Abstract

A granular media is a large conglomeration of discrete macroscopic particles. When compressed, the media is known to undergo a rigidity phase transition. We numerically model a $2D$ granular aggregate confined between two walls to demonstrate that this rigidity phase transition occurs at a specific packing density, ν_c. An emergent network of force chains is noted at and above this critical density. System properties were varied to study the effect on the phase transition and contact forces between particles. 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.

Chapter 1

Model Implementation

1.1 Molecular Dynamics Model

The Molecular Dynamics method [Pös05; Ris94; Cun79] for simulating granular media treats the particles as inelastic disks with a translational degree of freedom. The MD method employs an 'a posteriori' collision detection algorithm, i.e. collisions between particles are detected *after* the initial contact.

Two particles i and j are said to interact if the distance between their centers, $\mathbf{r}_{ij}$ is less than the sum of their radii, $\mathbf{R}_i + \mathbf{R}_j$. This overlap is interpreted as the physical deformation of the particles. The particles are non-cohesive, so when $\mathbf{R}_i + \mathbf{R}_j - \mathbf{r}_{ij} \geq 0$ the force is zero.

1.2 Calculation of the Normal Force

During a particle pair interaction, i.e. when $\mathbf{R}_i + \mathbf{R}_j - \mathbf{r}_{ij} < 0$ the force due to particle j on particle i has a Normal component. This Normal component is modeled by a linear spring and dashpot (Figure 1.1, Overleaf) and is calculated from the formula [Aha99]:

$$\tilde{\mathbf{F}}_{ij} = [k_n(\mathbf{R}_i + \mathbf{R}_j - \mathbf{r}_{ij}) - \gamma_n \mathbf{m}_{ij}\tilde{\mathbf{v}}_{ij}]\hat{\mathbf{n}} \tag{1.1}$$

where,

$\tilde{\mathbf{F}}_{\mathrm{ij}}$ = Normal force between particles i and j,

k_n = Normal elastic constant,

γ = Normal dissipation factor,

m_{ij} = Harmonic mass of particle i and j,

$\tilde{\mathbf{v}}_{ij}$ = Relative velocity between i and j,

$\hat{\mathbf{n}}$ = Normal unit vector.

From this equation we can infer that the contact force between two particles is proportional to the overlap/compression.

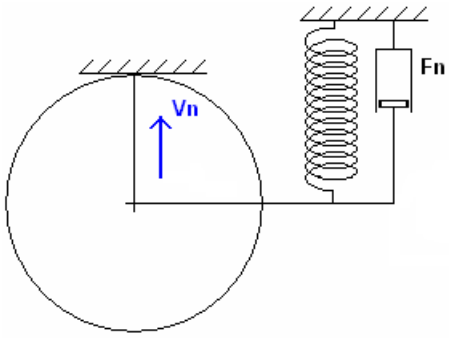

Figure 1.1: *Grains are modeled as a linear spring-dashpot.*

1.3 The Velocity Verlet Algorithm

The Velocity Verlet algorithm [Gio96] is used to calculate the positions and velocities at any instant in time for all particles. The Velocity Verlet is a 2^{nd} order finite difference approximation to the equations of motion. Time is divided into time-steps, $\Delta\mathbf{t}$. At time $\mathbf{t}$ the positions and velocities of all grains are known. From this information we can use (Equation 1.1) to calculate the force acting on each grain. The force is resolved into x and y components and from Newtons 2^{nd} law we can calculate the corresponding acceleration components of particle i:

$$\vec{a}_x(\mathbf{t}) = \frac{\tilde{\mathbf{F}}_x}{m_i}$$

$$\vec{a}_y(\mathbf{t}) = \frac{\tilde{\mathbf{F}}_y}{m_i}$$

The components of acceleration were then used to calculate the new x and y velocity components of particle i:

$$\tilde{\mathbf{v}}_x(\mathbf{t}) = \tilde{\mathbf{v}}_x(\mathbf{t} - \Delta\mathbf{t}) + \Delta\mathbf{t}\frac{\tilde{\mathbf{a}}_x(\mathbf{t} - \Delta\mathbf{t}) + \vec{\mathbf{a}}_x(\mathbf{t})}{2} \tag{1.2}$$

$$\tilde{\mathbf{v}}_y(\mathbf{t}) = \tilde{\mathbf{v}}_y(\mathbf{t} - \Delta\mathbf{t}) + \Delta\mathbf{t}\frac{\tilde{\mathbf{a}}_y(\mathbf{t} - \Delta\mathbf{t}) + \vec{\mathbf{a}}_y(\mathbf{t})}{2} \tag{1.3}$$

The current position, velocity and acceleration components are then used to calculate the x and y position components of particle i at the next value of $\Delta\mathbf{t}$:

$$x(\mathbf{t} + \Delta\mathbf{t}) = x(\mathbf{t}) + \Delta\mathbf{t}\vec{\mathbf{v}}_x(\mathbf{t}) + \frac{(\Delta\mathbf{t})^2}{2}\vec{a}_x(\mathbf{t}) \tag{1.4}$$

$$y(\mathbf{t} + \Delta\mathbf{t}) = y(\mathbf{t}) + \Delta\mathbf{t}\vec{\mathbf{v}}_y(\mathbf{t}) + \frac{(\Delta\mathbf{t})^2}{2}\vec{a}_y(\mathbf{t}) \tag{1.5}$$

1.4 Non-Dimensionalisation

When experimenting on a physically massive or minute true-scale system, it is best to experiment on a scaled-up (or scaled-down) system. All values are non-dimensionalised to bring them close to unit values with similar scaling, i.e. very large values will not be compared to very small values. In simulations presented here, all results are in dimensionless form.

Distance, $\mathbf{x}$, is measured in units of the average particle diameter $\mathbf{x}_0$. Time, $\mathbf{t}$, is scaled by the undissipated elastic wave travel time across this distance $\mathbf{t}_0$. Velocity, $\mathbf{v}$, is scaled by the elastic wave speed $\mathbf{v}_0$.

The dimensionless variables are shown below, denoted by asterisks:

$$\mathbf{x} = \mathbf{x}^*\mathbf{x}_0 \qquad \mathbf{t} = \mathbf{t}^*\mathbf{t}_0 \qquad \mathbf{v} = \mathbf{v}^*\mathbf{v}_0$$

where,

$\mathbf{x}_0 = 2\bar{R}$

$\mathbf{t}_0 = \sqrt{\dfrac{\bar{m}}{k_n}}$

$\mathbf{v}_0 = \dfrac{\mathbf{x}_0}{\mathbf{t}_0}$

$\bar{R}_i$ is the average particle radii, $\bar{m}_i$ is the mass of a particle with the average radius and k_n is the normal elastic constant.

Energy loss is governed by a normal restitution coefficient

$$e_n = \exp(\frac{-\gamma_n t_{col}}{2}) \tag{1.6}$$

where,

$$t_{col} = \frac{\pi}{\sqrt{\frac{k_n}{m_{ij}} - \frac{\gamma^2}{4}}} \tag{1.7}$$

See (Appendix **??**) for a description of the above.

In simulations presented here:

$$k_n = 1 \qquad m_i = \frac{R_i}{\bar{R}_i^2} \qquad \gamma_1 = 1 \qquad \gamma_2 = 1.5$$

So grains of average radii have a mass of 1.

1.5 Selection of Particle Radii

Polydispersivity was introduced to discourage crystallisation effects. Particle radii were randomly generated from a normal distribution with a mean, $\bar{R}_i = 0.5$, and a standard deviation, $\sigma = 0.1$. Particle radii ranged from $[0.2241, 0.7402]$.

1.6 Boundary Conditions

In order to compress the system of particles to a desired packing fraction, ν, the particles had to be confined to a box of height H and width W. Two types of boundary conditions were used to achieve this.

1.6.1 Static Boundaries

Static Boundaries are placed at the top and bottom of the simulation box to contain particles moving in the vertical direction. The mass and radius of each boundary particle is the average value, hence they have a non-dimensionalised width and mass of 1. The static boundaries, or wall particles, (Figure 1.2 as blue particles), form rough, rigid walls which contain the free particles, (Figure 1.2 as red particles), moving in the vertical direction from escaping the system. The bottom boundary is fixed in position while the top boundary can translate downwards to compress the free particles. Horizontal motion of the static boundaries was not allowed.

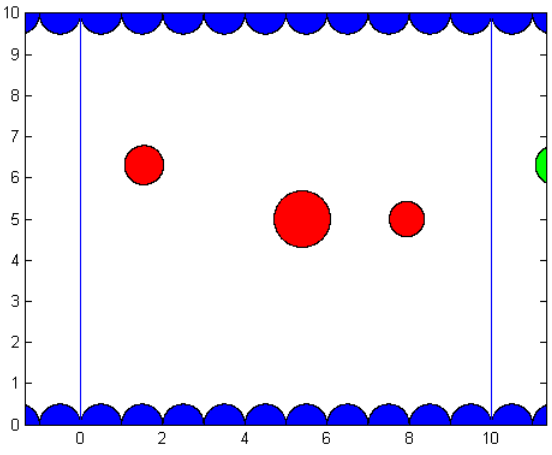

Figure 1.2: *Static boundaries.*

1.6.2 Periodic Boundaries

The number of particles in a granular media can number from hundreds to millions. Since carrying out tests on a system size that large would be expensive and time-consuming, making the system unbounded in the horizontal direction would be ideal, but in order to compress the system and reach desired values of ν we must somehow make the system bounded but *free of physical walls*. To do this we employ the use of **Periodic Boundary Conditions** [Pös05; Rap04]. By creating PBC's, any particle moving horizontally which crosses a boundary on one side will immediately re-enter on the other side. In effect we have created a finite system of particles which mimics an infinite system of particles. Free particles can still interact with each other near the centre of the system and free particles near the horizontal boundaries will interact with image particles of free particles. (Figure 1.3) below gives an example of how the PBC's in this simulation work.

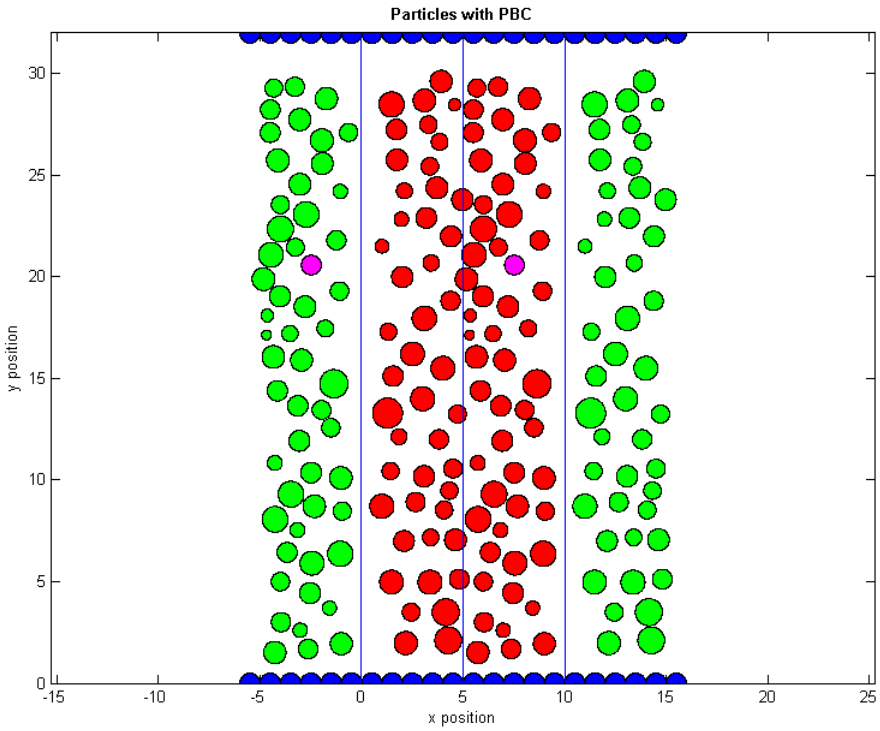

Figure 1.3: *Periodic boundaries.*

Let $\mathbf{x}_i$ represent the x-position of the centre of a free particle and W represent the non-dimensionalised width of the system, (denoted as the distance between the two vertical blue lines, $W = 10$ non-dimensionalised distance units)
If

$$0 \leq \mathbf{x}_i \leq \frac{W}{2}$$

then the free particle is assigned an image particle at x-position at

$$\mathbf{x}_i + W$$

Similarly, if

$$\frac{W}{2} < \mathbf{x}_i \leq W$$

then the free particle is assigned an image particle at x-position at

$$\mathbf{x}_i - W$$

In (Figure 1.3) this is emphasised by the two purple particles. The free particle lies within the region 5 and 10 which corresponds to $\frac{W}{2} < \mathbf{x}_i \leq W$ and as can be seen, an image particle is assigned at $\mathbf{x}_i - W$.

1.7 Flow Diagram

(Figure 1.4,Overleaf) shows a simplified flow diagram of the main algorithm. For every value of $\Delta\mathbf{t}$ the x and y components of the total force on each particle are calculated. To sum the total force on each a particle, pick a particle i, then cycle through all the remaining particles, j, in the system starting with $j = i + 1$. By taking into account Newtons 3^{rd} law, the force exerted on particle i by particle j is equal and opposite to the force exerted on particle j by particle i:

$$\vec{\mathbf{F}}_x^{ij} = -\vec{\mathbf{F}}_x^{ji} \tag{1.8}$$

$$\vec{\mathbf{F}}_y^{ij} = -\vec{\mathbf{F}}_y^{ji} \tag{1.9}$$

By incorporating (Equations 1.8 and 1.9) into the main algorithm, the number of calculations per particle *decreases* by 1 as i increases by 1.

To put this into perspective, if there are N free particles in the system, then the number of calculations carried out on any free particle, per value of $\Delta\mathbf{t}$, is reduced from N to $\dfrac{N}{2}$ calculations.

When all of the forces on particle i are known, they are summed to determine to x and y components of the force:

$$\vec{\mathbf{F}}_x^{i} = \sum_{j=i+1}^{N} \vec{\mathbf{F}}_x^{ij} \tag{1.10}$$

$$\vec{\mathbf{F}}_y^{i} = \sum_{j=i+1}^{N} \vec{\mathbf{F}}_y^{ij} \tag{1.11}$$

where,

$\vec{\mathbf{F}}_x^{ij} = x$-component of the force on i by particle j.

$\vec{\mathbf{F}}_y^{ij} = y$-component of the force on i by particle j.

$N =$ Total number of particles i can interact with.

The equations on the previous page are used to carry out force calculations on each free particle from other free particles, image particles and wall particles. Force calculations on image particles and wall particles are not carried out. These particles are not under observation and removing those calculations from the main algorithm reduces the runtime of the simulation. Please refer to (Appendix B) to see this in C code.

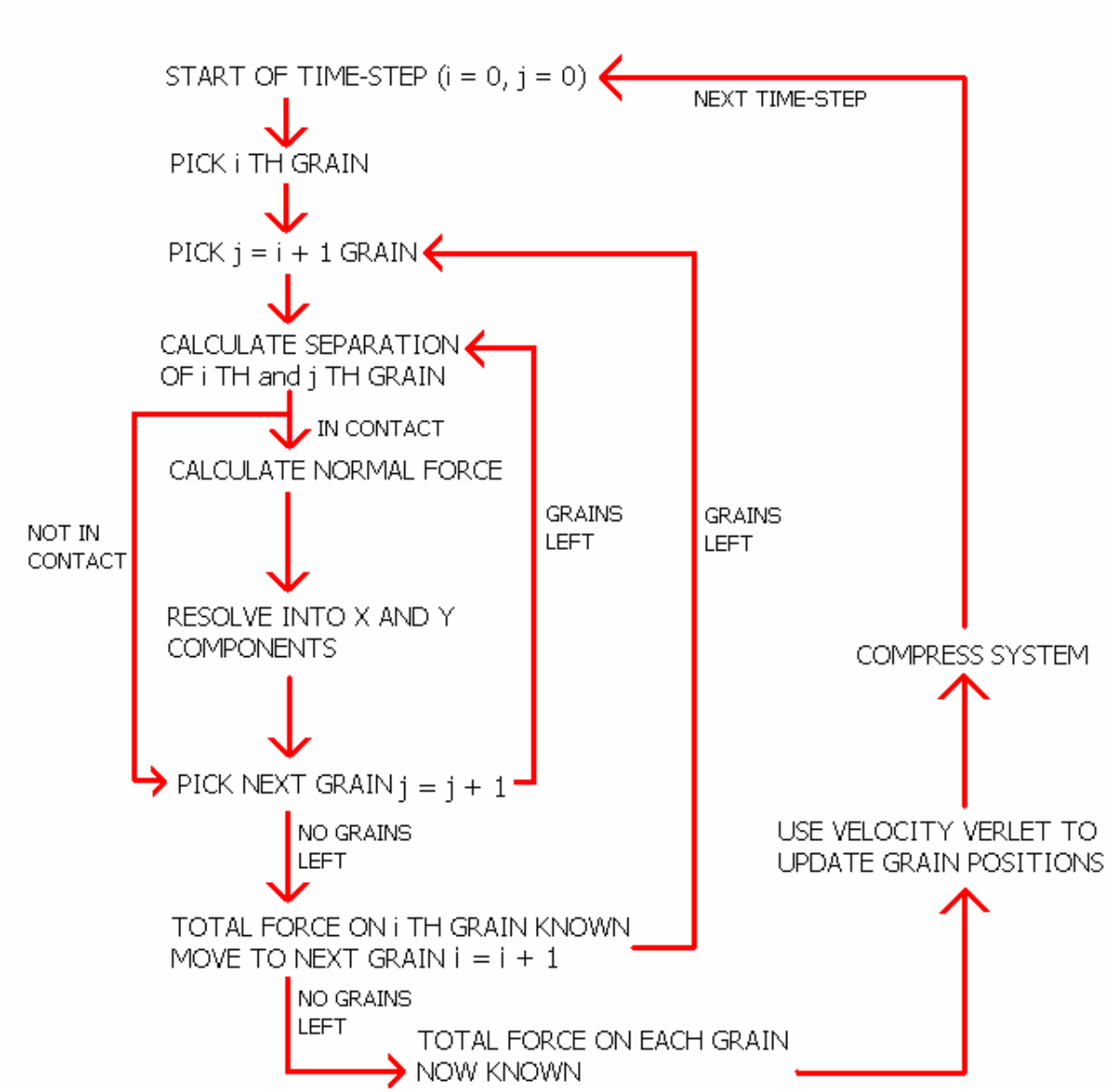

Figure 1.4: *Flow diagram.*

Appendix A

Approximating zero in the model

This appendix describes how quasi-static states were approximated as static states in the model.

In the simulation particles were deemed to be at rest, i.e zero velocity, when sum total kinetic energy of the free particles was less than or equal to the relaxation level. Ideally, zero would be an appropriate value but when running numerical simulations, it must be remembered that calculations are carried out in double precision (accurate to 16 decimal places) and exhibit roundoff errors in the last few terms. In short, calculations will *never* be exactly zero. Zero must be approximated for simulations and the choice of the approximated value will have an effect on runtime and accuracy of results. By approximating zero to a small value you will increase the runtime significantly. To large an approximation will give faster runtimes but yield less accurate results of static states. Aharonov & Sparks approximated zero as the point where the average velocity of all particles was $\leq 1x10^{-8}$ of the original average velocity.

In this simulation, the average kinetic energy of all free particles was chosen as the value from which zero would be approximated. When the average kinetic energy had decayed to $\leq 1x10^{-6}$, the system was considered to be at rest.

A similar problem occurred when calculating the packing fraction, ν. Measurements of ν were to be made in steps of $\Delta\nu = 0.01$. It seems trivial enough to do this, but we are used to working with less than 16 decimal places. When this was run in C code, the program would only measure the packing fraction if it increased in steps of $\Delta\nu = 0.0100000000000000$! Once again, due to roundoff errors, this proved to be impossible. To overcome this problem, a tolerance level had to be established. Instead of measuring just one value of ν for each value of $\Delta\nu$ on average 7 values of $\Delta\nu$ were measured. The C code is shown below, $max\_energy = 1x10^{-6}$:

```
if((kinetic_energy) <= max_energy)
 {
  /*Calculate Packing density*/
  packing = packing_density(free_particle,wall_particle);

  /*Check if packing is within tolerance*/
  if((fmod((packing*factor),divisor)<=tolerance))
   {
    printf("%lf\n",packing);
    coord_number = coordination_number(free_particle);   /*Calculate Z*/
    potential_energy = potential(x,image_x,wall_x);      /*Calculate Ep*/
   }
  /*Move the compression wall down*/
  translate_wall(wall_particle);
 }
```

You will also notice a value in line 4 of the code snippet above a two constants, *factor* and *divisor*. These were included to circumvent a problem that appears to be isolated to C compilers only. When using the $fmod(x, y)$ function in the *math.h* library, caution must be taken to ensure that $y > 0.01$ or else the fmod function

will return x as its result. This error is not noted in any texts on C but has been mentioned in online forums.

Appendix B

Force Calculation

This appendix reproduces the code used to calculate the force between the different particles previously discussed in (Section 1.7) and (Equation 1.1).

```
{
     int i, j;
     double nx,ny,R1R2,r1r2,red_mass,rel_vel_x,rel_vel_y;

     /*Reset the Coordination number to zero for all free_particles*/
     for(i = 0; i < n_max; i++)
      {
       free_particle[i][10][0] = 0.0;
       image_free_particle[i][10][0] = 0.0;
      }

     for(i = 0; i < 44; i++)
      {
       wall_particle[i][10][0] = 0.0;
      }
/*_____________________________________________________________________________*/
     /*Detect a collision between two free_particles*/
     for (i = 0; i < n_max; i++)
      {
       for (j = i+1; j < n_max; j++)
        {
         R1R2 = free_particle[i][9][0] + free_particle[j][9][0];
         r1r2 = sqrt((pow((free_particle[j][2][0]
          - free_particle[i][2][0]),2.0))
                +(pow((free_particle[j][3][0]
                - free_particle[i][3][0]),2.0)));
         red_mass = ((free_particle[i][8][0]*free_particle[j][8][0])
```

```
         /(free_particle[i][8][0] + free_particle[j][8][0]));
 rel_vel_x = (free_particle[i][0][0] - free_particle[j][0][0]);
        rel_vel_y = (free_particle[i][1][0] - free_particle[j][1][0]);

        /*If there is a collision*/
        if(r1r2 < R1R2)
         {
          /*Increase the Coordination Number by 1.0 for each collision*/
          free_particle[i][10][0] += 1.0;
          free_particle[j][10][0] += 1.0;

          /*Normal force x, component*/
          nx = (free_particle[i][2][0] - free_particle[j][2][0])/r1r2;
          /*Normal force y, component*/
          ny = (free_particle[i][3][0] - free_particle[j][3][0])/r1r2;

          /*Overlap distance of the two free_particles*/
          x[i][j] = R1R2 - r1r2;
          /*Repulsive Force, x component*/
          free_forces[i][j][0] = ((kn*(x[i][j]))
           -((gamma*red_mass)*(rel_vel_x*nx)))*nx;
          /*Repulsive Force, y component*/
          free_forces[i][j][1] = ((kn*(x[i][j]))
           -((gamma*red_mass)*(rel_vel_y*ny)))*ny;
         }
        else
         {
          free_particle[i][10][0] += 0.0;
          free_particle[j][10][0] += 0.0;
          nx = 0.0;
          ny = 0.0;
          x[i][j] = 0.0;
          free_forces[i][j][0] = 0.0;
          free_forces[i][j][1] = 0.0;
         }
        /*Use the symmetry of free_forces to reduce the number of calculations*/
        free_forces[j][i][0] = -(free_forces[i][j][0]);
        free_forces[j][i][1] = -(free_forces[i][j][1]);
        x[j][i] = x[i][j];
       }
     }
}
```

Bibliography

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

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

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

[Pös05] T. Pöschel and T. Schwager. *Computational Granular Dynamics: Models and Algorithms*. Springer, 2005.

[Rap04] D. Rapaport. *The Art of Molecular Dynamics Simulation*, pp. 15–17. Cambridge University Press, 2004.

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