INL/EXT-10-19305

PEBBLES Operation and Theory Manual

Joshua Cogliati
Abderrafi Ougouag

September 2010

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-10-19305

PEBBLES Operation and Theory Manual

Joshua Cogliati
Abderrafi Ougouag

September 2010

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

INL/EXT-10-19305: PEBBLES Operation and Theory Manual

Joshua Cogliati and Abderrafi Ougouag

September 15, 2010

Chapter 1

Overview of PEBBLES

The PEBBLES code is a computer program designed to simulation the motion, packing and vibration of spheres that undergo various mechanical forces including gravitation, Hooke's law force and various friction forces. The frictional forces include true static friction that allows non-zero angles of repose. Each pebble is individually simulated using the distinct element method.

Chapter 2

Overview of Running PEBBLES

Running PEBBLES generally requires three things. First is the geometry specification that describes the arrangement of the container for the reactor. Next is any type of motion inducing parameters such as an earthquake simulation or recirculation of the pebbles. Last are the tallies of variables that are desired. Sometimes multiple simulations are chained together manually, for example a recirculation that outputs the save data followed by a second run of PEBBLES that loads the data and simulates an earthquake.

Several simulation types that are supported by PEBBLES. The first is simulations to determine the effects and parameters of the recirculation. The second is to investigate packing arrangements of the pebbles. The third is to simulate the effects of vibration motion on the reactor. For all of these, a variety of tally types are provided including dust production estimation, force tallying, velocity tallying and also the raw position data is available.

Chapter 3

Theory

3.1 Overview of History of granular and Pebble simulation

A variety of simulations of the motion of discrete elements have been created for different purposes. Lu, Abdou and Ying applied a discrete element method (DEM) to determine the characteristics of packed beds used as fusion reactor blankets[13]. Jullien, Pavlovitch and Meakin used a DEM to determine packing fractions for spheres using different non-motion methods[14]. Soppe used a rain method (lower spheres down until they reach other ones) to determine pore structures in different sized spheres[15]. Freund et al. used a rain method for fluid flow for chemical processing[16].

The use of non-motion pebble packing methods provide an approximation of the positions of the pebble. Unfortunately, non-motion methods will tend to either under pack or over pack (sometimes both in the same model). For large pebble bed reactors, the approximately ten-meter height of the reactor core will result in different forces at the bottom than at the top. Without key physics including static friction and the transmittal of force, matching packing fractions over the entire reactor will be difficult. Non-physics based modeling can not be used for predicting the effect of changes in static friction or different methods of loading the pebbles into the reactor even when only the position data is required.

The first versions of the PEBBLES code simply minimized the sum of the gravitational and Hookes' law potential energies by adjusting pebble positions. However, that simulation was insufficient for determining flow and motion parameters. This lead to significant physics additions being created.

Kohring created a 3-D discrete element method simulation to study diffusional mixing and provided the time derivatives for the simulation, but not the details of the calculation of static friction[17]. Haile discusses both how to simulate hard spheres (which is impractical for a pebble bed due to the frequent and continuous contact between spheres) and soft spheres using just the potential energy which is insufficient for modeling the motion[18]. The dissertation by Ristow described multiple methods for simulation of granular materials. On Ristow's list was a model similar to that used in the preliminary work supporting this proposal. Ristow's dissertation provided an overview of simulating static friction[19]. Vu-Quoc and Zhang created a model for particle flow simulations which was used for simulation of particle flow in chutes[20]. Their model included a detailed description of static friction calculation that was computationally tractable yet still based on the complete and complicated friction model of Mindlin and Deresiewicz[5]. To determine particle flows, Wait developed a discrete element method which included dynamic friction, but the model did not include static friction[21]. Multiple other discrete element codes have been created, and PEBBLES is similar to several of the full motion models.

3.2 Methods of simulation of dynamic effects

The PEBBLES simulation tracks each individual pebble's velocity, position, and angular velocity. The following classical mechanics differential equations are used for calculating the time derivatives of those variables.

$$\frac{d\mathbf{v}_i}{dt} = \frac{m_i\mathbf{g} + \sum_{i\neq j}\mathbf{F}_{ij} + \mathbf{F}_{ci}}{m_i} \tag{3.1}$$

$$\frac{d\mathbf{p}_i}{dt} = \mathbf{v}_i \tag{3.2}$$

$$\frac{d\omega_i}{dt} = \frac{\sum_{i\neq j}\mathbf{F}_{\parallel ij} \times r_i\hat{\mathbf{n}}_{ij} + \mathbf{F}_{\parallel ci} \times r_i\hat{\mathbf{n}}_{ci}}{I_i} \tag{3.3}$$

$\mathbf{F}_{ij}$ is the force from pebble j on pebble i, $\mathbf{F}_{ci}$ is the force of the container on pebble i, $\mathbf{g}$ is the gravitational acceleration constant, m_i is the mass of pebble i, $\mathbf{v}_i$ is the velocity of pebble i, $\mathbf{p}_i$ is the position vector for pebble i, ω_i is the angular velocity of pebble i, $\mathbf{F}_{\parallel ij}$ is the tangential force between pebbles i and j, r_i is the radius of pebble i, I_i is the moment of inertia for pebble i, $\mathbf{F}_{\parallel ci}$ is the tangential force of the container on pebble i, $\hat{\mathbf{n}}_{ci}$ is the unit vector normal to the container wall on pebble i and $\hat{\mathbf{n}}_{ij}$ is the unit vector pointing from the position of pebble i to that of pebble j. The static friction model contributes to the $\mathbf{F}_{\parallel ij}$ term which is also part of the $\mathbf{F}_{ij}$ term.

The mass and moment of inertia are calculated assuming spherical symmetry and that the spheres have a different density in an inner spherical core. With this assumption the following equations hold:

$$m = \frac{4}{3}\pi\left[\rho_c r_c^3 + \rho_o(r_o^3 - r_c^3)\right] \tag{3.4}$$

$$I = \frac{8}{15}\pi\left[\rho_c r_c^5 + \rho_o(r_o^5 - r_c^5)\right] \tag{3.5}$$

where r_c is the radius of inner (fueled) zone of the pebble, r_o is the radius of whole pebble, ρ_c is the average density of center fueled region and ρ_o is the average density of outer non-fueled region.

The dynamic (or kinetic) friction model is based on the model described by Wait[12]. Wait's and PEBBLES model calculates the dynamic friction using a combination of the relative velocities and pressure between the pebbles, as shown in Eqns (3.6) and (3.7):

$$\mathbf{F}_{\perp ij} = hl_{ij}\hat{\mathbf{n}}_{ij} - C_{\perp}\mathbf{v}_{\perp ij}, l_{ij} > 0 \tag{3.6}$$

$$\mathbf{F}_{d\| ij} = -min(\mu|\mathbf{F}_{\perp ij}|, C_{\|}|\mathbf{v}_{\| ij}|)\hat{\mathbf{v}}_{\| ij}, l_{ij} > 0 \tag{3.7}$$

where $C_{\|}$ is the tangential dashpot constant, $C_{\perp}$ is the normal dashpot constant, $\mathbf{F}_{\perp ij}$ is the normal force between pebbles i and j, $\mathbf{F}_{d\| ij}$ is the tangential dynamic friction force between pebbles i and j, h is the normal Hooke's law constant, l_{ij} is the overlap between pebbles i and j, $\mathbf{v}_{\| ij}$ is the component of the velocity between two pebbles perpendicular to the line joining their centers, $\mathbf{v}_{\perp ij}$ is the component of the velocity between two pebbles parallel to the line joining their centers, $\mathbf{v}_{ij}$ is the relative velocity between pebbles i and j and μ is the kinetic friction coefficient. Eqns (3.8-3.11) relate supplemental variables to the primary variables:

$$\mathbf{F}_{ij} = \mathbf{F}_{\perp ij} + \mathbf{F}_{\| ij} \tag{3.8}$$

$$\mathbf{v}_{\perp ij} = (\mathbf{v}_{ij} \cdot \hat{\mathbf{n}}_{ij})\hat{\mathbf{n}}_{ij} \tag{3.9}$$

$$\mathbf{v}_{\| ij} = \mathbf{v}_{ij} - \mathbf{v}_{\perp ij} \tag{3.10}$$

$$\mathbf{v}_{ij} = (\mathbf{v}_i + \omega_i \times r_i\hat{\mathbf{n}}_{ij}) - (\mathbf{v}_j + \omega_j \times r_j\hat{\mathbf{n}}_{ji}) \tag{3.11}$$

The friction force is then bounded by the friction coefficient and the normal force, to prevent it from being too great:

$$\mathbf{F}_{f\| ij} = \mathbf{F}_{s\| ij} + \mathbf{F}_{d\| ij} \tag{3.12}$$

$$\mathbf{F}_{\| ij} = min(\mu|\mathbf{F}_{\perp ij}|, |\mathbf{F}_{f\| ij}|)\hat{\mathbf{F}}_{f\| ij} \tag{3.13}$$

where $\mathbf{F}_{s\| ij}$ is the static friction force between pebbles i and j, $\mathbf{F}_{d\| ij}$ is the kinetic friction force between pebbles i and j, h_s is the coefficient for force from slip, $\mathbf{s}_{ij}$ is the slip distance perpendicular to the normal force between pebbles i and j, $v_{\max}$ is the maximum velocity under which static friction is allowed to operate, and μ is the static friction coefficient when the velocity is less than $v_{\max}$ and the kinetic friction coefficient when the velocity is greater. These equations fully enforces the first requirement of a static friction method, $|\mathbf{F}_s| \leq \mu|\mathbf{F}_{\perp}|$.

3.3 Integration

When all the position, linear velocity, angular velocity and slips are combined into a vector $\mathbf{y}$, the whole computation can be written as a differential formulation in the form:

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \tag{3.14}$$

$$\mathbf{y}(t_0) = \mathbf{y}_0 \tag{3.15}$$

This can be solved by a variety of methods such as Euler's method:

$$\mathbf{y}_1 = \mathbf{y}_0 + \Delta t \mathbf{f}(t, \mathbf{y}_0) \tag{3.16}$$

In addition, both the Runge-Kutta method and the Adams-Moulton method can be used for solving this equation. These methods improve the accuracy of the simulation. However, they do not improve the speed at the lowest stable simulation, since the additional time required for computation negates the advantage of being able to use somewhat longer time-steps. In addition, when running on a cluster, more data needs to be transferred since the methods allow non-contacting pebbles to affect each other in a single 'time-step' calculation.

3.4 Geometry equations

For any geometry interaction, two things need to be calculated, the overlap distance l and the normal to the surface $\hat{n}$. The input is the radius of the pebble r and the position of the pebble, $\mathbf{p}$ with components $\mathbf{p}_x$, $\mathbf{p}_y$, and $\mathbf{p}_z$

For the floor contact this is:

$$l = (\mathbf{p}_z - r) - floor\_location \tag{3.17}$$

$$\hat{n} = \hat{z} \tag{3.18}$$

For cylinder contact on the inside of a cylinder this is:

$$pr = \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2} \tag{3.19}$$

$$l = (pr + r) - cylinder\_radius \tag{3.20}$$

$$\hat{n} = \frac{-\mathbf{p}_x}{pr}\hat{x} + \frac{-\mathbf{p}_y}{pr}\hat{y} \tag{3.21}$$

For cylinder contact on the outside of a cylinder this is:

$$pr = \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2} \tag{3.22}$$

$$l = cylinder\_radius + r - pr \tag{3.23}$$

$$\hat{n} = \frac{\mathbf{p}_x}{pr}\hat{x} + \frac{\mathbf{p}_y}{pr}\hat{y} \tag{3.24}$$

For contact on the inside of a cone defined by the $radius = mz + b$:

$$pr = \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2} \tag{3.25}$$

$$z_c = \frac{m(pr - b) + z}{m^2 + 1} \tag{3.26}$$

$$r_c = mz_c + b \tag{3.27}$$

$$x_c = (r_c/pr)\mathbf{p}_y \tag{3.28}$$

$$y_c = (r_c/pr)\mathbf{p}_x \tag{3.29}$$

$$\mathbf{c} = x_c\hat{x} + y_c\hat{y} + z_c\hat{z} \tag{3.30}$$

$$d = \mathbf{p} - \mathbf{c} \tag{3.31}$$

$$l = |d| + r, r_c < pr \tag{3.32}$$

$$\hat{n} = -\hat{d}, r_c < pr \tag{3.33}$$

$$l = r - |d|, r_c >= pr \tag{3.34}$$

$$\hat{n} = hatd, r_c >= pr \tag{3.35}$$

$$ \tag{3.36}$$

These equations are derived from minimizing the distance between the contact point **c** and the pebble position **p**.

For contact on a plane defined by $ax + by + cz + d = 0$ where the equation has been normalized so that $a^2 + b^2 + c^2 = 1$, the following is used:

$$dp = a\mathbf{p}_x + b\mathbf{p}_y + c\mathbf{p}_z + d \tag{3.37}$$

$$l = r - dp \tag{3.38}$$

$$\hat{n} = a\hat{x} + b\hat{y} + c\hat{z} \tag{3.39}$$

3.5 Simulation of Static Friction

The static friction model has two key tasks: the force from stuck slip must be updated based on relative motion of the pebbles, and the current direction of the force must be calculated, since the pebbles can rotate in space.

3.5.1 Use of Parallel Velocity for Slip Updating

The true method for updating the stuck slip force of elastic spheres is to use the R. D. Mindlin and H. Deresiewicz method[5], which requires computationally and memory intensive calculations to track the forces. However, a simpler method described by Cundall and Strack[6], uses the integration of the parallel relative velocity as the displacement, is used to approximate the force. The essential idea is that the farther the pebbles have stuck slipped at the contact point, the greater the counteracting static friction force needs to be. This is what happens under more accurate models such as Mindlin and Deresiewicz.

This assumption imposes two approximations: (1) the amount that the force changes is independent of the normal force, (2) the true hysteretic effects, which depend on loading history details, are ignored. For simulations where the exact dynamics of static friction are important, these could potentially be serious errors. However, since static friction only occurs when the relative speed is low, the dynamics of the simulation are usually unimportant. Thus, for most circumstances, the approximation that can be used to describe the rate of change of the magnitude and non-rotational change of the stuck slip is

$$\frac{d\mathbf{s}_{ij}}{dt} = \mathbf{v}_{\parallel ij}. \tag{3.40}$$

Vu-Quoc, Zhang, and Walton developed a 3-D discrete-element method for granular flows[7]. They used a simplification of the Mindlin and Deresiewicz model for calculating the stuck slip magnitude, and projected the stuck slip onto the tangent plane for each time step to rotate the stuck slip force direction. This correctly rotates the stuck slip, but requires that the rotation of the stuck slip be done as a separate step since this is not written in a differential form.

Silbert et al. describe a 3-D differential version of the Cundall and Strack method[8, 9]. The literature states that particle wall interactions are done identically to the particle-to-particle interactions (with no derivation or justification provided). The amount of computation of the model is less than the Vu-Quoc, Zhang, and Walton model. This model was used to model pebble bed flow[10, 11].

3.5.2 Rotation of Stuck Slip

The static friction force must also be rotated so that it is in the plane of contact between the two pebbles. This method works by noticing that the change in the direction in the stuck slip comes from changes in the relative pebble center location, which changes when there is a difference between the pebble's center velocities. That is:

$$\mathbf{p}_{in+1} - \mathbf{p}_{jn+1} \approx \mathbf{p}_{in} - \mathbf{p}_{jn} + (\mathbf{v}_{in} - \mathbf{v}_{jn})\Delta t \tag{3.41}$$

First, let $\mathbf{n}_{ijn} = \mathbf{p}_i - \mathbf{p}_j$ and $d\mathbf{n}_{ijn} = \mathbf{v}_i - \mathbf{v}_j$. The cross product $-d\mathbf{n}_{ijn} \times \mathbf{n}_{ijn}$ is perpendicular to both $\mathbf{n}$ and $d\mathbf{n}$ and signed to create the axis around which $\mathbf{s}$ is rotated in a right-handed direction. Then, using the cross product of the axis and $\mathbf{s}$, $-(d\mathbf{n}_{ij} \times \mathbf{n}_{ijn}) \times \mathbf{s}_{ijn}$ gives the correct direction that $\mathbf{s}$ should be increased.

Next is determining the factors required to make the differential the proper length. By cross product laws

$$|-(d\mathbf{n}_{ij} \times \mathbf{n}_{ijn}) \times \mathbf{s}_{ijn}| = |d\mathbf{n}_{ij}||\mathbf{n}_{ijn}||\mathbf{s}_{ijn}| \sin\theta \sin\phi \tag{3.42}$$

where θ is the angle between $\mathbf{n_{ijn}}$ and $d\mathbf{n_{ij}}$, and ϕ is the angle between $d\mathbf{n}_{ij} \times \mathbf{n}_{ijn}$ and $\mathbf{s}_{ijn}$. The relevant vectors are shown in Figure 3.1.

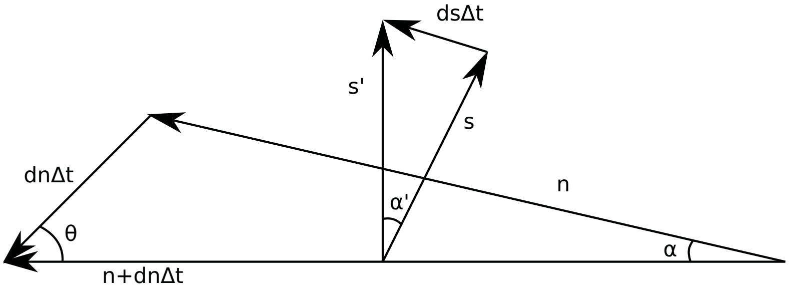

Figure 3.1: Static Friction Vectors

The goal is to rotate **s** by angle $\alpha\prime$, which is the 'projection' into the proper plane of angle α that **n** rotates by. Since the direction has been determined, for simplicity the figure leaves the indexes off and concentrates on determining the lengths. In Figure 3.1, **s** is the old slip vector, **s**′ is the new slip vector, and **n** is the vector pointing from one pebble to another. The vector $d\mathbf{n}\Delta t$ is added to **n** to get the new **n**′, $\mathbf{n} + d\mathbf{n}\Delta t$. The initial condition is that **s** and **n** are perpendicular. The final conditions are that **s**′ and **n**′ are perpendicular, **s** and **s**′ are the same length, and **s**′ is the closest vector to **s** as it can be while satisfying the other conditions. There is no requirement that **s** or **s**′ are coplanar with $d\mathbf{n}\Delta t$ (otherwise $\alpha\prime$ would be equal to α). From the law of sins we have:

$$\frac{|d\mathbf{n}\Delta t|}{\sin\alpha} = \frac{|\mathbf{n}|}{\sin\theta} \tag{3.43}$$

so

$$\sin\alpha = \frac{|d\mathbf{n}\Delta t|\sin\theta}{|\mathbf{n}|}. \tag{3.44}$$

The projection to the correct plane occurs in Figure 3.2. First, by using ϕ, the length of **s** is projected to the plane. Note that ϕ is the angle both to **s** and to **s**′. So, the length of the line on the $d\mathbf{n} \times \mathbf{n}$ plane is $|\mathbf{s}|sin\phi$, and the length of the straight line at the end of the triangle is $|\mathbf{s}|sin\phi sin\alpha$ (note that the chord length is $|\mathbf{s}|(sin\phi)\alpha$, but as Δt approaches 0 the other can be used). From these calculations, the length of the $d\mathbf{s}\Delta t$ can be calculated with

$$d\mathbf{s}\Delta t = \frac{|s|\sin\phi|d\mathbf{n}\Delta t|\sin\theta}{|\mathbf{n}|}. \tag{3.45}$$

Since $|-(d\mathbf{n}_{ij} \times \mathbf{n}_{ijn}) \times \mathbf{s}_{ijn}| = |d\mathbf{n}_{ij}||\mathbf{n}_{ijn}||\mathbf{s}_{ijn}|\sin\theta\sin\phi$, the formula for the rotation is

$$\mathbf{s}_{ijn+1} = -\frac{(d\mathbf{n}_{ijn} \times \mathbf{n}_{ijn}) \times \mathbf{s}_{ijn}}{\mathbf{n}^2}\Delta t + \mathbf{s}_{ijn}. \tag{3.46}$$

As a differential equation this is

$$\frac{d\mathbf{s}_{ij}}{dt} = -\frac{[((\mathbf{v}_i - \mathbf{v}_j) \times (\mathbf{p}_i - \mathbf{p}_j)) \times \mathbf{s}_{ijn}]}{|\mathbf{p}_i - \mathbf{p}_j|^2}. \tag{3.47}$$

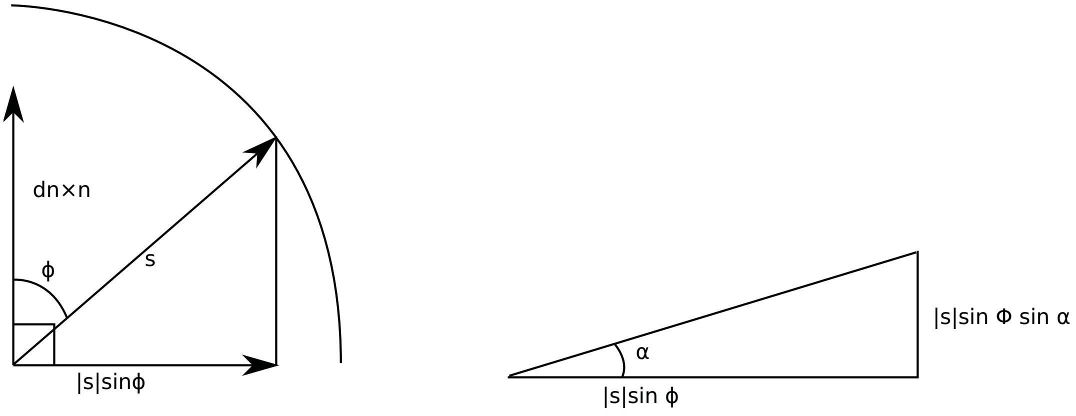

Figure 3.2: Projections to ds

By the vector property $a \times (b \times c) = b(a \cdot c) - c(a \cdot b)$ and, since $(\mathbf{p}_i - \mathbf{p}_j) \cdot \mathbf{s}_{ijn} = 0$, this can be rewritten per the version in Silbert et al. as

$$\frac{d\mathbf{s}_{ij}}{dt} = -\frac{(\mathbf{p}_i - \mathbf{p}_j)(\mathbf{s}_{ij} \cdot (\mathbf{v}_i - \mathbf{v}_j))}{|\mathbf{p}_i - \mathbf{p}_j|^2}. \tag{3.48}$$

Differential Equation for Surface Slip Rotating

It might seem that the wall interaction could be modeled the same way as the pebble-to-pebble interaction. For sufficiently simple wall geometries this may be possible, but actual pebble bed reactor geometries are more complicated and violate some of the assumptions that underpin the derivation. For a flat surface, there is no rotation so the formula can be entirely dropped. For a spherical surface, it would be possible to measure the curvature at pebble to surface contact point in the direction of relative velocity to the surface. This curvature could then be used as an effective radius in the pebble-to-pebble formulas.

The pebble reactor walls have additional features that violate assumptions made for the derivation. For surfaces such as a cone, the curvature is not in general constant, because the path can follow elliptical curves. As well, the curvature has discontinuities where different parts of the reactor join together (for example the transition from the outlet cone to the outlet chute). At these transitions, the assumption that the slip stays parallel to the surface fails because the slip is parallel to the old surface, but the new surface has a different normal.

The PEBBLES code uses an approximation of the rotation delta to deal with complications with using the pebble-to-pebble interaction. This is similar to the Vu-Quoc et al[7]. method of adjusting the slip so that it is parallel to the surface each time. Every time the slip is used, a temporary version properly aligned to the surface is computed and used for the force. When the derivatives are calculated, a rotation to move the slip to be more parallel to the surface is computed as follows:

Let the normal direction of the wall at the point of contact of the pebble be $\mathbf{n}$, and the old stuck slip be $\mathbf{s}$. Let a be the angle between $\mathbf{n}$ and $\mathbf{s}$, and $\mathbf{n} \times \mathbf{s}$ is perpendicular to both $\mathbf{n}$ and $\mathbf{s}$. Then $(\mathbf{n} \times \mathbf{s}) \times \mathbf{s}$ is perpendicular to this vector, so it is either pointing directly towards $\mathbf{n}$ if a is acute or directly away from $\mathbf{n}$ if a is obtuse. To get the correct direction, this vector is then multiplied by the scalar $\mathbf{s} \cdot \mathbf{n}$, which has the correct sign from $\cos a$. The magnitude of $(\mathbf{s} \cdot \mathbf{n})[(\mathbf{n} \times \mathbf{s}) \times \mathbf{s}]$ needs to be determined for reasonableness. The angle, which is between $(\mathbf{n} \times \mathbf{s})$ and $\mathbf{s}$, is defined as b. By these definitions the magnitude is $(|\mathbf{s}||\mathbf{n}| \cos a)[(|\mathbf{n}||\mathbf{s}| \sin a)|\mathbf{s}| \sin b]$. b is a right angle since $\mathbf{n} \times \mathbf{s}$ is perpendicular to $\mathbf{s}$, so $\sin b = 1$. Collecting terms gives the magnitude as $|\mathbf{s}|^3|\mathbf{n}|^2 \cos a \sin a$, which is divided by $|\mathbf{n} \times \mathbf{s}||\mathbf{n}||\mathbf{s}|$ to give the full term the magnitude $|\mathbf{s}| \cos a$. This is the length of the vector that goes from $\mathbf{s}$ to the plane perpendicular to $\mathbf{n}$. This produces Equation 3.49, which can be used to ensure that the wall stuck slip vector rotates towards the correct direction.

$$\frac{d\mathbf{s}}{dt} = (\mathbf{s} \cdot \mathbf{n}) \frac{[(\mathbf{n} \times \mathbf{s}) \times \mathbf{s}]}{|\mathbf{n} \times \mathbf{s}||\mathbf{n}||\mathbf{s}|} \tag{3.49}$$

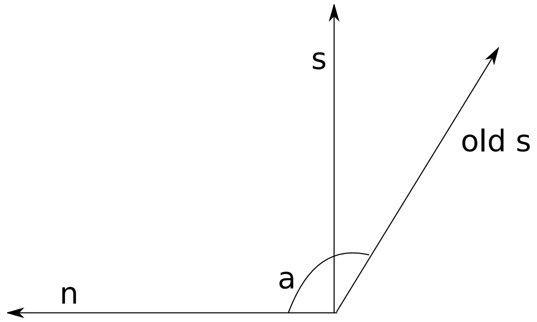

Figure 3.3: Static friction vectors for wall

3.5.3 Use of Slip

The value of the slip is used to calculate the force of static friction as in

$$\mathbf{F}_{s\|ij} = -h_s \mathbf{s}_{ij}, |\mathbf{v}_{v\|ij}| < v_{max}. \tag{3.50}$$

The friction force is then bounded by the friction coefficient and the normal force, to prevent it from being too great as

$$\mathbf{F}_{f\|ij} = \mathbf{F}_{s\|ij} + \mathbf{F}_{d\|ij} \tag{3.51}$$

and

$$\mathbf{F}_{\|ij} = min(\mu|\mathbf{F}_{\perp ij}|, |\mathbf{F}_{f\|ij}|)\hat{\mathbf{F}}_{f\|ij} \tag{3.52}$$

where $\mathbf{F}_{s\|ij}$ is the static friction force between pebbles i and j, $\mathbf{F}_{d\|ij}$ is the kinetic friction force between pebbles i and j, h_s is the coefficient for force from slip, $\mathbf{s}_{ij}$ is the slip distance perpendicular to the normal force between pebbles i and j, $v_{\max}$ is the maximum velocity under which static friction is allowed to operate, and μ is the static friction coefficient when the velocity is less than $v_{\max}$ and the kinetic friction coefficient when the velocity is greater. These equations fully enforces the first requirement of a static friction method, $|\mathbf{F}_{||}| \leq \mu|\mathbf{F}_{\perp}|$.

Chapter 4

Input Description

4.1 Introduction

directive *var1 var2*
Default: **directive** *num1 num2*

The variables are numbers input, and the default is what happens if no directive is given.

done

Terminates input and starts the program running. This is the only mandatory part of the input. Multiple done commands have no effect.

rem *stuff*

Remark. Allows comments in the input file. Allows commenting out lines. Ignored by pebbles.

4.2 Parameters

runs *number*
Default: **runs** *10000*

How many time-steps are run. This multiplied by alpha gives the total time that the simulation simulates.

number_of_pebbles *number*
Default: **number_of_pebbles** *1000*

The total number of pebbles in the simulation.

initial_time *time*
Default: **initial_time** *0.0*

The initial time used to start the simulation. This will allow the various times to be modified, such as the earthquake start times and such.

kinetic_friction *friction_coefficient*
Default: **kinetic_friction** *0.1*

The kinetic friction coefficient.

static_friction_new *friction_coefficient static_friction_hooke velocity_max_sqr*

The static friction coefficients. The friction_coefficient limits the amount of friction to friction_coefficient*normal_force. The static_friction_hooke is how much force there is related to the distance slipped. The velocity_max_sqr is the maximum velocity squared that static friction operates in.

static_friction_new2 *friction_coefficient static_friction_hooke surface_static_friction_hooke velocity_max_sqr*

The static friction coefficients. The friction_coefficient limits the amount of friction to friction_coefficient*normal_force. The static_friction_hooke is how much force there is related to the distance slipped between pebbles. The surface_static_friction_hooke is how much force there is related to the distance slipped between the pebble and the surface. The velocity_max_sqr is the maximum velocity squared that static friction operates in.

static_friction_new3 *pebble_pebble_friction_coefficient pebble_pebble_static_friction_hooke surface_friction_coefficient surface_static_friction_hooke velocity_max_sqr*

The static friction coefficients. The two friction_coefficient limit the amount of friction to friction_coefficient*normal_force. The pebble_pebble one is used between two pebbles, and the surface one is used between pebbles and the surface. The static_friction_hooke is how much force there is related to the distance slipped between pebbles. The surface_static_friction_hooke is how much force there is related to the distance slipped between the pebble and the surface. The velocity_max_sqr is the maximum velocity squared that static friction operates in. This version allows separate values for the pebble to pebble static friction coefficient and the pebble to surface coefficient.

alpha *time_step*
Default: **alpha** *0.0001*

The time step between each iteration of the model. Each time-step takes this long in simulation time. The unit is seconds.

dash_pot *dash_pot_constant*
Default: **dash_pot** *2.0*

The dashpot damping coefficient. Large values of this tend to make the model unstable (may need to decrease the timestep if this is increased).

dash_pot2 *normal_dash_pot_constant transverse_dash_pot_constant*
Default: **dash_pot2** *2.0 2.0*

The separate dashpot damping coefficients. The normal is used for normal velocity, and the transverse is used for transverse velocities.

seed *random_seed*
Default: **seed** *256*

The random number to seed the random number generator with. Put in different values to get different initial conditions. The random number generator is mostly used to find initial pebble locations.

pebble_pebble_hooke *constant*
Default: **pebble_pebble_hooke** *10000.0*

The pebble_pebble hooke constant in kg*m/(m*s**2)

pebble_reactor_hooke *constant*
Default: **pebble_reactor_hooke** *10000.0*

The pebble reactor hooke constant in kg*m/(m*s**2)

pebble_radius *center_radius outside_radius*
Default: **pebble_radius** *0.0 0.1*

The center radius is the radius of the fuel region. The outside radius is the radius of the pebble in meters. If the center_radius is 0.0 then the pebble has uniform density.

pebble_density *center_density outside_density*
Default: **pebble_density** *0.0 2.0*

The density of the pebble in kg/m**3. Used to calculate mass and moment of inertia. The first value is the density of the fuel region. The second value is the density of the region outside the fuel region.

recirculate *recirculation_height door_closed_time first_door_open_time hole_size hole_depth*

Creates a door at the bottom of the cone. The door will first open at first_door_open_time and then it will close for door_closed_time every time that a pebble falls through. The pebbles will be dropped from recirculation_height (make sure that is above all the pebbles). Makes the door pebble_radius * hole_size The hole_depth is the distance below the end of the cone that the door is and is in meters. This allows a short cylinder to be placed below the cone to simulate the drainage tube. It is recommended that recirculate_params be used instead since it splits out the geometry changes from the recirculation parameters.

recirculate_params *recirculation_height door_closed_time first_door_open_time*

Causes the pebbles to be recirculated. The door will first open at first_door_open_time and then it will close for door_closed_time every time that a pebble falls through. The pebbles will be dropped from recirculation_height (make sure that is above all the pebbles). Make sure that a exit chute has been created.

ppf_filename *base_filename*

The filename must be of the format abcdefgh00 where there are eight letters followed by two digits that are ignored. The last two characters are replaced by the MPI processor rank, and debugging and timing information is put in the file. This directive is ignored for the single processor pebbles7 and pebbles9 program.

linear_static_friction_cutoff

If this is turned on, then the model will use a 1- v_transvere/v_static_max for the static friction cutoff. Otherwise, 1 will be used.

sort_pebbles

If this is added, after the pebbles are initially loaded, it will sort them by height. In general, this should be included, since it speeds up the program by increasing memory locality which improves cache usage.

zero_long_slips_mult *mult*

Zeros slips that are longer than mult*normal_force*static_friction_coefficient

integration_method *method*

In pebbles9 this is used to determine the integration method used. Methods available are euler, runge_kutta, adams_bashforth and adams_moulton. In pebbles10 only euler and runge_kutta are available.

4.3 Tallies

dump_frequency *frequency*
Default: **dump_frequency** *100000*

How often a dump of the overlaps between pebbles is done. This is always done at the end. Units of timesteps.

display_frequency *frequency*
Default: **display_frequency** *1000*

How often the data for the pebbles is displayed. This includes the position, velocities and static friction loadings for all the pebbles. Unites in terms of timesteps. Note that these can be loaded from with appropriate processing, and used as restart files.

position_display_frequency *frequency*
Default: **position_display_frequency** *100000000*

How often just the positions of the pebbles are displayed. Units in terms of timesteps. This will dump less information to the output so disk space is not quite as much of an issue.

energy_display_frequency *frequency*
Default: **energy_display_frequency** *100*

How often to display the total energies. Units in terms of timesteps.

velocity_tally *start_time frequency*

Starts tallying the velocity from start time every frequency. Frequency is number of time steps to wait to do each tally.

probability_table *start_time frequency delta_size*

Starts the probability tally at start_time every frequency. Frequency is the number of time steps to wait to do each tally. delta_size specifies the size of the grid that the probabilities are done on.

calculate_pebble_wear

If this is turned on, calculates the sum of the normal force**(4/3)*distance for the pebble slips. This number is proportional to the volume of material produced by adhesive wear.

normal_force_wear_exponent *exponent*
Default: **exponent** *2.0/3.0*

Allows a different exponent to be used for the pebble wear calculation.

wear_velocity_bins *number list of bins*

This allows wear values to be calculated for different velocity bins. For example, if this is wear_velocity_bins 2 1.0 2.0 then three bins of wear tallies will be calculated, wear up to 1.0 m/s, wear from between 1.0 and 2.0, and wear from above 2.0.

calculate_force_tally

If this is turned on, a sum of the force on pebbles at different points in the reactor is done.

4.4 Packing

initial_packing *value*
Default: **initial_packing** *0.10*

The initial density of the pebbles. Should not be greater than 0.15 or the model may get into infinite loops trying to get started (or at least appears to act that way). In general random_packing_method should be used instead.

random_packing_method *mult*

An alternative method of packing. This uses the PRIMe method which generates a large number of pebble positions and finds the non-overlapping ones. This can produce higher packing fractions than the initial_packing method. The larger the mult, the higher the packing density, but the slower the method. Using 100000 results in a packing fraction above 0.5.

cone_packing *opening_size height*

Cone packing creates a cone above the reactor vessel with an opening radius size that is opening_size in meters and the bottom is at height and has a 45 degree angle. Then, the pebbles are initially put above the cone and drop down through the hole into the reactor. This is more realistic than the other packing method, but slower.

cone_packing_chute *opening_size height chute_height*

Cone packing creates a cone above the reactor vessel with an opening radius size that is opening_size in meters and the bottom is at height and has a 45 degree angle. Then, the pebbles are initially put above the cone and drop down through the hole into the reactor. This is more realistic than the other packing method, but slower. The chute_height is a cylinder starting at height and going down by chute_height meters and is to make the pebbles go straighter.

4.5 Loading and Saving

dump_positions *filename*

At the end of the run, dumps the positions of the pebbles to a file.

dump_positions_mult *filename mult*

Same as dump_positions, except that it multiplies the positions by the mult factor. Since the positions are in meters, if you say want them in cm, then the mult should be 100.0, and so forth.

load_positions *filename*

At the start of the run, loads the positions of the pebbles from a file. The positions should be in three floating point numbers seperated by spaces. They are x,y,z in meters.

load_positions_divide *filename divide*

Same as load_positions, except that it divides the positions by the divide factor. Since the positions are in meters, if you say want to load them from say cm, then the divide should be 100.0, and so forth.

load_pebble_info *filename*

Loads all the info from the file. Same format as the dumps that occur to standard out. load_pebble_save is generally preferred unless there is no slip or surface slip data.

load_pebble_slips

Loads the slips from a file in addition to all the other data. Ignored if load_pebble_info does not also appear in the file.

load_pebble_surface_slips

Loads the surface slips from a file in addition to all the other data. Ignored in if load_pebble_info does not also appear in the file.

load_pebble_save *filename*

Loads all the pebble info from the file. Same format as the dumps that occur to standard out. Loads both the slips and the surface slips.

4.6 Earthquake

earthquake_enable *number_of_waves*
Default: **earthquake_enable** *0*

Enables number_of_waves of earthquake waves. The number must at least be the number of earthquake waves that are later defined. Must appear before any definition of an earthquake wave.

earthquake_sine_wave *start_time end_time d_x d_y d_z period initial_cycle*

The wave runs between the start time and the end time (in terms of seconds). The d_x, d_y, and d_z are displacements that the earthquake does. So, d_x = d_y = 0 and d_z = 1.0 would be a earthquake that caused a 1.0 meter displacement in the upward direction, followed by a 1.0 displacement in the downward direction. The period is the time that the earthquake takes to cycle back to the beginning. The initial cycle is between 0 and 2*pi and determines where in the sine wave a cycle starts.

earthquake_sine_wave_offset *start_time end_time d_x d_y d_z period initial_cycle offset*

The same as earthquake sine wave, except there is a default offset. If the offset is zero, then it is exactly as the earthquake_sine_wave. If it is 1 then the displacement runs from 2.0*displacement to 0.0*displacement, instead of 1.0*displacement to -1.0*displacement.

tabular_earthquake *earthquake_start table_data_delta table_length filename*

The earthquake_start is the time that the earthquake data starts. The table_data_delta is the time between each piece of data in the table. Note that table_data_deltas less than the alpha value are not well handled in the current code (It won't crash, but the frequency spectrum will be destroyed). The table_length is the number of sample values. The filename is the name of the file to load the data from. The data should be in lines of x,y,z displacement values. The velocity is calculated by subtracting successive values.

4.7 Geometry Section

There can be one geometry section. It start with a `start_geometry` line and ends with a `end_geometry` line. In between, it can have unions, which consist of multiple objects that are combined together to form one object. A union is started with a `start_union` line and ended with a `end_union` line. In a union there can be the following objects. The geometry section can also have inlets, which specify where the recirculating pebbles enter.

inlet *px py pz vx vy vz*

Specifies the initial position x,y, and z and the initial velocity x, y, and z when a pebble is removed from the bottom of the reactor.

4.7.1 Geometry Objects

block *xmin ymin zmin xmax ymax zmax*

Creates a rectangular parallelepiped box with corners at xmin, ymin, zmin and xmax, ymax, and zmax. The minimum values must be less than the maximum values for each dimension.

cylinder *radius xcenter ycenter zmin zmax*

Creates a vertical cylinder between zmin and zmax centered at xcenter and ycenter with the supplied radius.

plane *a b c d*

Creates a plane defined by the equation $ax + by + cz + d = 0$. The normal for this plane is a vector formed by normalizing $a\hat{x} + b\hat{y} + c\hat{z}$. For example, `plane 0.0 0.0 1.0 -8.0` would form a floor at height 8.0, and `plane 0.0 0.0 -1.0 8.0` would form a ceiling at height 8.0, and `plane 0.0 0.0 -0.5 4.0` would also form a ceiling at height 8.0.

4.7.2 Indentations

Indentations are depressions in a wall. Right now, they are only supported on the main wall specified by the `reactor_radius` directive. A pattern is created by intersecting multiple planes. These patterns are then replicated between a minimum and a maximum height, at each row and at each angle specified for the row.

The indentations occur between the minimum and maximum height specified. Rows are specified by providing a minimum, and all the angles that indentations occur at in that row.

start_pattern

Starts a pattern block. Must be inside an indentation block. The only thing inside should be `plane` directives.

end_pattern

Ends a pattern block.

min_max *minimum maximum*

This must be inside an indentation block and before any rows. It specifies the minimum and the maximum heights that indentations are allowed.

start_row

This starts a row block. It must be inside an indentation block.

end_row

This ends a row block.

row_min *min*

This specifies the minimum location (and zero point) of a row. The row goes to either the next row's minimum location or the indentation's maximum location. This must be in a row block.

angle *angle*

This specifies an angle that an indention pattern is used. This must be in a row block, and multiple angles can be specified.

angles *angle_count angle1 angle2 ...*

This specifies multiple angles that an indention pattern is used. This must be in a row block, and this can be repeated. The angle_count is the number of angles that follow in that line.

```
reactor_radius 0.0 0.15
start_geometry
start_indentation
start_pattern
plane -1.0 0.0 0.0 0.25
plane 1.0 0.0 0.0 0.25
plane 0.0 -1.0 0.0 0.05
plane 0.0 1.0 0.0 0.05
plane 0.0 0.0 -1.0 0.5
plane 0.0 0.0 1.0 0.0
end_pattern
min_max 0.4 2.0
start_row
row_min 0.4
angle 0.0
end_row
start_row
row_min 1.0
angle 90.0
angle -45.0
angle 180.0
end_row
end_indentation
end_geometry
```

4.8 Other Geometry

reactor_radius *inside_radius outside_radius*
Default: **reactor_radius** *0.0 1.0*

The size of the cylinder the reactor is in. Pebbles go between the inside radius and the outside radius. If the inside_radius is 0.0 then the container is a cylinder instead of an annulus.

rectangular_vat *x_wall y_wall*

Creates a rectangular vat instead of a cylindrical. Use with a cone chute underneath is untested. The x_wall parameter is the distance that x_wall is from the center in meters, so the vat's floor area is 4.0*x_wall*y_wall.

floor_location *height*
Default: **floor_location** *0.0*

Sets the height of the floor. Below this value the pebbles are forced up by the hookes law force.

cone *location slope*

Creates a cone starting at location height and going down at slope slope. Slope 0.0 would be straight down, and 1.0 would be 45 degrees. Uses formula radius = m * z + constant

number_of_cones *number_of_cones*

Creates the specified number of cones located radially spaced around the reactor center.

offset_inlet_holes

Offsets the inlet holes and the cone packing cones from the holes that are in the base.

exit_chute *hole_size hole_depth*

Creates a door at the bottom of the cone. Makes the door pebble_radius * hole_size The hole_depth is the distance below the end of the cone that the door is and is in meters. This allows a short cylinder to be placed below the cone to simulate the drainage tube. If you want recirculation, use the recirculate_params function. Note that these parameters are also set if recirculate is used instead of recirculate_params.

Chapter 5

Methods of Packing

The PEBBLES code has three different methods of packing, or creating an initial condition. One of these needs to be chosen when an existing save file is not used. The simplest is the drop and compact method. For this method, an initial very low density packing is created (typically 15% filled) and then gravity is used to compact the pebbles. This method has two issues. First, the simulation can use a fair amount of time simulating the falling of the pebbles until they reach their final position. Second, depending on the number of pebbles and the cross sectional area of the vat, the pebbles that are above can cause excess compression of the pebbles that are below, and static friction can lock in part of this compression. This results in non-physical configurations.

The second method is the Pebble Removal Incremental Method (PRIMe). This method generates large numbers of potential pebble positions, and starting from the bottom chooses pebbles that are not currently overlapping[22]. In PEBBLES this is the simplest and fastest method to create an initial packing.

The third method of packing is the chute fill method. This method uses one of the other two methods to create a packing above chutes. The pebbles then fall down through the chutes and that creates the packing in the main container. To the extent that this resembles the physical method of packing, the modeling is improved.

Each of these methods will result in somewhat different initial configurations. The packing fraction and the vertical packing fraction distribution will be different depending on the method chosen and the parameters given to the method.

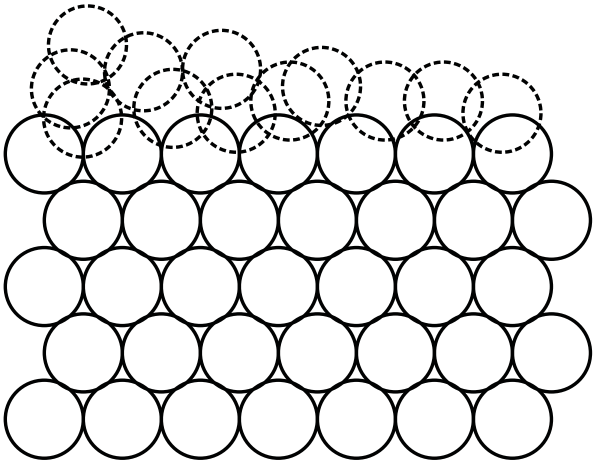

Figure 5.1: Prime method of operation

Chapter 6

Recirculation Simulation

The recirculation is simulated by moving a pebble from the bottom of one of the outlet chutes to an inlet location. Essentially, PEBBLES creates a flat door at the bottom of the outlet chutes. The lowest pebble is removed when the door is opened. If there are multiple outlet chutes, one pebble is removed from one of the chutes. The code cycles through all the outlet chutes, so that each time a different chute is used.

Pebbles are dropped in at the top. The location is selected by the input file. By default they are dropped in at the recirculation height directly above the center of the outlet chute with no initial velocity. They can also be dropped offset from the outlet chute. Alternatively the exact locations and the initial velocity of the pebbles can directly be specified in the geometry section.

Chapter 7

Earthquake Simulation

For each time, the simulation calculates both a displacement and a wall velocity.

For the sum of waves method, the displacement is calculated by:

$$\mathbf{d} = \sum_i \mathbf{D} \left[sin \left((t - S)\frac{2.0\pi}{p} + c \right) + o \right] \tag{7.1}$$

where t is the current time, S is the time the wave starts, p is the period of the wave, c is the initial cycle of the wave, o is the offset, and $\mathbf{D}$ is the maximum displacement.

The velocity is calculated by:

$$\mathbf{m} = \sum_i \frac{2\pi\mathbf{D}}{p} cos \left((t - S)\frac{2\pi}{p} + c \right) \tag{7.2}$$

For the tabular data, the displacement and velocity are calculated by:

$$\mathbf{d} = (1 - o)T_k + oT_{k+1} \tag{7.3}$$

$$\mathbf{m} = \frac{1}{\delta}(T_{k+1} - T_k) \tag{7.4}$$

where T_i is the displacement at the i'th timestep, o is a number between 0 and 1 that specifies where 0 is the start of the timestep and 1 is the end, and δ is the time in seconds between timesteps.

All the displacements are summed and the code then adjusts each pebble's velocity and position:

$$\mathbf{p} = \mathbf{p} + \mathbf{d} \tag{7.5}$$

$$\mathbf{v} = \mathbf{v} + \mathbf{m} \tag{7.6}$$

$$\tag{7.7}$$

Chapter 8

Sample inputs

These are sample inputs that can be used to testing pebbles or as the basis for other inputs.

8.1 Sample Packing Input

```
rem Sample Packing input
reactor_radius 0.0 0.5
pebble_radius 0.025 0.03
runs 10000
kinetic_friction 0.4
static_friction_new3 0.65 1.0e6 0.65 1.0e6 0.01
alpha 0.0001
dash_pot2 200.0 200.0
pebble_pebble_hooke 1.0e6
pebble_reactor_hooke 1.0e6
pebble_density 1883.0 1760.0
random_packing_method 100000
use_simple_integrator
floor_location 0.0
no_linear_static_friction_cutoff
decrease_long_slips 1.1 1.0
sort_pebbles
done
```

8.2 Sample AVR Input

```
rem AVR initial packing
rem AVR 90,000 pebbles in core
rem Single vat with diameter 3 meters pg 89 AVR book
rem Discharge chute 500 mm diameter pg 190 AVR book
```

```
reactor_radius 0.0 1.5
number_of_pebbles 90000
rem 30 degrees from surface, or 60 degrees from wall
cone 0.0 1.73205
start_geometry
inlet 1.0 1.0 2.83 0.0 0.0 0.0
inlet 1.0 -1.0 2.83 0.0 0.0 0.0
inlet -1.0 1.0 2.83 0.0 0.0 0.0
inlet -1.0 -1.0 2.83 0.0 0.0 0.0
rem offset inlet chute.  No good idea on velocity
inlet 0.1 0.0 3.13 -1.0 0.0 0.0
start_union
cylinder 0.15 0.0 1.0 -8.0 8.0
block -0.15 1.0 -8.0 0.15 2.0 8.0
end_union
start_union
cylinder 0.15 0.0 -1.0 -8.0 8.0
block -0.15 -2.0 -8.0 0.15 -1.0 8.0
end_union
start_union
cylinder 0.15 1.0 0.0 -8.0 8.0
block 1.0 -0.15 -8.0 2.0 0.15 8.0
end_union
start_union
cylinder 0.15 -1.0 0.0 -8.0 8.0
block -2.0 -0.15 -8.0 -1.0 0.15 8.0
end_union
start_indentation
start_pattern
rem back plane
plane_vp -1.0 0.0 0.0 0.015 0.0 0.0
rem top plane
plane_vp 0.0 0.0 -1.0 0.0 0.0 0.345
rem side planes
plane_vp -1.0 -2.0 0.0 0.0 0.045 0.0
plane_vp -1.0 2.0 0.0 0.0 -0.045 0.0
rem bottom angled plane
plane_vp -15.0 0.0 3 0.0 0.0 0.0
rem blocking plane (center of reactor)
plane 1.0 0.0 0.0 1.5
end_pattern
min_max 0.0 3.5
start_row
row_min 0.0
angles 7 -15.0 -25.0 -35.0 -45.0 -55.0 -65.0 -75.0
angles 7 -105.0 -115.0 -125.0 -135.0 -145.0 -155.0 -165.0
angles 7 15.0 25.0 35.0 45.0 55.0 65.0 75.0
angles 7 105.0 115.0 125.0 135.0 145.0 155.0 165.0
end_row
start_row
row_min 0.35
angles 6 -20.0 -30.0 -40.0 -50.0 -60.0 -70.0 -80.0
angles 6 -110.0 -120.0 -130.0 -140.0 -150.0 -160.0
angles 6 20.0 30.0 40.0 50.0 60.0 70.0 80.0
angles 6 110.0 120.0 130.0 140.0 150.0 160.0
end_row
start_row
row_min 0.70
angles 7 -15.0 -25.0 -35.0 -45.0 -55.0 -65.0 -75.0
angles 7 -105.0 -115.0 -125.0 -135.0 -145.0 -155.0 -165.0
angles 7 15.0 25.0 35.0 45.0 55.0 65.0 75.0
angles 7 105.0 115.0 125.0 135.0 145.0 155.0 165.0
end_row
start_row
row_min 1.05
angles 6 -20.0 -30.0 -40.0 -50.0 -60.0 -70.0 -80.0
angles 6 -110.0 -120.0 -130.0 -140.0 -150.0 -160.0
angles 6 20.0 30.0 40.0 50.0 60.0 70.0 80.0
angles 6 110.0 120.0 130.0 140.0 150.0 160.0
end_row
start_row
row_min 1.40
angles 7 -15.0 -25.0 -35.0 -45.0 -55.0 -65.0 -75.0
angles 7 -105.0 -115.0 -125.0 -135.0 -145.0 -155.0 -165.0
angles 7 15.0 25.0 35.0 45.0 55.0 65.0 75.0
angles 7 105.0 115.0 125.0 135.0 145.0 155.0 165.0
end_row
start_row
row_min 1.75
angles 6 -20.0 -30.0 -40.0 -50.0 -60.0 -70.0 -80.0
angles 6 -110.0 -120.0 -130.0 -140.0 -150.0 -160.0
angles 6 20.0 30.0 40.0 50.0 60.0 70.0 80.0
angles 6 110.0 120.0 130.0 140.0 150.0 160.0
end_row
start_row
row_min 2.10
angles 7 -15.0 -25.0 -35.0 -45.0 -55.0 -65.0 -75.0
angles 7 -105.0 -115.0 -125.0 -135.0 -145.0 -155.0 -165.0
angles 7 15.0 25.0 35.0 45.0 55.0 65.0 75.0
angles 7 105.0 115.0 125.0 135.0 145.0 155.0 165.0
end_row
start_row
```

```
row_min 2.45
angles 6 -20.0 -30.0 -40.0 -50.0 -60.0 -70.0 -80.0
angles 6 -110.0 -120.0 -130.0 -140.0 -150.0 -160.0
angles 6 20.0 30.0 40.0 50.0 60.0 70.0 80.0
angles 6 110.0 120.0 130.0 140.0 150.0 160.0
end_row
end_indentation
end_geometry
pebble_radius 0.025 0.03
rem 10.0 settle time, 10.0 second recirculation
runs 200000
kinetic_friction 0.4
static_friction_new3 0.65 1.0e6 0.65 1.0e6 0.01
alpha 0.0001
dash_pot2 200.0 200.0
pebble_pebble_hooke 1.0e6
pebble_reactor_hooke 1.0e6
pebble_density 1883.0 1760.0
random_packing_method 100000
dump_frequency 100000000
display_frequency 10000000
position_display_frequency 1000000
use_simple_integrator
floor_location -8.0
recirculate_params 3.5 0.25 10.0
exit_chute 8.33 6.0
seed 512
no_linear_static_friction_cutoff
decrease_long_slips 1.1 1.0
sort_pebbles
done
```

8.3 Sample Packing and Recirculation

```
rem flow test
reactor_radius 0.0 0.30
pebble_radius 0.025 0.03
rem 10% recirculation (500*0.1+1)/0.0001
runs 510000
number_of_pebbles 2000
kinetic_friction 0.4
static_friction_new3 0.65 1.0e6 0.65 1.0e6 0.01
alpha 0.0001
dash_pot2 200.0 200.0
pebble_pebble_hooke 1.0e6
```

```
pebble_reactor_hooke 1.0e6
pebble_density 1883.0 1760.0
random_packing_method 100000
use_simple_integrator
floor_location -8.0
cone 0.0 1.0
recirculate_params 1.6 0.25 1.0
exit_chute 5.5 0.0
no_linear_static_friction_cutoff
decrease_long_slips 1.1 1.0
sort_pebbles
seed 256
done
```

```
pebbles9 pack_and_recirc > pack_and_recirc.out
get_run pack_and_recirc.out 510000 | get_save_info > pack_and_recirc_save
```

8.4 Sample Earthquake input

The following loads from the packing and recirculation example, but could also be generated new by replacing the load_pebble_save with a random_packing_method directive.

```
rem earthquake sample input
reactor_radius 0.0 0.30
pebble_radius 0.025 0.03
runs 100000
number_of_pebbles 2000
kinetic_friction 0.4
static_friction_new3 0.65 1.0e6 0.65 1.0e6 0.01
alpha 0.0001
dash_pot2 200.0 200.0
pebble_pebble_hooke 1.0e6
pebble_reactor_hooke 1.0e6
pebble_density 1883.0 1760.0
use_simple_integrator
floor_location -8.0
cone 0.0 1.0
exit_chute 5.5 0.0
no_linear_static_friction_cutoff
decrease_long_slips 1.1 1.0
sort_pebbles
earthquake_enable 2
earthquake_sine_wave_offset 2.0 8.0 0.248 0.0 0.0 1.0 -1.5707963 1.0
load_pebble_save pack_and_recirc_save
seed 256
```

```
done
```

Chapter 9

Post processing tools

There are several tools that can be used for post processing the PEBBLES output. These include data splitters, data readers and packing fraction calculations.

get_run *time_step_number filename*

zget_run *time_step_number filename.gz*

The script `get_run` and the compressed file version `zget_run` get a specific time-step from the pebbles output. This is used by specifying the filename and the time-step number (like `get_run 1000 packing_output`).

get_save_info *[filename]*

This command takes either a filename or the standard input and outputs only the portions that are valid portions of a save file. This is usually used with the `get_run` command. Typical use would be `get_run pack_and_recirc.out 600000 | get_save_info > pack_and_recirc_save` which can then be used with `load_pebble_save` in a different input.

get_energy *filename*

zget_energy *filename.gz*

Extracts the energy displays from the program. This is a simple way to check the progress of an input since the time is included in the display. It also can be used to check if a packing run has sufficiently slowed down by looking at the linear and rotational energy. Geometry problems also show up if substantial amounts of energy are being injected into the system. However, earthquakes and recirculation also add energy to the system, and the compression energy is not tallied into the total.

split_data *filename directory_name [frequency [skips]]*

This command splits out the different time-steps of the file into the directory. This split up data can then be used by other programs such as plotters or the packing fraction calculation. The split can be done at different powers of 10 frequencies. Also, the only every skip can be calculated. For example, if the output was displayed every 100 times, and every thousandth output was desired, the command would be `split_data filename filename_dir 100 10`

bin_count *filename [cylinder_radius [cylinder_bottom cylinder_top [pebble_radius [center_radius]]]]*

This command calculates the packing fraction from a save file. This is calculated both vertically and radially. In order to do this, it needs to know the cylinder radius and the inner cylinder radius so that the area that is encompassed can be calculated. It also needs to know the pebble radius so that area and volume can be calculated, and for calculating the radial packing fraction, the cylinder bottom and top need to be specified so that area can be calculated correctly. Note that these should be included in the actual area that is filled with pebbles, so that the fraction calculation does not calculate over empty space.

For determining the volume of a sphere that is inside vertical slice the formula

$$a = max(-r, bot - z) \tag{9.1}$$

$$b = min(r, top - z) \tag{9.2}$$

$$v = \pi \left[r^2(b-a) + \frac{1}{3}(a^3 - b^3) \right] \tag{9.3}$$

where r is the pebble radius, *bot* is the bottom of the vertical slice,*top* is the top of the vertical slice and z is the vertical location of the pebble center.

To determine the area inside a vertical and radial slice, two auxilary functions are defined, one which has the area inside a radial 2d slice, and another which has the area outside a radial 2d slice.

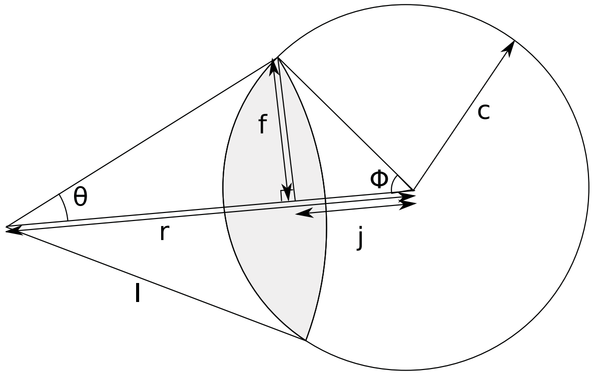

Figure 9.1: Area inside geometry

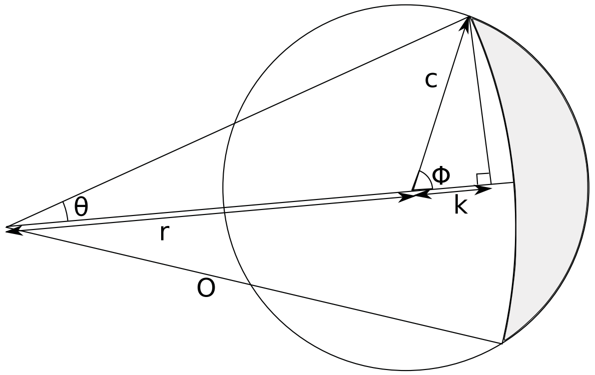

Figure 9.2: Area outside geometry

Figure 9.1 shows the area that is inside both a circle of radius c and a radial slice of I. The circle is r from the center of the radial circle. Auxiliary terms are defined, which include f, the distance from the intersection of the segment of the radial circle perpendicularly to the center line, j the distance to the intersection of f, θ the angle of segment, and ϕ the angle from the segment intersection on the circle side. The area_inside function has the following definition:

$$a_i = 0.0 \qquad \text{if}\, I < r - c \tag{9.4}$$

$$a_i = \pi c^2 \qquad \text{if}\, r + c < I \tag{9.5}$$

$$a_i = \pi I^2 \qquad \text{if}\, I < r + c\, \text{and}\, I < c - r \tag{9.6}$$

$$\text{otherwise} \tag{9.7}$$

$$j = \frac{r^2 + c^2 - I^2}{2r} \tag{9.8}$$

$$f = \sqrt{c^2 - j^2} \tag{9.9}$$

$$\theta = 2 \arccos \frac{I^2 + r^2 - c^2}{2Ir} \tag{9.10}$$

$$\phi = 2 \arccos \frac{r^2 + c^2 - I^2}{2rc} \tag{9.11}$$

$$a_i = \frac{1}{2} c^2 \phi + \frac{1}{2} I^2 \theta - fr \tag{9.12}$$

Figure 9.2 shows the area that is outside the radial slice, but inside the circle. The radial slice has a radius of O. The new auxiliary term k is the distance from the circle's center to the perpendicular intercept. The area_outside function has the following definition:

$$a_o = 0.0 \qquad \text{if}\, O > c + r \tag{9.13}$$

$$a_o = \pi c^2 \qquad \text{if}\, c - r > O \tag{9.14}$$

$$a_o = \pi c^2 - \pi O^2 \qquad \text{if}\, O < r + c\, \text{and}\, O < c - r \tag{9.15}$$

otherwise (9.16)

$$k = \frac{O^2 - r^2 - c^2}{2r} \tag{9.17}$$

$$m = \sqrt{c^2 - k^2} \tag{9.18}$$

$$\theta = 2 \arccos \frac{k + r}{O} \tag{9.19}$$

$$\phi = 2 \arccos \frac{k}{c} \tag{9.20}$$

$$a_o = (\frac{1}{2} c^2 \phi - mk) - (\frac{1}{2} O^2 \theta - m(k + r)) \tag{9.21}$$

Then, the total volume in a radial slice can be determined from the computation:

$$a = max(-r, bot - z) \tag{9.22}$$

$$b = min(r, top - z) \tag{9.23}$$

$$v_t = \pi \left[R^2 (b - a) + \frac{1}{3} (a^3 - b^3) \right] \tag{9.24}$$

$$v_i = \int_a^b \text{area\_inside}(c = \sqrt{R^2 - z^2}) dz \tag{9.25}$$

$$v_o = \int_a^b \text{area\_outside}(c = \sqrt{R^2 - z^2}) dz \tag{9.26}$$

$$v = v_t - v_i - v_o \tag{9.27}$$

Chapter 10

Estimating Run-times

The code theoretically scales as a function of the number of time-steps times the number of pebbles. In practice, the time for running increases somewhat faster because there are less edge pebbles (which have fewer pebble to pebble contacts) and less fit in the cache.

Bibliography

[1] S. T. Thornton and J. B. Marion, *Classical Dynamics of Particles and Systems, 5th Ed.*, Brooks/Cole, Belmont, California, USA (2004).

[2] Jacques Duran, *Sands, Powders, and Grains: An Introduction to the Physics of Granular Materials*, Springer, New York, New York, USA (1999)

[3] Matthias Sperl, "Experiments on corn pressure in silo cells – translation and comment of Janssens paper from 1895," *Granular Matter*, **8**, pp. 59-65 (2006)

[4] D. M. Walker, "An approximate theory for pressures and arching in hoppers," *Chemical Engineering Science*, **21** pp. 975-997 (1966)

[5] R. D. Mindlin and H. Deresiewicz, "Elastic Spheres in Contact Under Varying Oblique Forces," *Journal of Applied Mechanics*, **20**, pp. 327-344 (1953).

[6] P. A. Cundall and O. D. L. Strack, "A discrete numerical model for granular assemblies," *Géotechniqe*, **29**, pp. 47-65 (1979).

[7] L. Vu-Quoc, X. Zhang and O. R. Walton, "A 3-D discrete-element method for dry granular flows of ellipsoidal particles," *Computer Methods in Applied Mechanics and Engineering*, **187**, pp. 483-528 (2000)

[8] Leonardo E. Silbert, Deniz Ertas, Gary S. Grest, Thomas C. Halsey, Dov Levine, and Steven J. Plimpton, "Granular flow down an inclined plane: Bagnold scaling and rheology," *Physical Review E*, **64** 051302 (2001)

[9] James W. Landry, Gary S. Grest, Leonardo E. Silbert, and Steven J. Plimpton, "Confined granular packings: Structure, stress, and forces," *Physical Review E*, **67**,041303 (2003)

[10] Chris H. Rycroft, Martin Z. Bazant, Gary S. Grest, and James W. Landry, "Dynamics of random packings in granular flow," *Physical Review E*, **73** 051306 (2006)

[11] Chris H. Rycroft, Gary S. Grest, James W. Landry, and Martin Z. Bazant, "Analysis of granular flow in a pebble-bed nuclear reactor," *Physical Review E*, **74**, 021306 (2006)

[12] R. Wait, "Discrete Element Models of Particle Flows", *Mathematical Modeling and Analysis I*, **6**, pp. 156-164 (2001)

[13] Zi Lu, Mohamed Abdou, Alice Ying, "3D Micromechanical Modeling of Packed Beds", *Journal of Nuclear Materials*, (2001)

[14] Rmi Jullien, Andr Pavlovitch, Paul Meakin, *Journal Phys. A: Math. Gen.* , "Random Packings of Spheres built with sequential models", (1992)

[15] W. Soppe, *Powder Technology*, "Computer Simulation of Random Packings of Hard Spheres", (1990)

[16] Hannsjrg Freund, Thomas Zeiser, Florian Huber, Elias Klemm, Gunther Brenner, Franz Durst, Gerhand Emig, *Chemical Engineering Science*, "Numerical Simulations of single phase reacting flows in randomly packed fixed-bed reactors and experimental validation", (2003)

[17] G. A. Kohring, *Journal de Physique I*, "Studies of Diffusional Mixing in Rotating Drums via Computer Simulations", (1995)

[18] J. M. Haile,*Molecular Dynamics Simulation*, 1997

[19] G. H. Ristow, *Flow Properties of Granual Materials in Three-Dimensional Geometries*, (1998)

[20] Loc Vu-Quoc, Xiang Zhang, *Mechanics of Materials*, "An accurate and effcient tangential force-displacement model for elastic frictional contact in particle-flow simulations", (1999)

[21] R. Wait, *Mathematical Modeling and Analysis I*, "Discrete Element Models of Particle Flows", (2001)

[22] Abderrafi M. Ougouag, Joshua M. Cogliati and Jan-Leen Kloosterman, "Methods for Modeling the Packing of Fuel Elements in Pebble Bed Reactors", *ANS Topical Meeting in Mathematics and Computation, Supercomputing, Reactor Physics and Nuclear and Biological Applications* Palais des Papes, Avignon, France, September 12-15, 2005, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2005)