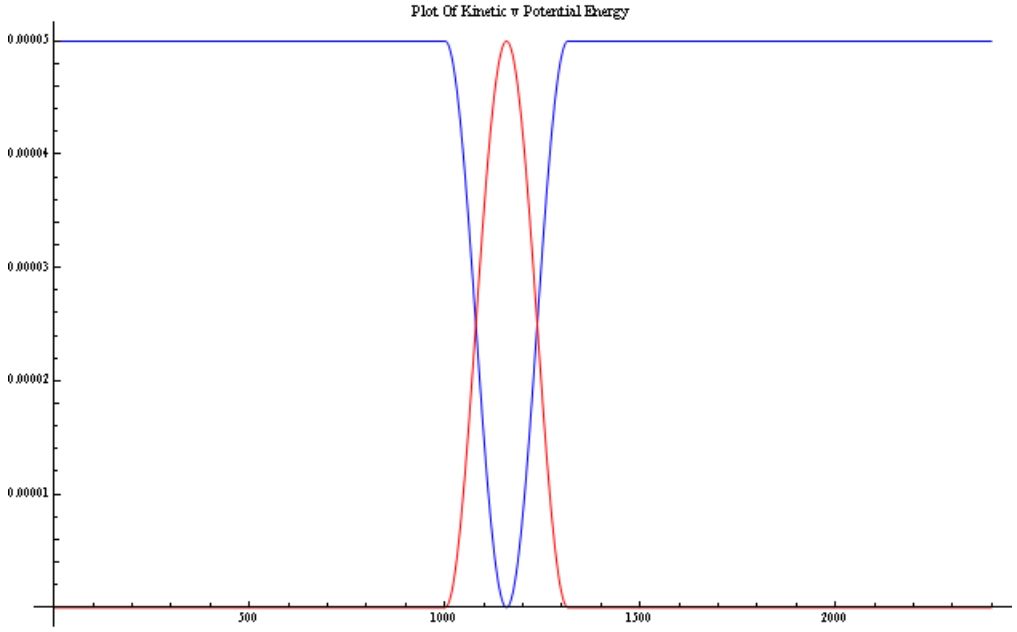


Figure 3.1: Conservation of Total Energy, $\mathbf{E}_T$.

(Figure 3.1) is a plot of E_k and E_p. When E_k goes through a minimum, E_p goes through a maximum. When the particles collide, their velocities begin to decrease and the kinetic energy of the system is converted into potential energy, causing the particles to slow down. The particles eventually have zero velocity, the particles stop, and all of the systems energy is in stored elastic potential energy. The stored potential energy is then converted back into kinetic energy, causing the particles to increase their velocities and repulse each other. It can be seen that at any time, the value of $E_k + E_p$ is constant.

Shown below (Figure 3.2) is an ObC between two free particles, i and j. Snapshots of the particles were taken at regular intervals of 10000 time-steps, to adequately show the particles change in direction and constant velocity. The particles have initial velocities:

$$\vec{\mathbf{v}}_i = 0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = -0.01_x + 0.00_y$$

Both particles experience repulsive forces in both the x and y directions. The final

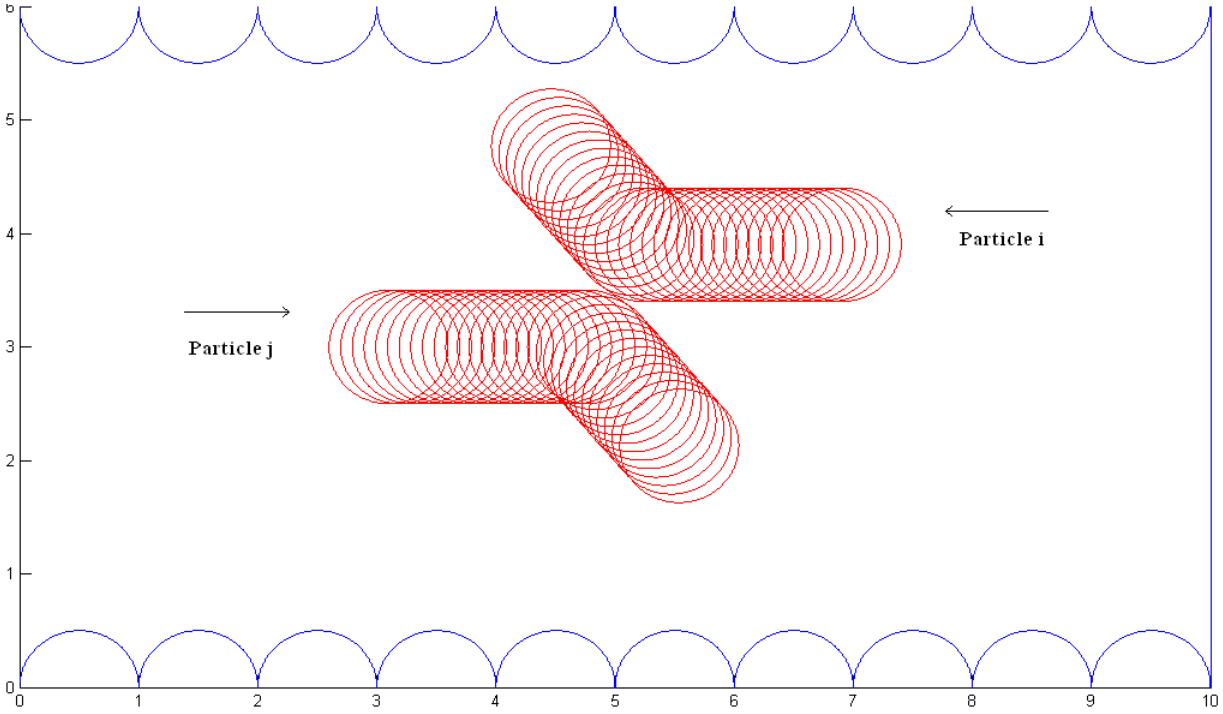


Figure 3.2: Non-dissipative Oblique Collision.

velocities of the particles are:

$$\vec{\mathbf{v}}_i = 0.006502_x - 0.007603_y \qquad \vec{\mathbf{v}}_j = -0.006502_x + 0.007603_y$$

Comparing the initial and final values of (Equation 3.2), E_T is still conserved.[1]

[1]The final values of $\vec{\mathbf{v}}_i$ and $\vec{\mathbf{v}}_i$ shown here are truncated. Calculations in the model were precise to 16 decimal places.

3.2.2 With Dissipation

The HoC described in the previous section was also tested with dissipation. The same initial conditions were used

$$\vec{\mathbf{v}}_i = 0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = -0.01_x + 0.00_y$$

and a dissipation factor of $\gamma = 1.0$ was used. In this test, the initial and final values of (Equation: 3.2) should *not* agree. Shown below are the final velocities of particles i and j:

$$\vec{\mathbf{v}}_i = -0.003_x + 0.00_y \qquad \vec{\mathbf{v}}_j = 0.003_x + 0.00_y$$

A dissipation factor of $\gamma = 1.0$ corresponds to a coefficient of restitution $\mathbf{e} = 0.3$ so the particles are expected to lose $\approx 70\%$ of their kinetic energy due to the collision. The initial and final values of particle velocities agree with this and hence validate the dissipation factor of the model.

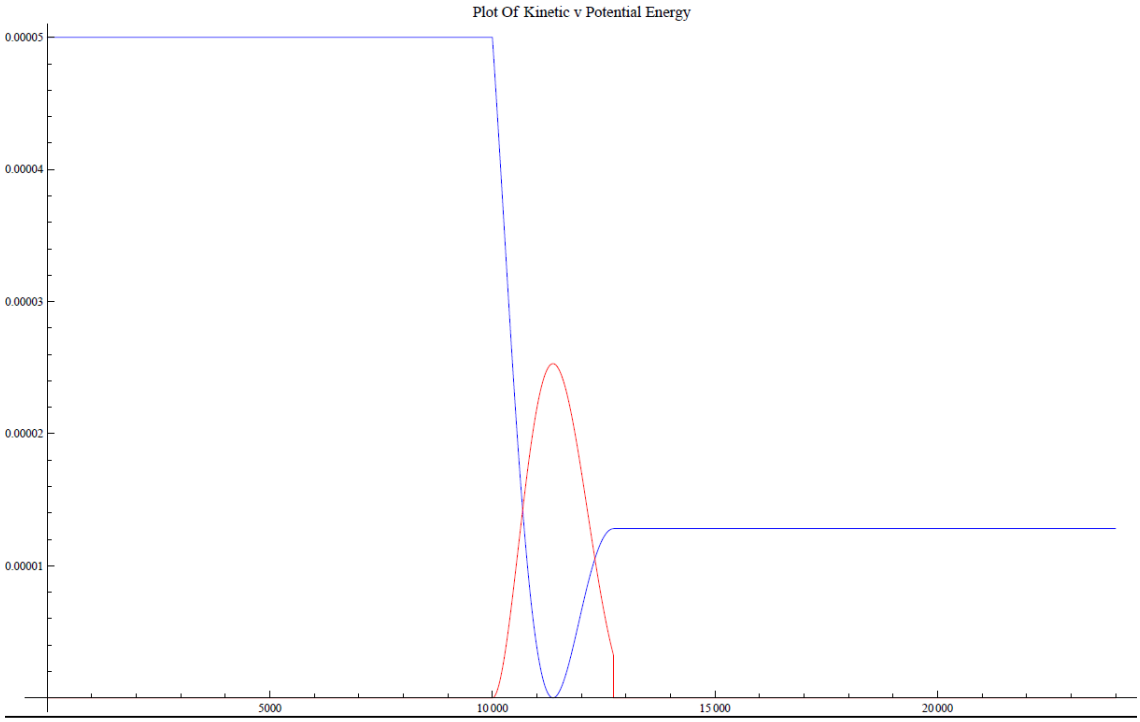


Figure 3.3: Total energy *not* conserved, $\mathbf{E}_T$.

Shown below (Figure 3.4) is another ObC between two free particles, i and j. Again, snapshots of the particles are taken at regular intervals of 10000 time-steps. The initial velocities of the particles are:

$$\vec{\mathbf{v}}_i = 0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = -0.01_x + 0.00_y$$

$\gamma = 1.0$ in this test so the particles should not only experience repulsive forces in the x and y-directions, but should also lose $\approx 70\%$ of their kinetic energy due to the collision.

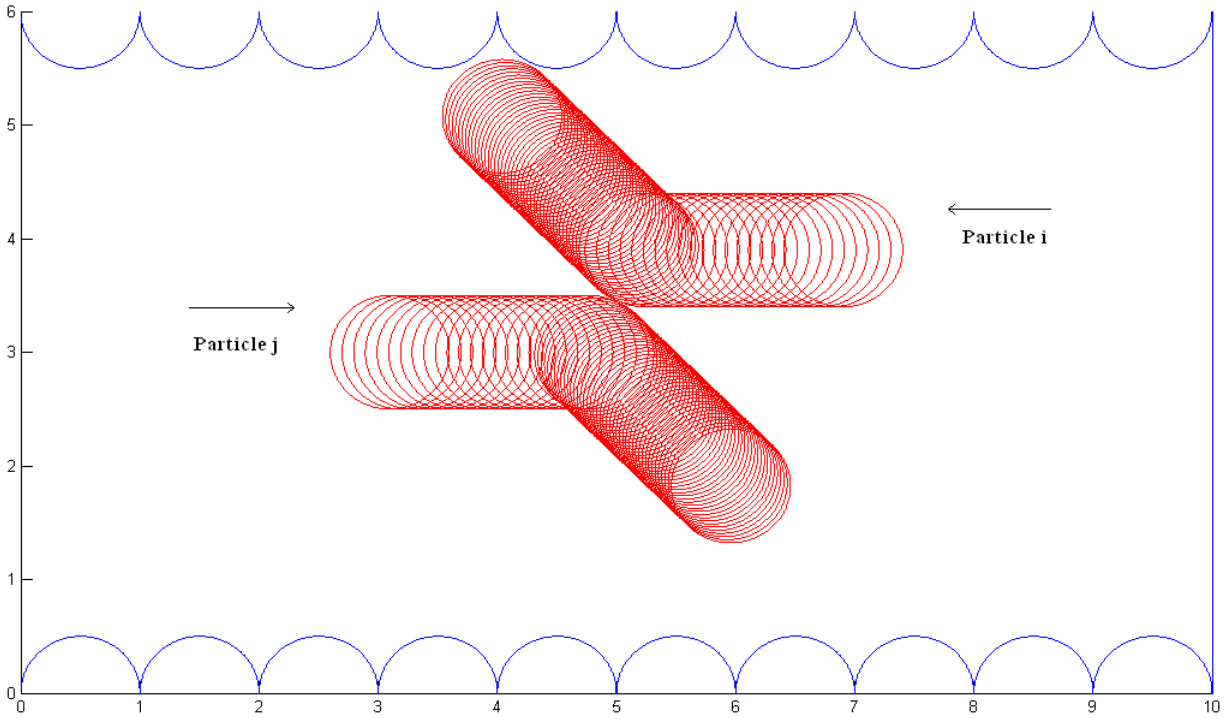


Figure 3.4: Dissipative oblique collision, $\gamma = 1.0$.

The final velocities of the particles are:

$$\vec{\mathbf{v}}_i = 0.002789_x - 0.002805_y \qquad \vec{\mathbf{v}}_j = -0.002789_x + 0.002805_y$$

The paticles have lost $\approx 70\%$ of their kinetic energy. (Figure 3.4) shows this. Initially the spacing between succesive snapshots of the particles is uniform. At the point of contact the spacing between snapshots begins to decrease and they appear closer together, corresponding to a decrease in the velocity of the particles. Compare this with (Figure 3.2) where the spacing between snapshots of a particle is constant throughout.

3.3 Conservation of Linear Momentum

The conservation of linear momentum states that, the total momentum after a collison must equal the total momentum before a collision in a closed system:

$$m_i\vec{\mathbf{v}}_i^u + m_j\vec{\mathbf{v}}_j^u = m_i\vec{\mathbf{v}}_i^v + m_j\vec{\mathbf{v}}_j^v \tag{3.4}$$

Rearranging (Equation 3.4) gives,

$$(m_i\vec{\mathbf{v}}_i^u + m_j\vec{\mathbf{v}}_j^u) - (m_i\vec{\mathbf{v}}_i^v + m_j\vec{\mathbf{v}}_j^v) = 0.0 \tag{3.5}$$

where,

$\vec{\mathbf{v}}_i^u$ = Initial velocity of particle i
$\vec{\mathbf{v}}_i^v$ = Final velocity of particle i
m_i = Mass of particle i

3.3.1 Without Dissipation

Two free particles, i and j, were placed in a HoC with initial velocities:

$$\vec{\mathbf{v}}_i = 0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = -0.01_x + 0.00_y$$

The particles collided and both experienced repulsive forces in the x-direction and no forces in the y-direction.
After the collision, the particles final velocities are inverted from their initial values:

$$\vec{\mathbf{v}}_i = -0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = 0.01_x + 0.00_y$$

Subbing these values of initial and final velocities into (Equation 3.5) yields conservation of linear momentum. (Figure 3.5) shows that the total linear momentum

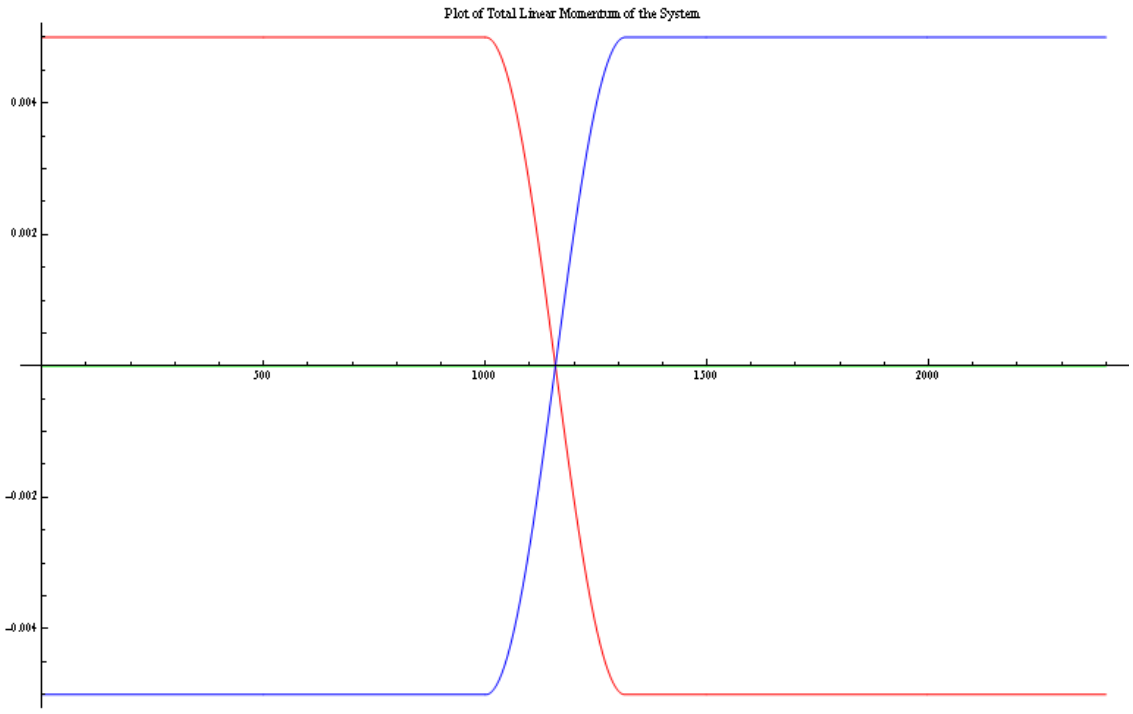


Figure 3.5: Conservation of linear momentum, $\vec{\mathbf{p}}_T$.

of the system remained constant at 0. As both particles had equal masses the linear momentum remained evenly distributed between the particles at all times. The green line shows the instantaneous linear momentum of the system at all times, which is always at 0.

3.4 Conservation of Angular Momentum

The conservation of angular momentum states that the total angular momentum of a system remains constant (is conserved) if the net external Torque acting on the system is zero:

$$J_i\vec{\omega}_i^u + J_j\vec{\omega}_j^u = J_i\vec{\omega}_i^v + J_j\vec{\omega}_j^v \tag{3.6}$$

Rearranging (Equation 3.6) gives,

$$J_i\vec{\omega}_i^u + J_j\vec{\omega}_j^u - J_i\vec{\omega}_i^v + J_j\vec{\omega}_j^v = 0.0 \tag{3.7}$$

where,

$\vec{\omega}_i^u$ = Initial angular velocity of particle i
$\vec{\omega}_i^v$ = Final angular velocity of particle i
J_i = Moment of Inertia of particle i

3.4.1 With Dissipation

To test for the conservation of angular momentum it was necessary that dissipation was present. If $\gamma = 0$ then the particles would be frictionless and in an ObC would only glance off of each other without any change in their respective angular velocity, ω. For this test, $\gamma = 1$.

Two particles, i and j were given initial positions that would ensure they underwent an oblique collision, initial translational velocities of:

$$\vec{\mathbf{v}}_i = -0.01_x + 0.00_y \qquad \vec{\mathbf{v}}_j = 0.01_x + 0.00_y$$

and initial angular velocities of:

$$\vec{\omega}_i = 0.0 \qquad \vec{\omega}_j = 0.0$$

The particles instantaneous angular velocity was then calculated and plotted.

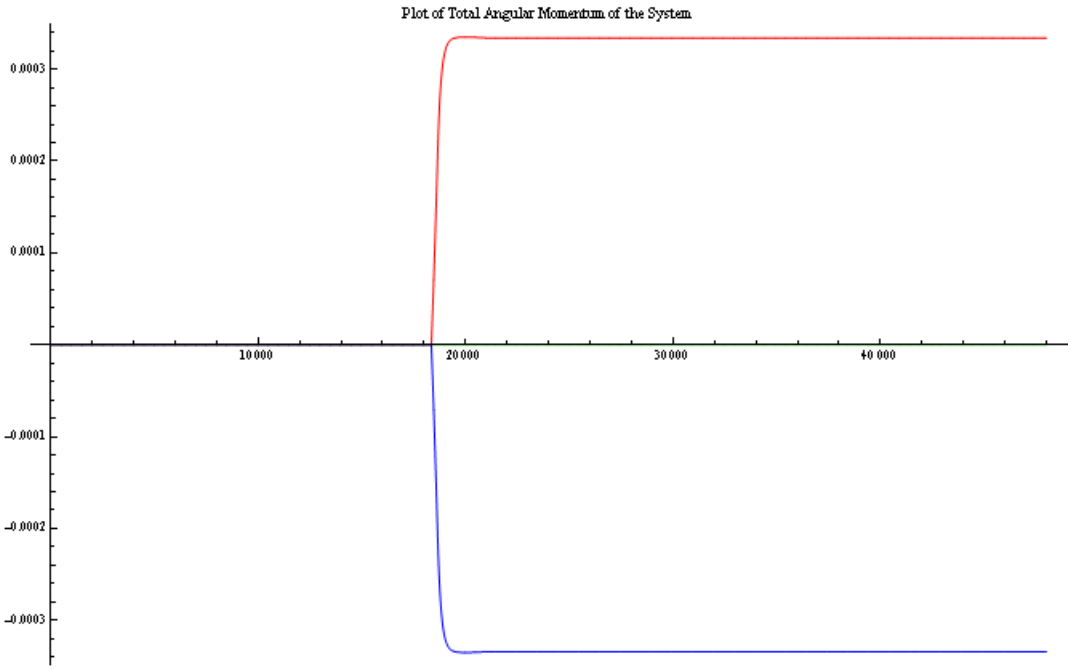


Figure 3.6: Conservation of angular momentum, $\vec{\mathbf{L}}_T$.

(Figure 3.6) above shows a plot of the angular velocity of particles i and j at each timestep of the test. Also shown is the sum total of the angular momentum of the system, shown by the green line. At each instant its sum is 0. When i and j collided obliquely a normal force *and* frictional force were generated and exerted on both

particles equally. These forces gave rise to both linear and angular accelerations, $\vec{\mathbf{a}}$ and $\vec{\phi}$ and caused a subsequent change in the linear velocity and angular velocity of the particles. Both particles i and j have angular velocities of equal magnitude but of opposing direction, as to be expected from an ObC between two particles of identical mass and linear momentum.

Chapter 4

Aspects of Granular Media

The developed software is used to simulate and investigate some aspects of granular media when energy is added to the system in different forms: gravitational potential energy, and vertical mechanical energy. This is done to perform *brief* investigations of phenomena which arise in granular systems when this energy is added, specifically granular flow and size segregation. Finally, a brief study on the distribution of contact forces in different system sizes is carried out. Each section contains its own discussion and conclusions. The purpose of this section is to demonstrate the flexibility of the software. It can be used to simulate different systems and aspects. If the user wishes to investigate a particular aspect in detail, then the code may be modified to suit such a simulation.

4.1 Granular Flow

4.1.1 Background Theory & Simulation Setup

Background Theory

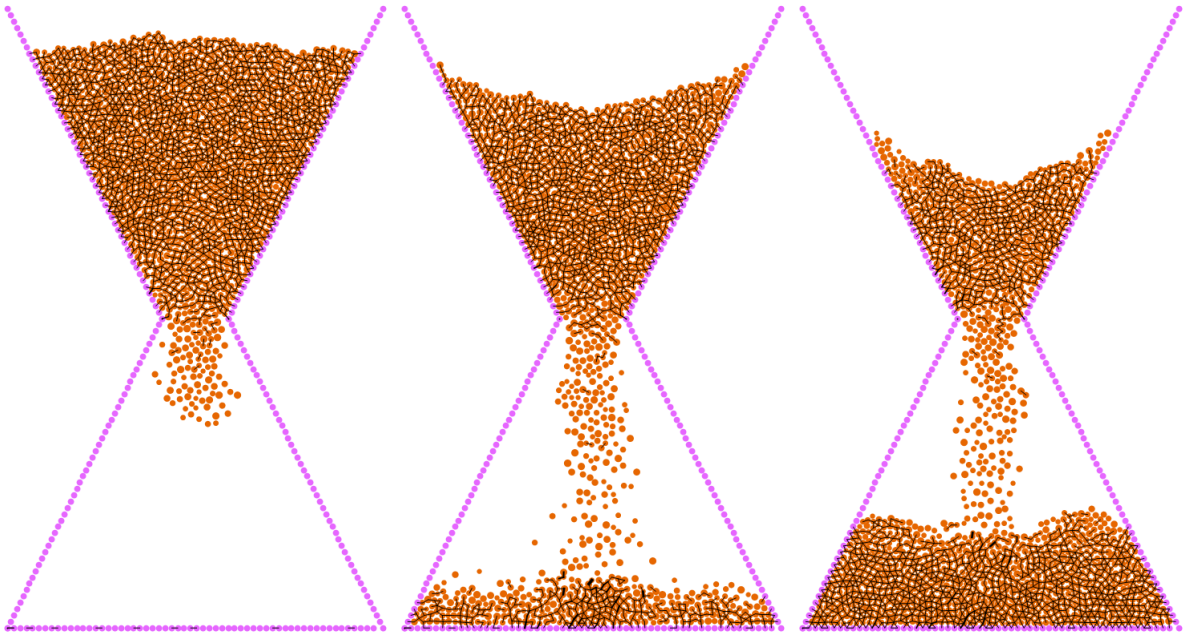
Granular materials display a surprisingly complex range of properties which make them appear solid-like or liquid-like depending on the applied energy, [Jae96a]. Because the interaction between the grains is dissipative and the thermal energy scale is small compared with the energy required to move grains, such materials quickly come to rest unless external energy is supplied constantly. Granular flows driven purely by gravitational potential energy occur in nature, the most common example is an Hour Glass, where a dense granular flow is used to keep fairly regular time. Dense granular flows currently have no fundamental statistical theory available to describe their properties. One reason for this situation is the lack of quantitative data which can be used to test and develop models of dense granular flow. In this brief study, we focus on a simulated granular flow inside an Hour Glass geometry in order to elucidate the nature of the flow.

Figure 4.1: The dense arrangement of particles within the hour glass leads to a quasi-regular flow downwards.

Simulation Setup

A granular sample of 1859 with radii of $\bar{R} = 0.005 \pm 0.001mm$ in an Hour Glass geometry were prepared using the sedimentation method and then allowed to fall under the influence of gravity for 8 seconds. Snapshots were taken every 0.1 seconds.

4.1.2 Contact Network



(a) $t = 0.1s$ (b) $t = 0.5s$ (c) $t = 1.0s$

Figure 4.2: The changing force chain network in Hour Glass flow.

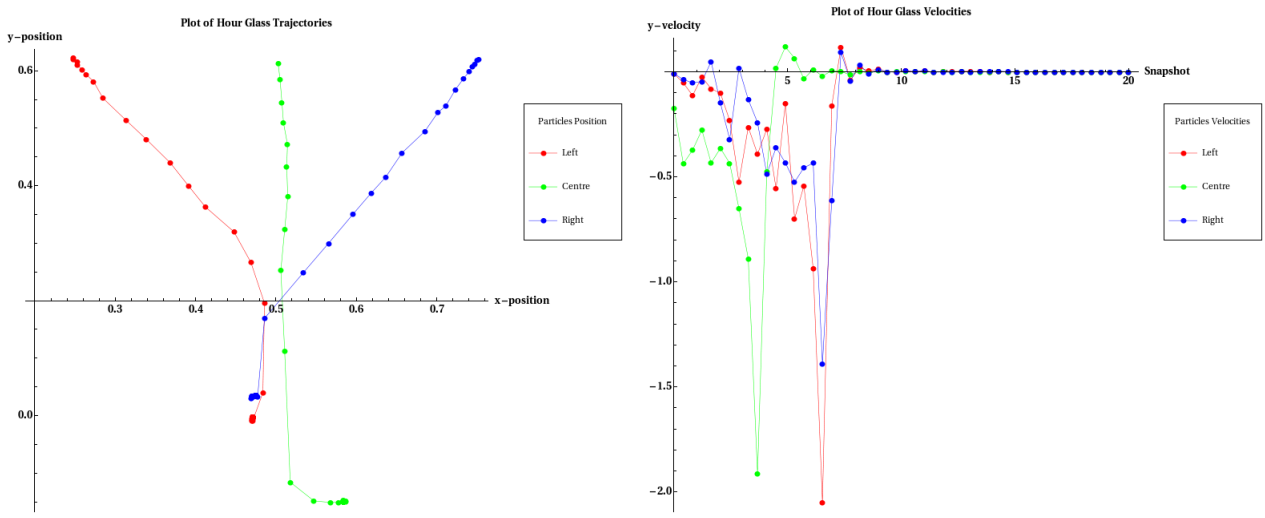
Above are 3 snapshots of the contact force network in the sample as the particles flowed downwards, the thickness of a line is proportional to the magnitude of the contact force. In Figure (4.2a) a network of contact forces, known as force chains, shows that the majority of particles are in contact and that chains can span the entire width of the system - except for a small region just above the center of the hour glass orifice where contacts are weak or non-existent. Particles along the hour glass wall feel the strongest contact force. This is due to dissipative collisions with the stationary wall particles and the contact force from neighbour particles. These particles dissipate their kinetic energy faster due to the collisions and increase their elastic potential energy, becoming temporarily stationary until they can rearrange. These side particles carry most of the stress of the contact network and create temporary system spanning arc's that support the particles above them and allow those underneath to lower their elastic potential energy by rearranging their positions. Particles near the center of an arc feel the weakest contact force and are more likely to roll and slide into new positions which lower their potential energy. Particles underneath the lowest system spanning arch are free to fall under the influence of gravity and occasionally take part in horizontal collisions with other free falling neighbours.

In Figure (4.2b) the particles which formed the center of the system spanning force chains have rearranged their positions to lower their potential energy, thereby weakening the force chain or have broken free from the chain altogether and moved downwards closer to the orifice. The side particles are still the strongest part of the force chains, although they have moved downwards it is only by a little amount and still experience stronger collisions. This manifests itself as the creation of a small depression at the top of the free particles - the inverse of the initial configuration. The highest points occur at both edges and the lowest point is now in the center. Particles on the center line of the orifice, regardless of height appear to have a faster flow velocity.

Figure (4.2c) shows similar behaviour as Figure (4.2b) with particles closest to the orifice feeling weak or no contact force and all particles on the center line of the orifice moving the fastest.

4.1.3 Trajectories & Velocities of Particle Triplets

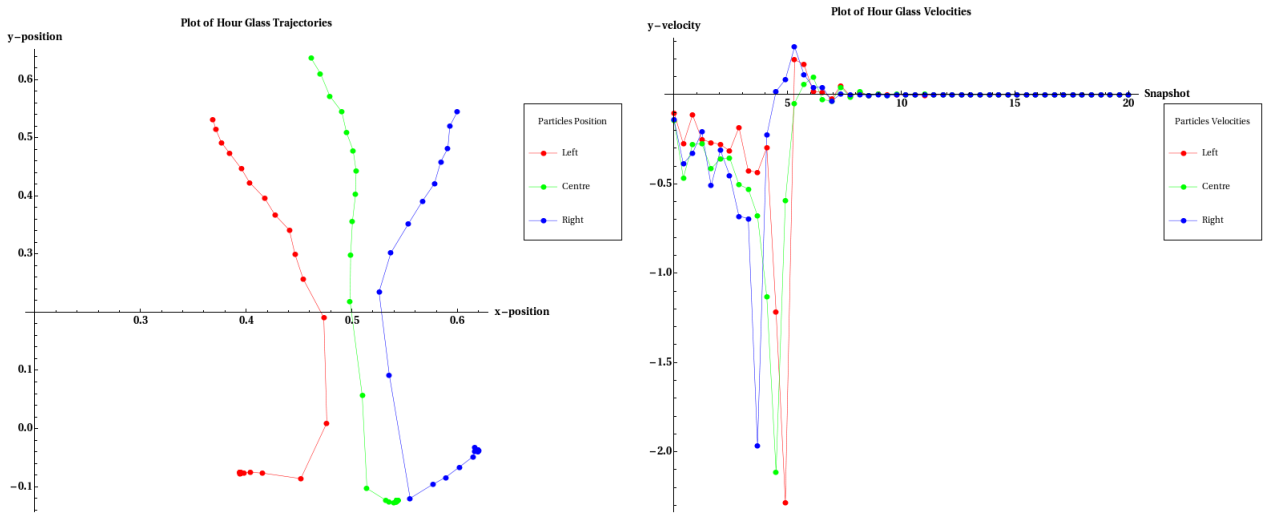
In order to investigate the flow further, the trajectories and velocities of triplets of particles were tracked.



(a) Trajectories (b) Velocities

Figure 4.3: Triplet A, Not Neighbours.

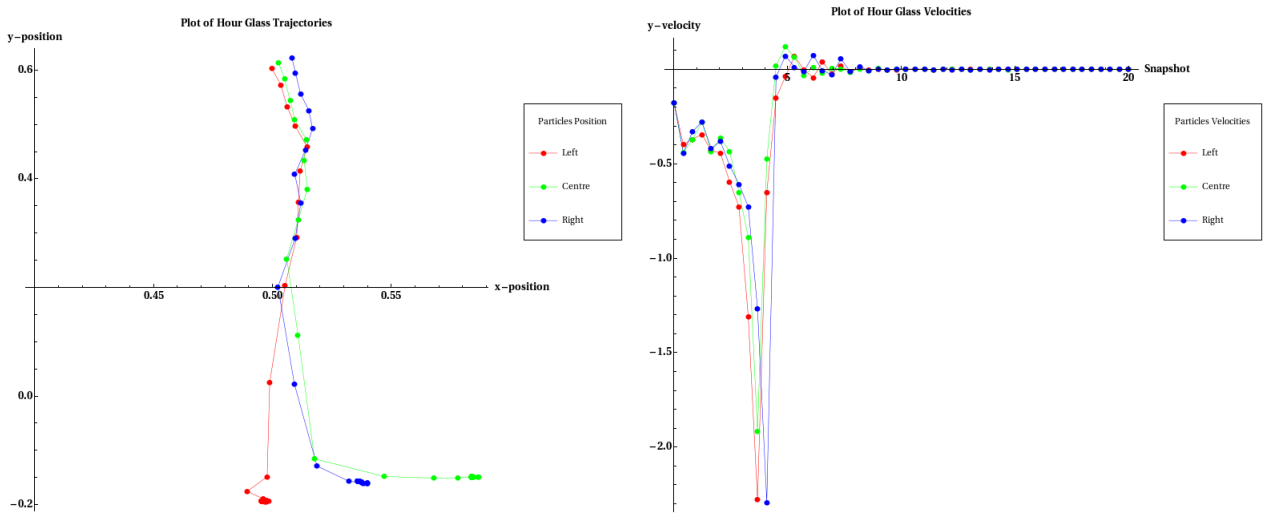
The first triplet, Triplet A, was chosen to highlight the similar behaviour of particles near the top left and right walls and the contrasting behaviour of a particle on the center line of the hour glass at the top of the particle heap. In figure (4.3a), the horizontal axis represents the position of the orifice and each data point represents the particles position at intervals of 0.1 seconds (This is also the case for Figures (4.4a) & (4.5a)). The distance between successive data points can be used as an indication of instantaneous velocity, small distances indicate a low velocity and vice versa. Initially, the left and right particles do not move quickly due to collisions with the wall but eventually increase in downward velocity as they approach the orifice, then pass through and settle in the lower half. The behaviour is almost symmetric for both sides. The center particle moves downward with a much faster velocity that increases with every data point. It passes through the orifice well before the wall particles, < 0.9 seconds. Figure (4.3b) confirms this. It is a plot of the $y-$velocity of the particles as a function of time. The center particle has a greater *average* downward velocity than the left and right particles and is the first to collide with the particles that aggregated in the lower half (Shown by the sudden change from negative to positive velocity). The plot does show that the left particle had the *greatest* downward velocity before impact with the lower particles.



(a) Trajectories (b) Velocities

Figure 4.4: Triplet B, Distant Neighbours.

Triplet B consists of the same central particle as Triplet A but two different left and right particles which are closer to the center and initially lower down than the center particle. Figure (4.4a) shows that the left and right particles have a different velocity to the left and right particles of Triplet A. These particles are surrounded by other downward moving free particles which synchronise their velocities due to horizontal collisions and it appears that there are different velocity profiles for different regions of the hour glass. Figure (4.4b) shows that the downward velocity of all three particles is similar, although the central particle still moves with the greatest initial downward velocity, although the left and right particles are the first to pass through the orifice. This is due to their initial $y-$position being lower than the center particle's.



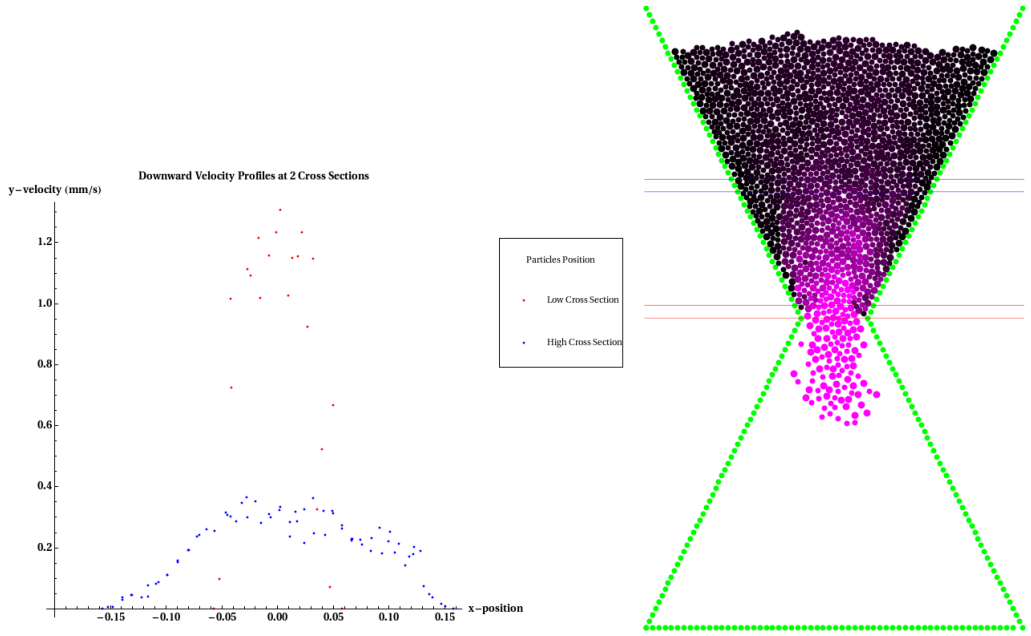
(a) Trajectories (b) Velocities

Figure 4.5: Triplet C, Close Neighbours.

Triplet C is comprised of 3 close neighbours. The center particle is the same as the previous triplets and the left and right particles are now its immediate left and right neighbours. Figure (4.5a) & (4.5b) clearly show that the particles have synchronised their velocities and travel downwards in a fast moving section of the hour glass. The particles cross through the orifice at roughly the same time and speed before settling in their rest positions in the lower half. Clearly, there are differing velocity profiles in the hour glass which are associated with location from the center line on the orifice.

4.1.4 Cross Sectional Velocity Profiles

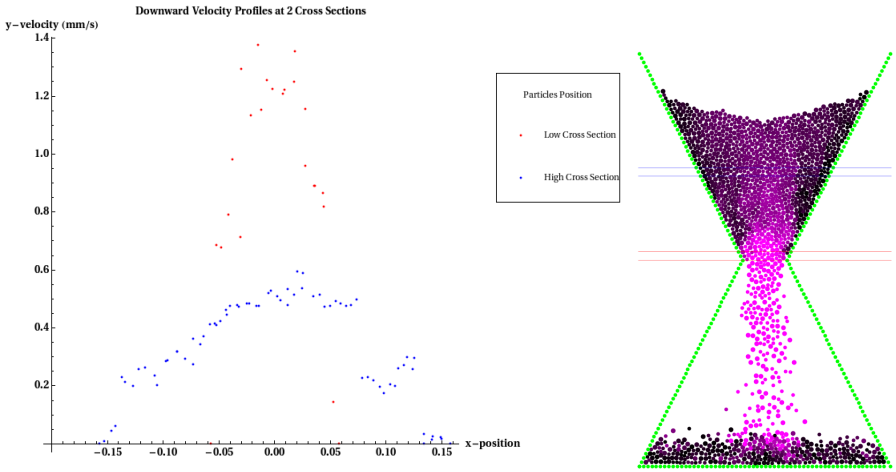
In order to further investigate the presence of velocity profiles in the simulation, cross-sectional velocities were plotted for two regions of the hour glass. The Red data points and horizontal lines in Figure (4.6a) & (4.6) correspond to particles near the orifice and the Blue data points and horizontal lines refer to particles away from the orifice. The absolute values of the velocities were plotted as a function of their horizontal distance from the center of the orifice and a false colour plot of the particles was created. Brighter colours denote faster moving particles (Green represents the stationary wall particles).



(a) Downward Velocity Profiles (b) False Colour Plot

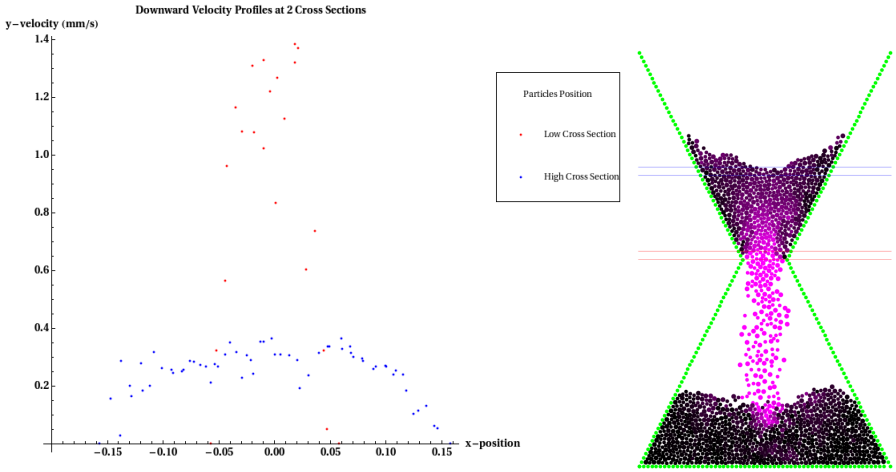
Figure 4.6: Downward Velocity Profiles & Corresponding False Colour Plots at $t = 0.1s$

In Figure (4.6) shows that the maximum velocity is at the orifice and appears to 'diffuse' upwards,[Cho04; JC05]. Particles near the wall form a stagnant region, even some that are close to the orifice are included in it until they begin to increase their velocity and pass through. Figure (4.6a) confirms that the velocity of a particle is dependent upon its proximity to the orifice, with those closest to its center line traveling the fastest. Overleaf are more plots showing the system at different time values.



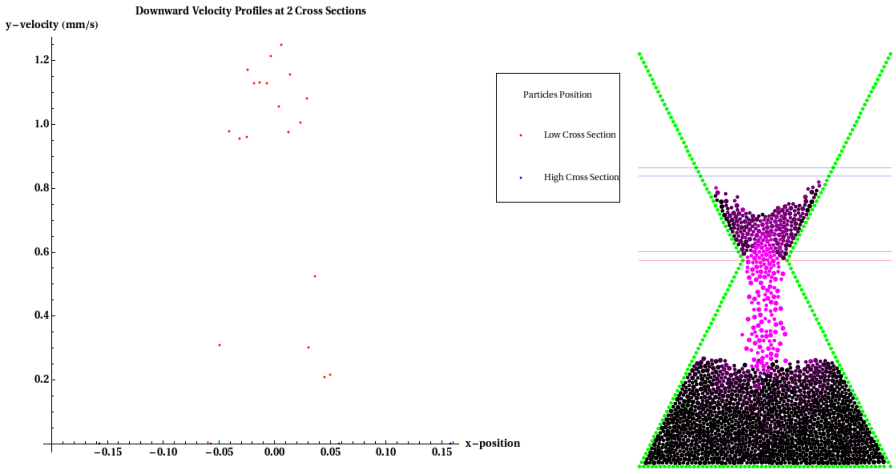
(a) Downward Velocity Profiles (b) False Colour Plot

Figure 4.7: $t = 0.5s$



(a) Downward Velocity Profiles (b) False Colour Plot

Figure 4.8: $t = 1.0s$



(a) Downward Velocity Profiles (b) False Colour Plot

Figure 4.9: $t = 1.5s$

Particles near the orifice have velocities which range from $1 - 1.4\ mm\ s^{-1}$ and tend to cluster around $1.2\ mm\ s^{-1}$. For most particles, regardless of its initial position, a particle passing through the orifice will have a velocity within this range unless repeated collisions with the wall particles reduce its velocity. Particles further up, move within a range of $0 - 0.4\ mm\ s^{-1}$ with the greatest velocity occurring on the center line of the orifice. As these particles move downwards they increase their velocity before passing through the orifice.

4.1.5 Discussion & Conclusions

Wall collisions and the contact network between particles create the gravity driven granular flows we are accustomed to in an hour glass. Particles closest to the walls feel the greatest contact force. This is caused by bearing the greatest stress in system spanning contact chains, which forces them towards the walls where they are compressed. The particles at the center of a system spanning chain feel the weakest force and will abate the chain or even break free from it in order to lower their potential energy. This forming, breaking and reforming of contact chains creates a downward flow of particles which is greatest a the center and is the cause of the parabolic shape of the free particle boundary. This allows particles close to the wall to know lower their potential energy as they are no longer part of system spanning chains and some flow into the center of the parabola. Further, particles move downwards with synchronised velocities dependant upon how close to the center of the orifice they are. The synchronisation is created by repeated horizontal collisions

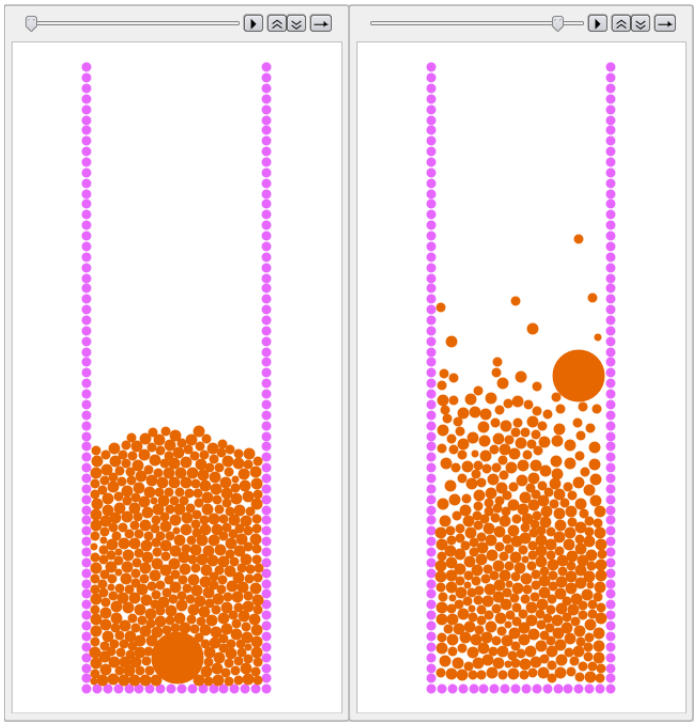
with neighbour particles. The weakening and diminishing of the contact network does allow all free particles to fall under the influence of gravity and synchronisation of velocities leads to a quasi-regular flow rate from the orifice.

4.2 Granular Size Segregation

4.2.1 Background Theory & Simulation Setup

Background Theory

An important effect occurs in many industrial situations in which granular mixtures of particles of different sizes are used.When the particles are vibrated or shaken, the larger particles rise to the top. This size segregation occurs even if the large particles are significantly denser than the smaller ones and even when the size ratio is near 1. The resulting non uniformity is usually an undesirable property, although there are some applications in which shaking is employed as a means of separating particles of differing sizes. Size segregation effects are important in powder metallurgy, pharmaceuticals, and the glass and paint industries.



(a) Initial State (b) After Shaking

Figure 4.10: Size Segregation

Previous experiments and simulations, [Haf86; AR87], supported the earlier hypothesis that the transport of larger particles to the top was caused by the smaller particles filling the void beneath a larger particle during shaking. More recently, a different mechanism has been identified as a possible cause of size segregation - *Granular Convection Currents*, [Gal92; Kni93].

Simulation Setup

537 particles were generated with a silo geometry and subjected to *vertical vibration* for 8 seconds. 1 large particle, with a radii and mass 5 times that of the average radii and mass was placed on the bottom of the silo. Its motion is tracked during the simulation along with the motion of 5 smaller particles to discern if convection currents are present and how they may occur. The vibration follows a sinusoidal trajectory. The $y-$coordinate of each wall particle was augmented at each time step, t, by:

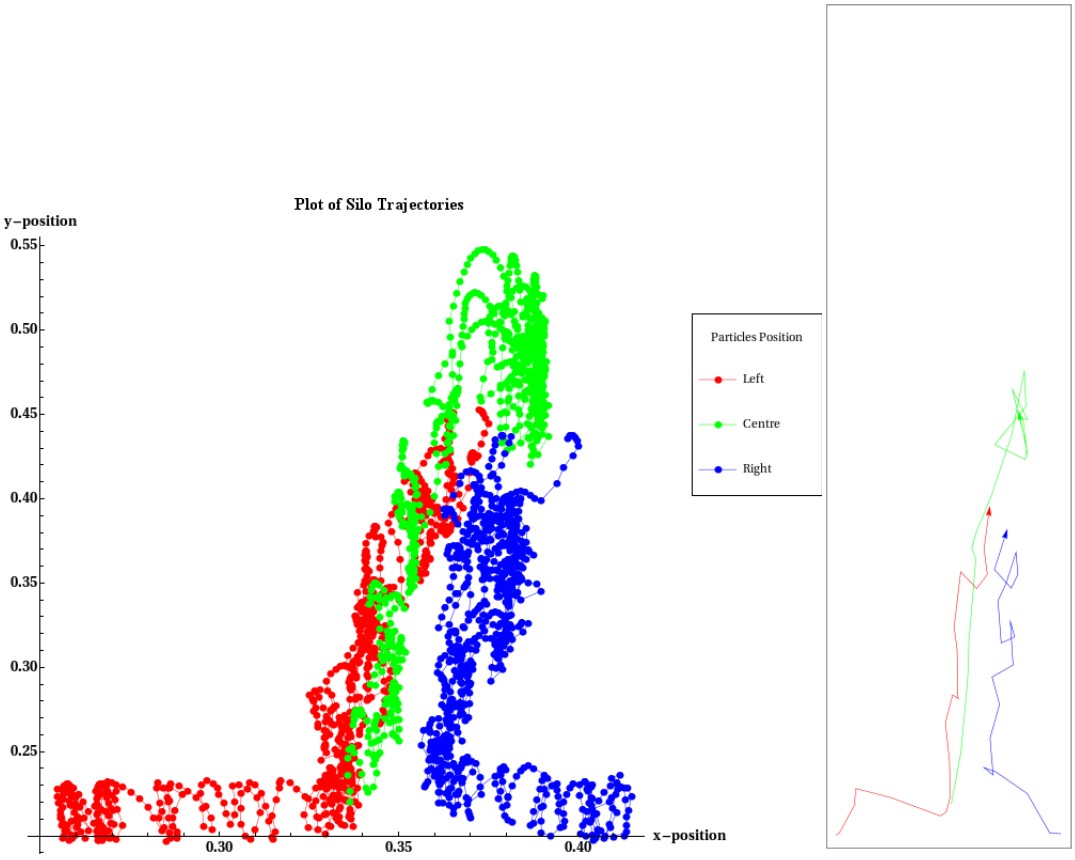
$$\vec{y}(t) = \vec{y}(t) + (0.02 * Sin(60t)) * 0.1 \tag{4.1}$$

and the $y-$velocity by:

$$\vec{\dot{y}}(t) = 0.02 * (60 * Cos(60t)) \tag{4.2}$$

Snapshots were taken at time steps of 0.001 to track the particle motion in detail.

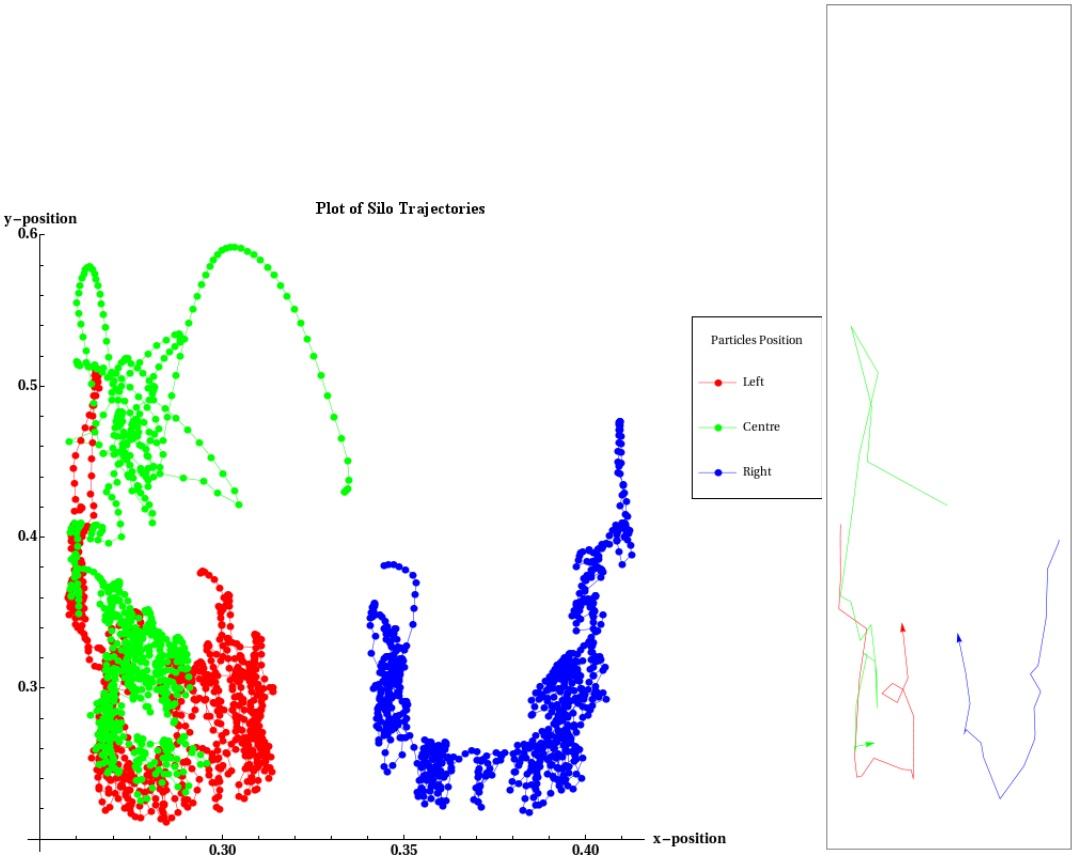
4.2.2 Granular Convection Currents



(a) Triplet A (b) Overall Motion

Figure 4.11: Convective Motion of Particles, Triplet A

Above are the trajectories of 3 particles, including the large particle, initially on the bottom of the silo. The largest particle is represented by Green data points and lines. What is immediately obvious in Figure (4.11a) is that the small particles do not travel exclusively upwards. They move inwards first, away from the container walls before reaching the center and moving up. The large particle moves vertically, with a slight horizontal component towards the right. The motion of all three particles indicates that there are opposing 'currents' in the media. On the left the current is counter-clockwise and on the right, clockwise. The center appears to be where the currents meet and cause upward motion. Figure (4.11b) shows a low-res plot of the trajectories within the Silo geometry, with an arrow to denote the final position. It displays the overall motion of the particles, with much of the upward jerks caused by the vibrations not shown. what is important is the the large center particle stays at the top once it breaches the upper layer and does not recede downwards.



(a) Triplet B (b) Overall Motion

Figure 4.12: Convective Motion of Particles, Triplet B

The upward convection motion is visible in Figures (4.11a) & (4.11b). Figure (4.12a) & (4.12b) show the trajectories of another Triplet of particles, 3 average diameter particles whose initial position was at the top of the free particles and to the left, center and right of the system and displaying downward convection motion. The left and right particles move downwards in streams close to the walls of the container before moving horizontal along the base of the container and upwards when they reach the center. Counter-clockwise motion on the left and clockwise on the right. The center particle, Green, is initially flung to the left by the applied upward vibration, then follows a downward path along the left wall. The particles have recessed back into the system from a starting position a the top, in contrast to the large particle of the previous triplet.

4.2.3 Discussion & Conclusions

The origin of the convection currents and the motion they induce in the free particles can be explained by the effect of rapidly vibrating the particles upward. Upon upward translation of the container base and walls, the packing becomes quite compressed for a short time. Upward motion is opposed by tangential friction with wall particles. When the free particles begin to move downwards under gravity, the packing becomes less dense and the tangential friction decreases. Overall, the net downward motion of the free particles is much greater than the net upward motion, as investigated by Gallas et al,[Gal92]. The downward moving free particles near the walls push the bottom most free particles horizontally. The direction of the pushing is away from the nearest vertical wall, as the particles cannot pass through it and can only move in one other horizontal direction. This manifests itself as a counter-clockwise current at the left wall and a clockwise current near the right wall. When the particles from opposing currents meet, they are pushed upwards. When large particles reach the top surface they remain their. This is due to the diameter of the particles being larger than the width of the left and right current 'channels'. This appears to be related to the width of the container, as in Section (5.2.3) large particles were able to recede back into the system in a much wider container geometry.

- Granular convection currents can form in a vertically perturbed granular media.
- Where two currents form, one will travel in a counter-clockwise direction and the other in a clockwise direction.
- The currents will translate small particles down the side walls, along the bottom and at the center, where the currents meet, particles will move upwards.
- The closer a particle is to a Vertical wall or a corner, the faster it moves. Motion is greatest in the downward direction and away from the corner.
- Further away from the walls, nearer to the center where the currents meet, the velocity appears to be much slower and synchronised.
- Large particles are translated to the surface at the center and remain at the top if the channel width is smaller then their diameter. Channel width appears to be related to the width of the system. A narrow system will give rise to narrow channels and a wide system will have wide channels.

- A combination of these mechanisms results in size segregation of particles in shaken granular media.

4.3 Distribution of Forces in a Granular Media

4.3.1 Background Theory & Simulation Setup

Background Theory

As observed in section (4.1), the force network between particles appears to be inhomogeneous. One quantitative way of analysing the inhomogeneities of these forces is to measure the Probability Distribution, ($P< \mathbf{f} >$), of the normalised forces between particles in contact. Previous experiments, simulations and statistical analysis of granular media, [Mue98], have found that the probability distribution of these normalised contact forces follows an exponential distribution above the average normalised contact force, and follows a power law distribution below the average normalised contact force.

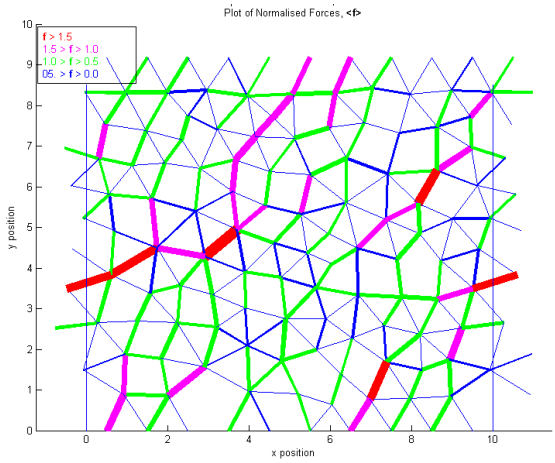


Figure 4.13: False colour force plot.

The image shown above is a false colour image displaying the network of *normalised* force chains. Red represents the strongest, blue represent the weakest. The image clearly shows that contact forces between particles are distributed unevenly.

Simulation Setup

The probability distribution was generated by normalising the contact forces for 5 granular assemblies of, with system sizes ranging from 911 to 3885 particles. Normalisation is carried out by taking the contact force between two particles, $\mathbf{F}$, and dividing by the value of the average force in the system, $\bar{\mathbf{F}}$. This will yield a normalised value of the contact force, $< \mathbf{f} >$. The values of $< \mathbf{f} >$ of each contact force were then binned. The bins ranged from $[0.0 : 5.0]$ in steps of 0.2.

4.3.2 Probability Distribution of Contact Forces

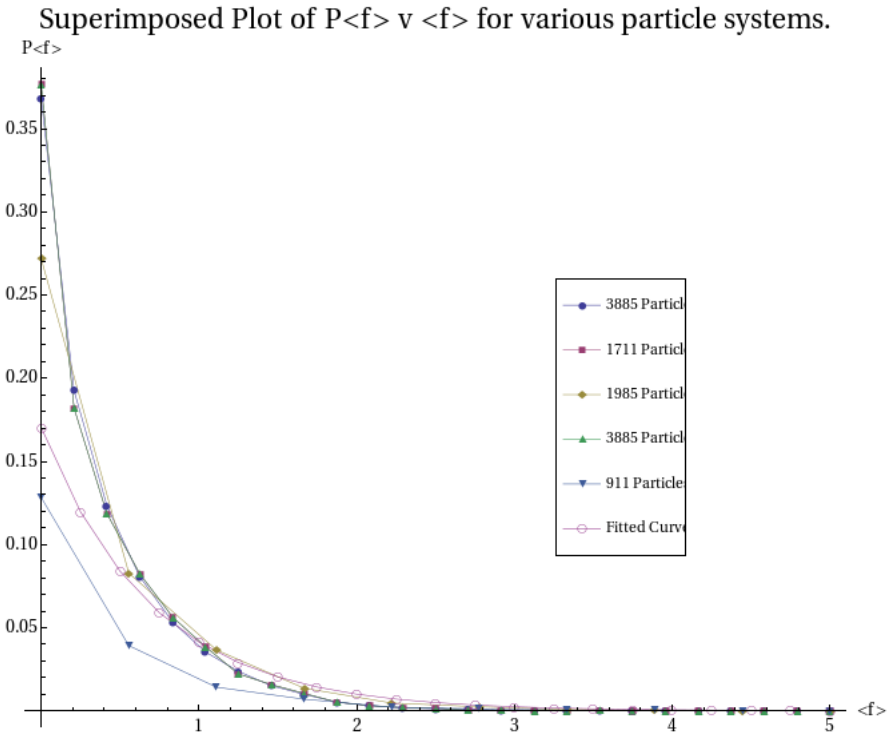


Figure 4.14: $P(< \mathbf{f} >)$

Figure (4.14) shows that the probability of finding a contact force greater than the average forces is exponentially rare. When $< \mathbf{f} >$ is greater than 1 the following relationship is observed:

$$P(< \mathbf{f} >) \propto \exp^{-\beta < \mathbf{f} >} \tag{4.3}$$

The slope of Figure (4.14) was fit for Equation (4.3) with a value of $\beta = 1.77$. Previous simulations [Aha02] have calculated the value of β as 1.47. When $< \mathbf{f} >$ is less than 1, the distribution can no longer be characterised as exponential:

$$P(< \mathbf{f} >) \propto < \mathbf{f} >^{\alpha} \tag{4.4}$$

The above plot was obtained by averaging the values of $P(< \mathbf{f} >)$ from all experiments. Regardless of γ or relaxation energy, contact forces between particles are exponentially rare above $P(< \mathbf{f} >) = 1$. It is also worth noting that almost 60% of the particles in the system experience a contact force that is *less* than the average force. The majority of the stress in the system is carried by the minority of particles.

4.3.3 Discussion & Conclusions

The probability distribution ($P< \mathbf{f} >$) shows a power law behaviour for $< \mathbf{f} >$ is less than 1 and an exponential decay if $< \mathbf{f} >$ is greater than 1:

$$P(< f >) \propto \begin{cases} < f >^{\alpha}, & \text{if } < f > \ < 1 \\ \exp^{-\beta < \mathbf{f} >}, & \text{if } < f > \ > 1 \end{cases} \tag{4.5}$$

This agrees with previous physical experiments [Mue98; Bla01], simulations [Mai07; Rad99; Mak00] and purely statistical analysis [Edw03] of granular media.

The value of the exponential decay constant, $\beta = 1.77$, differs from the A&S result, $\beta = 1.47$. The exponent α is deemed to have a positive value but could not be determined accurately, as not enough data points were present. The different values of α and β can be attributed to the system size. 5 system sizes were simulated and studied but did not generate enough particle contacts to accurately represent the distribution of contact forces in a real compacted granular media. A simulation of one, much larger, system size should see the value of β decrease and allow sufficient resolution of the weaker forces to calculate α.

Chapter 5

Periglacial Landforms

5.1 Periglacial Environments

In the high latitude regions of the Northern and Southern hemispheres and in some areas at high elevation elsewhere, prevailing temperatures are so low that the ground remains frozen for much or all of the year. In this environment the effect of repeated freezing and thawing of the ground and the growth of ice masses is so ubiquitous that the formation of Periglacial landforms occurs, which are a characteristic of the environment.

Periglacial environments are commonly underlain by permanently frozen ground termed *Permafrost*. However not all Periglacial regions contain a Permafrost substructure, so the most accurate definition of a Periglacial region is locations where intense *Frost Action* and the ground is, at least, seasonly snow free. Using this delimination, Periglacial environments may be categorised as such:

Polar Lowlands Mean temperature of coldest month $< -3°C$. Zone is characterised by Ice caps, bare rock surfaces and Tundra vegetation.

Subpolar Lowlands Mean temperature of coldest month $< -3°C$ and warmest month $> 10°C$. Roughly coincides with the Northern hemisphere summer tree line.

Mid-latitude Lowlands Mean temperature of coldest month $< -3°C$ but mean temperature is $> 10°C$ for at least 4 months per year.

Highlands Climate influenced by altitude as well as latitude.

5.1.1 Permafrost

Although we have previously said that Permafrost is not essential to Periglacial environments, it is an important agent in many Periglacial processes[1]. The surface of the permanently frozen layer is known as the *Permafrost Table*. Above the table, annual Summer thawing produces a layer of material known as the *Active Layer* which refreezes every Winter. This active layer may be up to 3 m deep. As a result of freezing penetrating downwards at unequal rates, unfrozen water may be trapped between the Permafrost table and the active layer. These water pockets are called *Taliks*. It is the active layer of the Permafrost which is most excessively engaged with Periglacial processes during the annual Freeze-Thaw cycle. Of particular note is the Periglacial landforms which arise from processes such as *Frost Action* which lead to *Patterned Ground* in Periglacial environments.

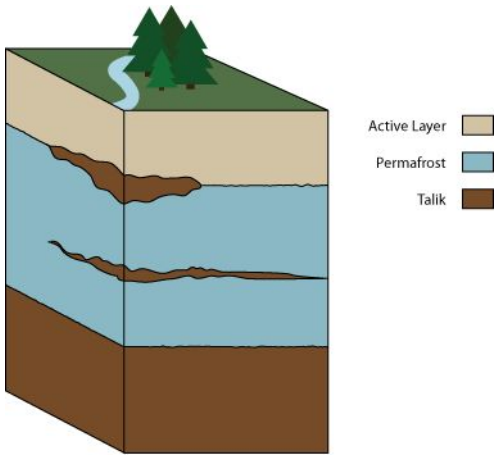


Figure 5.1: A cross-section of the soil in a Permafrost region

5.1.2 Patterned Ground

Intricate patterns of stones and mud decorate the ground in Periglacial landscapes, which contrast sharply with their bleak and disorganised surrounding geography. Many theories as to how these patterns arise have been put forward, including Aqueous Convection Currents in the ground, [Nor09] and the ejection of stones by freezing, [Nan22]. Only recently has it become clear that it is a combination of three processes, over very long spatial scales and very long temporal ranges that give rise to self-organising beaviour in stone domains in these cold climates, [Kes03]. The three processes are the up-freezing of stones, the compression of stone domains by

[1]Please refer to Appendix A for a map of Permafrost dispersal in the Northern Hemisphere.

expansion of the adjacent soil during freezing and lastly, gravitational redistribution of heaved stones.

Patterned ground is one of the most conspicuous features of Periglacial environments. It can range from structures of a few cm across to structures that are 100 m across or more! These patterns *may* occur in other climates but are most prevalent in surfaces which are subject to intense Frost action. 5 basic Patterned ground types are recognised:

Circles Occur singly or in groups. Typical dimensions $0.5 - 3\ m$. Non-sorted types rimmed by vegetation. Sorted types rimmed by stones which tend to increase in size with size of circle.

Figure 5.2: Stone Circles in Spitzbergen, Finland.

Polygons Occur in groups. Non-sorted polygons range from $1 - 100$ m across. Sorted polygons attain a maximum diameter of 10 m.

Figure 5.3: Polygons in Spitzbergen, Finland.

Nets A transitional form between circles and polygons. Usually small, < 2 m across.

Figure 5.4: Sorted Stone Nets in Nunavut, Canada.

Steps Found on relatively steep slopes. They develop either parallel to the slope contours or become elongated downslope into Lobate forms.

Figure 5.5: Sorted Stone Steps in Montana, USA.

Stripes Tend to form on slopes steeper than steps. Sorted stripes composed of alternating stripes of coarse and fine material.

Figure 5.6: Sorted Stone Stripes in Montana, USA.

5.2 Frost Heave

Frost heave contributes greatly to patterned ground. In soils that are variable in coarseness, ice segregations will form most effectively in the finer material. Frost heave will then be irregular forming a hummocky surface with many small *Frost Mounds*. In many instances, ground ice will form beneath the larger stones which lose heat more rapidly by conduction, so that these are raised relative to the surrounding finer particles. Over a period of time, there will be a high degree of vertical sorting as larger particles are translated to the surface and the finer particles are washed down by melt-water.

5.2.1 Simple Frost Heave model

The **MD** simulation was used to create a basic model that mimics the action of Frost heave in an effort to:

- Demonstrate that large particles can be translated vertically to the surface by Frost heave.
- Determine how these large particles advance.
- Discover if particle diameter causes different translation patterns.

Since no Ice Wedges or interstitial fluid such as water was present in our model, we conceived a simple way to mimic the behaviour of soil and stones subject to frost heave.
Soil at the bottom of the active layer is pushed upwards by Ice. Ice is also present in the space between particles including the space between the neighbour particle immediately above the lowest particle.
The lowest soil particle, dubbed the i^{th} particle, experiences a heave of Δy_i relative to the Permafrost table and the neighbour immediately above experiences a linear combination of heaves, $\Delta y_i + \Delta y_{i+1}$. Other soil particles above experience a similar linear combination of heaves so that the nearer a particle is to the surface, the greater the heave experienced during a Freeze-Thaw cycle. The diagram overleaf shows this simplified assumption:

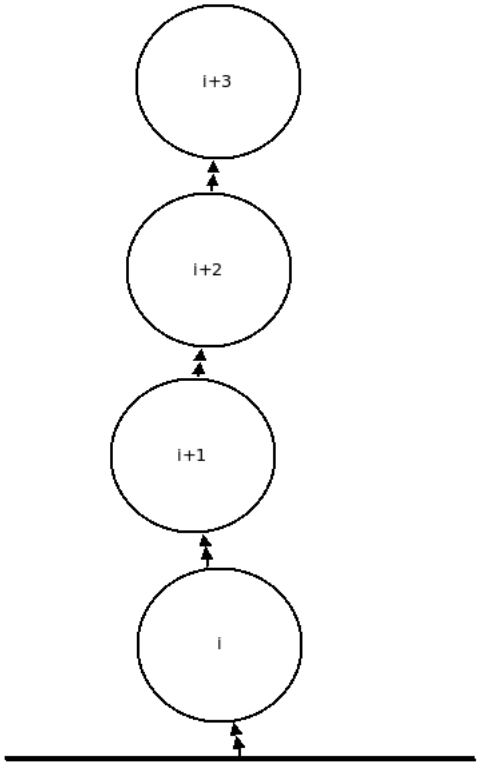


Figure 5.7: A linear combination of heaving, relative to the Permafrost Table

To acheive this in the model, during a freezing cycle each free particles $y-$coordinate was multiplied by a *Heave Factor*, $\delta_H > 1$. This ensured that the lowest particles experienced the the lowest amount of heaving whilst the particles closest to the surface experienced the greatest.

A freezing cycle was triggered in the model when the system Kinetic Energy was less than a user-defined tolerance, $E_k < \epsilon$. When $E_k < \epsilon$ the particles were deemed to be at rest and in a stable configuration. When triggered, the freezing cycle altered the $y-$coordinate then immediately switched back to a thawing cycle. This was repeated until a user-defined maximum time was reached.

5.2.2 Results

Shown below is a series of snapshots taken at regular intervals during a frost heave simulation.

The simulation began with a soil sample of 3885 particles in a window box geometry. The average radii of the soil particles is $\bar{R} = 0.005\ mm$ which corresponds to Silt. The range of the silt particle radii was $\bar{R} \pm 0.001\ mm$. 3 of the particles had radii which are greater than $\bar{R}$. These are referred to as stone particles. Their initial resting positions were on or near the Permafrost table in the active layer, spread across its horizontal range:

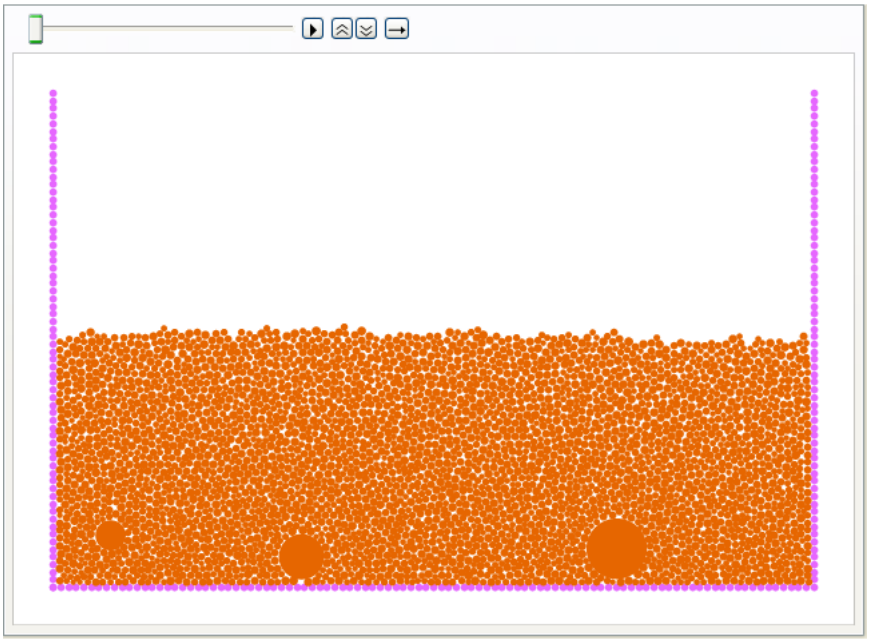


Figure 5.8: Initial configuration.

The trajectories of the 3 stone particles and 2 silt particles were tracked during the whole simulation. We wished to observe how the stone and silt particles moved during frost heaving. One silt particle was chosen from the bottom of the active layer and one was chosen from the top. The heaving factor used was, $\delta_H = 1.5$ and the value of $\epsilon = 10^{-6}$.

The snapshot below shows the positions of the stones after a few freeze-thaw cycles.
Clearly the 3 stones have moved upwards but by varying amounts. The largest stone has been translated vertically by the most. The smallest stone has only been shifted upwards by a small amount. All 3 stones have experienced a lateral displacement but in two different directions and appear to be converging to a point.

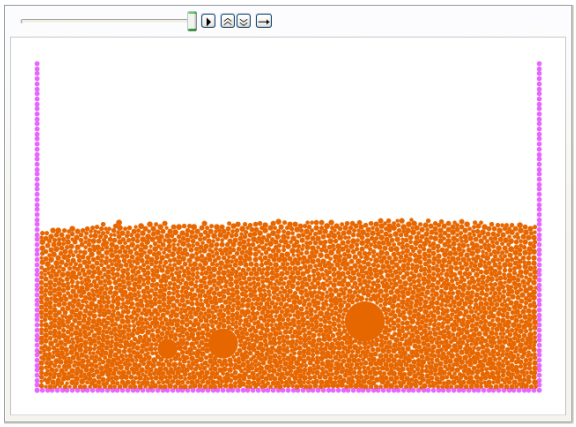


Figure 5.9: After a few Freeze-Thaw Cycles.

Snapshot no.3 shows *slightly* different behaviour. Again all 3 stones are translated vertically upwards and laterally. Again the amount of vertical translation is proportional to the stone diameter. However, instead of all 3 stones converging and moving upwards to the surface the largest particle has inverted the direction of its lateral movement while the smaller stones have slowed down the pace of theirs but kept the direction the same.

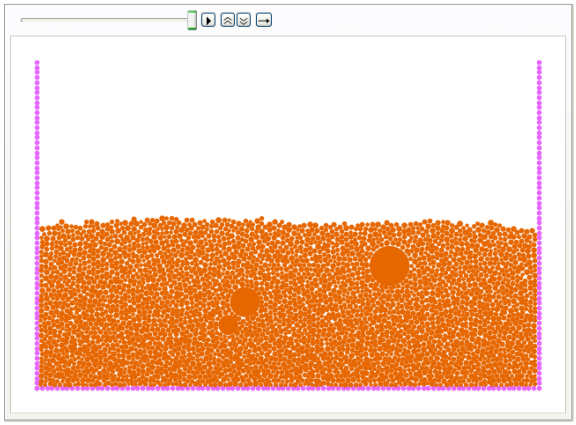


Figure 5.10: Slightly different behaviour.

Snapshot no.4 shows the first emergence of the largest stone at the surface. The 2 smaller stones are still being translated upwards and laterally but yet again, the rates of these translations are small and evidently decreasing.

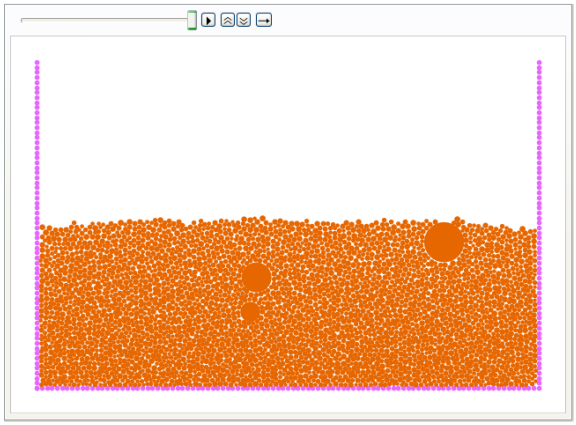


Figure 5.11: First Emergence at the Surface.

The final snapshot shows that the largest stone has receded back into the active layer entirely. Its lateral motion is still the same direction yet its vertical motion is strictly *downwards*.
The 2 smaller stones are still moving upwards but, have now switched their lateral motion to the opposing direction. It is interesting to note that the smaller stone appears to be following in the wake of the medium stone.

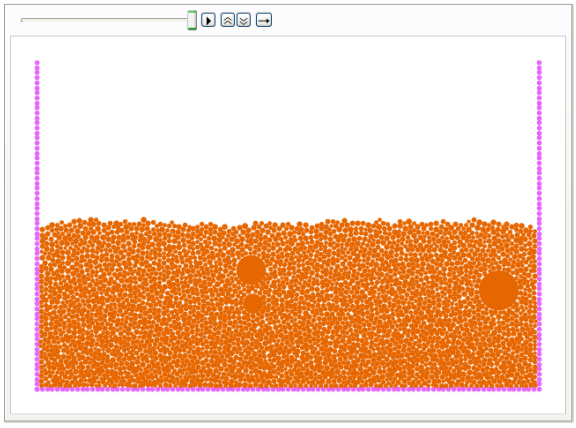


Figure 5.12: Recession into the Active Layer.

Shown below is a Low-res graph[2] which displays the tracked motion of the 3 stones and the 2 silt particles during the entire series of freeze-thaw cycles.

The spike in each trajectory can be simply explained as the vertical translation of the $y-$coordinate of each particle during one particular freezing cycle.

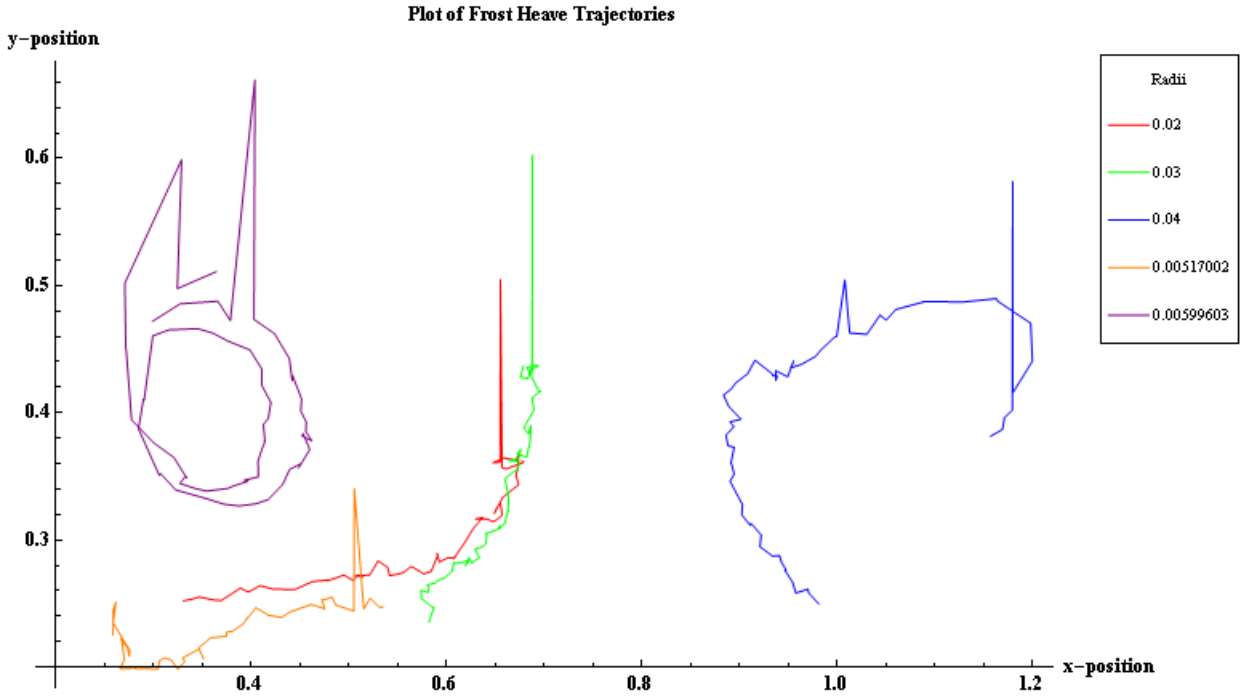


Figure 5.13: Low-res Particle Trajectories

[2]Plese refer to Appendix B to see a High-res map of the same simulation.

5.2.3 Discussion & Conclusions

Discussion

Overall, it appears that there are two opposing 'convection currents' in the soil sample. A clockwise current on the right and a counter-clockwise current on the left. Interestingly, the more massive a particle is, say the large stone (shown in the low-res graph as a blue trail), the greater it is influenced by by a current. The large stone is most certainly swept upwards strongly then pushed back down again in a clockwise motion. The medium & smallest stones are also swept up in a current but to a weaker degree in the counter-clockwise direction. The silt particle at the top travels in a counter-clockwise motion but is at closer to the center of the convection current. Its trajectory is smaller but it is trapped in a cycle and ultimately stays near the top of the soil sample. The silt particle at the bottom is primarily pushed downwards, following the counter-clockwise flow, then translated to the right and slightly upwards. It is highly possible that it may have moved to the top, following the current.

Such behaviour contrasts with convection currents in a liquid, where the lightest particles particles feel the effects of the current more strongly.
The Granular convection current behaviour, however, can be explained. The simple Frost heave model we created acts upon the particles centre of mass coordinates in a manner which is similar to vertically vibrating them. Only the $y-$component is altered explicitly and the $x-$component is altered due to collisions and friction. When rapidly translated upwards, the packing becomes quite compressed for a short time. Upward motion is opposed by tangential friction with wall particles. When the free particles begin to move downwards under gravity, the packing becomes less dense and the tangential friction decreases. Overall, the net downward motion of the free particles is much greater than the net upward motion, as investigated in Section (4.2) and the literature. The downward moving free particles near the walls push the bottom most free particles horizontally. The direction of the pushing is away from the nearest vertical wall, as the particles cannot pass through it and can only move in one other horizontal direction. In the window box geometry, two opposing convection currents are formed from the same mechanism. This explains why the currents are fastest near the vertical walls and horizontal walls, due to tangential friction and pushing respectively, and why the currents have different directions. When the particles from opposing currents meet, they are pushed upwards (away from the lower horizontal wall) and their velocities begin to decrease and synchronise due to friction.

Conclusions

These results were unexpected, as were were trying to mimic frost-heave action in Periglacial soils and not granular convection currents. However the results are exciting. Clearly the model is not a very strong candidate for Periglacial modeling in its current form as stones do not stay on the surface when they breach it. It is possible that this convection current process does actually take place during frost-heave but much more powerful and detailed simulations,[Kes03], have ruled out aqueous convection currents in the soil as a probable cause of patterned ground. Clearly a more dedicated model needs to be constructed, with the presence of Ice wedges and unfrozen water.

From the results of the simulation though, we can draw up some conclusions about granular convection currents:

- Granular convection currents can form in a *weakly* vertically perturbed granular media.
- The closer a particle is to a Vertical wall or a corner, the faster it moves. Motion is greatest in the downward direction and away from the corner.
- Further away from the walls, nearer to the center where the currents meet, the velocity appears to be much slower and synchronised.
- Large particles are translated vertically upwards at a greater rate. Particles which are just larger than the average diameter also are moved upwards but at a slower rate. They can sink back down somewhat amongst the smallest particles when allowed to fall downwards.
- Due to the convection currents, large particles can be dragged back down if the size of the convection channel is wider than the particle. A stenotic relationship exists between the channel width and the system width.

Chapter 6

Critical Discussion of the Model & Future Work

6.0.4 Critical Discussion of the Model

A numerical model of a granular media was constructed to study the properties of such a collection of particles in differing geometries and with energy applied to the particles in differnt ways. An efficient force summation technique was implemented to increase the system size and the time taken to run a simulation. The model was tested for robustness by obeying the conservation of total energy and linear momentum, before simulations were carried out. The model was extended to create a very simple Periglacial model of soil. The model has a very simple collision detection system, if the distance between particles is less than the sum of their radii then they are in contact and long range interactions are not included. This allows for simulations with thousands of particles to be carried out at a low computation cost. Material properties may be varied and studied and different geometries may be constructed. Particle types may be created and adapted into the model also. The choice of output can be of high detail or low detail. However there are some drawbacks to this model.

For small relative particle velocities, the tangential force vanishes. Simulation of static packings such as sandpiles or clogged hoppers are not suited to this model. There are force laws which do give very good results of static friction but are not used here as the study was on the dynamic behaviour of granular media.

Thee is a well founded theory behind the normal force law but no the tangential force law. Choosing the correct values of tangential material parmeters is only achieved by fine tuning the values to match results of real world experiments.

6.0.5 Future Work

Further Investigation of Phenomena

The simulations presented here are small studies into different granular phenomena. Each of these is an exciting field of study and by developing this software further to suit a chosen topic, much more interesting results will be yielded. Varaitions on the angle of the hour glass walls and orifice width may lead to different velocity profiles. Previous simulations have found a critical angle associated with granular flow from a hopper/hour glass, it would be very easy to test this with the developed software. The distribution of forces in a granular media is know to follow an exponential decay, above the average force in the system. The distribution of forces below the average force is known to follow a poer law distribution. By increasing the system size the value of the coeficients of these distribution laws may be resolved. Also, some study into the effect of slow driven compaction of a granular system and horizontal shearing is possible with the software.

Extension to 3 Dimensions

The developed software runs simulations in 2D. By extending the model to 3D we wish to observe if the same phenomena explored in the report are also present in a 3D sim, or are just restricted to 2D. By moving to 3 dimensions, the software will be much more realistic.

Parallel Code

Although the Verlet List algorithm resulted in shorter simulation times, one processor simulating ~ 4000 particles still required in excess of 4 days to evelove the particles through time until they came to rest at a required minimum energy. It is felt that by distributing the particles across a cluster of PC's using the Message Passing Interface (MPI), the time taken could be furthere reduced. Some work has done during this thesis on MPI Granular Simulation, with the creation of a new $\mathrm{MPI}_{D}ata_{t}ypetorepresenttheparticles$.

Periglacial Model

Finally, the developed software does not only have to be used to simulate non-cohesive granular media. A working Periglacial model which includes frozen and unfrozen water with more well defined freezing and thawing cycles can be created

by using this software as a base system. This area is very interesting and with a 3D model running MPI code the model could be very powerful.

Appendix A

Permafrost Map

This appendix shows the distribution of continuous and discontinuous Permafrost in the Northern Hemisphere.

Figure A.1: Permafrost in the Northern Hemisphere.

Appendix B

High-res Trajectory Graph

The appendix displays the High-res graph of the trajectories, including the peaks caused by the freezing cycles during a frost heave simulation.

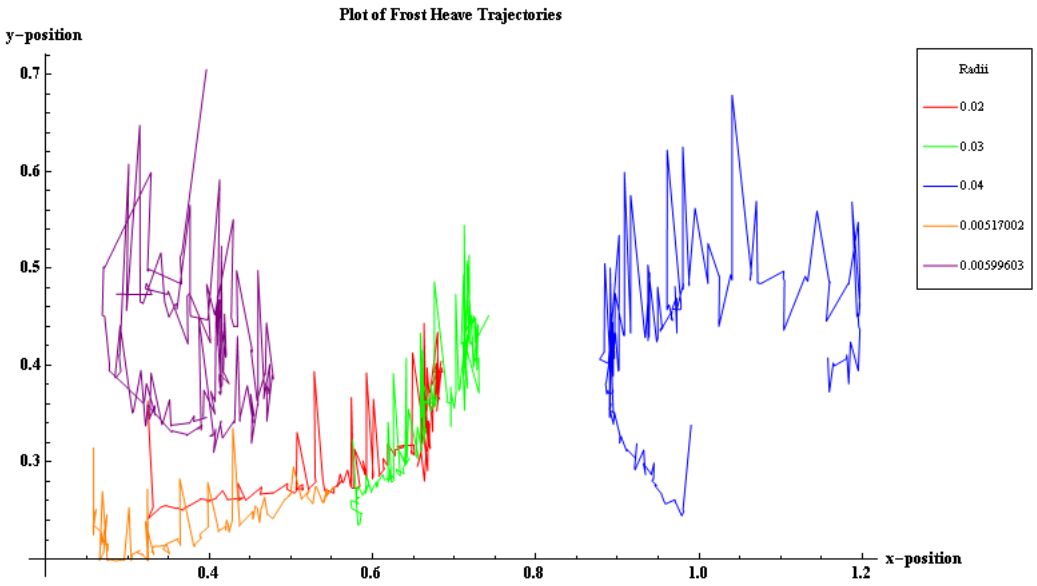


Figure B.1: High-res Particle Trajectories.

Appendix C

Visualisation

This appendix shows the Mathematica code used to visualise the results of a simulation and how to export it to Flash (*.swf) format.

Change the working directory to that of the input and output files and load the data:

Execute this code block to run in Unix, at home:

```
SetDirectory["//home//bryan//Desktop//Dropbox//Thesis//New_Work//Week13//Code//C++//Output_Files//Pc-Lab"];
data = Import["PC-Lab01_frost_heave01.out"];
energy = Import["Output\\eng2.out"];
linmmn = Import["Output\\lmn2.out"];
angmmn = Import["Output\\amn2.out"];
coord = Import["Output\\cdn.out", "Table"];
particles = Take[data[[1]]];
data = Drop[data, 5];
```

Plot the average Coordination Number of the System:

```
ZPlot = Table[coord[[i, 1]], {i, 1, Length[coord] − 1}];
ListPlot[ZPlot,
PlotLabel → "Plot of the Average Coordination Number of the System, Z, versus the Number of Itterations, t",
AxesLabel->{"Itterations", "Z"}, PlotRange → All]
```

Plot the Total Energy of the System:

```
kinetic = Table[energy[[i, 1]], {i, 1, Length[energy]}];
potential = Table[energy[[i, 2]], {i, 1, Length[energy]}];
total = Table[energy[[i, 3]], {i, 1, Length[energy]}];
Show[ListLinePlot[kinetic, PlotStyle → {Blue}, PlotRange → All],
ListLinePlot[potential, PlotStyle → {Red}, PlotRange → All],
ListLinePlot[total, PlotStyle → {Green}, PlotRange → All],
PlotLabel → "Plot Of Kinetic v Potential Energy"]
```

Plot the Total Linear Momentum of the System:

```
particle1 = Table[linmmn[[i, 1]], {i, 1, Length[linmmn]}];
particle2 = Table[linmmn[[i, 2]], {i, 1, Length[linmmn]}];
total = Table[linmmn[[i, 3]], {i, 1, Length[linmmn]}];
Show[ListLinePlot[particle1, PlotStyle → {Blue}, PlotRange → All],
ListLinePlot[particle2, PlotStyle → {Red}, PlotRange → All],
```

```
ListLinePlot[total, PlotStyle → {Green}, PlotRange → All],
PlotLabel → "Plot Of the System Linear Momentum"]
```

Plot the Total Angular Momentum of the System:

```
particle1 = Table[angmmn[[i, 1]], {i, 1, Length[angmmn]}];
particle2 = Table[angmmn[[i, 2]], {i, 1, Length[angmmn]}];
total = Table[angmmn[[i, 3]], {i, 1, Length[angmmn]}];
Show[ListLinePlot[particle1, PlotStyle → {Blue}, PlotRange → All],
ListLinePlot[particle2, PlotStyle → {Red}, PlotRange → All],
ListLinePlot[total, PlotStyle → {Green}, PlotRange → All],
PlotLabel → "Plot Of the System Angular Momentum"]
```

Plot the Particle Trajectories and Rotation:

These are the Graphics functions:

```
fps = 5;
top = 0.8;
half = (top − bottom)/2. + bottom;
bottom = 0.1;
bottomleft = 0.15;
bottomright = 1.3;

DrawParticle[ls_, k_, l_]:=Show[Graphics[Table[{{RGBColor[0.9, .4, ls[[j, 9]]],
Disk[{ls[[j, 1]], ls[[j, 2]]}, ls[[j, 8]]]}},
{j, k, l ∗ particles[[1]]}]]];

DrawSmallFrame:=Graphics[{Line[{{bottomleft, bottom}, {bottomleft, top}}],
Line[{{bottomright, bottom}, {bottomright, top}}],
Line[{{bottomleft, bottom}, {bottomright, bottom}}],
Line[{{bottomleft, top}, {bottomright, top}}],
Line[{{bottomleft, half}, {bottomright, half}}]}];

DrawLargeFrame:=Graphics[{Line[{{−25, 0}, {−25, 25}}], Line[{{25, 0}, {25, 25}}],
Line[{{−25, 0}, {25, 0}}], Line[{{−25, 25}, {25, 25}}],
Line[{{−25, 12.5}, {25, 12.5}}]}];

DrawForce[ls_, pj_, i_]:=Graphics[{Red, Thickness[0.01Fx],
Line[{{ls[[i, 1]], ls[[i, 2]]}, {ls[[i, 3]], pj[[i, 4]]}}]}];

DrawRotation[ls_, k_, l_]:=Show[Graphics[Table[{Black, Thickness[.0005],
Line[{{ls[[j, 1]], ls[[j, 2]]}, {ls[[j, 1]] + ls[[j, 8]]Cos[ls[[j, 3]]], ls[[j, 2]] + ls[[j, 8]]Sin[ls[[j, 3]]]}}],
{j, k, l ∗ particles[[1]]}]]];
```

Graphing Section:

```
movie = {};
k = l = 1;
For[j = 1, j < (Length[data]/particles[[1]]), j++;
movie = Append[movie, Show[{DrawParticle[data, k, l], (*DrawRotation[data, k, l], *)DrawSmallFrame},
ImageSize → {600}]];
l+=1;
k+=particles[[1]];
];
ListAnimate[movie, AnimationRunning → False, Alignment → {Center}, DisplayAllSteps → True]
```

Export to *.swf format:

```
Export["movie.swf", movie];
```

Appendix D

Hopper

This is the code used to Generate a hopper geometry with 400 free particles.

```
#include<fstream>
#include<stdlib.h>
#include<stdio.h>
#include<time.h>

//A function which writes the particle information to file
void dump_particle(ostream & os, double x, double y, double phi, double vx, double vy,
double omega, double mass, double radius, int type, double mu,
double gamma, double Y, double A){
os << x << "\t" << y << "\t" << phi << "\t" << vx << "\t" << vy << "\t" << omega
 << "\t" << mass << "\t" << radius << "\t" << type << "\t" << mu << "\t" << gamma
 << "\t" << Y << "\t" << A << "\n";
}
int main(){
ofstream fout ("Input/large_hopper.in");

fout << 1711 << "\n" << 0 << "\n" << 5 << "\n" << 0.000001 << "\n"
<< 0 << "\t" << -9.81 << "\t" << 0 << "\n";

//Base of the Hopper
for(int i = 0; i < 11; i++){
dump_particle(fout,0.45+i*0.01,0.19,0,0,0,0,1,0.005,1,0.5,10,0.0003,0.01);
}

//Walls of the Hopper
for(int i = 0; i < 50; i++){
dump_particle(fout,0.55+(i+0.5)*0.005,i*0.01+0.2,0,0,0,0,1,0.005,1,0.5,10,100000,0.01);
dump_particle(fout,0.45-(i+0.5)*0.005,i*0.01+0.2,0,0,0,0,1,0.005,1,0.5,10,100000,0.01);
}

double Rmax = 0.006;
double Rmin = 0.004;

//Particles above the hopper
for(int i = 0; i < 20; i++){
for(int k = 0; k < 20; k++){
double centerx = 0.35+0.013*i;
double centery = 0.65+0.013*k;
double r = Rmin + rand() * (Rmax - Rmin) / RAND_MAX;
dump_particle(fout,centerx,centery,0,0,0,0,(r*r/(Rmax*Rmax)),r,0,0.5,10,100000,0.01);
}
}
fout.close();

}
```

Appendix E

Granular Flow

This appendix shows the Mathematica code used to visualise the trajectories and velocities of particles that exhibit granular flow.

Change the working directory to that of the input and output files and load the data:

```
Off[Part::partw];
Off[SetDirectory::cdir];
SetDirectory["//home//bryan//Desktop//Dropbox//Thesis//New_Work//Week15//Code//C++//Output_Files//PC-Lab01"];
data = Import["hour_glass01.out", "Table"];
particles = Take[data[[1]]];
data = Drop[data, 5];
```

These are the lists which will hold the centre of mass coordinates and the y-velocities of the tracked particles. If you wish to stich multiple runs together, then only execute this block once at the beginning of the notebook.

```
listLeft = {};
listLeftVel = {};
listCentre = {};
listCentreVel = {};
listRight = {};
listRightVel = {};
```

Gather the centre of mass coordinates for the particles and the y-velocities. By changing the initial j values you can track different particles. You may also add more particles to track instead of just 5. Then plot the data on the same graph.

```
pLeft = 1304;
pCentre = 1535;
pRight = 1298;
For[j = pLeft, j < Length[data],
listLeft = Append[listLeft, {data[[j, 1]], data[[j, 2]]}];
```

```
listLeftVel = Append[listLeftVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r1 = data[[pLeft, 8]];

For[j = pCentre, j < Length[data],
listCentre = Append[listCentre, {data[[j, 1]], data[[j, 2]]}];
listCentreVel = Append[listCentreVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r2 = data[[pCentre, 8]];

For[j = pRight, j < Length[data],
listRight = Append[listRight, {data[[j, 1]], data[[j, 2]]}];
listRightVel = Append[listRightVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r3 = data[[pRight, 8]];

max = Max[Abs[listCentreVel]];
listLeft = Table[listLeft[[i]], {i, 1, Length[listLeft] - 1}];
listLeftVel = Table[listLeftVel[[i, 2]], {i, 1, Length[listLeftVel] - 1}];
plot1 = ListLinePlot[listLeft, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];
plot1vel = ListLinePlot[listLeftVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

listCentre = Table[listCentre[[i]], {i, 1, Length[listCentre] - 1}];
listCentreVel = Table[listCentreVel[[i, 2]], {i, 1, Length[listCentreVel] - 1}];
plot2 = ListLinePlot[listCentre, PlotRange → All,
ImageSize → {800}, PlotStyle → {Green}];
plot2vel = ListLinePlot[listCentreVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

listRight = Table[listRight[[i]], {i, 1, Length[listRight] - 1}];
listRightVel = Table[listRightVel[[i, 2]], {i, 1, Length[listRightVel] - 1}];
plot3 = ListLinePlot[listRight, PlotRange → All,
ImageSize → {800}, PlotStyle → {Blue}];
plot3vel = ListLinePlot[listRightVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

Needs["PlotLegends"];
DataPlot = ListPlot[{listLeft, listCentre, listRight},
Joined → True,
AxesOrigin->{0.4, 0.2},
PlotRange → All,
AspectRatio → 1,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium],
PlotLabel → "Plot of Hour Glass Trajectories",
PlotMarkers → {" ● ", " ● ", " ● "},
PlotStyle → {Red, Green, Blue},
PlotLegend → {"Left", "Centre", "Right"},
LegendLabel → "Particles Position",
LegendShadow → None,
LegendSize → 0.5,
LegendPosition → {0.9, −0.0},
ImageSize->{800}
]

DataPlot = ListPlot[{listLeftVel, listCentreVel, listRightVel},
Joined → True,
AxesOrigin->{0.0, 0.0},
DataRange->{0, 20},
PlotRange → All,
AspectRatio → 1,
AxesLabel->{"Snapshot", "y-velocity"},
LabelStyle → Directive[Black, Bold, Medium],
PlotLabel → "Plot of Hour Glass Velocities",
PlotMarkers → {" ● ", " ● ", " ● "},
PlotStyle → {Red, Green, Blue},
PlotLegend → {"Left", "Centre", "Right"},
LegendLabel → "Particles Velocities",
LegendShadow → None,
LegendSize → 0.5,
LegendPosition → {0.9, −0.0},
```

```
ImageSize->{800}
]
```

These are the lists which will hold the centre of mass coordinates and the y-velocities of the tracked particles but at larger time intervals and create a clearer plot of the trajectories.
If you wish to stich multiple runs together,
then only execute this block once at the beginning of the notebook.

```
listLeftInterval = Table[listLeft[[i]], {i, 1, Length[listLeft] − 1, 1}];
listCentreInterval = Table[listCentre[[i]], {i, 1, Length[listCentre] − 1, 1}];
listRightInterval = Table[listRight[[i]], {i, 1, Length[listRight] − 1, 1}];
listLeftIntervalPlot = Graphics[{Red, Arrow[listLeftInterval]}];
listCentreIntervalPlot = Graphics[{Green, Arrow[listCentreInterval]}];
listRightIntervalPlot = Graphics[{Blue, Arrow[listRightInterval]}];
Needs["PlotLegends"];
topleft = 0.2025;
topright = 0.7975;
bottomleft = 0.2025;
bottomright = 0.7975;
top = 0.69;
bottom = −0.31;
half = 0.19;
halfleft = 0.45;
halfright = 0.55;
DrawSmallFrame:=Graphics[{Line[{{bottomleft, bottom}, {bottomright, bottom}}],
Line[{{topleft, top}, {topright, top}}],
Line[{{topright, top}, {halfright, half}}],
Line[{{topleft, top}, {halfleft, half}}],
Line[{{bottomright, bottom}, {halfright, half}}],
Line[{{bottomleft, bottom}, {halfleft, half}}]}];
DataPlot = Show[{listLeftIntervalPlot, listCentreIntervalPlot, listRightIntervalPlot, DrawSmallFrame},
ImageSize->{800}
]
```

Create a Manipulate object of the particle trajectories over time.

```
Manipulate[
listA = {};
listB = {};
listC = {};
listA = Table[listLeft[[i]], {i, 1, Steps}];
listB = Table[listCentre[[i]], {i, 1, Steps}];
listC = Table[listRight[[i]], {i, 1, Steps}];
plotA = ListLinePlot[listA, PlotRange → All,
ImageSize → {600}, PlotStyle → {Red}, DataRange → {.2, 1.3}];
plotB = ListLinePlot[listB, PlotRange → All,
ImageSize → {600}, PlotStyle → {Green}, DataRange → {.2, 1.3}];
plotC = ListLinePlot[listC, PlotRange → All,
ImageSize → {600}, PlotStyle → {Blue}, DataRange → {.2, 1.3}];
Show[plotA, plotB, plotC,
PlotLabel → "Plot of Hour Glass Trajectories",
AxesOrigin → {0.2, 0.2}, PlotRange → All,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium]]
, {Steps, 1, Length[listLeft], 1}]
```

Appendix F

Size Segregation

This appendix shows the Mathematica code used to visualise the trajectories of particles that exhibit size segregation.

Change the working directory to that of

the input and output files and load the data:

```
Off[Part::partw];
Off[SetDirectory::cdir];
SetDirectory["//home//bryan//Desktop//Dropbox//Thesis//New_Work
//Week15//Code//C++//Output_Files//PC-Lab02//Silo"];
data = Import["PC-Lab02_silo60.out", "Table"];
particles = Take[data[[1]]];
data = Drop[data, 5];
```

These are the lists which will hold the centre of mass coordinates

and the y-velocities of the tracked particles.

If you wish to stich multiple runs together,

then only execute this block once at the beginning of the notebook.

```
listLeft = {};
listLeftVel = {};
listCentre = {};
listCentreVel = {};
listRight = {};
listRightVel = {};
```

Gather the centre of mass coordinates for the particles

and the y-velocities.

By changing the initial j values you can track different particles.

You may also add more particles to track instead of just 5.

Then plot the data on the same graph.

```
pLeft = 175;
pCentre = 375;
pRight = 533;
```

```
For[j = pLeft, j < Length[data],
listLeft = Append[listLeft, {data[[j, 1]], data[[j, 2]]}];
listLeftVel = Append[listLeftVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r1 = data[[pLeft, 8]];

For[j = pCentre, j < Length[data],
listCentre = Append[listCentre, {data[[j, 1]], data[[j, 2]]}];
listCentreVel = Append[listCentreVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r2 = data[[pCentre, 8]];

For[j = pRight, j < Length[data],
listRight = Append[listRight, {data[[j, 1]], data[[j, 2]]}];
listRightVel = Append[listRightVel, {data[[j, 4]], data[[j, 5]]}],
j+=particles[[1]]
];
r3 = data[[pRight, 8]];

max = Max[Abs[listCentreVel]];
listLeft = Table[listLeft[[i]], {i, 1, Length[listLeft] - 1}];
listLeftVel = Table[listLeftVel[[i, 2]], {i, 1, Length[listLeftVel] - 1}];
plot1 = ListLinePlot[listLeft, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];
plot1vel = ListLinePlot[listLeftVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

listCentre = Table[listCentre[[i]], {i, 1, Length[listCentre] - 1}];
listCentreVel = Table[listCentreVel[[i, 2]], {i, 1, Length[listCentreVel] - 1}];
plot2 = ListLinePlot[listCentre, PlotRange → All,
ImageSize → {800}, PlotStyle → {Green}];
plot2vel = ListLinePlot[listCentreVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

listRight = Table[listRight[[i]], {i, 1, Length[listRight] - 1}];
listRightVel = Table[listRightVel[[i, 2]], {i, 1, Length[listRightVel] - 1}];
plot3 = ListLinePlot[listRight, PlotRange → All,
ImageSize → {800}, PlotStyle → {Blue}];
plot3vel = ListLinePlot[listRightVel, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

Needs["PlotLegends`"];
DataPlot = ListPlot[{listLeft, listCentre, listRight},
Joined → True,
AxesOrigin->{0.25, 0.2},
PlotRange → All,
AspectRatio → 1,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium],
PlotLabel → "Plot of Hour Glass Trajectories",
PlotMarkers → {" ● ", " ● ", " ● "},
PlotStyle → {Red, Green, Blue},
PlotLegend → {"Left", "Centre", "Right"},
LegendLabel → "Particles Position",
LegendShadow → None,
LegendSize → 0.5,
LegendPosition → {0.9, -0.0},
ImageSize->{800}
]

DataPlot = ListPlot[{listLeftVel, listCentreVel, listRightVel},
Joined → True,
AxesOrigin->{0.2, 0.0},
(*DataRange->{0, 20}, *)
PlotRange → All,
AspectRatio → 1,
AxesLabel->{"Snapshot", "y-velocity"},
LabelStyle → Directive[Black, Bold, Medium],
PlotLabel → "Plot of Hour Glass Velocities",
PlotMarkers → {" ● ", " ● ", " ● "},
PlotStyle → {Red, Green, Blue},
PlotLegend → {"Left", "Centre", "Right"},
LegendLabel → "Particles Velocities",
LegendShadow → None,
```

```
LegendSize → 0.5,
LegendPosition → {0.9, −0.0},
ImageSize->{800}
];
```

These are the lists which will hold the centre of mass coordinates and the y-velocities of the tracked particles but at larger time intervals and create a clearer plot of the trajectories.

If you wish to stich multiple runs together,

then only execute this block once at the beginning of the notebook.

```
listLeftInterval = Table[listLeft[[i]], {i, 1, Length[listLeft] − 1, 40}];
listCentreInterval = Table[listCentre[[i]], {i, 1, Length[listCentre] − 1, 40}];
listRightInterval = Table[listRight[[i]], {i, 1, Length[listRight] − 1, 40}];
listLeftIntervalPlot = Graphics[{Red, Arrow[listLeftInterval]}];
listCentreIntervalPlot = Graphics[{Green, Arrow[listCentreInterval]}];
listRightIntervalPlot = Graphics[{Blue, Arrow[listRightInterval]}];
Needs["PlotLegends`"];
bottomleft = 0.25;
bottomright = 0.42;
top = 0.78;
bottom = 0.19;
DrawSmallFrame:=Graphics[{Line[{{bottomleft, bottom}, {bottomleft, top}}],
Line[{{bottomright, bottom}, {bottomright, top}}],
Line[{{bottomleft, bottom}, {bottomright, bottom}}],
Line[{{bottomleft, top}, {bottomright, top}}]}];
DataPlot = Show[{listLeftIntervalPlot, listCentreIntervalPlot, listRightIntervalPlot, DrawSmallFrame},
ImageSize->{800}
]
```

Create a Manipulate object of the particle trajectories over time.

```
Manipulate[
listA = {};
listB = {};
listC = {};
listA = Table[listLeft[[i]], {i, 1, Steps}];
listB = Table[listCentre[[i]], {i, 1, Steps}];
listC = Table[listRight[[i]], {i, 1, Steps}];
plotA = ListLinePlot[listA, PlotRange → All,
ImageSize → {600}, PlotStyle → {Red}, DataRange → {.2, 1.3}];
plotB = ListLinePlot[listB, PlotRange → All,
ImageSize → {600}, PlotStyle → {Green}, DataRange → {.2, 1.3}];
plotC = ListLinePlot[listC, PlotRange → All,
ImageSize → {600}, PlotStyle → {Blue}, DataRange → {.2, 1.3}];
Show[plotA, plotB, plotC,
PlotLabel → "Plot of Hour Glass Trajectories",
AxesOrigin → {0.2, 0.2}, PlotRange → All,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium]]
, {Steps, 1, Length[listLeft], 1}]
```

Appendix M

Force Chains

This appendix shows the Mathematica code used to visualise the force chain network of a system of particles.

Change the working directory to that of the input and output files and load the data:

```
SetDirectory["C:\\Documents and Settings\\SOMPG07\\My Documents\\Visual Studio 2010\\Projects\\print_chains\\Debug\\Output"];
chains = Import["chains.out", "Table"];
SetDirectory["C:\\Documents and Settings\\SOMPG07\\My Documents\\Visual Studio 2010\\Projects\\print_chains\\Debug\\Input"];
particles = Import["PC-Lab02_silo.dat", "Table"];
particles = Drop[particles, 5];
```

Plot the Force Chain Network and the Particles:

These are the Graphing functions:

```
top = 0.8;
half = (top − bottom)/2. + bottom;
bottom = 0.15;
bottomleft = 0.2;
bottomright = 0.48;

DrawSmallFrame:=Graphics[{Line[{{bottomleft, bottom}, {bottomleft, top}}],
Line[{{bottomright, bottom}, {bottomright, top}}],
Line[{{bottomleft, bottom}, {bottomright, bottom}}],
Line[{{bottomleft, top}, {bottomright, top}}],
Line[{{bottomleft, half}, {bottomright, half}}]}];

DrawParticle[ls_, k_]:=Show[Graphics[Table[{{RGBColor[0.9, .4, ls[[j, 9]]],
Disk[{ls[[j, 1]], ls[[j, 2]]}, ls[[j, 8]]]}},
{j, 1, k}]]];

DrawForce[ls_, k_]:=Show[Graphics[Table[{Black, Thickness[(ls[[j, 11]]/ max) * 0.005],
Line[{{ls[[j, 3]], ls[[j, 4]]}, {ls[[j, 5]], ls[[j, 6]]}}]}, {j, 1, k}]],
ImageSize → {600}];

max = chains[[1, 11]];
```

Graphing Section:

```
For[i = 2, i < Length[chains], i++;
If[chains[[i, 11]] > max,
max = Max[chains[[i, 11]]];
]
```

```
]
Show[DrawParticle[particles, Length[particles]], DrawForce[chains, Length[chains]], DrawSmallFrame, ImageSize → {800}]
```

Appendix N

Frost Heave

This appendix shows the Mathematica code used to visualise the trajectories of particles that undergo frost heaving.

Change the working directory to that of the input and output files and load the data:

```
Off[Part::partw];
Off[SetDirectory::cdir];
SetDirectory["//home//bryan//Desktop//Dropbox//Thesis//New_Work/
/Week15//Code//C++//Output_Files//PC-Lab05//Frost_Heave"];
data = Import["PC-Lab05_frost_heave04b.out", "Table"];
particles = Take[data[[1]]];
data = Drop[data, 5];
```

These are the lists which will hold the centre of mass coordinates of the tracked particles.

If you wish to stich multiple runs together,

then only execute this block once at the beginning of the notebook.

```
list1 = {};

list2 = {};
list3 = {};
list4 = {};
list5 = {};
```

Gather the centre of mass coordinates for the particles.

By changing the initial j values you can track different particles.

You may also add more particles to track instead of just 5.

Then plot the data on the same graph.

```
ClearAll[j]

p1 = 683;
p2 = 684;
p3 = 685;
```

```
p4 = 237;
p5 = 2701;

For[j = p1, j < Length[data],
list1 = Append[list1, {data[[j, 1]], data[[j, 2]]}],
j+=particles[[1]]
];
r1 = data[[p1, 8]];

For[j = p2, j < Length[data],
list2 = Append[list2, {data[[j, 1]], data[[j, 2]]}],
j+=particles[[1]]
];
r2 = data[[p2, 8]];

For[j = p3, j < Length[data],
list3 = Append[list3, {data[[j, 1]], data[[j, 2]]}],
j+=particles[[1]]
];
r3 = data[[p3, 8]];

For[j = p4, j < Length[data],
list4 = Append[list4, {data[[j, 1]], data[[j, 2]]}],
j+=particles[[1]]
];
r4 = data[[p4, 8]];

For[j = p5, j < Length[data],
list5 = Append[list5, {data[[j, 1]], data[[j, 2]]}],
j+=particles[[1]]
];
r5 = data[[p5, 8]];

list1 = Table[list1[[i]], {i, 1, Length[list1] - 1}];
plot1 = ListLinePlot[list1, PlotRange → All,
ImageSize → {800}, PlotStyle → {Red}];

list2 = Table[list2[[i]], {i, 1, Length[list2] - 1}];
plot2 = ListLinePlot[list2, PlotRange → All,
ImageSize → {800}, PlotStyle → {Green}];

list3 = Table[list3[[i]], {i, 1, Length[list3] - 1}];
plot3 = ListLinePlot[list3, PlotRange → All,
ImageSize → {800}, PlotStyle → {Blue}];

list4 = Table[list4[[i]], {i, 1, Length[list4] - 1}];
plot4 = ListLinePlot[list4, PlotRange → All,
ImageSize → {800}, PlotStyle → {Orange}];

list5 = Table[list5[[i]], {i, 1, Length[list5] - 1}];
plot5 = ListLinePlot[list5, PlotRange → All,
ImageSize → {800}, PlotStyle → {Purple}];

Needs["PlotLegends"];
DataPlot = ListPlot[{list1, list2, list3, list4, list5},
Joined → True,
AxesOrigin->{0.2, 0.2},
PlotRange → All,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium],
PlotLabel → "Plot of Frost Heave Trajectories",
PlotStyle → {Red, Green, Blue, Orange, Purple},
PlotLegend → {r1, r2, r3, r4, r5},
LegendLabel → "Radii",
LegendShadow → None,
LegendSize → 0.5,
LegendPosition → {0.9, -0.0},
ImageSize->{800}
]
```

These are the lists which will hold the centre of mass coordinates and the y-velocities of the tracked particles but at larger time intervals and create a clearer plot of the trajectories.
If you wish to stich multiple runs together,
then only execute this block once at the beginning of the notebook.

```
list1Interval = Table[list1[[i]], {i, 1, Length[list1] − 1, 5}];
list2Interval = Table[list2[[i]], {i, 1, Length[list2] − 1, 5}];
list3Interval = Table[list3[[i]], {i, 1, Length[list3] − 1, 5}];
list4Interval = Table[list4[[i]], {i, 1, Length[list4] − 1, 5}];
list5Interval = Table[list5[[i]], {i, 1, Length[list5] − 1, 5}];
list1IntervalPlot = Graphics[{Red, Arrow[list1Interval]}];
list2IntervalPlot = Graphics[{Green, Arrow[list2Interval]}];
list3IntervalPlot = Graphics[{Blue, Arrow[list3Interval]}];
list4IntervalPlot = Graphics[{Orange, Arrow[list4Interval]}];
list5IntervalPlot = Graphics[{Purple, Arrow[list5Interval]}];
Needs["PlotLegends`"];
bottomleft = 0.25;
bottomright = 1.25;
top = 0.84;
bottom = 0.19;
DrawSmallFrame:=Graphics[{Line[{{bottomleft, bottom}, {bottomleft, top}}],
Line[{{bottomright, bottom}, {bottomright, top}}],
Line[{{bottomleft, bottom}, {bottomright, bottom}}],
Line[{{bottomleft, top}, {bottomright, top}}]}];
DataPlot = Show[{list1IntervalPlot, list2IntervalPlot, list3IntervalPlot, list4IntervalPlot, list5IntervalPlot, DrawSmallFrame},
ImageSize->{800}
]
```

Create a Manipulate object of the particle trajectories over time.

```
Manipulate[
listA = {};
listB = {};
listC = {};
listD = {};
listE = {};
listA = Table[list1[[i]], {i, 1, Steps}];
listB = Table[list2[[i]], {i, 1, Steps}];
listC = Table[list3[[i]], {i, 1, Steps}];
listD = Table[list4[[i]], {i, 1, Steps}];
listE = Table[list5[[i]], {i, 1, Steps}];
plotA = ListLinePlot[listA, PlotRange → All,
ImageSize → {600}, PlotStyle → {Red}, DataRange → {.2, 1.3}];
plotB = ListLinePlot[listB, PlotRange → All,
ImageSize → {600}, PlotStyle → {Green}, DataRange → {.2, 1.3}];
plotC = ListLinePlot[listC, PlotRange → All,
ImageSize → {600}, PlotStyle → {Blue}, DataRange → {.2, 1.3}];
plotD = ListLinePlot[listD, PlotRange → All,
ImageSize → {600}, PlotStyle → {Orange}, DataRange → {.2, 1.3}];
plotE = ListLinePlot[listE, PlotRange → All,
ImageSize → {600}, PlotStyle → {Purple}, DataRange → {.2, 1.3}];

Show[plotA, plotB, plotC, plotD, plotE,
PlotLabel → "Plot of Frost Heave Trajectories",
AxesOrigin → {0.2, 0.2}, PlotRange → All,
AxesLabel->{"x-position", "y-position"},
LabelStyle → Directive[Black, Bold, Medium]]
, {Steps, 1, Length[list1], 1}]
```

Appendix G

Disk Class File

This is the C++ Class file that allows a user to create Objects of type disk.

```
#ifndef _disk_h
#define _disk_h
#include<iostream>
#include<fstream>
#include "Vector.h"
using namespace std;
extern Vector G;

inline double normalise(double dx, double L){
while(dx< -L/2){
dx+=L;
}
while(dx>=L/2){
dx-=L;
}
return dx;
}

class disk{
//Calculate the force between two particles
friend void force(disk & p1, disk & p2, double lx, double ly, int tag);
//Calculate the packing density of the system
friend double density(disk iArray[], double lx, double ly);
friend double avg_coordination_number(disk iArray[]);
friend double Distance(const disk & p1, const disk & p2,
 double lx, double ly){
double dx=normalise(p1.rtd0.x()-p2.rtd0.x(),lx);
double dy=normalise(p1.rtd0.y()-p2.rtd0.y(),ly);
return sqrt(dx*dx+dy*dy);
}

friend istream & operator >> (istream & is, disk & p1);
friend ostream & operator >> (ostream & os, const disk & p1);

public:
//Constructor
disk():rtd0(null),rtd1(null),rtd2(null),rtd3(null),rtd4(null){}

//Access private members
//Position vector
Vector & pos() {return rtd0;} //Access a dynamic vector
Vector pos() const {return rtd0;} //Access a static vector

//x-position
double & x() {return rtd0.x();}
```

```
double x() const {return rtd0.x();}

//y-position
double & y() {return rtd0.y();}
double y() const {return rtd0.y();}

//Angle
double & phi() {return rtd0.phi();}
double phi() const {return rtd0.phi();}

//Velocity vector
const Vector & velocity() const {return rtd1;}

//x-velocity
double & vx() {return rtd1.x();}
double vx() const {return rtd1.x();}

//y-velocity
double & vy() {return rtd1.y();}
double vy() const {return rtd1.y();}

//Angular velocity
double & omega() {return rtd1.phi();}
double omega() const {return rtd1.phi();}

//Particle properties
//Radius
double & r() {return _r;}
double r() const {return _r;}

//Mass
double & m() {return _m;}
double m() const {return _m;}

//Particle type
int & type() {return _type;}
int type() const {return _type;}

//Material Properties
double & mu() {return _mu;}
double & gamma() {return _gamma;}
double & Y() {return _Y;}
double & A() {return _A;}

//Coordination number
int & z() {return _z;}
int z() const {return _z;}

//Functions that operate on the _force vector of a particle
void predict(double dt);
void add_force(const Vector & f) {_force+=f;}
void correct(double dt);
void set_force_to_zero() {_force=null;}
void boundary_conditions(int n, double dt, double t);

//Other functions
void inc_coordination_number() {_z++;}
void reset_coordination_number() {_z=0;}
double kinetic_energy() const;
double linear_momentum() const;
double angular_momentum() const;

private:
double _r; //Radius
double _m; //Mass
double _J; //Moment of Inertia
int _z; //Coordination Number
int _type; //Type of particle
double _mu,_gamma,_Y,_A;//Material properties
```

```
Vector rtd0,rtd1,rtd2,rtd3,rtd4; //Position vector and its higher order time derivatives
Vector _force; //Force vector, has x, y and angular components
};
#endif
```

Appendix H

Vector Class File

This is the C++ Class file that allows a user to create Objects of type vector.

```
#ifndef _Vector_h
#define _Vector_h
#include<iostream>
#include<math.h>
using namespace std;

class Vector {
//Friend overloaded binary functions
friend istream & operator >> (istream & is, Vector & v){
is >> v._x >> v._y >> v._phi;
return is;
}

friend ostream & operator << (ostream & os, const Vector & v){
os << v._x << " " << v._y << " " << v._phi;
return os;
}

//Vector addition
friend Vector operator + (const Vector & v1, const Vector & v2){
Vector res(v1);
res+=v2;
return res;
}

//Vector subtraction
friend Vector operator - (const Vector & v1, const Vector & v2){
Vector res(v1);
res-=v2;
return res;
}

//Scalar multiplication
friend Vector operator * (double c, const Vector & p){
Vector res=p;
res *= c;
return res;
}

friend Vector operator * (const Vector & p, double c){
return c*p;
}

public:
//Explicit constructor
```

```
explicit Vector(double x=0, double y=0, double phi=0): _x(x), _y(y), _phi(phi){};
//explicit Vector(const Vector & ivec): _x(ivec._x),_y(ivec._y),_phi(ivec._phi){};

//Access private members
double & x(){return _x;}
double x() const {return _x;}
double & y(){return _y;}
double y() const {return _y;}
double & phi(){return _phi;}
double phi() const {return _phi;}

//Overloaded binary operators
//Addition
const Vector & operator += (const Vector & p){
_x += p._x; _y += p._y; _phi += p._phi;
return *this;
}

//Subtraction
const Vector & operator -= (const Vector & p){
_x -= p._x; _y -= p._y; _phi -= p._phi;
return *this;
}

//Multiplication by a scalar
const Vector & operator *= (double c){
_x *= c; _y *= c; _phi *= c;
return *this;
}

private:
double _x; //x-component
double _y; //y-component
double _phi; //angular-component
};
const Vector null(0,0,0);
#endif
```

Appendix I

Boundary Condition Code

This is the C++ code of the Boundary Conditions used for different particle types.

```
void disk::boundary_conditions(int n, double dt, double t)
{
  switch(type()){
  case(0): break;
  case(1): break;
  case(2): {
      x()=0.5-0.4*cos(10*t);
      y()=0.1;
      vx()=10*0.4*sin(t);
      vy()=0;
    } break;
  case(3): {
      double xx=x()-0.5;
      double yy=y()-0.5;
      double xp=xx*cos(dt)-yy*sin(dt);
      double yp=xx*sin(dt)+yy*cos(dt);
      x()=0.5+xp;
      y()=0.5+yp;
      vx()=-yp;
      vy()= xp;
      omega()=1;
    } break;
  case(4): {
      x()=0.5+0.1*cos(t) + 0.4*cos(t+2*n*3.14159265/128);
      y()=0.5+0.1*sin(t) + 0.4*sin(t+2*n*3.14159265/128);
      vx()=-0.1*sin(t) - 0.4*sin(t+2*n*3.14159265/128);
      vy()= 0.1*cos(t) - 0.4*cos(t+2*n*3.14159265/128);
      omega()=1;
    } break;
  case(5): {//Vertical Vibrating Particle
  y()=y()+((0.02*sin(30*t))*0.1);
      vx()=0;
      vy()=0.02*30*cos(30*t);
    } break;
  case(6): {
      int i=n/2;
      y()=i*0.02+0.1+0.02*sin(30*t);
      vx()=0;
      vy()=0.02*30*cos(30*t);
    } break;
  case(7): {//Expander
  r()+=0.001;
}break;
  case(8): {//Contractor
  r()-=0.001;
```

```
}break;
  case(9): {//Compaction wall
  if(kinetic_sum < 1e-006){
  y()-=0.001;
  }
  else
  y()=y();
}break;
  default: {
      cout << "ptype: " << type() << " not implemented\n";
      abort();
    }
  }
}
```

Appendix J

Verlet List Construction Code

This is the C++ code which was executed to create and update the Verlet List.

```
bool make_verlet(){
bool ok = true;

//cout << "Making the new Verlet list...\n";
verlet.resize(particles);

//cout << "Resized...\n";
for(int ix = 0;ix < vnx; ix++){
for(int iy = 0; iy < vny; iy++){
celllist[ix][iy].clear();
}
}

//cout << "Celllist cleared...\n";
for(unsigned int i = 0; i < particle.size(); i++){
int ix = int((particle[i].x()-x_0)/dx);
int iy = int((particle[i].y()-y_0)/dy);
celllist[ix][iy].push_back(i);
//cout << "Remaking...\n";
}

//cout << "Cells remade...\n";
for(unsigned int i = 0; i < particles; i++){
set<int> oldverlet=verlet[i];
verlet[i].clear();
int ix = int((particle[i].x()-x_0)/dx);
int iy = int((particle[i].y()-y_0)/dy);
for(int iix = ix - 1; iix <= ix + 1; iix++){
for(int iiy = iy - 1; iiy <= iy + 1; iiy++){
int wx = (iix+vnx)%vnx;
int wy = (iiy+vny)%vny;
for(unsigned int k = 0; k < celllist[wx][wy].size(); k++){
int pk = celllist[wx][wy][k];
if(pk < (int)i){
if(Distance(particle[i],particle[pk],lx,ly) < particle[i].r() + particle[pk].r() + verlet_distance){
if((particle[i].type()==0) || (particle[pk].type()==0)){
verlet[i].insert(pk);
if(oldverlet.find(pk)==oldverlet.end()){
if(do_touch(i,pk)){
ok=false;
}
}
}
}
}
}
}
```

```
}
}
}
}
return ok;
}

bool do_touch(int i, int k){
return(Distance(particle[i],particle[k],lx,ly) < particle[i].r() + particle[k].r());
}

bool verlet_needs_update(){
for(unsigned int i = 0; i < particle.size(); i++){
if(Distance(particle[i],safe[i],lx,ly) >= verlet_ratio*verlet_distance){
return true;
}
}
return false;
}
```

Appendix K

Predictor-Corrector Code

This is the C++ code of the Gears Predictor-Corrector algorithm.

```
//Predictor
void disk::predict(double dt){
double a1 = dt;
double a2 = a1 * dt/2;
double a3 = a2 * dt/3;
double a4 = a3 * dt/4;

rtd0 += a1*rtd1 + a2*rtd2 + a3*rtd3 + a4*rtd4;
rtd1 += a1*rtd2 + a2*rtd3 + a3*rtd4;
rtd2 += a1*rtd3 + a2*rtd4;
rtd3 += a1*rtd4;
}

//Corrector
void disk::correct(double dt){
static Vector accel, corr;
double dt_recip = 1/dt;

const double coeff0 = double(19)/double(90)*(dt*dt/double(2));
const double coeff1 = double(3)/double(4)*(dt/double(2));
const double coeff3 = double(1)/double(2)*(double(3)*dt_recip);
const double coeff4 = double(1)/double(12)*(double(12)*(dt_recip*dt_recip));

accel=Vector(((1/_m)*_force.x())+G.x(),((1/_m)*_force.y())+G.y(),((1/_J)*_force.phi())+G.phi());

corr=accel-rtd2;
rtd0 += coeff0*corr;
rtd1 += coeff1*corr;
rtd2 = accel;
rtd3 += coeff3*corr;
rtd4 += coeff4*corr;
}
```

Appendix L

Force Calculation Code

This is the C++ code which was executed to calculate the contact force between particles.

```
void force(disk & p1, disk & p2, double lx, double ly, int tag){
double dx=normalise(p1.x()-p2.x(),lx);
double dy=normalise(p1.y()-p2.y(),ly);
double rr=sqrt(dx*dx+dy*dy);
double r1=p1.r();
double r2=p2.r();
double m1 = p1.m();
double m2 = p2.m();
double xi=r1+r2-rr;

if(xi>0){
p1.inc_coordination_number();
p2.inc_coordination_number();
potential_sum += (1*(xi*xi))/2;
double Y=p1._Y*p2._Y/(p1._Y+p2._Y);
double A=0.5*(p1._A+p2._A);
double mu = (p1._mu<p2._mu ? p1._mu : p2._mu);
double gamma = (p1._gamma<p2._gamma ? p1._gamma : p2._gamma);
gamma=100.0;
double gamma_t = 100.0;
double kn = 300000.0;
double reff = (r1*r2)/(r1+r2);
double meff = (m1*m2)/(m1+m2);
double dvx=p1.vx()-p2.vx();
double dvy=p1.vy()-p2.vy();
double rr_rez=1/rr;
double ex=dx*rr_rez;
double ey=dy*rr_rez;
double xidot=-(ex*dvx+ey*dvy);
double vtrel=-dvx*ey + dvy*ex + p1.omega()*p1.r()-p2.omega()*p2.r();
double fn,fx,fy;
if(tag==0){
fn = kn*xi + gamma*meff*xidot; //1st force law
}
else if(tag==1){
fn=sqrt(xi)*Y*sqrt(reff)*(xi+A*xidot); //2nd force law
}
else if(tag==3){
fx = kn*xi - (gamma*meff*(dvx*ex)); //3rd force law
fy = kn*xi - (gamma*meff*(dvy*ex));
}
double ft=-gamma_t*vtrel;
if(fn<0) fn=0;
if(ft<-mu*fn) ft=-mu*fn;
if(ft>mu*fn) ft=mu*fn;
```

```
if((tag==0)||(tag==1)){
if(p1.type()==0) {
p1.add_force(Vector(fn*ex-ft*ey, fn*ey+ft*ex, r1*ft));
}
if(p2.type()==0) {
p2.add_force(Vector(-fn*ex+ft*ey, -fn*ey-ft*ex, -r2*ft));
}
}
else{
if(p1.type()==0) {
p1.add_force(Vector(fx*ex-ft*ey, fy*ey+ft*ex, r1*ft));
}
if(p2.type()==0) {
p2.add_force(Vector(-fx*ex+ft*ey, -fy*ey-ft*ex, -r2*ft));
}
}
}
}
```

Appendix O

Other Functions

This is the C++ code of functions used during testing and to output particle properties to a file.

```
//-----------------------------------------Functions-----------------------------------------

void dump_grain(ostream & os, double x, double y, double phi, double vx, double vy,
double omega, double mass, double radius, int type, double mu,
double gamma, double Y, double A){
os << x << "\t" << y << "\t" << phi << "\t" << vx << "\t" << vy << "\t" << omega
<< "\t" << mass << "\t" << radius << "\t" << type << "\t" << mu << "\t" << gamma
<< "\t" << Y << "\t" << A << "\n";
}

double density (){
double sum; //The sum of the disk areas, the system width and height
double result;
sum = 0.0;

//Calculate the sum of the disk areas, from r^2 * PI
for(unsigned int j = 0; j < particle.size(); j++){
sum += pow((particle[j].r()),2.0)*3.14159265;
}
result = sum/(lx*ly);
return result;
}

//The kinetic energy of a particle
double disk::kinetic_energy() const{
return _m*((rtd1.x()*rtd1.x())/2 + (rtd1.y()*rtd1.y())/2) + _J*rtd1.phi()*rtd1.phi()/2;
}

//The total kinetic energy of the system
double total_kinetic_energy()
{
  double sum=0.0;
  for(unsigned int i=0;i<particle.size();i++){
    if(particle[i].type()==0){
      sum+=particle[i].kinetic_energy();
    }
  }
  return sum;
}

//The Linear Momentum of a particle
double disk::linear_momentum() const{
return _m*(rtd1.x() + rtd1.y()) + _J*rtd1.phi();
}
```

```
//The System Linear Momentum
double total_linear_momentum(){
double sum= 0.0;

for(unsigned int i = 0; i < particle.size(); i++){
if(particle[i].type()==0){
sum += particle[i].linear_momentum();
}
}
return sum;
}

//The Angular Momentum of a particle
double disk::angular_momentum() const{
return _J*rtd1.phi();
}

//The System Angular Momentum
double total_angular_momentum(){
double sum= 0.0;

for(unsigned int i = 0; i < particle.size(); i++){
if(particle[i].type()==0){
sum += particle[i].angular_momentum();
}
}
return sum;
}

//The avergae Coordination number
double avg_coordination_number(){
double sum=0.0;
double result;

for(unsigned int a = 0; a < particle.size(); a++){
sum+=particle[a].z();
}

result = sum/particles;
return result;
}
```

Appendix P

FrostHeave.exe

This is the code used to Generate a hopper geometry with 400 free particles.

```
// frost_heave.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include "disk.h"
#include "Vector.h"
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<assert.h>
#include<time.h>
#include<stdlib.h>
#include<set>
#include<vector>
#include<math.h>
#include<iterator>
#include<list>
#include<stdio.h>
#include<tchar.h>
#define length(a) ( sizeof ( a ) / sizeof ( *a ) )//Length of an array
using namespace std;

//Function Headers
void dump_grain(ostream & os, double x, double y, double phi, double vx, double vy,
double omega, double mass, double radius, int type, double mu,
double gamma, double Y, double A);

//Calculate the packing density of the system
double density();
//Calculate the Total Kinetic Energy of the System
double total_kinetic_energy();
//Calculate the Total Linear Momentum of the System
double total_linear_momentum();
//Calculate the Total Angular Momentum of the System
double total_angular_momentum();
//Calculate the Average Coordination Number of the System
double avg_coordination_number();

//The Verlet List functions
bool make_verlet();
bool do_touch(int i, int k);
bool verlet_needs_update();
void predict(double dt);
void correct(double dt);

//Simulation variables
```

```
int particles; //The number of particles in the system
int tag; //Used to select the force law
Vector G; //The Gravity vector
vector<disk> particle; //The Vector Array of particles
double Z = 0.0; //The System Coordination number
double potential_sum = 0.0; //The sum of the system potential energy
double kinetic_sum = 0.0; //The sum of the system kinetic energy
double linear_momentum_sum = 0.0; //The sum of the system linear momentum
double angular_momentum_sum = 0.0; //The sum of the system angular momentum
bool freezing; //If true, the system is on a freeze cycle
bool thawing; //If true, the system is thawing
bool ok_to_freeze; //If true, the system will imminently enter a freezing cycle

//Verlet variables
vector<set<int> > verlet; //The Verlet List
vector<vector<vector<int> > > celllist;//The Grid List
vector<disk> safe;
double verlet_distance = 0.00005;
double verlet_ratio = 0.6;
double verlet_grid = 0.05;
double verlet_increase = 1.1;
double dx,dy;
int vnx,vny;
double lx = 2.0;
double ly = 2.0;
double x_0 = 0.0;
double y_0 = 0.0;
double Timesafe;

int _tmain(int argc, _TCHAR* argv[])//Main program
{
double dt,tmax; //The timestep and the max time
time_t start, end; //Timer variables
double diff; //The difference in time
int j; //Counters for loops
int counter = 0; //Loop counter
int swap = 1; //Swap the free-thaw cycle
int swap_counter = 0;//Keep track of how many freeze-thaw cycles have taken place
int output = 500000;//Used to dump the phase properties

ofstream out1 ("Output/PC-Lab03_frost_heave01b.out"); //File writer object, writes phase information at predetermined
ofstream out2 ("Input/PC-Lab03_frost_heave02b.in"); //File writer object, writes the final phase information at the en
//ifstream in ("input.dat");//File reader object
ifstream in ("Input/PC-Lab03_frost_heave01b.in");//File reader object, change this to run any tests

if(!out1){//If the file is not accessible
cout << "Cannot open the first file to be written to!\n";
return 1;
}

if(!out2){//If the file is not accessible
cout << "Cannot open the seccond file to be written to!\n";
return 1;
}

if(!in){//If the file is not present
cout << "Cannot access the input file!\n";
return 1;
}

cout << "The input file is being read:\n";
//Read data from file, Gravity
in >> particles;
in >> tag;
in >> tmax;
in >> dt;
in >> G.x();
in >> G.y();
in >> G.phi();
```

```
//Print Pre-processing details to the screen
cout << "The Simulation is of " << particles << " particles" << ".\n"
<< "It simulates "<< tmax << " seconds at a time step of " << dt << ".\n"
<< "Using force law " << tag << ".\n";

particle.resize(particles);
safe=particle;
Timesafe=0.0;
vnx = int(lx/verlet_grid);
vny = int(ly/verlet_grid);

if(vnx == 0){
vnx = 1;
}

if(vny == 0){
vny = 1;
}

dx = lx/vnx;
dy = ly/vny;

celllist.resize(vnx);
for(int i = 0; i < vnx; i++){
celllist[i].resize(vny);
}

make_verlet();

cout << "The particles are being assigned their properties:\n";

//Input the particle properties from a file
for(unsigned int i = 0; i < particle.size(); i++){
in >> particle[i].x();
in >> particle[i].y();
in >> particle[i].phi();
in >> particle[i].vx();
in >> particle[i].vy();
in >> particle[i].omega();
in >> particle[i].m();
in >> particle[i].r();
in >> particle[i].type();
in >> particle[i].mu();
in >> particle[i].gamma();
in >> particle[i].Y();
in >> particle[i].A();
particle[i].omega()=0.0;
}
make_verlet();

//Print to the file
//Pre-processing details
//Phase Plot, goes to run1.out
out1 << particles << "\n" << tag << "\n" << tmax << "\n" << dt << "\n"
<< G.x() << "\t" << G.y() << "\t" << G.phi() << "\n";

//Phase Plot, goes to run2.in
out2 << particles << "\n" << tag << "\n" << tmax << "\n" << dt << "\n"
<< G.x() << "\t" << G.y() << "\t" << G.phi() << "\n";

//Particle Properties, goes to run1.out
for(unsigned int j = 0; j < particles; j++){
dump_grain(out1,particle[j].x(),particle[j].y(),particle[j].phi(),
particle[j].vx(),particle[j].vy(),particle[j].omega(),
particle[j].m(),particle[j].r(),particle[j].type(),particle[j].mu(),
particle[j].gamma(),particle[j].Y(),particle[j].A());
}
```

```
//make verlet
//Start time
time(&start);
thawing=true;
freezing=false;
ok_to_freeze=false;

//Print it to screen
kinetic_sum = total_kinetic_energy();
cout << "The Main loop has started:\n";
cout << "The minimum Kinetic Energy required to trigger a freezing cycle is: 1e-006\n";

for(double t = 0; t <= tmax; t+=dt){
counter++;
swap++;

if((swap%10000)==0){
ok_to_freeze=true;
}

if((kinetic_sum <= 1e-006)&&(ok_to_freeze==true)){
swap_counter++;
freezing=true;
thawing=false;
swap = 1;
ok_to_freeze=false;
cout << "Freezing cycle: " << swap_counter << " and a time of: " << t << "\n";
}

bool ok = true;
bool newverlet = false;
if(verlet_needs_update()){
ok = make_verlet();
newverlet = true;
}

if(!ok){
particle = safe;
t = Timesafe;
verlet_distance *= verlet_increase;
make_verlet();
}

if(newverlet && ok){
safe = particle;
Timesafe = t;
}

//-------------------Freezing----------------------------------
if(freezing==true){
for(unsigned int i = 0; i < particle.size(); i++){
if(particle[i].type()==0){
particle[i].rtd0.y() *= 1.4;
}
}
make_verlet();
freezing=false;
thawing=true;
swap = 1;
ok_to_freeze=false;
}

//-------------------Thawing-----------------------------------
if(thawing==true){
for(unsigned int i = 0; i < particle.size(); i++){
//particle[i].reset_coordination_number();
if(particle[i].type()==0){
particle[i].set_force_to_zero();
particle[i].predict(dt);
```

```
}
else {
particle[i].boundary_conditions(i,dt,t);
 }
}

//Cycle through the particles and look for collisions
for(unsigned int i = 0; i < particle.size(); i++){
set<int>::iterator iter;
for(iter = verlet[i].begin(); iter != verlet[i].end(); iter++){
force(particle[i],particle[*iter],lx,ly,tag);
}
}

//Correct the positions and reset the Coordination Numbers
for(unsigned int i = 0; i < particle.size(); i++) {
if(particle[i].type()==0){
particle[i].correct(dt);
}
}
//}//Thawing-----------------------------------------------------

//Output to File, goes to run1.out
if(counter%output==0){
cout << "The time is: " << t << "\n";
cout << "The Kinetic Energy is: " << kinetic_sum << "\n";

for(j = 0; j < particles; j++){
dump_grain(out1,particle[j].x(),particle[j].y(),particle[j].phi(),
particle[j].vx(),particle[j].vy(),particle[j].omega(),
particle[j].m(),particle[j].r(),particle[j].type(),particle[j].mu(),
particle[j].gamma(),particle[j].Y(),particle[j].A());
}
}
kinetic_sum = total_kinetic_energy();
}

//End Time
time(&end);

//Final Phase Plot, goes to run2.out
for(j = 0; j < particles; j++){
dump_grain(out2,particle[j].x(),particle[j].y(),particle[j].phi(),
particle[j].vx(),particle[j].vy(),particle[j].omega(),
particle[j].m(),particle[j].r(),particle[j].type(),particle[j].mu(),
particle[j].gamma(),particle[j].Y(),particle[j].A());
}

//Calculate the time to run the loop
diff = difftime(end,start);
cdn << "It has taken " << diff/60 << " minutes to run the main loop.\n";

//Close the input and output files
in.close();
out1.close();
out2.close();
return 0;
}
```

Bibliography

[Aha99] E. Aharonov and D. Sparks. Rigidity phase transition in granular packings. *Physical Review E*, vol. 60(6):pp. 6890–6896, 1999.

[Aha02] E. Aharonov and D. Sparks. Shear profiles and localization in simulations of granular materials. *Physical Review E*, vol. 65(5):p. 51302, 2002.

[Ald57] B. Alder and T. Wainwright. Phase Transition for a Hard Sphere System. *Journal of Chemical Physics*, vol. 27:p. 1208, 1957.

[Ald59] B. Alder and T. Wainwright. Studies in Molecular Dynamics .I. General Methods. *Journal of Chemical Physics*, vol. 31:p. 459, 1959.

[Ald60] B. Alder and T. Wainwright. Studies in Molecular Dynamics .II. Behaviour of a small number of Elastic Spheres. *Journal of Chemical Physics*, vol. 33:p. 1439, 1960.

[AR87] F. P. A. Rosato, K. J. Strandburg and R. H. Swendsen. Why brazil nuts are on top: Size segregation of particulate matter by shaking. *Phys Rev Lett*, vol. 58(10):p. 1038, 1987.

[Bag05] K. Bagi. An algorithm to generate random dense arrangements for discrete element simulations of granular assemblies. *Granular Matter*, vol. 7(1):pp. 31–43, 2005.

[Bla01] D. Blair, N. Mueggenburg, A. Marshall, H. Jaeger, and S. Nagel. Force distributions in three-dimensional granular assemblies: Effects of packing order and interparticle friction. *Physical Review E*, vol. 63(4):p. 41304, 2001.

[Cho04] J. Choi, A. Kudrolli, R. R. Rosales, and M. Z. Bazant. Diffusion and mixing in gravity-driven dense granular flows. *Phys Rev Lett*, vol. 92(17):p. 174301, Apr 2004.

[Cun79] P. Cundall and O. Strack. A discrete element model for granular assemblies. *Geotechnique*, vol. 29(1):pp. 47–65, 1979.

[Edw03] S. Edwards and D. Grinev. Statistical mechanics of granular materials: stress propagation and distribution of contact forces. *Granular Matter*, vol. 4(4):pp. 147–153, 2003.

[Gal92] J. A. C. Gallas, H. J. Herrmann, and S. Sokolowski. Convection cells in vibrating granular media. *Phys Rev Lett*, vol. 69(9):pp. 1371–1374, Aug 1992.

[Gio96] N. Giordano. *Computaional Physics*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1996.

[Gra09] J. Gray and M. Elliott. *Ecology of Marine Sediments: From Science to Management*. Oxford University Press, 2009.

[Haf86] P. Haff and B. Werner. Computer simulation of the mechanical sorting of grains. *Powder Technology*, vol. 48(3):pp. 239 – 245, 1986.

[Jae96a] H. Jaeger, S. Nagel, and R. Behringer. Granular solids, liquids, and gases. *Reviews of Modern Physics*, vol. 68(4):pp. 1259–1273, 1996.

[Jae96b] H. Jaeger, S. Nagel, and R. Behringer. The Physics of Granular Materials. *Physics Today*, vol. 49(4):pp. 32–38, 1996.

[JC05] A. K. Jaehyuk Choi1 and M. Z. Bazant. Velocity profile of granular flows inside silos and hoppers. *Journal of Physics: Consensed Matter*, vol. 17(24):p. 2533, 2005.

[Kes03] M. Kessler and B. Werner. Self-Organisation of Sorted Patterned Ground. *Science*, vol. 299(5605):pp. 380–383, 2003.

[Kni93] J. B. Knight, H. M. Jaeger, and S. R. Nagel. Vibration-induced size separation in granular media: The convection connection. *Phys Rev Lett*, vol. 70(24):pp. 3728–3731, Jun 1993.

[Mai07] K. Mair and J. Hazzard. Nature of stress accommodation in sheared granular material: Insights from 3D numerical modeling. *Earth and Planetary Science Letters*, vol. 259(3-4):pp. 469–485, 2007.

[Mak00] H. Makse, D. Johnson, and L. Schwartz. Packing of Compressible Granular Materials. *Physical Review Letters*, vol. 84(18):pp. 4160–4163, 2000.

[Mic] Michael de Podesta. http://www.physicsofmatter.com/NotTheBook/CriticalOpal/Explanation.html. http://www.physicsofmatter.com/NotTheBook/CriticalOpal/Explanation.html.

[Mue98] D. M. Mueth, H. M. Jaeger, and S. R. Nagel. Force distribution in a granular medium. *Phys Rev E*, vol. 57(3):pp. 3164–3169, Mar 1998.

[Nan22] Nansen, F. *Spitzbergen*. Brockhaus, 1922.

[Nor09] Nordenskjold, O. *Die Polarwelt*. Tuebner, 1909.

[PÖ5a] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*, p. 50. Springer, 2005.

[PÖ5b] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*. Springer, 2005.

[PÖ5c] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*, p. 20. Springer, 2005.

[PÖ5d] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*, p. 54. Springer, 2005.

[PÖ5e] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*, p. 58. Springer, 2005.

[Pre07] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[Rad99] F. Radjai, S. Roux, and J. Moreau. Contact forces in a granular packing. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 9:p. 544, 1999.

[Ris94] G. Ristow. Granular Dynamics: A review about recent Molecular Dynamics simulations of granular materials. *Annual Reviews of Computational Physics I*, 1994.