THE UNIVERSITY OF CALGARY

Discrete Mechanics of Granular Matter

by

Dmitri Gavrilov

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE CROSS-DISCIPLINARY DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MECHANICAL AND MANUFACTURING ENGINEERING

and

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

JUNE, 1999

ABSTRACT

A discrete model of granular type materials is presented in this dissertation. The model has three submodels: the Molecular Dynamics model, the Quasi-Static Model and the Multibody Dynamics model. Each of the submodels is a valid granular material model with its own applicability limits. By combining the three submodels, a unique model with a very wide range of applicability is obtained. The model can adapt to dynamic changes in the granular system by "morphing" into the most applicable submodel.

The Multibody Dynamics model is presented first, together with the underlying equations and algorithms. The application of this model to the simulation of a shaker ball mill is presented. Two grinding models, a statistical and an analytical, are also presented.

Then, the Quasi-Static model is presented. The original Recursive Inverse Matrix Algorithm (RIMA) that enables the inverse stiffness matrix of a granular system to be maintained dynamically is presented. It is then shown how the model handles local instabilities in the system, such as slips and micro-avalanches. The application of the Quasi-Static model to the simulation of a silo is presented.

Finally, the unified model based on the original Multibody Dynamics approach is presented. The model contains the Molecular Dynamics and Quasi-Static models as submodels and uses them to represent various parts of the granular system depending on the current conditions. The application of the unified model to the simulation of hopper flow is presented.

ACKNOWLEDGEMENTS

I gratefully acknowledge the support, guidance and encouragement that I received from my supervisor, Dr. Oleg Vinogradov, throughout the course of my studies. I thank him for careful reading, valuable suggestions and helpful discussions of my dissertation. I would like to thank my co-supervisor, Dr. Jon Rokne, for his help, advice and encouragement.

I am thankful for the financial support in the form of scholarships from the Killam Foundation, Pan-Canadian Petroleum Corporation and Robert B. Paugh Foundation.

Finally, my special thanks to my son, Andrei, for giving me inspiration and motivation to explore. And my very special thanks to my wife, Marina Gavrilova, for her love and support. It is to her that I dedicate my dissertation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1. Computer simulation as a virtual experimentation tool

Computer simulation is an extremely powerful tool, which is used in both research and industry applications. Before computers were introduced, physical *experimentation* was the primary tool available to researchers to test their theories and hypotheses. It is through experimentation that the first laws of physics, such as the law of gravitation, were discovered. Experimentation was also used by engineers, when they were testing various designs and materials for building machines and devices.

Experimentation was often a difficult, expensive and even dangerous practice. This is the main reason why people started to come up with various rules and algorithms, which would allow them predicting the outcome of an experiment for a given set of initial conditions without actually performing the experiment. These rules and algorithms gradually evolved into some generic formulas or principles, which became known as laws of physics. Discovering such laws has become one of the primary objectives of scientific research.

Experimentation always will be used as a major tool used by scientists and engineers. It is considered as an ultimate approach to validate a discovered law, proposed theory or design. The main advantage of this approach is that it is usually very close to practice, i.e. to the final environment, where the law or design will be practically utilized. Also, experimentation does not usually require introduction of any hypotheses, constraints or models in addition to the one being tested.

The disadvantages of the approach include such factors as the cost of building the experimental equipment (considering that in many cases it will become useless or even destroyed during the experiment), time factor (some experiments must run for a prolonged period of time), as well as other factors. The main disadvantage of the approach is that it usually requires introduction of special measurement devices into the

system being tested. These devices are "foreign" to the system and their presence may falsify the results of the experiment. Another problem is that sometimes the parameters of interest simply can not be measured directly either because of the extreme range of the parameter values (such as the temperature at the core of a nuclear reactor), or because of the very small scale of the parameter values (such as the interaction between molecules of a gas). To measure such parameters, certain implicit methods are used. This, however, introduces additional inaccuracy into the measurement. In addition, a theory or a law allowing the interpretation of the implicitly measured parameter values must be developed and tested.

An alternative to the experimentation approach is the *logical inference* approach. This approach is the direct opposite to experimentation, because it is completely theoretical (paper-based). The approach assumes a set of *axioms*, or generally accepted facts and rules, and tries to logically infer a new law (or a theorem) based on the axioms and some generic induction rules. Very often, this approach uses a set of previously obtained and confirmed laws and theories. The sequence of inference steps leading to the statement of a law (or theorem) is called the *proof*.

The logical inference approach is the primary discovery tool in many areas, mathematics being the key example. The main advantage of the approach is that the results obtained are usually absolutely precise (as opposed to the approximately measured experimentation results), well defined, logically sound and concise. The main disadvantage of the approach is that it can be extremely difficult or even impossible to obtain a proof for a law. Using logical inference to prove theories is an art, i.e. in most cases it can not be standardized or automated. Still, logical inference remains the only approach, which ultimately yields 100% correct results (unless, of course, there is an error in the proof).

Logical inference and experimentation approaches were often used hand in hand, when an experiment would give a hint to an existence of a fundamental law, which was later proven logically with the logical inference approach.

Simulation is the third approach available to test a theory or a design. It combines the advantages (and disadvantages) of both experimentation and logical inference. Simulation can be generically considered as *numerical experimentation*, where a numeric (virtual) experiment (as opposed to a physical experiment) is performed either on paper or on a computer. Simulation was known before computers were introduced, but it became widely used only after computers started to appear in scientific and industry institutions. The reason for this is that simulation usually involves very large amounts of computation, which is practically impossible to perform without a computer. This is why the terms "simulation" and "computer simulation" are often used as synonyms.

Similarly to the logical inference approach, a simulation experiment is based on a set of axioms and generally accepted laws, which constitute the basis of the *simulation model*. Another part of the simulation model is a set of assumptions and constraints defining the applicability of the model. The final part of a simulation model is the set of evolution laws, which define how the model evolves or changes during the virtual experiment. The evolution laws are usually based on some known laws.

A simulation experiment is conducted very much like a practical experiment. An initial set of parameters is entered into the model. Then, the evolution laws are applied to the parameters in an iterative manner and the parameters of the model are being observed as they evolve.

A simulation model usually has to be *validated*. This is achieved by comparing the simulation results to the experimental results obtained for a given initial parameter set. Another validation technique is related to observing certain behaviors in the simulated system, which are usually found in physical systems (such as conformance to certain physical laws). One of the widely used validation techniques is based on various conservation principles (energy conservation, for example). The conservation principles are often used by the simulation model itself to control the numerical errors and, possibly, to correct the system evolution paths. Another validation technique is comparing the

results produced by different simulation models. This validation technique is, however, somewhat limited and, thus, can not be used alone to validate a model.

The main advantage of the simulation approach is that it is usually virtual, i.e. it is not required to build any experimental equipment (except for acquiring the computer hardware, which can however be considered as ultimately universal experimental equipment). Another major advantage of virtual experimentation is that many of the parameters of interest may be readily available in the model or can be computed based on the model parameters. This might include the parameters, which are impossible to measure in a physical experiment. In some cases, simulation can obtain a result much faster than a practical experiment would take. This is based on the fact that simulation is usually based on the notion of "virtual time", which can be artificially sped up or slowed down.

The main disadvantage of the simulation approach is that the validity of the experimental results is heavily dependent on the quality of the simulation model being used. It might be easy to define a realistic and correct simulation model, but it is much more complicated to define the applicability limits of this model. Also, it often happens that after an attractive model has been defined, it starts to be used in many different applications without considering if it is suited for them. This might lead to falsification of simulation results. The construction of a good simulation model is also complicated by the fact that it might be impossible to know in advance which of the factors influencing the model are important and which are not important, because this can only be found out as a result of simulation and validation.

Another problem related to the simulation approach is the fact that a simulation model usually includes a multitude of external parameters, which affect the performance and sometimes the validity of the model. Some of these parameters can be related to physical quantities (such as stiffness or damping coefficients), and others are completely computation-related (such as various tolerance values or the integration timestep). Ultimately, the validity of the simulation experiment depends on the proper selection of

these values. Very often, as a simulation model is being validated, the acceptable values for the parameters are being obtained by trial-and-error. Such a validation experiment does, indeed, prove that the model is correct. The practical applicability of the model is however limited to the cases that are similar to the one for which the validation was performed, because for a different set of initial conditions the values of model parameters must be updated. If a simulation model does not include guidelines (or, at least, recommendations) for parameter selection, this might render the model completely unusable.

The simulation approach becomes inefficient compared to practical experimentation when the simulated system is very complex or when very strict requirements are imposed on the accuracy of computations. For example, the simulations of granular materials are generally constrained by the total number of particles in the system. Even on a supercomputer, simulating several thousands of particles might become a quite time-consuming task (see, for example, [Potapov and Campbell 96]). This is in contrast to practical experimentation, where working with billions of sand particles is not a problem. Because of this computational efficiency limitation, many researchers are working on parallelizing the simulation algorithms, which is an obvious direction towards improving the simulation efficiency [Baezner et. al. 94]. The other direction is, of course, getting a faster computer, but this might be limited by technological and financial constraints.

1.2. Computer simulation of granular materials

Granular materials include such physical systems as sand, flowing ice, grain in an elevator or on a transporter, coal in a hopper, and many others. Applications related to granular materials are found in many areas, including mechanical engineering, chemical engineering, civil engineering, manufacturing, mining, agriculture and many others [Rapaport 95].

Granular matter is a very interesting physical medium, which possesses the qualities of solids, liquids and sometimes even gases [Rapaport 95]. Another extremely interesting

quality of granular systems is their chaotic nature, which is related to the fact that they consist of many elements and, thus, their behavior is defined as a combination of behaviors of smaller elements. Therefore, the laws describing the granular matter often have a statistical or "average" meaning (such as the pressure on a silo wall, which is averaged among many particle contacts in a unit area). It has also been noted that certain types of granular systems exhibit the "self-organized criticality" (SOC) phenomenon ([Bak et. al. 88, Carreras et. al. 96, Dendy and Helander 97]). The main property of a SOC system is that it is a stochastic system that does not have a temporal or spatial scale.

Because of the complexity of a granular system, explicit laws defining the system can not be obtained. Thus, a number of different theories and models have been introduced, ranging from continuum-based models to cellular automata models (see Section 2.2 below for a detailed review). Different approaches provide various degrees of accuracy and have various applicability domains.

Because of the nature of a granular system, it is difficult to measure the internal properties of the system experimentally. A granular system is one of the systems, where the behavior is altered by the introduction of external objects [Claudin and Bouchaud 97]. Thus, only external or implicit measurements are possible. Therefore, simulation is considered to be a very attractive approach to investigate granular systems. It might, however, be quite difficult to validate the simulation results because the original system's parameters are difficult to measure. Thus, a simulation model can be validated by comparing it to another simulation model.

In this work, an attempt was made to design and develop a universal granular material model, which can be used in a wide range of application domains, from quasi-static systems (such as stable sand or silos), to intermediate systems (such as flowing ice or hoppers), to dynamic systems (such as shakers and fluidized beds). The model adjusts to varying conditions and "morphs" to an appropriate state as they change. The model has an adjustable degree of accuracy, so that a more accurate result can be obtained if desired

(at the expense of increased simulation time). The model can therefore be used as a measurement tool, which can be used for grading and comparison with other models.

1.3. Thesis content

This thesis is structured as follows.

The first chapter introduces the concept of computer simulation in comparison to other scientific discovery methods and describes the general problem of granular material simulation.

The second chapter gives a detailed definition of the granular-type material. It reviews the existing approaches to granular material modeling and compares them. It also introduces the general schema of the proposed universal simulation model, which consists of three submodels.

The third chapter considers the first submodel, called the Molecular Dynamics (MD) model. This model is well suited for high-energy granular systems, such as shaker ball mills. A detailed description of the model is given, followed by a discussion of the applicability domain. Finally, an application example (shaker ball mill simulation) is presented.

The fourth chapter considers the second submodel, called the Quasi-Static (QS) model. This model is best suited for simulating low-energy granular systems, whose topology does not change too much throughout the simulation. Examples of such models are sand heaps and silos. The model is based on the original Recursive Inverse Matrix Algorithm (RIMA). A detailed description of the model is given, followed by a discussion of the applicability domain and an application example (silo simulation).

The fifth chapter considers the unified model, called the Multibody Dynamics (MBD) model. This model contains the MD and QS models as its submodels. The MBD model consists of several layers of objects. A detailed description of objects at each level is

given together with the underlying simulation and physical models. The chapter is concluded with an application example (hopper flow).

Finally, conclusions are presented and future work is discussed.

CHAPTER 2. PRELIMINARIES AND REVIEW

This chapter introduces the concept of a granular-type material and discusses its properties. A brief review of the existing approaches to granular materials simulation is presented. Finally, the proposed universal simulation model is introduced and briefly described.

2.1. Definition of a granular material

A *granular material* (granular-type material, granular matter) can be defined as a system consisting of a number of particles interacting with each other. A granular material confined within some boundaries is called a granular system. The particles move acted upon by an external force field and interact with each other as well as with the boundaries. These interactions are of contact-type only, i.e. if two objects do not touch each other, then they do not affect each other directly. Two particles, which are not in direct contact can still however affect each other through a connected chain of other particles.

Examples of granular materials include sand, flowing ice, avalanches, grain, coal, etc. Granular materials are used in such industrial systems as grain elevators, hoppers, bins, shaker and drum mills. Many problems in civil engineering (building bridges that are able to withstand ice pressure, designing avalanche fortifications), hydrotransportation (moving sand, coal and such through pipelines), chemical engineering applications, and many others, also involve granular materials.

The above definition of granular-type material excludes certain kinds of interaction between particles, such as gravitation forces between molecules or planets. Note that such systems can still be modeled with the suggested approach, however, they will not be included into consideration in this thesis.

Note that for some sophisticated external force field models the particles can affect each other indirectly by disturbing the external force field. For example, a model of a turbulent

flow might consider how the motion of particles affects the turbulence of the flow. Thus, the motion of one particle might affect the motion of other particles without a direct contact.

Certain applications may consider moving boundaries. An example of such an application is a shaker ball mill (see Chapter 3 for a detailed description). Generally speaking, the motion of the boundaries of the system may or may not be affected by the motion of particles.

In this thesis, only dry granular materials are considered. It is assumed that the contacts between objects are one-sided, i.e. the objects never "stick" to each other. Again, systems with adhesive particle properties can be successfully simulated using the proposed model. However, this will not be considered here.

The number of particles in a granular system can range from just a few (e.g. 16 balls in a billiard simulation) to thousands or even millions (in a sand or avalanche simulation). Simulating systems with a very large number of particles can be quite computationally expensive. With the proposed model, it is also possible to simulate a system consisting of a single particle (such as a cylinder rolling down an inclined plane), but such a system could hardly be regarded as a granular system.

2.2. Existing approaches to granular materials simulations

This section will describe the existing approaches that can be applied to simulation of granular materials. These can be categorized into the following six groups:

- continuum-based models;
- finite-element method (FEM) models;
- cellular-automata models;
- molecular dynamics models;
- distinct element method (DEM) models and

- the multibody dynamics model.

These models have different applicability domains and different accuracy levels. Generally speaking, the simulation efficiency decreases as the accuracy increases. Similarly, the simulation efficiency decreases as the number of objects in the system increases.

2.2.1. Continuum-based models

Continuum-based models were the first models ever developed for granular matter. One of the first works on this subject was Janssen's theory of load distribution in silos [Janssen 1895]. Various continuum-based models were being developed ever since [Brown and Richard 66, Ragneau and Aribert 93, Bouchaud et. al. 95, Claudin and Bouchaud 98, Bouchaud and Cates 98, Wang and Hutter 99, Matuttis and Schinner 99].

A continuum-based model represents the granular material as a continuous medium and tries to obtain the differential equations governing the stress field in this medium. In the simplest scalar case, the equations can be obtained as follows [Bouchaud et. al. 98]: Consider the particle configuration presented in the Figure 2.1.

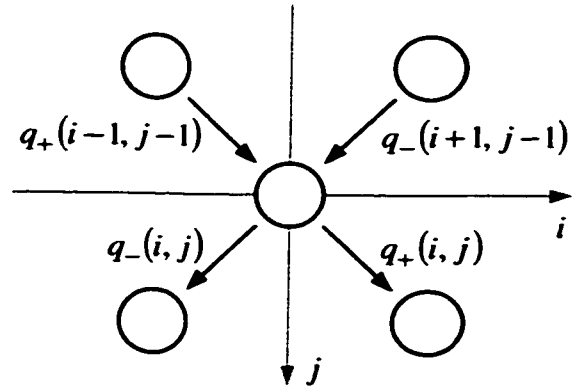


Figure 2.1. The 'q-model' with two neighbors.

It is assumed that each particle is connected to two particles underneath it. Assume that only the vertical component of the stress (the weight) is taken into account. Then the total force acting in a vertical direction on the middle particle can be written as

$$w(i,j) = w_g + q_+(i-1,j-1)w(i-1,j-1) + q_-(i+1,j-1)w(i+1,j-1), \qquad (2.1)$$

where w_g is the weight of a particle and $q_\pm(i,j)$ are the "transmission" coefficients that give the fraction of the weight, which the particle (i,j) transmits to its neighbors below. The mass conservation equation implies that $q_-(i,j)+q_+(i,j)=1$. In the simplest case, when the particles have the same radii and the packing lattice is uniform, $q_-(i,j)=q_+(i,j)=\frac{1}{2}$.

To take into account random particle sizes and irregularities of packing, $q_\pm(i,j)$ can be selected as independent random variables (subject to the above constraint). This model is similar to a diffusion equation with the random convection term [Bouchaud et. al. 98].

Now consider a similar model in 2 dimensions [Bouchaud et. al. 98] (see Figure 2.2). The top particle transmits a fraction of its weight $p \le 1$ to its downward neighbor, and the remaining weight is equally split among the particles on the left and right.

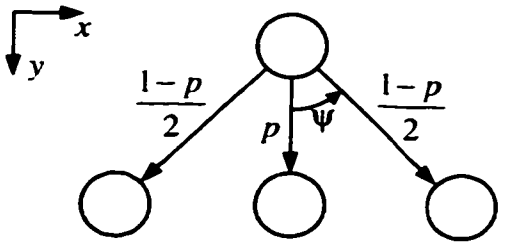


Figure 2.2. Three-neighbor configuration.

The force distribution equations in 2 dimensions can be written as

$$F_x(i,j) = \frac{1}{2}\left(F_x(i-1,j-1) + F_x(i+1,j-1)\right) + \frac{(1-p)\tan\psi}{2}\left(F_y(i-1,j-1) + F_y(i+1,j-1)\right), \qquad (2.2)$$

$$F_y(i,j) = w_g + pF_y(i,j-1) + \frac{1-p}{2}\left(F_y(i-1,j-1) + F_y(i+1,j-1)\right) + \frac{1}{2\tan\psi}\left(F_x(i-1,j-1) + F_x(i+1,j-1)\right), \tag{2.3}$$

where ψ is the angle between particles (see Figure 2.2). Taking the continuum limit on the equations (2.2) and (2.3), the following differential equations can be obtained.

$$\begin{aligned} &\frac{\partial}{\partial y}F_y + \frac{\partial}{\partial x}F_x = \rho, \\ &\frac{\partial}{\partial y}F_x + c_0^2\frac{\partial}{\partial x}F_y = 0, \end{aligned} \tag{2.4}$$

where $c_0^2 = (1-p)\tan^2\psi$ and ρ is a constant. Note that by eliminating one variable (say, F_x), these equations can be converted to a wave equation for the other variable (assuming that y coordinate is taken as time):

$$\frac{\partial^2}{\partial^2 y}F_y + c_0^2\frac{\partial^2}{\partial^2 x}F_y = 0, \tag{2.5}$$

where c_0 represents the speed of the wave. Thus, it can be seen that the stress propagates along a fixed direction $\varphi = \arctan c_0$, where in the general case $\varphi \neq \psi$.

A number of more sophisticated models can be derived using similar techniques (see [Herrmann et. al. 98] for a review). The models can take into account friction between particles (i.e. the maximum ratio between the normal and tangential stress components) and other parameters, try to model slips (or contact failures) between particles and introduce stochastic properties into the equations. Indeed, experiments show that the stress distribution in a granular system can vary in a significant range for a series of identical physical experiments [Brown and Richard 66], i.e. the stress distribution depends on the history of the construction of a granular system.

In order to obtain a closed system of equations defining the stress distribution based on the physical properties of the material, it is required to introduce an additional postulate, called the constitutive equation. One of the recent areas of research is concerned with the finding of an appropriate constitutive equation. One of the approaches is based on the "fixed principal axis" (FPA) constitutive equation [Wittmer et. al. 97]. The numerical results obtained using this constitutive equation are very close to the experimental ones. The approach is based on the assumption that the stress paths in a sandpile are oriented along a fixed direction, and that the orientation of the stress is fixed at the time of particle burial, i.e. it does not change under further loading [Bouchaud et. al. 98].

The continuum-based models are attractive because they can model large granular systems. With the recent advances in the area, the models show a very good correspondence with the experimental results. While being quite accurate in the average case, a continuum model fails to describe the micro-mechanical properties of a granular material (i.e. the stresses acting on a specific particle), which can be of interest in many applications. Also, a continuum model can become quite complex, if dynamic changes have to be considered (such as unloading a hopper). As the model becomes more and more complex, more assumptions and parameters are introduced into the system, which, in turn, makes the model less usable. The introduction of additional assumptions also limits the applicability domain of the model.

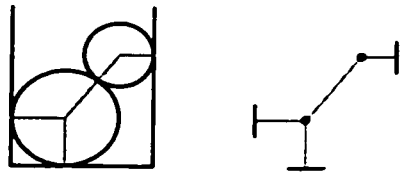
Generally speaking, a continuum model "locks" the system in a predetermined range of states that is defined by the model. Thus, continuum models fail to describe local instabilities or global jamming, because the physical granular system undergoes a state transformation, which must be reflected by a corresponding transformation of the numerical model.

2.2.2. Finite element method (FEM) models

The finite element method models are based on a well-known approach, which considers a material as a system of interconnected elements (finite elements) [Bickford 95]. Each

element's properties are described by a system of stress-strain relationships. The finite element method has been around for quite a while and a range of public domain as well as commercial software packages are available. This makes the method quite attractive for granulates simulations [Wan et. al. 90, Li and Bagster 93, Briscoe et. al. 93, Tejchman and Gudehus 93, Ooi and She 97, Meng et. al. 97, Wan and Guo 98, Tejchman 98, Nuebel and Karcher 98, Karlsson et. al. 98].

The essence of the method can be described as follows. Consider a simple granular system consisting of two disks and a box (see Figure 2.3a).



(a) (b)

Figure 2.3. A granular system (a) and the corresponding FE model (b).

In the simplest case, each of the contacts between either the disks or a disk and a box can be modeled by an elastic two-node element (see Figure 2.3b). Thus, a system with two free and three fixed nodes is obtained. Each of the elements is described by one longitudinal and one transversal (normal and tangential) stiffness. It is assumed that the element is only subject to small deformations. Then the relationship between the displacements $\mathbf{D}$ of the element nodes and the force $\mathbf{S}$ in the element can be expressed in a linear form:

$$\mathbf{S} = \begin{bmatrix} \mathbf{C} & -\mathbf{C} \\ -\mathbf{C} & \mathbf{C} \end{bmatrix} \mathbf{D}, \tag{2.6}$$

where $\mathbf{S} = (s_{1x}, s_{1y}, s_{2x}, s_{2y})^{\mathrm{T}}$ is the vector of forces acting on the left and right nodes of the element, $\mathbf{D} = (d_{1x}, d_{1y}, d_{2x}, d_{2y})^{\mathrm{T}}$ is the vector of nodes displacements,

$\mathbf{C} = \mathbf{T}_\varphi^T \mathbf{C}_0 \mathbf{T}_\varphi$, where $\mathbf{C}_0$ is a diagonal stiffness matrix and $\mathbf{T}_\varphi$ is the coordinate transformation matrix (from local to global):

$$\mathbf{C}_0 = \begin{bmatrix} K_\eta & 0 \\ 0 & K_\tau \end{bmatrix}, \quad \mathbf{T}_\varphi = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}, \tag{2.7}$$

where K_η and K_τ are the normal and tangential stiffnesses of the element, respectively, and φ is the angle of the element.

Now, it is possible to obtain a system of linear equations defining the displacements of the free nodes based on the values of the external forces acting on these nodes (such as gravity forces). These equations are obtained from the equilibrium condition: the sum of all forces acting on a free node (including the external force and the reaction forces in links) must be equal to zero. When these equations are coupled with the equations constraining the displacements of the fixed nodes, a complete linear system is obtained. By solving the system, one obtains the values of free node displacements for given values of external forces. Then, by analyzing node displacements, it is possible to obtain the values for the stresses acting in the internal links of the system.

More advanced implementations of FEM approach may consider particles as rigid bodies rather than modeling them by point nodes connected by links. See [Herrmann et. al. 98] for a review of various FEM-based simulation approaches.

The FEM-based approach works very well for static particle configurations, where the connectivity between particles (the system topology) does not change. Once a slip or a disconnection is detected in a link, it is necessary to regenerate the system and solve it again for the new configuration. This becomes quite computationally expensive when the system is large or when it undergoes many topological changes (such as in hopper unloading).

2.2.3. Cellular-automata models

Cellular-automata models constitute an interesting class of granular models, which is somewhat different from the rest of the granular models. These models consider a granular system as a cellular automaton, i.e. a state machine that evolves by certain rules. These rules are usually quite simple, which allows effective modeling of very large systems. The evolution rules however are usually defined as certain heuristics, i.e. they are not obtained from the known laws of mechanics, but rather are based on some real-life observations and considerations. Thus, the cellular-automata models are best suited for investigating phenomenological behaviors occurring in granular systems, but not for determining the actual values of stress distribution fields or parameters of particle motion and interaction. Examples of cellular-automata models can be found in [Claudin and Bouchaud 97, Goles et. al. 98, Ktitarev and Wolf 98].

An interesting application of a cellular-automata model was used by [Bak et. al. 88] to describe the self-organized criticality (SOC) phenomenon. A sandpile is modeled by a simple lattice, where at each lattice site (i, j) an integer variable z_{ij} is defined. The value of the variable z can represent the amount of sand at this site as well as the "slope" of the pile at this site relative to its neighboring sites. The sand is added to the sandpile at random sites near to the middle of the pile. If the slope at some site exceeds some predetermined value, say four, then this site will *topple*, such that four particles are removed from this site and placed at its neighboring sites:

$$\begin{aligned} z_{ij}^{new} &= z_{ij} - 4, \\ z_{i+1,j}^{new} &= z_{i+1,j} + 1, \\ z_{i-1,j}^{new} &= z_{i-1,j} + 1, \\ z_{i,j+1}^{new} &= z_{i,j+1} + 1, \\ z_{i,j-1}^{new} &= z_{i,j-1} + 1, \end{aligned} \tag{2.8}$$

where the subscript *new* indicates the updated value of the variable. The redistribution is performed simultaneously at all points, where the stability condition $z_{ij} < 4$ is violated. Note that if toppling occurs at a boundary of the lattice, then the particles are simply removed from the system (they fall off the boundary).

Obviously, placing a single particle may involve formation of avalanches, which can differ in size. The sizes of avalanches were measured in the numerical experiments and it appeared that this sandpile model exposed the self-organized criticality phenomenon. The avalanches were occurring at random intervals of time and had random sizes both varying in a maximum possible range, i.e. the system had neither temporal nor spatial scales (see [Bak et. al. 88] for a more detailed description of the experiment and a discussion).

2.2.4. Molecular dynamics models

The molecular dynamics (MD) approach is a popular approach in granulates simulations. It is best suited for high-energy systems, where the particles move with high velocities, collide with each other and with the boundaries, and no long-lasting contacts are formed between particles. Examples of such systems include gas dynamics, shaker ball mills, fluidized beds and similar systems. Descriptions of various MD models can be found in [Rapaport 95, Xie 97, Luding and McNamara 98, Fuji et. al. 98, Gavrilov et. al. 99].

The MD approach assumes that a granular system comprises a number of hard particles, which move independently in some external force field. The particles can collide with each other as well as with the boundaries. These collisions are assumed to be instantaneous and one-to-one only. As a result of a collision, the velocities of the particles participating in the collision are changed instantly according to a collision law. When the particles are represented by disks or spheres, a simple collision law based on the *restitution coefficient* can be obtained (see Chapter 3 below for a more detailed description).

The simulations are usually carried on using a discrete-event simulation scheme, where the collisions between objects are regarded as events, which are placed in a priority queue in sorted order according to their timestamps. The events are retrieved from the event queue one-by-one and processed by the simulation engine. Processing of an event (handling a collision) involves updating the current system state and scheduling new events (checking for collisions with other particles).

The requirement that all collisions are instantaneous and one-to-one only is related to the fact that a collision involving more than two particles can not be resolved statically in the general case. Thus, the dynamics of the impact must be considered in order to obtain the velocities of the particles after the collision.

This requirement places certain limitations on the applicability domain of the MD models. Indeed, the systems where long lasting contacts are formed among particles can not be effectively simulated by the MD approach because such a contact is modeled by a dense sequence of low-velocity collisions, which can essentially bring the simulation to a halt. Various approaches are suggested to deal with this problem (see, for example [Luding and McNamara 98]), however, they are generally artificial. Note that in order to simulate the long-lasting contacts properly, it is required to consider the dynamics of these contacts, which lies beyond the scope of the MD model.

2.2.5. Distinct element method (DEM) models

The Distinct Element Method (DEM)[1] was originally introduced in [Cundall and Strack 79]. The method has become the dominant tool in granular materials research due to its simplicity and ease of implementation. Examples of DEM models in application to various granular problems can be found in [Ristow and Herrmann 95, Sakaguchi et. al.

---

[1] Contrary to the original naming, the method is more commonly referred to as the Discrete Element Method.

93, Hirshfeld et. al. 97, Rong et. al. 97, Potapov and Campbell 98, Zhang and Whiten 98, Gera et. al. 98, McCarthy and Ottino 98, Kano and Saito 98, Kawaguchi et. al. 98].

The method considers a granular system consisting of a number of particles[1]. Each particle is considered separately from other particles. The particles move in an external force field. At each time step, the positions of all particles are examined and the overlapping particle pairs are detected. For a pair of disks, the amount of the overlap is computed as

$$z = \left\| \mathbf{x}_i(t) - \mathbf{x}_j(t) \right\| - (r_i + r_j), \tag{2.9}$$

where $\mathbf{x}_i(t)$ and $\mathbf{x}_j(t)$ are the positions of centers of disks and r_i and r_j are the radii of the disks. Note that if $z > 0$ then there is no contact between particles.

Each overlap is considered as a contact between particles and a restoring force is applied to the particles in contact. The magnitude of this restoring force is selected to be proportional to the maximum value of the overlap. After forces at all overlapping pairs of particles have been determined, the total forces acting on each particle are computed and the positions of the particles are updated for the beginning of the next time step under the assumption that the forces remain constant during this time step:

$$\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \mathbf{v}_i(t)\Delta t + \frac{\mathbf{F}_i(t)}{2m_i}\Delta t^2, \mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \frac{\mathbf{F}_i(t)}{m_i}\Delta t, \tag{2.10}$$

where $\mathbf{x}_i(t)$ is the position of the particle, $\mathbf{v}_i(t)$ is the velocity of the particle, m_i is the mass of the particle, $\mathbf{F}_i(t)$ is the total force acting on the particle at the beginning of the timestep and Δt is the timestep. The simulation proceeds in this manner until the specified final time is reached. Note that the simulation is not event-driven, i.e. the

---

[1] Generally speaking, the particles can be arbitrarily shaped. The original work [Cundall and Strack 79] assumed that they are represented by discs.

overlaps are not detected immediately, but only at the next time grid-point. Also, in order to ensure that the energy level in the system does not grow, it is necessary to apply damping at all contacts.

Originally, the method was developed for quasi-static granular assemblies generally found in such fields as soil mechanics. Soon the method started to be used in a wide variety of applications, including high-energy systems. Unfortunately, the method is very often misused. One of the main assumptions of the method is that "the time step chosen may be so small that during a single time step disturbances can not propagate from any disc further than its immediate neighbors" [Cundall and Strack 79]. This assumption can be easily violated when the particles are moving with high velocities because in this case the time step must be selected as a very small value, which is quite computationally expensive. Also, the model has a large number of input parameters, such as the coefficient of proportionality, which is used to compute restoring forces in contacts, the damping coefficient, as well as the timestep value. The selection of the timestep proves to be a crucial task, which affects the validity of a simulation significantly [Vinogradov 99].

2.3. Suggested approach – morphing models

The primary objective of the research described in this thesis was to develop a more universal model for simulation of granular-type materials. The model is, indeed, a combination of three models: one is based on the molecular dynamics approach, another (the Quasi-Static model) is similar to the FEM models, and the third one, which is used to drive the simulation, is based on the Multibody Dynamics (MBD) approach, first introduced by Vinogradov [Vinogradov 86a].

The MBD approach provides a numerically stable alternative to the DEM. It treats an interconnected system of particles (a cluster) as a whole and obtains a numerical solution of the coupled system of differential equations defining the motion of a cluster. The approach yields a more accurate solution, and also provides better means of error control. The MBD approach was originally developed at the University of Calgary and then

continually evolved [Vinogradov 86a, Vinogradov and Springer 90, Wierzba and Vinogradov 91, Vinogradov 92a, Vinogradov 92b, Vinogradov 93, Sun et. al. 94, Vinogradov and Sun 97]. The proposed universal model uses the MBD model as a base and extends it with the concept of "morphing" models, which allows adjusting dynamically to the system being simulated.

The three submodels constituting the overall simulation model (the MD, QS and MBD models) are constantly transforming (morphing) into each other, depending on the state of the system. The MD model is used to simulate particles that are moving independently; the Quasi-Static model is used to simulate stable configurations of disks called the Quasi-Rigid Bodies (QRBs); and the MBD model is used for simulating the systems in transient states, as well as to drive the simulation. The use of different models is not exclusive, i.e. the MD model can be used to simulate a subset of the system, while at the same time the QS and MBD models are used to simulate other parts of the system.

The simulation model is event-driven, which ensures that the contacts between particles, slips and breaking of contacts are detected and handled immediately as they occur.

The model was implemented in a form of an object-oriented library, consisting of several layers of objects. This allowed creating a system of interchangeable software components, which can be easily updated. An example of an independent software component is the *ExternalForceField* object, which is used to compute the external forces acting on a particular body. This object communicates with the rest of the simulation system via a generic interface. Thus, if a new *ExternalForceField* object needs to be introduced, it is done transparently such that the rest of the system does not need to be changed.

Each of the models was implemented and tested. The results compare favorably with the experimental results as well as with the theory. A number of papers were published based on the various aspects of this work [Gavrilov and Vinogradov 94, Gavrilov and Vinogradov 95, Gavrilov et. al. 95, Gavrilov and Vinogradov 96, Gavrilov et. al. 97,

Gavrilov and Vinogradov 97a, Gavrilov and Vinogradov 97b, Gavrilov and Vinogradov 98a, Gavrilov and Vinogradov 98b, Gavrilov et. al. 99, Gavrilov and Vinogradov 99].

CHAPTER 3. MOLECULAR DYNAMICS MODEL

The Molecular Dynamics (MD) model is the simplest and most commonly used model for simulating granular matter. Each particle is considered separately from the rest of the particles. Particles do not form lasting contacts with each other or with the boundaries. Interactions between particles and boundaries are limited to instantaneous collisions only. Between two consecutive collisions a particle is subjected to external forces only.

The model is relatively simple. The differential equations governing the motion of particles (*the equations of motion*) are not coupled and can be easily integrated. In some cases an analytical solution is available, for example, when the external force field is constant (e.g. the gravitational field). Even when the equations can not be analytically integrated, a high-precision numerical solution can be obtained at a low computational cost. It is easy to introduce an error control scheme based on the total energy of the system.

The interactions between particles are limited to collisions. Since they are considered to be instantaneous, the forces do not participate in the collision equation. The *collision laws* define how the state of the colliding objects changes after the collision. In the MD model, the collision equations are based solely on energy and momentum conservation principles.

The MD simulations are usually event-driven. Two main types of events can be distinguished: the predict-trajectory events and the collision events. The predict-trajectory events represent time steps in the numerical integration of the equation of motion of a particle. A collision event is scheduled at times when a collision is detected between a pair of objects (particle-to-particle or particle-to-boundary collisions). In order to compute the time of a collision, the *collision detection equation* has to be solved.

If an analytical solution of the equation of motion is available, then the trajectory of a particle can be predicted for an infinite period of time. In this case, the predict-trajectory

events are only scheduled when the state of a particle is changed unexpectedly, i.e. due to a collision.

Since the model is relatively simple, it is very popular. A straightforward implementation of the MD model is an easy programming exercise. Since each particle moves separately from the other particles and since the interactions are limited to a very close vicinity of the particle, it is usually possible to parallelize the algorithm based on the spatial partitioning of the simulation space. Some very large simulation experiments involving over a billion particles have been performed in distributed computational environments [Stadler et. al. 97].

The MD model has certain limitations. The main limitation is related to the assumption that no lasting contacts are formed between the particles and no inter-particle forces are taken into consideration. If a lasting contact occurs in the simulated system (e.g. a ball is lying on a horizontal surface), it might bring the simulation to a halt. The contact is essentially modeled by a sequence of low-velocity collisions between the objects in contact. These collisions are very close together on the time scale. This effect tends to slow the simulation down significantly, or even bring it to a complete halt. In order to move the simulation ahead, artificial corrections can be applied, such as changing particles' velocities and/or positions. Such corrections are not justified mechanically and may lead to falsification of the simulation results.

Another class of applications where the MD model does not work very efficiently is high-density systems, e.g. densely packed shaker mills. In such systems, collisions are very frequent and, as a result, the energy is dissipated quite rapidly, which, in turn, might lead to the formation of lasting contacts.

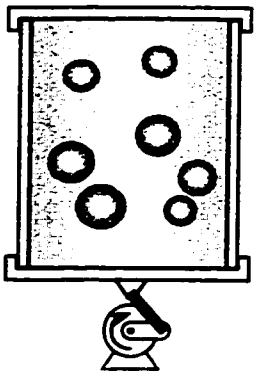


Figure 3.1. Shaker ball mill.

The MD model has many applications where it works very well. Many high-energy systems, where the particles move with high velocities can be efficiently simulated. One of the main areas of application of the MD model is, indeed, molecular dynamics [Rapaport 95]. Others include chemical applications (fluidized beds) [Kawaguchi et. al. 98, Gera et. al. 98] and ball mills used in mechanical alloying [Gavrilov and Vinogradov 98a, Gavrilov et. al. 99] (see Figure 3.1). One classical application of the MD model is to the simulation of billiards (see Figure 3.2).

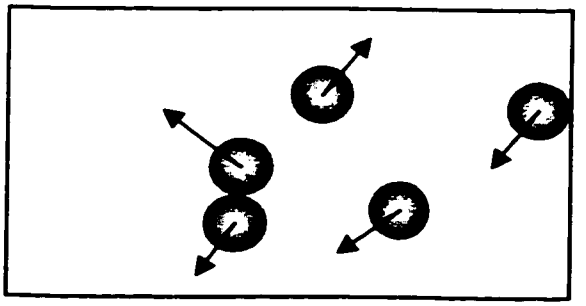


Figure 3.2. Billiard simulation.

3.1. System structure in the MD model

In the molecular dynamics model the simulation system includes objects of two types: particles and boundaries. *Particles* are most commonly represented by disks or spheres. Then, the collision detection equations can be solved analytically in many cases. When

the particles are represented by more complex shapes, the collision detection equations can only be resolved numerically, which can be computationally expensive. The equations of motion and the collision equations become more complicated as well.

Boundaries are usually represented by line segments or planar facets. In some cases, curvilinear boundaries (such as circular arcs or spherical segments) can be introduced without complicating the collision detection equations too much. Sometimes, it is necessary to introduce *moving boundaries*. For example, in shaker ball mill simulations [Gavrilov and Vinogradov 98a, Gavrilov et. al. 99], the boundaries are subjected to an oscillatory motion. It is assumed that the motion of boundaries is predefined, i.e. it is not affected by the collisions.

Assumption 3.1. The collisions between particles and boundaries are assumed to be central, frictionless and instantaneous. Thus, the following conditions are satisfied:

1) the tangential velocity of particles is conserved during the collision;
2) the rotation of particles does not affect the collision and the angular velocity of rotation is not changed due to a collision; and
3) the collisions are one-to-one only (i.e. three or more objects never participate in a collision).

Since the collisions are assumed to be instantaneous, the probability of several collisions happening simultaneously is zero. In practical implementations, however, several collisions can occur simultaneously due to round-off errors. In this case, they are processed in a random order, which is equivalent to an infinitesimal perturbation of particle positions.

Note that a collision between three or more objects can not be resolved in the MD framework in the general case (there are more unknowns than equations). In order to obtain the correct distribution of momentum in such a collision, the dynamics of the collision has to be considered.

3.2. Laws of motion

The equation governing the motion of a particle can be obtained from Newton's second law:

$$\ddot{\mathbf{x}} = \frac{1}{m}\mathbf{F}(\mathbf{x}, t, ...), \tag{3.1}$$

where $\mathbf{x} = \mathbf{x}(t)$ is the position of the particle's center of mass (the center of the disk or sphere for circular/spherical particles), m is the particle mass and $\mathbf{F}(\mathbf{x}, t, ...)$ is the external force field. $\mathbf{F}$ and $\mathbf{x}$ are either 2-dimensional or 3-dimensional vector functions. In the general case, the external force field can depend on the position of the particle, time and other parameters (e.g. the velocity of particle, the particle radius, etc.). Since the collisions do not affect the angular velocity of the particles, it is not necessary to consider particle rotation.

In the general case, in order to obtain the trajectory of the particle, the equation of motion (3.1) has to be numerically integrated. In many applications, however, the force field is constant. For example, in shaker ball mill simulations, it is the gravitational field $\mathbf{F}(\mathbf{x}, t, ...) = m\mathbf{g}$, where $\mathbf{g}$ is the gravitational constant. In billiard simulation there are no external forces at all, i.e. $\mathbf{F}(\mathbf{x}, t, ...) = 0$. In these and some other cases it is possible to obtain an analytical solution of the equation of motion.

3.2.1. Constant force field

For the case when the external force field is a constant $\mathbf{F}(\mathbf{x}, t, ...) = \mathbf{F}_0$, the solution can be written as

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0(t - t_0) + \frac{\mathbf{F}_0(t - t_0)^2}{2m}, \quad \mathbf{v}(t) = \dot{\mathbf{x}}(t) = \mathbf{v}_0 + \frac{\mathbf{F}_0(t - t_0)}{m}, \tag{3.2}$$

where $\mathbf{x}_0 = \mathbf{x}(t_0)$ and $\mathbf{v}_0 = \dot{\mathbf{x}}(t_0)$ are the position and velocity of particle at the initial moment of time t_0, respectively. Note that when the velocity of a particle changes due to

a collision at a moment of time t_c, the laws of motion (3.2) can be used to predict the trajectory of the particle for the infinite time interval $[t_c,+\infty)$. In fact, the particle will move along this parabolic (or linear) trajectory only until the next collision.

Note that if the particles move in a gravitational field, their relative velocities are only linear in respect to time (because the second-degree term cancels out). This means that an analytical solution for the collision detection equations can be obtained (see Section 3.3 below).

3.2.2. Variable force field

In the general case, the equation of motion (3.1) does not have an analytical solution. It has to be solved numerically using some integration method [Burden et. al. 78]. Thus, the solution is obtained as a sequence of values at fixed timesteps. In order to perform collision detection, the trajectory has to be represented by a continuous curve. It is possible to use cubic splines or another interpolation to obtain a smooth approximation. In this case, however, the collision detection equations would add excessive computational complexity. Instead, linear splines are used to represent the trajectory during a time interval $[t_i,t_{i+1}]$, where $t_{i+1}=t_i+h$ and h is the timestep:

$$\mathbf{x}(t_i+\tau)=(1-\alpha)\mathbf{x}_i+\alpha\mathbf{x}_{i+1},\ \mathbf{v}(t_i+\tau)=\frac{\mathbf{x}_{i+1}-\mathbf{x}_i}{h}, \tag{3.3}$$

where $\tau\in[0,h]$, $\alpha=\frac{\tau}{h}$, $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are the solution values obtained by numerical integration at times t_i and t_{i+1}, correspondingly. Note that the curve representing the trajectory is continuous, but not smooth (see Figure 3.3). Indeed, the velocity is kept constant during the timestep and is changed instantly at the beginning of the next timestep.

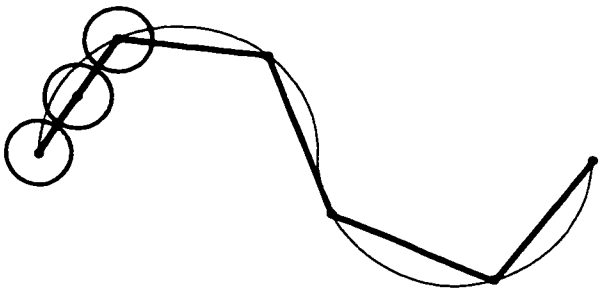


Figure 3.3. Linear spline trajectory interpolation.

When the trajectory is approximated by a piecewise-linear function the collision detection equations can be solved analytically. As in any numerical method, a greater precision can be achieved by decreasing the value of the timestep h. The accuracy can not be increased infinitely, however, because of the round-off errors in computations.

3.3. Collision detection equations

In order to detect a collision between a particle P_i and an object Q_j (either a particle or a boundary), the minimal positive root t_c of the collision detection equation

$$d(P_i, Q_j) = 0 \tag{3.4}$$

has to be found (if it exists). Here $d(P_i, Q_j)$ denotes the general distance function, which is defined as

$$d(P, Q) = \min_{\mathbf{p} \in P, \mathbf{q} \in Q} \|\mathbf{p} - \mathbf{q}\|, \tag{3.5}$$

where $\|\cdot\|$ denotes the Euclidean norm. Note that the positions of P and Q are changing with time and, consequently, the distance $d(P, Q)$ is a function of time as well.

In the general case, when the trajectories of P and Q are given by some functions, the equation (3.4) can only be solved numerically. Moreover, when particles and boundaries have arbitrary shapes, this problem becomes a minimization problem.

In some cases the collision detection equation can be solved analytically.

3.3.1. Collision detection between two disks (spheres)

The distance function (3.5) between two disks (spheres) P and Q can be written as

$$d(P,Q)=\|\mathbf{x}_p-\mathbf{x}_q\|-(r_p+r_q), \tag{3.6}$$

where $\mathbf{x}_p$, $\mathbf{x}_q$, r_p and r_q are the positions of the centers of P and Q and their radii, respectively (see Figure 3.4).

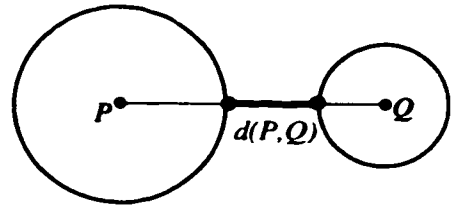


Figure 3.4. The distance between two disks (spheres).

Assume that the relative velocity $\mathbf{v}=\mathbf{v}_p-\mathbf{v}_q$ is a constant. Note that this is the case for both parabolic trajectories in a gravitational field (Eqn. (3.2)) and linearly interpolated trajectories (Eqn. (3.3)). Denote the initial relative displacement of disks (spheres) by $\mathbf{x}=\mathbf{x}_{p0}-\mathbf{x}_{q0}$. Then, the collision detection equation (3.4) can be rewritten as

$$\|\mathbf{x}+\mathbf{v}t\|^2=r^2, \tag{3.7}$$

where $r=r_p+r_q$. This is a quadratic equation in t, and the solution can be obtained as

$$t_{1,2}=-\lambda\pm\sqrt{\lambda^2-\frac{\|\mathbf{x}\|^2-r^2}{\|\mathbf{v}\|^2}},\quad \lambda=\frac{(\mathbf{x},\mathbf{v})}{\|\mathbf{v}\|^2}, \tag{3.8}$$

where $(\cdot,\cdot)$ denotes the scalar product. Note that the quantity λ represents the length of the projection of $\mathbf{x}$ onto $\mathbf{v}$.

The minimum positive root of the equation is selected as the collision time t_c. If the root does not exist or if both roots are negative, then P and Q will not collide, i.e. $t_c=+\infty$.

If the relative velocity $\mathbf{v}$ is zero, then the solution (3.8) does not apply. In this case, however, P and Q will not collide as well (note that they do not intersect at the initial state and do not move relatively to each other). If $t_1 = t_2$, then it is assumed that the collision does not occur, since P and Q just touch each other (i.e. their relative velocity at the time of the collision is zero).

3.3.2. Collision detection between a disk (sphere) and a line (plane) segment

The distance function (3.5) between a disk (sphere) $P = (\mathbf{x}_p, r_p)$ and a line (plane) $Q = (\mathbf{x}_q, \mathbf{n}_q)$, where $\mathbf{x}_q$ is a reference point which belongs to Q and $\mathbf{n}_q$ is the normal to Q [1] (see Figure 3.5) can be written as

$$d(P,Q) = (\mathbf{x}_p - \mathbf{x}_q, \mathbf{n}_q) - r_p . \tag{3.9}$$

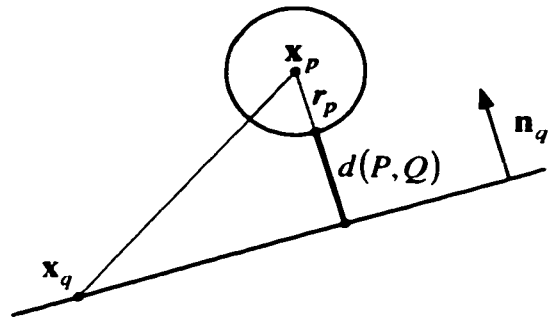


Figure 3.5. The distance between a disk (sphere) and a line (plane).

Assume that both P and Q move along parabolic trajectories, i.e.

$$\mathbf{x}_p(t) = \mathbf{x}_{p0} + \mathbf{v}_{p0}t + \frac{\mathbf{a}_p t^2}{2},\ \mathbf{x}_q(t) = \mathbf{x}_{q0} + \mathbf{v}_{q0}t + \frac{\mathbf{a}_q t^2}{2}, \tag{3.10}$$

---

[1] Q usually represents a boundary. Then $\mathbf{n}_q$ is selected as a normal to Q which points out of the boundary.

where $\mathbf{x}_{p0}$, $\mathbf{x}_{q0}$, $\mathbf{v}_{p0}$, $\mathbf{v}_{q0}$, $\mathbf{a}_p$ and $\mathbf{a}_q$ are the initial positions, initial velocities and the accelerations of P and Q, respectively.

Then, the collision detection equation (3.4) can be rewritten as

$$x_n + v_n t + a_n \frac{t^2}{2} - r_p = 0, \tag{3.11}$$

where $x_n = (\mathbf{x}_{p0} - \mathbf{x}_{q0}, \mathbf{n}_q)$, $v_n = (\mathbf{v}_{p0} - \mathbf{v}_{q0}, \mathbf{n}_q)$ and $a_n = (\mathbf{a}_p - \mathbf{a}_q, \mathbf{n}_q)$, i.e. the problem is transformed to a one-dimensional parabolic collision detection problem.

If $a_n = v_n = 0$, then the equation does not have a solution, i.e. $t_c = +\infty$ (note that P and Q do not intersect at the initial position and do not move relatively to each other). If $a_n = 0$, $v_n \neq 0$, then the collision time t_c is found as $t_c = -\frac{x_n - r_p}{v_n}$ (note that $t_c > 0$ only if $v_n < 0$, i.e. when P moves towards Q).

If $a_n \neq 0$, then the equation has two solutions

$$t_{1,2} = -\lambda \pm \sqrt{\lambda^2 - \frac{2(x_n - r)}{a_n}}, \quad \lambda = \frac{v_n}{a_n}. \tag{3.12}$$

Again, the minimum positive root of the equation is selected as the collision time t_c. If the root does not exist or both roots are negative, then P and Q will not collide, i.e. $t_c = +\infty$. If $t_1 = t_2$, then it is assumed that the collision does not occur, since P and Q just touch each other (i.e. their relative velocity at the time of the collision is zero).

3.4. Collision laws

Collision laws describe how the velocities of particles change after a collision occurs. Note that it is assumed that the tangential components of particle velocities conserve in a collision, i.e. only the normal components of velocities change (see Assumption 3.1).

Also recall that the collisions are assumed to be one-to-one only. The collision laws will be obtained based on two fundamental conservation principles: energy conservation and momentum conservation.

The energy loss due to a collision can be characterized by the *coefficient of restitution* ε, $0 \le \varepsilon \le 1$. It is defined as

$$\varepsilon = \frac{|v_2 - v_1|}{|v_{20} - v_{10}|}, \tag{3.13}$$

where v_{i0} and v_i, $i = 1,2$ are the normal velocities of particles before and after collision, respectively. When $\varepsilon = 1$ the collision is said to be *absolutely elastic*, i.e. no energy is lost in the collision. When $\varepsilon = 0$ the collision is said to be *absolutely plastic*, i.e. the maximum possible amount of energy is dissipated. In this case, the normal velocities of particles become equal after the collision. The values for the restitution coefficients are determined experimentally for balls made of various materials [Metals 84].

The energy and momentum conservation laws are written as

$$\frac{m_1 v_{10}^2}{2} + \frac{m_2 v_{20}^2}{2} = \frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} + \Delta E, \tag{3.14}$$

$$m_1 v_{10} + m_2 v_{20} = m_1 v_1 + m_2 v_2, \tag{3.15}$$

respectively, where m_i, $i = 1,2$ are the masses of particles and ΔE is the energy lost in the collision.

Solving the system of three equations (3.13), (3.14) and (3.15) for v_1, v_2 and ΔE, the following solution is obtained

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{bmatrix} \alpha - \varepsilon & 1 - \alpha + \varepsilon \\ \alpha & 1 - \alpha \end{bmatrix} \begin{pmatrix} v_{10} \\ v_{20} \end{pmatrix}, \quad \Delta E = \frac{m_1 m_2}{m_1 + m_2} \left(1 - \varepsilon^2\right) \frac{(v_{20} - v_{10})^2}{2}, \tag{3.16}$$

where $\alpha = \dfrac{(1+\varepsilon)m_1}{m_1+m_2}$.

Note that the above solution was found for the case when two particles collide. The solution for a particle-to-boundary collision can be obtained by assuming that the boundary is represented by the second particle, which is assigned an infinite mass $m_2 = +\infty$. In this case $\alpha = 0$ and the solution can be written as

$$v_1 = v_{20} + \varepsilon(v_{20} - v_{10}),\ \Delta E = m_1\left(1-\varepsilon^2\right)\frac{(v_{20}-v_{10})^2}{2}, \tag{3.17}$$

where v_{20} is the velocity of the boundary at the time of the collision. Note that according to (3.16) $v_2 = v_{20}$, i.e. the boundary is unaffected by the collision. If the boundary is moving then the amount of energy C contributed to the system due to a collision can be computed as

$$C = 2m_1 v_{20}(v_{20} - v_{10}). \tag{3.18}$$

Note that this amount can be of either sign.

If particles and/or boundaries made of different materials are present in the system, then individual restitution coefficients ε_{ij} can be defined for each pair of objects in the system.

3.5. General simulation algorithm

The general scheme used for MD simulations is now defined. The simulations are carried out using the discrete event-driven simulation scheme. The above formulas are used in computations of particle trajectories and for detecting collisions.

3.5.1. Discrete event-driven simulation

The general algorithm used in discrete event-driven simulations is as follows.

initialize event queue:
    insert initialization events for objects in the system
    insert *finish simulation* event
while *finish simulation* event has not been encountered
    retrieve the next event from the event queue
    process the event:
        advance the simulation clock to the time of the event
        update system state depending on the type of event
        cancel the events that have been invalidated by the update
        detect and schedule new events
end (while)
stop

Each event is attributed with the following parameters: time of the event, event type, plus additional parameters depending on the type of the event (e.g. a collision event contains references to the colliding objects). The event queue contains a list of events, sorted according to their time attribute.

The following types of events are used in MD simulations.

3.5.2. Predict-trajectory event

This event is scheduled for each particle individually. If the equations of motion are numerically integrated and the trajectory is interpolated, then the predict-trajectory events are scheduled regularly for each timestep. Note that the predict-trajectory events need not be synchronized for all particles. Therefore, a variable timestep can be introduced, which depends on the current state of each particle, a procedure, which, in turn, improves the simulation efficiency.

When the law of motion is given in an explicit form, the predict-trajectory events are only scheduled after the particle's state changes due to a collision.

The handling of a predict-trajectory event for a particle includes canceling all pending events scheduled for this particle (since the trajectory is being updated), detecting and scheduling collisions with other particles and/or boundaries and also scheduling the next predict-trajectory event (if needed).

Note that when a predict-trajectory event is handled, it is not necessary to advance all of the objects in the system to the current time (this would be quite computationally expensive). Indeed, only the object, whose predict-trajectory event is being handled, needs to be advanced. This has certain implications related to the collision detection formulas (equations (3.8) and (3.12)). Indeed, a pair of objects being tested for a collision may not be synchronized at the initial state. This can be overcome by advancing the second object to the current time before applying the formula.

When a collision detection optimization algorithm [Gavrilova 98] is used, not every pair of objects is checked for collisions, but only the closest neighbors of the object. Then only these closest neighbors are advanced to the current time during the collision detection stage.

When detecting collisions, it is necessary to detect and schedule *all possible* collisions with other objects, not the first one only. Consider an example in the Figure 3.6. Assume that the trajectories are defined by explicit functions (billiard application).

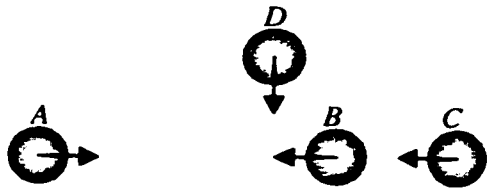


Figure 3.6. All possible collisions have to be scheduled.

Assume that a predict-trajectory event for particle A is being handled. Two collisions involving A can be detected: AB and AC. Clearly, AB is the first collision. Consider what will happen if the collision AC is left unscheduled. The particle D will collide with the particle B and the predict-trajectory events will be scheduled for both of them. Assume

that neither *B* nor *D* will collide with *A*. Then the collision between *A* and *B* will be canceled. The particles *A* and *C* will move towards each other unobstructed and will have to collide eventually. This collision, however, will be missed because no more predict-trajectory events are scheduled for either *A* or *C* and a new collision can only be detected during handling a predict-trajectory event.

Note that for interpolated trajectories it is usually known when the next predict-trajectory event will occur for each particle. Then it does not make sense to schedule any collisions for this particle occurring after the predict-trajectory event because they will be cancelled.

When a moving boundary is present in the system, it is possible that its motion is described by a piecewise-defined function as well. In this case, predict-trajectory events are also scheduled for this boundary. The same rules apply for scheduling collisions and the following predict-trajectory events.

3.5.3. Collision event

The collision events are attributed with references to two colliding objects. One of them is necessarily a particle, and the other one may be either a particle or a boundary.

When a collision event is handled, both objects participating in the collision are advanced to the current time. Then the collision law (equation (3.16) or (3.17)) is applied to update the current velocity of the particle(s) participating in the collision. Finally, a predict-trajectory event is scheduled at a current time for each particle participating in the collision. Note that it is not necessary to cancel any pending events or detect new collisions because this will be done when the predict-trajectory events will be handled.

3.5.4. Timestep selection

When the laws of motion are not explicitly defined and the trajectories are interpolated, the selection of a timestep value becomes a critical task. Introducing a timestep that is too large may bring the simulation to an infinite loop. Consider an example in the Figure 3.7.

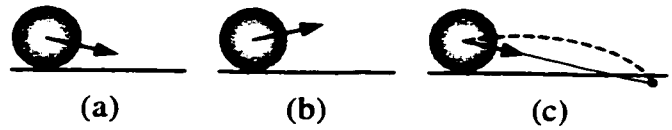


Figure 3.7. Infinite loop in a MD simulation.

Assume a collision between a particle and a boundary is being handled. The particle has a relatively small normal velocity of approach (Figure 3.7a). After the collision is handled, the particle is assigned a new velocity (Figure 3.7b). A predict-trajectory event is scheduled for the particle and handled right away (Figure 3.7c). The predicted trajectory is shown in a dashed line in the Figure 3.7c. Note that the predicted particle position is below the boundary. The parabolic trajectory is interpolated by a straight line and the velocity is updated such that it points toward the boundary again. The system ended up in exactly the same state, as it was in the Figure 3.7a. A collision between the particle and the boundary has to be scheduled at the current time. This brings the system to an infinite loop: an infinite sequence of events (collision, predict, collision, predict, etc.) is being processed.

Note that this situation can be dealt with by selecting a smaller timestep. Unfortunately, such situations can neither be avoided nor predicted. The timestep might need to be adjusted if an infinite loop is detected (i.e. when the simulation clock does not advance after a considerably long sequence of events has been processed). Reducing the timestep, however, might slow the simulation down considerably.

The root of the problem is that a lasting contact is about to be formed between the particle and the boundary. The MD model is inapplicable in this case, because it can not handle lasting contacts and tends to represent them by infinite sequences of low-velocity collisions.

3.6. Application – shaker ball mill [Gavrilov et. al. 99]

The application of the molecular dynamics model is demonstrated by the simulations of the shaker ball mill, which is used in mechanical alloying.

Mechanical alloying has been around since the late 1960's; the principles are well established and explained in [Benjamin 70]. Little information currently exists with respect to the design and efficient operation of grinding equipment for the production of this material [Metals 84]. Commercial ball mills of a number of different types have been constructed and used, but most design and operating information is obtained by trial and error. A computer simulation approach can be a powerful tool in studying and optimizing the grinding processes in shaker ball mills [Gavrilov et. al. 97].

The process of grinding in a shaker ball mill starts when two components in a granular form are placed in a shaker and ends when the final size of granules of these components becomes less then some specified level. The event of grinding takes place when a material particle gets in between the colliding balls. The efficiency of grinding, defined as the time needed to reduce the size of granules from the original to the final one, depends on the frequency of shaking, the number and size of balls, and the geometry of the shaker. The number of balls should be such that their relative density is small compared to the shaker volume. The toughness of the material to be ground is usually much smaller than that of the steel balls. During milling the energy supplied to the balls from the moving walls is transferred to the material particles at each collision, and the latter results in more particles with refined sizes. Thus the efficiency of milling is determined by the number of collisions between the balls and the energy released in collisions. The molecular dynamics model is well suited for simulating the process of milling. The process of refinement, in terms of number of particles with a reduced size, can then be deduced based on the rate of collisions. Both the rate of collisions and the rate of refinement were investigated.

The simulation procedure includes two steps. The first step deals with the determination of the number of collisions in a specified time for a given set of parameters. These parameters include shaker geometry (radius, height, and lid configuration: flat or spherical), number, diameters and materials of balls, and frequency and amplitude of shaking.

The second step deals with the investigation of the grinding process in terms of the distribution of particle radii during the milling process. Two approaches were developed: one, based on the random sampling from all possible system states, and another, on the conservation of mass after each event of collision. The first approach results in a discrete Monte Carlo simulation model, while the second one in a continuous model. Both approaches give similar distribution of particle sizes after a specified number of ball collisions, and compare favorably with the corresponding experimental results.

The validity of simulation was confirmed when the optimum parameters were identified based on numerical experiments so that the efficiency of grinding was increased almost ten-fold.

3.6.1. Problem definition

The motion of balls inside the shaker ball mill is simulated in a 3-D space. The simulation system consists of the objects representing the cylinder itself and the balls moving inside the cylinder. Since the balls move in a gravitational field, their motion is defined by explicit formulas (equation (3.2), where $\mathbf{F}_0 = m_i\mathbf{g}$, m_i is the mass of a particle).

The motion of the cylinder is harmonic in the vertical direction z with an amplitude A and a frequency f. For simplicity of calculations, it is assumed that the cylinder moves with constant acceleration during each half-period of oscillations, i.e. along a parabolic trajectory:

$$z(t) = v_0 t + \frac{at^2}{2}, \tag{3.19}$$

where for the first half-period ($0 \le t < T/2$, $T = 1/f$ is the period of vibrations) $a = -32Af^2$, $v_0 = 8Af$, and for the second half-period ($T/2 \le t < T$) $a = 32Af^2$, $v_0 = -8Af$. This approximation allows computing collision times directly (see equation (3.17)), while in the case of a sinusoidal form, a numerical scheme would be required.

The purpose of the simulations was to investigate the energy-transfer properties of the system, i.e. how efficient the external energy of shaker vibrations is transformed into the internal energy of balls inside the shaker. With each collision the lost energy is transformed into heating the balls as well as grinding. For a ball-to-ball collision this energy is defined by equation (3.16) and for a ball-to-cylinder collision by equation (3.17). When a ball collides with the moving lid of the cylinder, the energy is contributed to the system of moving balls. One part of this energy is spent on heating the ball and grinding (equation (3.17)) and another one on changing the kinetic energy of the ball. The energy contributed to the system as a result of a ball-to-cylinder collision is defined by equation (3.18).

3.6.2. Statistical grinding model

The results of the simulation obtained using the molecular dynamics model are used to simulate the grinding process occurring in the shaker ball mill. The objective is to investigate how the parameters of the ground particles, namely, their radii, are changing with time.

In a straightforward approach, one could try to simulate the system of material particles and steel balls as one system in which the number of material particles (modeled by spheres) grows in time. The split of a material particle takes place every time it is caught between the two colliding balls. In this scenario, however, the number of particles becomes so large with time (i.e., one 1-mm particle can be split into one million 0.01-mm pieces during first seconds of shaking), that it becomes computationally inefficient to keep track of position and velocity of each particle. In this respect, a statistical model offers an alternative way of describing the distribution of particle radii during the milling process.

The random process of grinding can be represented by a directed graph with system states represented by the vertices and transition states by the edges. Each edge is weighted with the probability of transition to another state, and the sum of weights of all edges going

out of a vertex is always equal to 1. This is demonstrated on a simple example shown in Figure 3.8, where the numbers indicating the probabilities are given as illustrative.

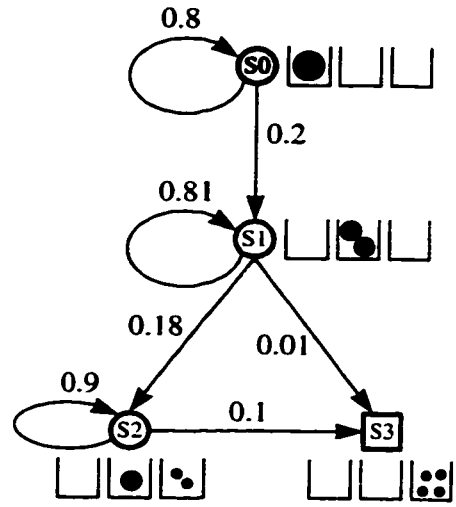


Figure 3.8. An illustrative example of the graph describing the grinding process.

In this example a single particle in the state *S0* is transformed into a two-particle system in the state *S1*, and then into a three-particle system (different particle sizes) in the state *S2*, and eventually into a four-particle system in the state *S3* (final particle size in this example). Note that different particle sizes are placed in different "baskets". In this example there are only three baskets: B_0, B_1 and B_2. Thus the initial state *S0* has 1 particle in the basket B_0, and no particles in baskets B_1 and B_2. When a collision occurs, the particle in B_0 may break up into two smaller ones with probability 0.2, and then the system will move into the state *S1* with 0 particles in basket B_0, 2 particles in basket B_1, and none in B_2. The other possibility is to stay in the same state *S0*, i.e. the particle was not involved in the collision. The probability of the latter event is 0.8.

When the system is in the state *S1* with 2 particles in basket B_1, there are three possible outcomes of a collision. In the first scenario, both particles are broken and moved into the basket B_2. The probability of this is 0.01. With this transition the system is moved into

the final state *S3*, when all particles are in the last basket B_2. The second possible outcome is when only one of two particles in basket B_1 is broken. The probability of this is 0.18, and with this transition the system moves into the state *S2* with 1 particle in basket B_1 and 2 particles in basket B_2. The third outcome is when no particles are broken due to a collision, and the system stays in the state *S1*. The probability of this is the remaining 0.81. Finally, for the state *S2*, which has one particle in B_1, there are two possible outcomes: one, when the particle is broken (with probability 0.1) and the system is moved into the final state *S3*, or, second, when the particle is not broken (with probability 0.9), and the system stays in the state *S2*.

The above example represents a graph of all possible outcomes during the grinding process from the initial to the final state. It is clear that if the initial particle is to be reduced by many orders of magnitude (e.g. from 4 mm to 1 μm), the graph becomes very complicated, and the determination of all intermediate states becomes time consuming. Instead, a random sampling technique can be used to obtain a statistically representative sample of outcomes after a specified number of transitions from the initial to the final state. A single simulation run then represents a random path from the initial (*S0*) to the final state (*SN*). The probabilities of transition from state to state can be found using the Monte Carlo simulation technique.

The statistical approach is based on the assumption that the distribution of particles is uniform in space for each particle size. It is also assumed that the condition for a particle to be split is associated with the possibility of its location between the colliding balls within the so-called "grinding volume" (see Figure 3.9, where the grinding volume is shaded).

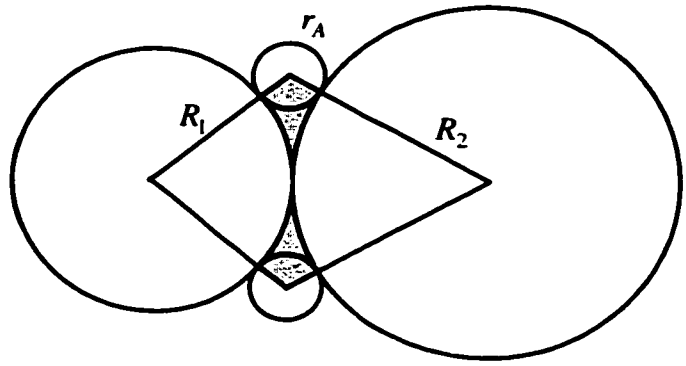


Figure 3.9. The grinding volume.

It follows then from these two assumptions that the probability of a particle of a specific size r_A to be inside the grinding volume is equal to the ratio of this volume V_{gr} to the total volume occupied by the particles in the shaker $V_{avail} = V_{shaker} - V_{balls}$, where V_{shaker} is the volume of the shaker, and V_{balls} is the total volume of balls moving inside the shaker. The grinding volume can be calculated approximately by the following formula

$$V_{gr} \approx 4\pi \frac{R_1 R_2}{R_1 + R_2} r_A^2, \tag{3.20}$$

where r_A is the radius of the particle A, and R_1 and R_2 are the radii of two collided balls.

Then, since it is assumed that the particle can be located at any point of the available volume V_{avail} with equal probability, the probability that the particle A is located inside the grinding volume is

$$P_{gr} = \frac{V_{gr}}{V_{avail}}, \tag{3.21}$$

Equation (3.21) is used to calculate the probability of grinding for each particle in the system. It can be seen that the probability of grinding decreases as the square of particle

radius. This is consistent with the experimental fact that the milling rate is greatly reduced when particles become small (see Sections 3.6.4 and 3.6.5 below).

It is assumed that if a particle with radius r_A participates in a collision, then it is split into two equal particles with radii $\frac{r_A}{\sqrt[3]{2}}$ (so that the total volume is conserved). Assuming that initially all particles have equal radii r_0, it is sufficient to keep track only of the number of particles N_i for each of the radii

$$r_i = r_0 2^{-i/3}\ ,\ i = 0..k\ . \tag{3.22}$$

Here r_k is the minimum particle radius, such that the particles with this radius do not break into smaller particles anymore. Usually, this radius is about 1 μm [Shaw et. al. 93].

As in the example of Figure 3.8, the distribution of particles according to their radii can be viewed as their placement in the corresponding baskets so that each basket B_i holds particles of radius r_i. Initially, all particles have radius r_0 and are placed into basket B_0. Denote the number of particles in a basket B_i as N_i. If a particle from basket B_i was involved in a collision, then one particle is removed from basket B_i and two particles are added into basket B_{i+1} (see Figure 3.10).

Since all particles in a basket have the same radius, the grinding probabilities are calculated by the same formula (equation (3.21)). Then, the number of particles from the same basket involved in the collision is described by the binomial distribution. Thus, the number N_i^c of particles with radius r_i participated in a collision is distributed as

$$P\left(N_i^c = x\right) = \binom{x}{N_i} p_i^x (1-p_i)^{N_i - x},\ x = 0..N_i, \tag{3.23}$$

where p_i is the grinding probability $P_{gr}(r_i)$ (see equation (3.21)) for a particular particle radius r_i. Note that the volume required for the number of crashed particles N_i^c can exceed the grinding volume V_{gr}. For the binomial distribution, however, the probability of such situation decreases with the number of particles.

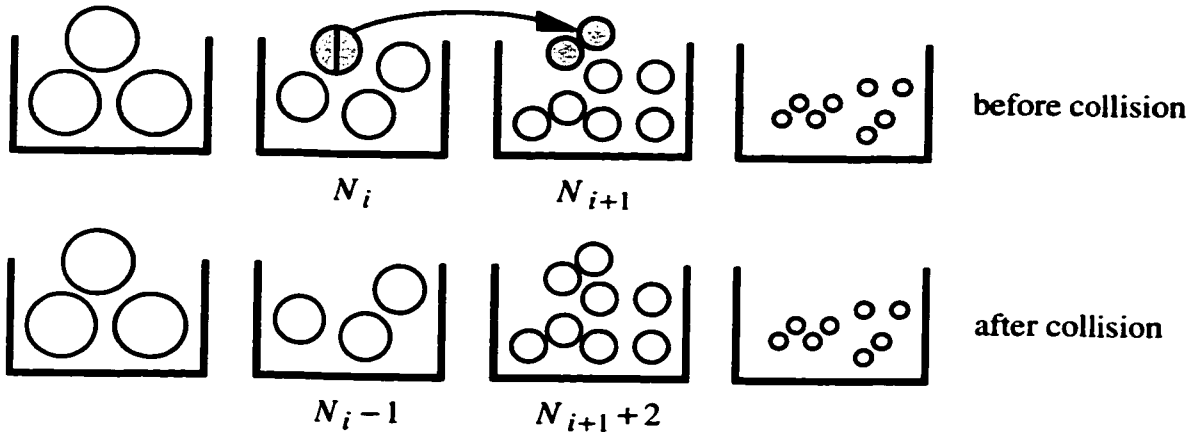


Figure 3.10. Effect of collision on redistribution of particles in baskets.

In summary, when a collision between two balls takes place, the particles are redistributed in baskets. For each basket B_i, $i = 0..k-1$, a random number of particles $0 \le N_i^c \le N_i$ involved in the collision is determined, and then N_i^c is deducted from N_i, and $2N_i^c$ is added to N_{i+1}. N_i^c is a discrete random variable distributed by the binomial law with parameters (p_i, N_i). The process is stopped when all particles are in the last basket B_k, i.e. all material is ground. The number of collisions, as well as the distribution of particle sizes during the grinding, are the objectives of the simulation.

3.6.3. Analytical grinding model

Although the model described above is much more efficient than the analysis of the complete graph of grinding events, it is still quite slow. In a normal run, the number of collisions required to grind the material can be as large as 10^{12}. Since each collision must

be handled separately, the run time could be quite long even on a supercomputer. To solve this problem, a continuous analytical model is introduced, which represents the average-case behavior of the statistical model.

The analytical model also considers baskets containing balls with the same radii. However, instead of recording the number of particles in a basket, the total mass of particles in each basket is considered. Then the incremental mass balance equation for a single basket is derived, which leads to a differential form of the mass flow equation.

Denote the mass in each basket by X_i, $i = 0..k$. Clearly, if a mass of one particle with radius r_i is m_i (which is proportional to r_i^3), then $X_i = N_i m_i$. Consider how masses of particles in baskets are redistributed after a collision. Since N_i^c is a discrete random variable distributed according to the binomial law with parameters (p_i, N_i), its mean value is $p_i N_i$, i.e., on average this many particles from basket B_i are ground due to a collision. Then the mass transferred from basket B_i into basket B_{i+1} is

$$p_i N_i m_i = p_i X_i, \; i = 0..k-1, \tag{3.24}$$

The mass transferred from the last basket B_k is always zero, since particles in this basket are not ground any further. Now a differential equation describing the change of mass of the material in baskets in time will be obtained. Assume that collisions occur in the system with frequency λ. Then during a time interval dt there will be λdt collisions. The mass transferred from basket B_i into basket B_{i+1} can be represented as $\lambda p_i X_i dt$. At the same time, the mass $\lambda p_{i-1} X_{i-1} dt$ is transferred into basket B_i from basket B_{i-1} for all $i = 1..k$ (no mass is transferred into the first basket B_0). Then the total change of mass in basket B_i during the time dt is

$$dX_i = \lambda(p_{i-1} X_{i-1} - p_i X_i)dt, \; i = 1..k-1, \tag{3.25}$$

Basket B_0 does not have any mass transferred into it, and therefore, $dX_0 = -\lambda p_0 X_0 dt$. The last basket B_k does not have any mass transferred out, thus, $dX_k = \lambda p_{k-1} X_{k-1} dt$. The resulting differential system describing the average-case behavior of the grinding process is

$$\dot{\mathbf{X}} = \lambda \mathbf{P} \mathbf{X}, \tag{3.26}$$

where $\mathbf{X} = \{X_i\}^T$, $i = 0..k$, and

$$\mathbf{P} = \begin{bmatrix} -p_0 & 0 & 0 & \dots & 0 & 0 & 0 \\ p_0 & -p_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -p_1 & p_2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -p_{k-2} & 0 & 0 \\ 0 & 0 & 0 & \dots & p_{k-2} & -p_{k-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & p_{k-1} & 0 \end{bmatrix}. \tag{3.27}$$

The initial conditions are set as

$$X_0(0) = M, \text{ and } X_i(0) = 0,\ i = 1..k., \tag{3.28}$$

This linear differential system has the following analytic solution:

$$X_m(t) = M \prod_{i=0}^{m-1} p_i \sum_{i=0}^{m} \frac{e^{-\lambda p_i t}}{\prod_{j=0, j \neq i}^{m} (p_j - p_i)}, m = 0..k. \tag{3.29}$$

The numerical results (see Sections 3.6.4 and 3.6.5 below) confirm that the continuous analytical model provides a very good approximation of the discrete statistical model. Moreover, by the law of large numbers, the average values of statistical model parameters calculated for a large number of runs converge to the analytical solution. Equation (3.29) gives an explicit dependence of the outcome of the grinding process on the system parameters.

3.6.4. Numerical experiments

A number of numerical experiments have been made. In the first series of experiments the effect of the external parameters of the shaker, such as the frequency and amplitude of vibrations, on the internal parameters, such as the rate of energy dissipation due to the heating of the balls, the number of collisions, the kinetic energy of the balls, and the rate of energy flow into the system was investigated. One of the interesting results was that the dissipation of energy inside the system H is proportional to the amplitude of shaker vibrations A and to the third degree of frequency f, $H \sim Af^3$ (see Figure 3.11 and Figure 3.12, where the following data was used in experiments: 100 steel balls, each of radius 3.2 mm; cylinder: height 103 mm, radius 31.5 mm).

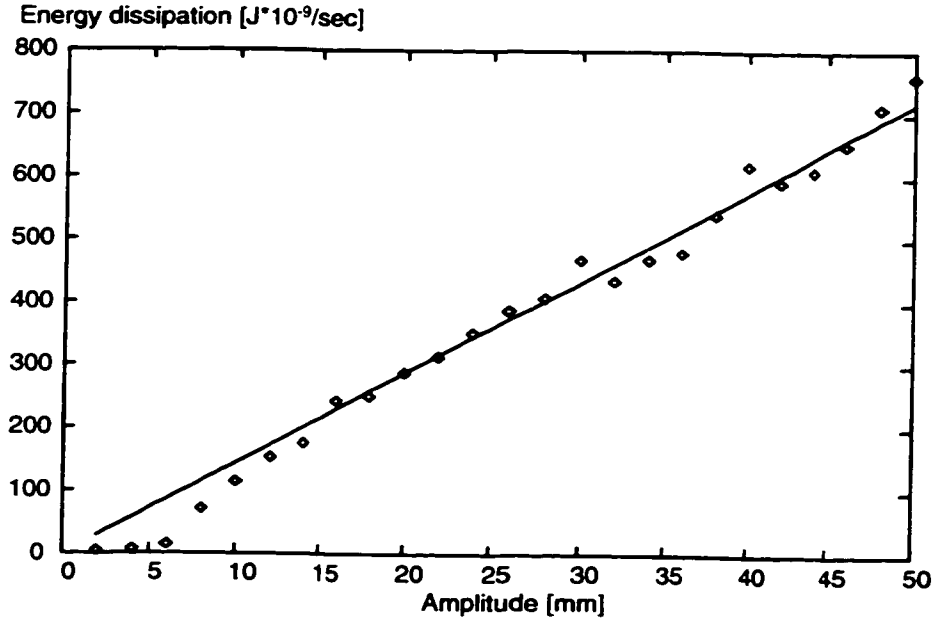


Figure 3.11. Energy dissipation vs. amplitude ($f = 52.6Hz$).

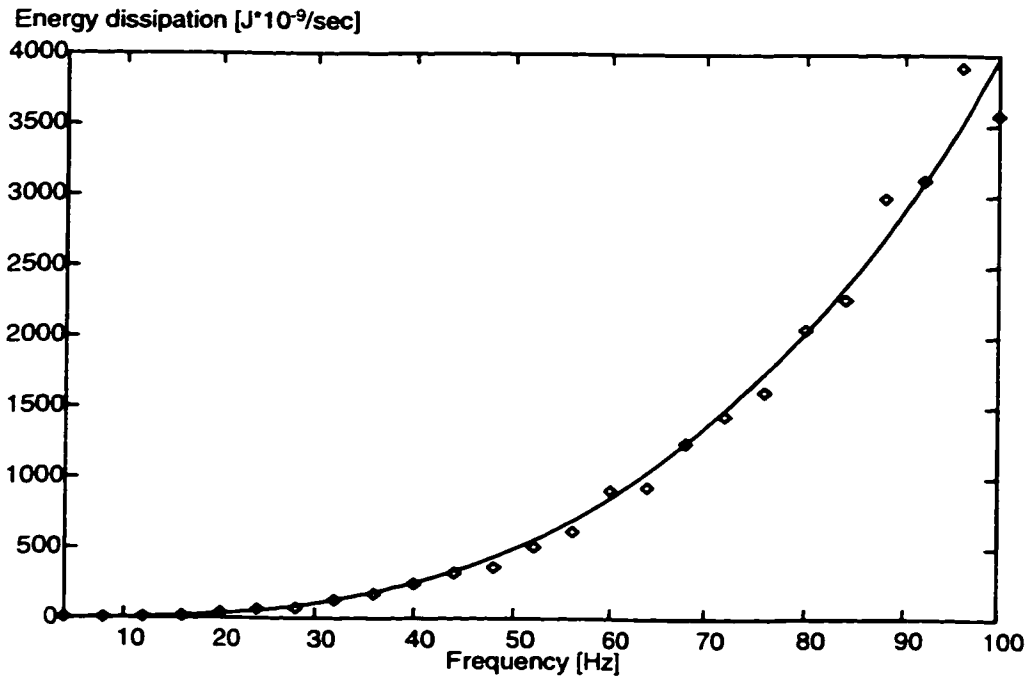


Figure 3.12. Energy dissipation vs. frequency of vibrations ($A = 10.9mm$).

The result that the internal milling energy is proportional to the third power of the operating frequency is consistent with the results of Streletskii [Streletskii 93], however the linear behavior of the energy with the amplitude is not.

3.6.5. Validation

A model of an ideal gas was constructed to validate the simulation software. The ideal gas was simulated by a system of perfectly elastic small balls representing molecules of ideal gas moving inside the motionless cylinder. All collisions were elastic, i.e. there was no energy loss in the system. Initially the all balls were assigned equal velocities in random directions. The relationship between the kinetic energy of the balls (which remains constant), pressure on the walls of the cylinder, and the volume of the cylinder was investigated. Numerical experiments have shown that the $PV = RT$ gas law is satisfied. Here P is the pressure of the gas, V is the volume of the cylinder, T is the temperature of the gas (which is assumed to be proportional to the internal energy of the

gas, i.e. the kinetic energy of balls), and R is a constant. This result is of a particular interest, because this relationship does not depend on the shape of the cylinder, i.e. the ratio between the height and the radius.

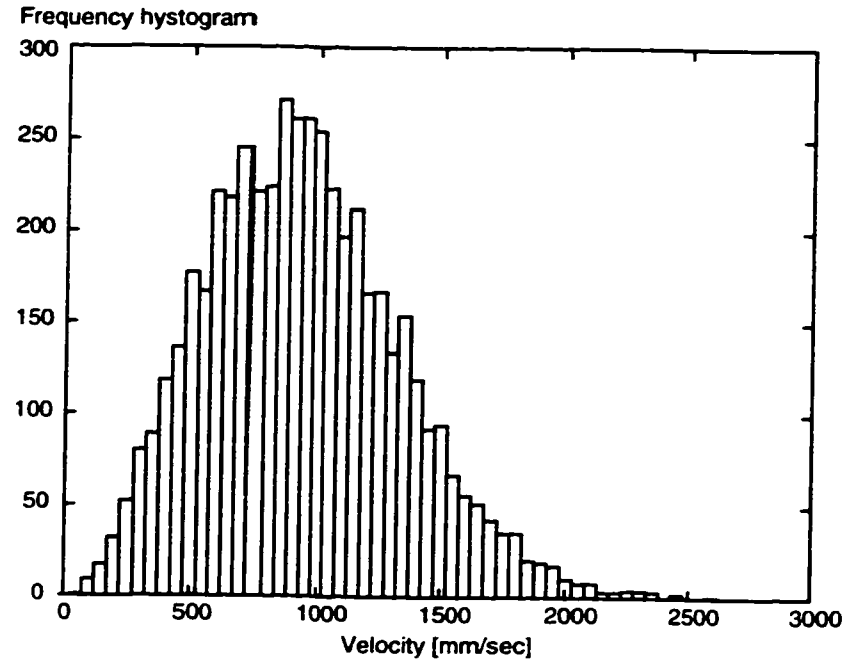


Figure 3.13. Distribution of velocities in an ideal gas model.

Another validating result is that the distribution of velocities of molecules, which are all assigned equal values at the beginning of the simulation, soon converges to the Maxwell distribution of velocities (see Figure 3.13). The initial magnitude of velocities in this experiment was set to 1000 mm/sec for all balls.

In the second series of experiments the particle radii distribution during the milling process was investigated. The results presented in the following figures were obtained for the following shaker configuration: ball-to-ball coefficient of restitution $\varepsilon_{bb} = 0.98$, ball-to-wall coefficient of restitution $\varepsilon_{bw} = 0.95$, shaker radius $R_s = 31.5mm$, shaker height $H_s = 103.0mm$, shaking frequency $f = 52.6Hz$, shaking amplitude $A = 10.9mm$, ball radius $R = 3.2mm$, the number of balls $N = 125$, ball density $d = 7850kg/m^3$ (steel).

For this set of parameters the collision rate of approximately 100,000 collisions/sec was obtained. The collision rate becomes quite stable after 0.1 sec of shaking. For this reason the simulation time to determine the number of collisions was 1 sec, and the computed effective collision rate was $\lambda = 100{,}000$ collisions/sec.

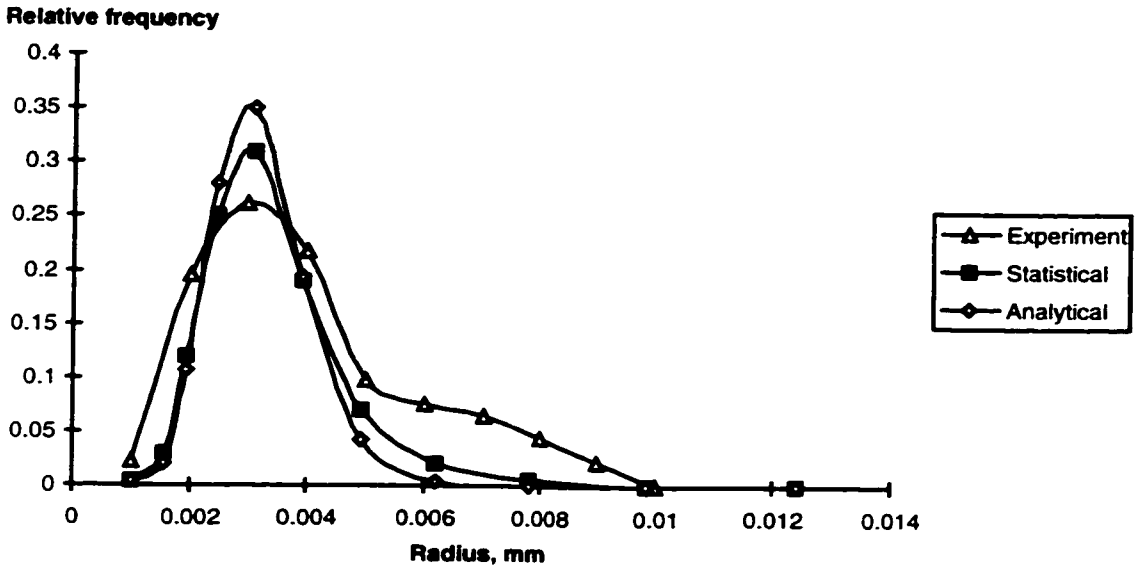


Figure 3.14. Particle radii distribution after 24 hours of milling.

The distribution of radii of particles after 24 hours of milling is presented in the Figure 3.14. Since the effective rate of collisions for the simulation was set to 100,000 collisions per second, the resulting distribution is obtained after approximately $8.64 \cdot 10^9$ collisions. The experimental data in the Figure 3.14 is taken from [Shaw et. al. 93]. The close correspondence between both simulation and experimental results is clearly seen. The solutions obtained from statistical and analytical models are almost equivalent.

Another validating result is that the distribution of radii is almost independent of the initial particle radius, i.e. the milling rate it significantly higher for larger particles, which means that they are very quickly ground to the smaller ones. Another very interesting result, which is also in agreement with the experiments [Shaw et. al. 93], is that the milling rate is independent of the amount of material put into the mill.

3.7. Conclusions

The Molecular Dynamics (MD) model was introduced and described in this chapter. The laws of motion, collision detection equations and collision laws were obtained. The discrete event-driven simulation scheme together with the types of events was described.

The MD model use was demonstrated in application to the simulation of a shaker ball mill used in mechanical alloying. The simulation allowed investigating the collision rate and energy transfer parameters of shaker ball mills. The input parameters of the model included geometry of the cylinder, filling ratios and ball characteristics. Discrete statistical and continuous deterministic models of the grinding process were introduced and compared favorably against each other and against experimental results.

CHAPTER 4. QUASI-STATIC MODEL

The Quasi-Static (QS) model is based on an approach similar to the Finite Element Method approach. Thus, the model is best suited for simulating granular matter in a steady or near-to-steady state, i.e. when dynamic changes in the system's topology do not occur frequently. Examples of such systems are silos, bins, piles [Claudin and Bouchaud 97], as well as many applications found in soil mechanics [Gudehus 96, Wan and Guo 98], fracture mechanics [Bazant and Ozbolt 90] and many other fields.

As opposed to the Molecular Dynamics approach, only long-range contacts between the particles are taken into account. The motion of particles in the system is considered to be very slow, so that the inertia forces are not taken into consideration. Moreover, the system is simulated in a quasi-static mode, i.e., the model does not have a time scale.

The system of particles is, in fact, modeled by a *truss system* (or a *beam system*), where each particle is represented by a node and each contact between two particles is represented by a spring. It is then possible to determine the stress distribution in the system under the influence of external forces (such as gravity forces) by solving a system of linear equations. This system can be obtained by analyzing stress-strain relationships in the nodes. Rather than solving the linear system each time when the external conditions change, the inverse stiffness matrix of the system is found and maintained. Thus, the stress distribution in the particle system can be obtained by direct matrix multiplication.

The Recursive Inverse Matrix Algorithm (RIMA) is introduced. It is used to maintain the inverse stiffness matrix as the topology of the system changes. Specifically, it defines the formulas for updating the inverse matrix when a link is added or deleted from the system. These operations are less expensive than recomputing the whole inverse matrix each time the topology of the system changes.

Certain topological changes, such as slips or micro-avalanches, can occur in quasi-static granular systems [Claudin and Bouchaud 97]. The algorithms for dealing with such micro-changes are introduced and analyzed.

The main advantage of the Quasi-Static model is that it computes the stress distribution in a quasi-static system directly using explicit formulas and it does not require solving differential or algebraic equations.

The QS model has certain limitations. The main limitation of the model is that it does not take into account inertia forces and, thus, is incapable of handling global topological changes, which lead to physical redistribution of particles, such as avalanches on the surface of the pile (as opposed to micro-avalanches related to slips). Another limitation of the model is that it can not handle free-flowing systems, i.e. the simulated system must contain at least one link to a fixed boundary.

4.1. Introduction

In many applications, such as fracture mechanics, granular mechanics, composite materials with delamination, structures in the process of construction, etc., either the connectivity between the discrete/finite elements or the number of these elements changes as a function of time or load/deformation. The conventional way of handling such systems is to generate and solve the governing equations each time the topology of the system changes. For large systems such an approach is computationally expensive.

In [Vinogradov 86b] an algorithm was presented which constructs an inverse matrix for an arbitrary finite element system by adding the elements one by one to an expanding sequence of subsystems. In other words, if the inverse matrix for a subsystem j is known, then the inverse matrix for an expanded system $(j+1)$ formed by this subsystem plus a new element is obtained by updating and expanding the inverse matrix for a subsystem j taking into account the properties of the new element. The effect of element subtraction is found in a similar manner. The algorithm thus is suitable for systems with variable topology since it does not require the solution of the system of equations each time the physical system changes.

In this chapter a customized version of this algorithm called the Recursive Inverse Matrix Algorithm (RIMA) is considered in detail for granular mechanics applications. In

granular systems some particles may form clusters which remain geometrically invariant during some time [Drescher and Josselin de Jong 72, Gustafson and Gustafson 96, Drake 90]. Such a cluster was called a Quasi-Rigid Body (QRB) in [Vinogradov and Springer 90] and further investigated in [Gavrilov and Vinogradov 96]. A QRB is thus an overconstrained system of particles and its integrity remains intact as long as all internal forces are compressive and no sliding between the particles takes place. In other words, a QRB is a system of particles with no internal degrees of freedom. Any specific cluster may have, however, a limited life span. Any attachment or detachment of particles changes the cluster size or topological connectivity between the particles. Thus, a QRB is a system with variable, in discrete moments of time, connectivity and size.

A similar approach for granular mechanics simulation was presented in [Takeuchi and Kawai 88]. Their Rigid Bodies Spring Model was used to represent a cluster of interconnected particles, and a customized incremental finite element scheme was employed to calculate internal stresses in the system.

4.2. Recursive Inverse Matrix Algorithm [Gavrilov and Vinogradov 97a]

Consider an arbitrary subsystem Q_k with k interconnected two-node elements and assume that the inverse matrix $\mathbf{Z}^k$ for this subsystem is known. The problem is to express the inverse matrix for an expanded subsystem Q_{k+1} through the matrix $\mathbf{Z}^k$ and the stiffness matrix for the $\mathbf{r}_{k+1}$ element.

It is assumed that the force-displacement relationship for an arbitrary element $\mathbf{r}_j$ is given in the form

$$\mathbf{P} = \mathbf{S}^j \mathbf{u}, \tag{4.1}$$

where $\mathbf{P}$ and $\mathbf{u}$ are the $2m$-order force and displacement vectors, respectively, $\mathbf{S}^j = \left[\mathbf{s}_{ik}^j\right]_{2\times 2}$, $\mathbf{s}_{ik}^j$ are the $m\times m$ blocks of the stiffness matrix, and m is the number of

degrees of freedom of the node. It is assumed that $\mathbf{S}^j$ is symmetric, which also guarantees that the inverse matrix $\mathbf{Z}^k$ is also symmetric.

When a new element $\mathbf{r}_{k+1}$ connecting two nodes α and β is added to the system, four different cases are considered based on the topology of the new link (see Figure 4.1).

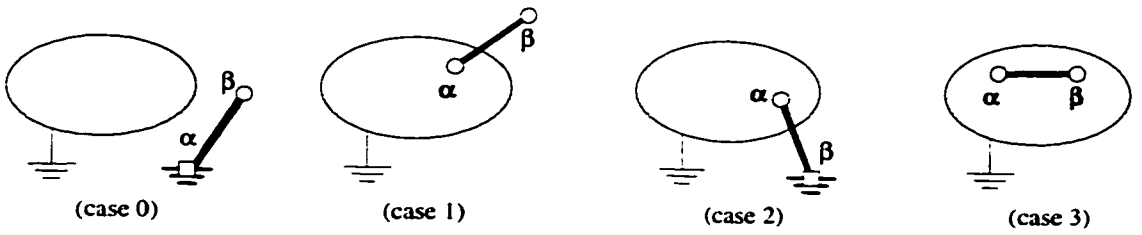


Figure 4.1. New element attachment cases.

Case 0: a new free node β is connected to a fixed node α;

Case 1: a new free node β is connected to a free node α already in the system;

Case 2: a fixed node β is connected to a free node α already in the system;

Case 3: two free nodes α and β already in the system are connected.

Note that in all cases the system is "grounded", i.e. it has at least one fixed node.

Each of these cases is considered separately.

4.2.1. Element attachment case 0

The element $\mathbf{r}_{k+1}$ is not coupled with the subsystem Q_k. In this case, the inverse matrix is simply extended with an additional row as

$$\mathbf{Z}_{ij}^{k+1} = \begin{cases} \mathbf{Z}_{ij}^{k} & \text{if } i,j<n \\ 0 & \text{if } i<n, j=n, \\ \mathbf{s}_{11}^{-1} & \text{if } i=j=n \end{cases} \tag{4.2}$$

where $\mathbf{Z}_{k+1}$ is the updated inverse stiffness matrix and $\mathbf{Z}_k$ is the known inverse stiffness matrix for the original subsystem Q_k.

4.2.2. Element attachment case 1

The element $\mathbf{r}_{k+1}$ is coupled with the subsystem Q_k at the node α while the other node β is unconstrained. The coupling of an element is accomplished by satisfying two requirements: 1) the compatibility of node displacements, and 2) the equality of internal node forces, whereas the latter, the vector $\mathbf{q}_\alpha$, is unknown. Treating this force as an external one, the displacements of the nodes of the subsystem Q_k are

$$\mathbf{u}^*_{(n-1)} = \mathbf{Z}^k \mathbf{F}^*_{(n-1)}, \tag{4.3}$$

where

$$\mathbf{F}^*_j = \begin{cases} \mathbf{F}_j & \text{if } j \neq \alpha \\ \mathbf{F}_j - \mathbf{q}_\alpha & \text{if } j = \alpha \end{cases} \tag{4.4}$$

where $\mathbf{Z}^k$ is the known inverse matrix, $j = 1..n$, $\alpha \leq n$, * denotes the subsystem Q_k as a free-body system, and the subscript in brackets indicates the order of the block vector.

The force-displacement relationship for the element $\mathbf{r}_{k+1}$ given by the equation (4.1) in this case is

$$\begin{bmatrix} \mathbf{q}_\alpha \\ \mathbf{F}_n \end{bmatrix} = \begin{bmatrix} \mathbf{s}_{11} & \mathbf{s}_{12} \\ \mathbf{s}_{21} & \mathbf{s}_{22} \end{bmatrix}^{k+1} \begin{bmatrix} \mathbf{u}_\alpha \\ \mathbf{u}_n \end{bmatrix}. \tag{4.5}$$

Satisfying the compatibility requirements, namely, that at the node α $\mathbf{u}^*_\alpha = \mathbf{u}_\alpha$, the displacement $\mathbf{u}_\alpha$ is found from the equation (4.3) to be

$$\mathbf{u}_\alpha = \mathbf{Z}^k_\alpha \mathbf{F}_{(n-1)} - \mathbf{Z}^k_{\alpha\alpha} \mathbf{q}_\alpha, \tag{4.6}$$

where $\mathbf{Z}^k_\alpha$ is the α-th row of the matrix $\mathbf{Z}^k$.

Solving equations (4.5) and (4.6) together, the unknown forces and displacements are obtained

$$\begin{bmatrix}\mathbf{q}_\alpha\\ \mathbf{u}_n\end{bmatrix}=\begin{bmatrix}\mathbf{a}_\alpha\\ \mathbf{b}_n\end{bmatrix}(\mathbf{F}_1,\ldots,\mathbf{F}_n)^{\mathrm{T}}, \tag{4.7}$$

where $\mathbf{a}_\alpha^{k+1}=\left[\mathbf{a}_{\alpha j}^{k+1}\right]_{1\times n}$ and $\mathbf{b}_n^{k+1}=\left[\mathbf{b}_{nj}^{k+1}\right]_{1\times n}$, and the block elements $\mathbf{a}_{\alpha j}^{k+1}$ and $\mathbf{b}_{nj}^{k+1}$ are given in the form

$$\begin{bmatrix}\mathbf{a}_{\alpha j}\\ \mathbf{b}_{nj}\end{bmatrix}=\left(\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}+\begin{bmatrix}\mathbf{s}_{11} & \mathbf{s}_{12}\\ \mathbf{s}_{21} & \mathbf{s}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{Z}_{\alpha\alpha}^{k} & \mathbf{0}\\ \mathbf{0} & -\mathbf{I}\end{bmatrix}\right)^{-1}\begin{cases}\begin{bmatrix}\mathbf{s}_{11}^{k+1}\mathbf{Z}_{\alpha j}^{k}\\ \mathbf{s}_{21}^{k+1}\mathbf{Z}_{\alpha j}^{k}\end{bmatrix} & \text{if } j\le n-1\\ \begin{bmatrix}\mathbf{0}\\ -\mathbf{I}\end{bmatrix} & \text{if } j=n\end{cases} \tag{4.8}$$

Using $\mathbf{q}_\alpha$ and $\mathbf{u}_n$ from the equation (4.7), the solution for the expanded substructure Q_{k+1} can now be written in the form

$$\mathbf{u}_{(n)}=\mathbf{Z}^{k+1}\mathbf{F}_{(n)}, \tag{4.9}$$

and, as a result, the following recursive relations between the block entries of the two inverse matrices associated with two consecutive substructures Q_k and Q_{k+1} can be established:

$$\mathbf{Z}_{ij}^{k+1}=\begin{cases}\mathbf{Z}_{ij}^{k}-\mathbf{Z}_{i\alpha}^{k}\mathbf{a}_{\alpha j}^{k+1} & \text{if } i,j<n\\ -\mathbf{Z}_{i\alpha}^{k}\mathbf{a}_{\alpha j}^{k+1} & \text{if } i<n,\ j=n\\ \mathbf{b}_{nj}^{k+1} & \text{if } i=n\end{cases}. \tag{4.10}$$

4.2.3. Element attachment case 2

The element $\mathbf{r}_{k+1}$ is coupled with the subsystem Q_k at the node α while the other node β is fixed. In this case $\mathbf{u}_n^{k+1}$ is zero in the equation (4.7), and the latter is reduced to

$$\mathbf{q}_\alpha=\mathbf{a}_\alpha(\mathbf{F}_1,\ldots,\mathbf{F}_{n-1})^{\mathrm{T}}, \tag{4.11}$$

where T is the transposition sign, and the block components of $\mathbf{a}_\alpha$ are defined by

$$\mathbf{a}_{\alpha j} = \left(\mathbf{I} + \mathbf{s}_{11}^{k+1}\mathbf{Z}_{\alpha\alpha}^{k}\right)^{-1}\mathbf{s}_{11}^{k+1}\mathbf{Z}_{\alpha j}^{k},\ j = 1..n-1. \tag{4.12}$$

Following the same procedure as before, the recursive relations between the block entries of the two inverse matrices characterizing substructures Q_k and Q_{k+1} can be obtained in the form

$$\mathbf{Z}_{ij}^{k+1} = \mathbf{Z}_{ij}^{k} - \mathbf{Z}_{i\alpha}^{k}\mathbf{a}_{\alpha j},\ i, j = 1..n-1. \tag{4.13}$$

4.2.4. Element attachment case 3

The element $\mathbf{r}_{k+1}$ is coupled with the substructure Q_k at two nodes α and β ($\alpha < \beta$). The two unknown internal forces are found in a similar way, and they are equal to

$$\begin{bmatrix}\mathbf{q}_\alpha \\ \mathbf{q}_\beta\end{bmatrix} = \begin{bmatrix}\mathbf{a}_\alpha \\ \mathbf{a}_\beta\end{bmatrix}(\mathbf{F}_1,\ldots,\mathbf{F}_{n-1})^{\mathrm{T}}, \tag{4.14}$$

where the block matrices $\mathbf{a}_{\alpha j}$ and $\mathbf{a}_{\beta j}$ are given by

$$\begin{bmatrix}\mathbf{a}_{\alpha j} \\ \mathbf{a}_{\beta j}\end{bmatrix} = \left(\begin{bmatrix}\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}\end{bmatrix} + \begin{bmatrix}\mathbf{s}_{11} & \mathbf{s}_{12} \\ \mathbf{s}_{21} & \mathbf{s}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{Z}_{\alpha\alpha}^{k} & \mathbf{Z}_{\alpha\beta}^{k} \\ \mathbf{Z}_{\beta\alpha}^{k} & \mathbf{Z}_{\beta\beta}^{k}\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbf{s}_{11}^{k+1}\mathbf{Z}_{\alpha j}^{k} + \mathbf{s}_{12}^{k+1}\mathbf{Z}_{\beta j}^{k} \\ \mathbf{s}_{21}^{k+1}\mathbf{Z}_{\alpha j}^{k} + \mathbf{s}_{22}^{k+1}\mathbf{Z}_{\beta j}^{k}\end{bmatrix}, \tag{4.15}$$

where $j = 1..n-1$.

In this case the recursive relations are as follows

$$\mathbf{Z}_{ij}^{k+1} = \mathbf{Z}_{ij}^{k} - \mathbf{Z}_{i\alpha}^{k}\mathbf{a}_{\alpha j} - \mathbf{Z}_{i\beta}^{k}\mathbf{a}_{\beta j},\ i, j = 1..n-1. \tag{4.16}$$

Equations (4.2), (4.10), (4.13) and (4.16) can be used to compute an inverse matrix for any irregular system consisting of two-node finite elements. Note that the size of the inverse matrix at every step is equal to the current number of nodes in the system, and that the resulting block matrix $\mathbf{Z}^{k+1}$ is symmetric. Therefore, it is sufficient to compute only a lower diagonal half of $\mathbf{Z}^{k+1}$ in applications.

4.3. Cluster as an equivalent beam system

In a 2D (3D) cluster of disks (spheres) the center line connecting any two particles in contact can be considered as a line of an equivalent beam system, i.e. each link has a longitudinal and a transverse stiffness (two transverse stiffnesses). The latter property is due to the tangential stiffness of two particles in contact, whereas the former is due to their normal stiffness. In a planar case, a link is represented by a 2 degrees of freedom beam, while in a most general case all 6 degrees of freedom of a 3D beam can be considered.

Thus a link between two spheres is considered as a two-node element with d degrees of freedom at each node. The stiffness of such an element in a global coordinate system is characterized by a block matrix $\mathbf{S}$

$$\mathbf{S}=\begin{bmatrix} \mathbf{C} & -\mathbf{C} \\ -\mathbf{C} & \mathbf{C} \end{bmatrix}, \tag{4.17}$$

where $\mathbf{C}$ is a $d\times d$ matrix, $\mathbf{C}=\mathbf{T}_\varphi^T\mathbf{C}_0\mathbf{T}_\varphi$, $\mathbf{C}_0$ is a diagonal matrix, and $\mathbf{T}_\varphi$ is the coordinate transformation matrix (from local to global).

For a plane beam system,

$$\mathbf{C}_0=\begin{bmatrix} K_\eta & 0 \\ 0 & K_\tau \end{bmatrix},\ \mathbf{T}_\varphi=\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}, \tag{4.18}$$

where K_η and K_τ are the normal and tangential stiffnesses of the beam, respectively.

For a particular case of the element described by the equation (4.17), equations (4.2), (4.10), (4.13) and (4.16) can be simplified.

The equation for case 0 is written as

$$\mathbf{Z}^{k+1}=\left[\begin{array}{c:c} \mathbf{Z}^k & 0 \\ \hdashline 0 & \mathbf{C}^{-1} \end{array}\right]. \tag{4.19}$$

When expression (4.17) is substituted into the equation (4.7), $\mathbf{a}_{\alpha j}$ and $\mathbf{b}_{nj}$ can be obtained as solutions of the following linear systems:

$$\begin{bmatrix} \mathbf{I}+\mathbf{C}\mathbf{Z}_{\alpha\alpha}^{k} & \mathbf{C} \\ -\mathbf{C}\mathbf{Z}_{\alpha\alpha}^{k} & -\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{a}_{\alpha j} \\ \mathbf{b}_{nj} \end{bmatrix} = \begin{cases} \begin{bmatrix} \mathbf{C}\mathbf{Z}_{\alpha j}^{k} \\ -\mathbf{C}\mathbf{Z}_{\alpha j}^{k} \end{bmatrix} & \text{if } j \le n-1 \\ \begin{bmatrix} \mathbf{0} \\ -\mathbf{I} \end{bmatrix} & \text{if } j = n \end{cases} . \tag{4.20}$$

The solution of the latter is found to be in the form:

$$\begin{bmatrix} \mathbf{a}_{\alpha j} \\ \mathbf{b}_{nj} \end{bmatrix} = \begin{cases} \begin{bmatrix} \mathbf{0} \\ \mathbf{Z}_{\alpha j}^{k} \end{bmatrix} & \text{if } j \le n-1 \\ \begin{bmatrix} -\mathbf{I} \\ \mathbf{Z}_{\alpha\alpha}^{k}+\mathbf{C}^{-1} \end{bmatrix} & \text{if } j = n \end{cases} . \tag{4.21}$$

By substituting the equation (4.21) into the equation (4.10), the following expression for the inverse matrix $\mathbf{Z}^{k+1}$ for the case 1 is obtained:

$$\mathbf{Z}^{k+1} = \left[\begin{array}{c:c} \mathbf{Z}^{k} & \left(\mathbf{Z}_{\alpha}^{k}\right)^{\mathrm{T}} \\ \hdashline \mathbf{Z}_{\alpha}^{k} & \mathbf{Z}_{\alpha\alpha}^{k}+\mathbf{C}^{-1} \end{array}\right], \tag{4.22}$$

where $\mathbf{Z}_{\alpha}^{k} = \left[\mathbf{Z}_{\alpha j}^{k}\right]$, $j = 1..n$, is the α-th row of the matrix $\mathbf{Z}^{k}$.

Thus, the updated matrix $\mathbf{Z}^{k+1}$ is found by performing only one calculation for the $(k+1)$-th diagonal block and copying the contents of the α-th row.

For the remaining two cases, the matrix updates are obtained in a similar fashion in the form

$$\mathbf{Z}^{k+1} = \mathbf{Z}^{k} + \Delta^{k},\ \Delta^{k} = \left[\Delta_{ij}^{k}\right],\ i, j = 1..n, \tag{4.23}$$

where for the case 2

$$\Delta_{ij}^{k} = -\left(\mathbf{Z}_{i\alpha}^{k}\right)^{\mathrm{T}} \mathbf{X} \mathbf{Z}_{\alpha j}^{k}, \tag{4.24}$$

$$\mathbf{X} = \left(\mathbf{C}^{-1} + \mathbf{Z}_{\alpha\alpha}^{k}\right)^{-1}, \tag{4.25}$$

and for the case 3

$$\Delta_{ij}^{k} = -\left(\mathbf{Z}_{i}^{*}\right)^{\mathrm{T}} \mathbf{X} \mathbf{Z}_{j}^{*}, \tag{4.26}$$

$$\mathbf{Z}_{i}^{*} = \mathbf{Z}_{\alpha i}^{k} - \mathbf{Z}_{\beta i}^{k}, \tag{4.27}$$

$$\mathbf{X} = \left(\mathbf{C}^{-1} + \left(\mathbf{Z}_{\alpha}^{*} - \mathbf{Z}_{\beta}^{*}\right)\right)^{-1}. \tag{4.28}$$

Note that for the first two cases the matrix is bordered with an additional row and a column, and for the last two cases, the entire matrix is updated, while its size does not change. It should be pointed out that the update is obtained by a simple cross-multiplication of a row and a column, which can be performed very efficiently on a parallel computer. Thus, the form of the updated matrix has a potential for an efficient handling of the variability of the system. In addition, since the matrix $\mathbf{Z}$ is symmetrical, it is sufficient to store and update only a half of it, for example, the lower left triangle including the main diagonal.

4.4. Algorithm description

The particles are added to the system one by one. The addition of a particle may lead to a sequence of slides and changes in the topology of the system of particles. The next particle is added after the previous one has settled and no more slides and topological changes have been detected.

The particles are represented by disks (spheres). The contact between two disks or between a disk and a boundary is modeled by an elastic element connecting particle centers. The element is defined by normal stiffness K_{η} and tangential stiffness K_{τ}. Damping is not taken into account because the system is considered to be in a quasi-static

mode. The Coulomb friction is modeled by the friction coefficient μ. In the following, a contact between two particles or a particle and a boundary will be referred to as a link.

The simulation starts with addition of the first particle. It is connected to a boundary with one or more links. The first link is inserted and the inverse stiffness matrix is computed using equation (4.19) (case 0). The rest of the links are classified as case 2 and are processed using equation (4.22). When each of the remaining particles is added to the system, it is usually connected by 2 or more links. Each of the links is then classified and processed using the appropriate formula.

4.4.1. Computation of stresses in links

By applying the above algorithm, the inverse stiffness matrix of the system is known at each intermediate step. Assume now that n particles are added to the system. Then the stresses in links are calculated as follows.

First, the particle displacements $\mathbf{d}_i = (d_{xi}, d_{yi})^T$, $i = 1..n$ are found by multiplying the inverse stiffness matrix by the vector of external forces:

$$\mathbf{D} = \mathbf{Z}\mathbf{F}, \tag{4.29}$$

where $\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \\ \dots \\ \mathbf{d}_n \end{bmatrix}$, and $\mathbf{F} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \dots \\ \mathbf{f}_n \end{bmatrix}$ is the column of external forces acting on the particles, $\mathbf{f}_i = (f_{xi}, f_{yi})^T$, $i = 1..n$.

Let us consider a link l connecting two nodes α and β, and let us call α as the left node, and β as the right node. Then the relative displacement of two adjacent nodes is found as

$$\boldsymbol{\delta} = (\delta_x, \delta_y)^T = (d_{x\alpha} - d_{x\beta}, d_{y\alpha} - d_{y\beta})^T. \tag{4.30}$$

The vector $\boldsymbol{\delta}$ can be described in a coordinate system associated with the link:

$$\boldsymbol{\delta}^* = (\delta_\eta, \delta_\tau)^T = \mathbf{T}_\varphi \boldsymbol{\delta}, \tag{4.31}$$

where $\mathbf{T}_\varphi$ is the coordinate transformation matrix (see the equation (4.18)).

Finally, the normal and tangential stresses in the link are found as

$$\mathbf{s} = (s_\eta, s_\tau)^T = \mathbf{C}_0 \boldsymbol{\delta}^* = (K_\eta \delta_\eta, K_\tau \delta_\tau)^T, \tag{4.32}$$

where $\mathbf{C}_0$ is the normalized stiffness matrix (see the equation (4.18)). Note that both normal and tangential stresses can be of both signs.

4.5. Slides and topology changes

A granular system is said to be in *static equilibrium* if none of the following conditions apply:

1) there exists a link with a negative normal stress (stretched link);
2) there exists a pair of disks, which overlap (i.e. the distance between their centers is less than the sum of their radii), but are not connected by a link;
3) there exists a link, where the absolute value of tangential stress $|s_\tau|$ exceeds the maximum value $s_\tau^{\max}$, which is determined by the friction coefficient $s_\tau^{\max} = \mu s_\eta$.

If at any step of the simulation the system leaves the static equilibrium state, then a special algorithm is applied, which modifies the system topology so that it is returned to the static equilibrium. This algorithm is described below.

The previous section described an algorithm, which allows computation of the stress distribution in the system on each step provided that the topology remains invariant. This is not the case, however, in granular systems, and the following three cases of topological changes that occur in the system must be taken into account:

disconnection: a link with a negative normal stress (negative link) must be removed.

connection: if two disconnected nodes come into contact, a link must be inserted.

slide: if the ratio between the tangential and normal stresses exceeds the friction coefficient, then a limitation on the tangential stress is imposed.

4.5.1. Detection of connections and disconnections

The disconnection of a link can be detected by analyzing the normal stresses in a link. A link must be disconnected if the normal stress becomes negative ($s_\eta < 0$). When a disconnected link is detected, it is removed from the system by attaching an imaginary link with negative stiffness in parallel to the original link, which results in disabling of the original link.

The connection is detected by analyzing the distances between the centers of particles. The condition of connection between the two nodes α and β is

$$\left\| \mathbf{p}_\alpha - \mathbf{p}_\beta \right\| \le r_\alpha + r_\beta , \tag{4.33}$$

where $\mathbf{p}_\alpha$, $\mathbf{p}_\beta$, r_α and r_β are the coordinates of centers and radii of particles, respectively. Note that for a fixed node the radius is assumed to be zero.

4.5.2. Slide detection and analysis

The condition of the slide in the link is as follows:

$$|s_\tau| > \mu s_\eta , \tag{4.34}$$

where μ is the friction coefficient in the link, s_τ is the tangential stress, and s_η is the normal stress.

The maximum value of the tangential stress is $s_\tau^{max} = \mu s_\eta$. Consequently, the maximum relative tangential displacement in the link is $\delta_\tau^{max} = \frac{s_\tau^{max}}{K_\tau} = \mu \frac{K_\eta}{K_\tau} \delta_\eta$. A slide in the link decreases the absolute value of the tangential stress to s_τ^{max} and the absolute value of tangential displacement to δ_τ^{max}. Thus, the slide distance δ_{slide} is determined as $|\delta_{slide}| = |\delta_\tau| - \delta_\tau^{max}$, and $\text{sgn}(\delta_{slide}) = -\text{sgn}(\delta_\tau)$. Note that if a slide is detected then the absolute value of the original displacement $|\delta_\tau|$ is greater than the maximum displacement δ_τ^{max}, which ensures that $|\delta_{slide}| > 0$. Also note that the direction of the slide is always opposite to the initial displacement δ_τ. The slide in a link is illustrated in the Figure 4.2.

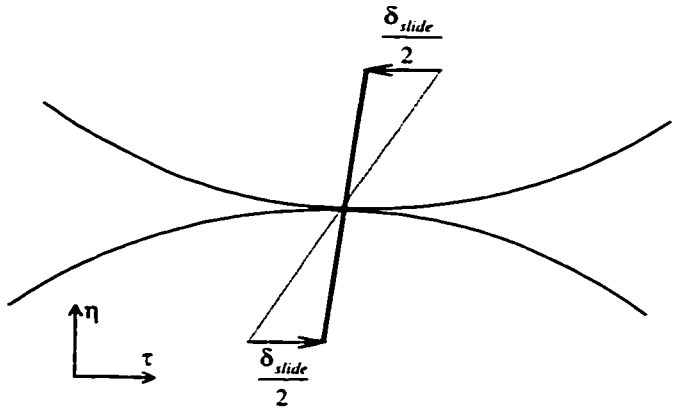


Figure 4.2. Slide in a link.

One of the ways to handle a slide in a link is to remove the link and then insert the updated link, which is a link with the same stiffness, but attached to a pair of shifted nodes. Insert/remove operations of this sort are computationally expensive, because they require a full matrix update. This operation can be simplified by taking into account that the stiffness matrix $\mathbf{C}$ of the updated link is almost identical to that of the original link because only small deformations are assumed in the analysis of a given truss system. It

follows that the link is rotated by a very small angle, since the slide distance is very small compared to the length of the link. Then, instead of removing a link and inserting an almost identical one, it is sufficient to introduce an internal force (a *prestress*), which is equal to $K_\tau \delta_{slide}$. As a result, the force $K_\tau \delta_{slide}$ is applied to the left end of the link and the force $-K_\tau \delta_{slide}$ is applied to the right end of the link (see Figure 4.2). Taking into account these internal stresses, the equation (4.32) becomes

$$\mathbf{s} = (s_\eta, s_\tau)^T = (K_\eta \delta_\eta, K_\tau (\delta_\tau - \delta_{slide}))^T . \tag{4.35}$$

Note that the current value of the slide distance δ_{slide} is stored for each link and the consecutive stress computations use this value.

The energy transformed to heat due to a slide is computed as

$$E_{slide} = |K_\tau \delta_\tau \delta_{slide}| . \tag{4.36}$$

4.5.3. Dependent topology changes

At each step, after a disk has been added to the system, the matrix is updated and the stress distribution is recomputed. In the process, a number of potential topological changes may be detected. In some cases, these topological changes are not independent from each other and this leads to an algorithmic indeterminism. Consider the example presented in the Figure 4.3.

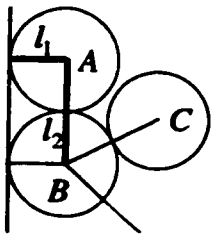


Figure 4.3. Dependent topological changes.

Assume that after attachment of the disk C the negative normal stresses are detected in two links l_1 and l_2. If both links are disconnected simultaneously, then the disk A

becomes completely disconnected from the system, thus "hanging in the air", which is not a correct configuration in this scenario. If the link l_2 is disconnected first, then the disk A is left "hanging" on the link l_1, which then will have to be disconnected on the next iteration. Finally, the only reasonable action is to disconnect the link l_1, thus letting the disk A to rest on top of the disk B.

To resolve the indeterminism, it is necessary to consider the dynamics of the system in order to find out how the stresses change in time after the addition of a new disk. Then, it would be possible to detect which of the links became disconnected first and thus follow the topological changes in time. Since the dynamic stress changes can not be computed in a quasi-static formulation, a number of a priori algorithms for the topology updates were considered. The following iterative algorithm was selected based on the best performance and plausible results.

Assume that after a new link was inserted, a number of topological changes were detected. As indicated above, there are three groups of topological changes: disconnections, connections and slides. The candidate links for the topological changes are placed in three lists: *DisconnectionsList*, *ConnectionsList* and *SlidesList*, correspondingly. Then the iterative algorithm works as follows:

Step 1. If the *ConnectionsList* is empty, then proceed to Step 2.

Otherwise,

Select the link l_{max} with the maximum projected stress from the list (the projected stress is calculated based on the overlap between the disconnected disks).
Perform the connection for l_{max}.
Recalculate stresses in the system.
Goto Step 1.

Step 2. If the *DisconnectionsList* is empty, then proceed to Step 3.

Otherwise,

Select the link l_{max} with minimum value of the normal stress s_η from the list (note that all links in the list have negative normal stress and thus the link with the maximum absolute value of normal stress is selected).

Disconnect link l_{max}.

Recalculate stresses in the system.

Goto Step 1.

Step 3. If the *SlidesList* is empty, stop (no more topological changes).

Otherwise,

Perform the slide operation for *all* links in the *SlidesList*.

Recalculate stresses in the system.

Goto Step 1.

The rationale for the suggested algorithm is as follows.

The first step finds a connected link with the maximum projected stress. The connection of the link may introduce some additional support, and, as a result, some of the disconnections and slides may be canceled. Only one link is selected because its connection might cancel other connections.

The second step disconnects the most negatively stressed link. Again, there might have been a chain of negative links "hanging" on the top one (as in the example in Figure 4.3). The disconnection of the most stressed link (usually, the topmost one) allows the disks in the whole chain to "slide" down and restore compressive contacts between the particles in the chain. Often such a disconnection leads to a sequence of slides.

The third step only runs if there are no connections or disconnections left unhandled. Unlike in the first two steps, the whole list of slid links is processed. This is because the slides are not as dependent from each other as connections and disconnections. Note that in most cases when a batch of slides is processed, it usually leads to more slides being detected, often in the same place. Such slides may continue for several iterations, until

the system comes to a stable state. This phenomenon is noted in granular systems, and is generally referred to as static avalanches or micro-avalanches or self-organization.

4.6. Limitations of the quasi-static model

The main limitation of the QS model is related to the fact that the dynamics of the simulated system is not being considered. This may lead to the appearance of infinite loops in simulation for certain configurations.

Consider the example presented in the Figure 4.4. After the particle A has been added to the system, an infinite sequence of slides occurs in the links marked in bold in the Figure 4.4. The reason for that is that the system has arrived to an unstable state, where particles A, B and C have to slide down the slope. This situation can not be handled by the quasi-static model because it can only consider very small movements of particles relative to their initial position. In this case, the dynamics of a cluster of particles has to be taken into account.

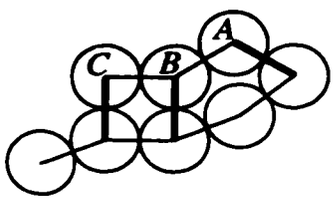


Figure 4.4. Infinite sequence of slides.

Another limitation of the quasi-static model is also related to the model's inability to handle dynamic changes on a global scale. The model does not consider the dynamic impact distribution in the system when a particle is dropped onto the pile with some initial velocity.

4.7. Application – silo simulation [Gavrilov and Vinogradov 99]

The state of particles in a silo during the filling process is of interest from macro- and micro-mechanical points of view. Both of these aspects were studied extensively using different approaches.

The main macro-mechanical property of interest is the distribution of particle weight on the walls and the bottom of the silo. It was long known [Janssen 1895] that the weight supported by the bottom of the silo is only a fraction of the total weight of the silo contents, while the most of the weight is supported by the silo walls [Brown and Richard 66, Nedderman 92]. A very interesting phenomenon is that the stress distribution can change dramatically with minor fluctuations in temperature, which lead to small changes in particle sizes [Liu and Nagel 92, Claudin and Bouchaud 97]. The stress distribution can change in a wide range during the consecutive fillings of the silo [Claudin and Bouchaud 97], i.e. it is strongly dependent on the history of the filling.

The micromechanics is associated with some local structural changes in the system, which take place at some moments of time during the filling process. These local changes are sudden and reflect the local instabilities, as such are often referred to as micro-avalanches or static avalanches [Claudin and Bouchaud 97]. The topology changes result in stress redistribution and are leading to a new equilibrium state, which means that the total potential energy of the system jumps to a lower level, as particles slide down to a more stable position.

There has been a number of models introduced for granular media in a silo, including continuum-mechanics models [Janssen 1895, Bouchaud et. al. 95, Ragneau and Aribert 93], finite element analysis [Ooi and She 97, Meng et. al. 97], statistical analysis [Chen et. al. 96], cellular automata models [Claudin and Bouchaud 97] and molecular dynamics models [Ristow and Herrmann 95, Sakaguchi et. al. 93, Hirshfeld et. al. 97, Rong et. al. 97]. The experimental work on stress distribution in silos has also been done [Connelly 83, Eibl 84, Jarrett et. al. 96, Tejchman and Gudehus 93].

It is considered to be computationally expensive to model the dynamics of the filling of a silo, because of a large number of topological changes occurring in the system, each requiring the regeneration and solution of the equations describing the system. Therefore, many of the models are somewhat simplified or limited in their applications.

Continuum-mechanics models (see [Schwedes and Feise 95] for an overview) were introduced more than 100 years ago [Janssen 1895]. It is known [Wittmer et. al. 96], that the generic formulation based on balance of forces acting on a small element can not be used to describe the system completely – an additional physical postulate is required to close the equations. There has been a number of approaches suggested, e.g. the assumption that the stresses propagate along some fixed directions [Wittmer et. al. 97]. This particular approach explains the presence of "arches" and the pressure minimum under the apex of a sandpile. However, the continuum-mechanics approach can not model the micro-mechanical properties of contacts between particles. It also can not be used to investigate the dynamics of slides and micro-avalanches during the filling of a silo.

Finite element models can be used to model both macro- and micro-mechanical properties of granulate medium in a silo. The accuracy of FE models is usually quite high at the expense of the high computational effort required to solve large algebraic systems. The most significant limitation of the FE models is that if the topology of the system changes (e.g. due to a slide or micro-avalanche), the whole system must be reformulated and solved. This may become very inconvenient for modeling of silo filling, since frequent topology changes take place.

Statistical models attempt to find the appropriate values of undetermined parameters used in continuum-mechanics models by analyzing the experimental data. Thus, while they can be used to obtain statistically-correct results for some specific configurations, they inherit the lack of ability to analyze the micro-mechanical properties of granulates in the silo.

Cellular automata models can simulate very large granulate systems at very little computational expense. When the evolution laws are appropriately selected, the models can produce some feasible output. However, only macro-mechanical properties can be modeled, since the micro-mechanics is predefined by a set of simple evolution laws.

Molecular dynamics models are based on the Distinct Element method (DEM) [Cundall and Strack 79]. DEM has become a dominant tool in granular materials research due to its flexibility and simplicity of implementation. It can model both micro- and macro-mechanical properties of a granular solid. The main limitation of the approach is the lack of error-control mechanisms. In order to obtain a reasonably accurate result, the timestep of the algorithm should be very small, which can be quite computationally expensive for a large system. Another limitation is that there is no explicit algorithm for timestep selection, i.e. it has to be determined by trial-and-error for each particular configuration.

A more sophisticated tool to study micro events in a silo during its filling is developed. The stress distribution is recomputed after each particle is dropped into the silo and settled. The method is based on the maintenance of the inverse stiffness matrix of the system, which allows exact computation of the forces acting in the system. Rather than being recomputed explicitly on each step, the matrix is updated to reflect changes in the system's topology. The Recursive Inverse Matrix Algorithm (RIMA) [Gavrilov and Vinogradov 97a] is applied to maintain the inverse matrix.

In order to account for the topological changes and slides in the system due to the stress redistribution, the whole system is analyzed after each particle is dropped. When the tangential stress in a contact between two adjacent particles exceeds the limit imposed by the friction coefficient, a slide occurs in that contact. Usually, a slide triggers a sequence of slides among the neighboring particles, thus leading to a micro-avalanche in the system. Such avalanches result in global stress redistribution representing a more stable system state, in which it remains until the next avalanche occurs.

4.7.1. Problem definition

A particle is dropped without initial velocity into a planar rectangular silo through the orifice at its top (see Figure 4.5). The addition of a particle may lead to a sequence of slides and changes in the topology of the system of particles. The next particle is dropped

after the previous one has settled and no more slides and topological changes have been detected.

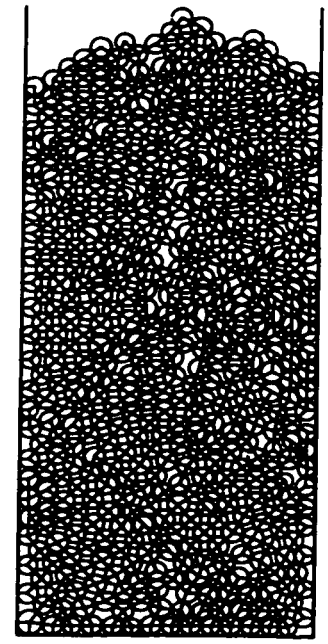


Figure 4.5. A silo with 500 particles.

The particles are represented by disks with normally distributed radii. The contact between two disks or between a disk and a silo wall is modeled by an elastic element connecting particle centers.

The mechanical properties of a link can be defined based [Mindlin and Deresiewicz 53]. Under small deformations, the stress-displacement relationship can be linearized:

$$s = K\delta, \tag{4.37}$$

where s is the stress in the contact (either normal or tangential), K is the constant stiffness coefficient (either normal or tangential) and δ is the displacement (again, either normal or tangential).

Consider a contact between two spheres of the same radius R. Assume that a link is subjected to a constant compressive normal stress s_η. The maximum absolute value of

the tangential stress is defined as $s_\tau^{max} = \mu s_\eta$, where μ is the friction coefficient. When the computed value of the tangential stress s_τ exceeds $s_{\tau max}$, a *slide* occurs in the link. The maximum tangential displacement in the link is defined as

$$\delta_\tau^{max} = \frac{3(2-\vartheta)\mu s_\eta}{16\vartheta a} \text{ [Mindlin and Deresiewicz 53],} \tag{4.38}$$

where $a = \left(\frac{3(1-\nu^2)}{4E} s_\eta R\right)^{1/3}$ is the contact radius from Hertz theory [Timoshenko and Goodier 51], ν is Poisson's ratio, E is Young's modulus and ϑ is the shear modulus of the material. Then the tangential stiffness can be approximated as

$$K_\tau = \frac{s_\tau^{max}}{\delta_\tau^{max}} = \frac{16\vartheta a}{3(2-\vartheta)}. \tag{4.39}$$

And the normal stiffness of a sphere is

$$K_\eta = \frac{4\vartheta a}{1-\nu} \tag{4.40}$$

The stiffnesses K_η and K_τ define the contact properties of a link while $|s_\tau| \le s_\tau^{max}$, i.e. until a slide occurs. If a slide takes place, to be correct, a rigid body motion should be considered. Using this dynamic approach is beyond the scope of the QS model. Instead, a quasi-static slide resolution algorithm is used (see Section 4.5).

The dimensions of the silo and the material properties are selected to match the generic parameters provided in the CA-SILO Collaborative Action document [Rotter et. al. 95]. The main goal of this project is to compare various approaches to silo simulation. A decision to use these parameters was made in order to compare our results with a number of other implementations solving the same problem. The parameters provided include the values for the normal and tangential stiffnesses characterizing the contacts between disks and disks and the silo walls, friction coefficients and some other parameters.

The particles are randomly generated and dropped through an orifice above the silo. A cellular-automata algorithm is used to find the position of the particle after it has been dropped. The particle "rolls" down the slope of the heap until it finds a stable position, i.e. a position where it is supported from left and right by the previously placed particles or a silo wall. When a particle is added to the system, two new links are inserted (see Figure 4.6).

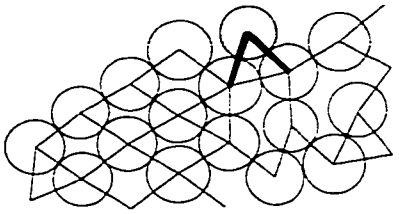


Figure 4.6. A new particle is connected by two links.

A system of particles with the links connecting them can be interpreted as a truss system with gravity forces applied to the centers of the particles. However, this "truss" system changes with any slide or placement of a new particle or disconnection between any two particles. Thus a system of particles during the filling is treated as a system with variable topology.

The aim of the simulation is to obtain the history of stress distributions in the links, of the slides and topology changes occurring in the system, and of the energy losses due to slides during the filling of the silo.

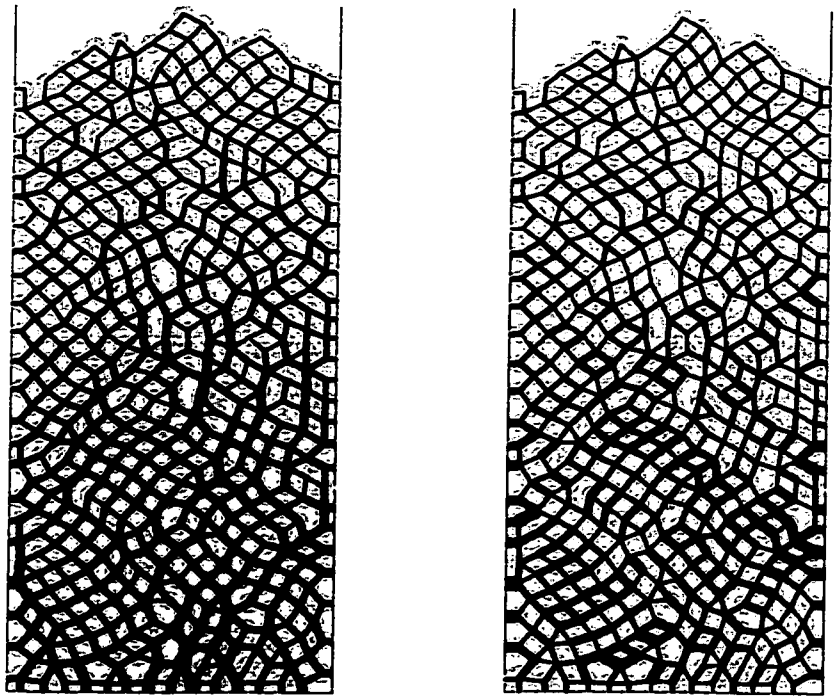
4.7.2. Numerical experiments

The algorithms were implemented in Borland Delphi 3 and the numerical experiments were run on a Pentium-166 with 32Mb of RAM. A silo filled with 500 disks was selected as a sample problem (see Figure 4.5). The parameters of the silo are taken from [Rotter et. al. 95]. The width of the silo is $0.3m$ and the depth is $1m$. Radii of disks are distributed normally with mean $0.01m$ and variance 5%. The density of disks is $\rho = 1190 kg\, m^{-3}$. For a link between two particles the stiffness and friction coefficients are defined as

$K_\eta = 1.0\times10^6\, Nm^{-1}$, $K_\tau = 1.0\times10^5\, Nm^{-1}$, and $\mu = 0.43$. For a link between a particle and a silo wall the coefficients are defined as $K_\eta = 1.5\times10^6\, Nm^{-1}$, $K_\tau = 1.5\times10^5\, Nm^{-1}$ and $\mu = 0.33$.

In the first series of experiments the stress distributions occurring during the filling of a silo were investigated. The final distribution of normal and tangential stresses is presented in the Figure 4.7. The thickness of the line represents the level of stress (to a scale). The stress paths or "arches" can be clearly seen. It can also be noted that the angle of the stress paths is different from the angle of repose.

In the second series of experiments, dynamic topology changes (such as micro-avalanches) were investigated. The slides occurred after the 463th disk was dropped are marked with bold in the Figure 4.8.



Normal stress Tangential stress

Figure 4.7. Final stress distributions.

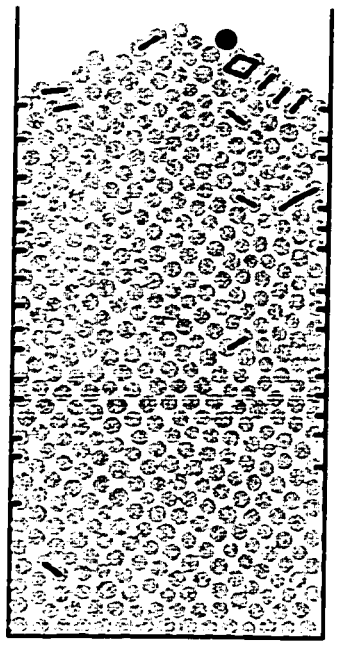


Figure 4.8. Slides occurred after a new particle (marked in black) has been added.

The numerical experiments show that the micro-avalanches generally occur in the following areas:

a) on the top levels of particles in the neighborhood of the dropped disk;
b) along the side walls of the silo, as the system is slowly moves down;
c) in some critical regions, where the particles have formed an irregular pattern (see Figure 4.9).

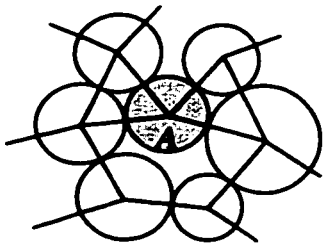
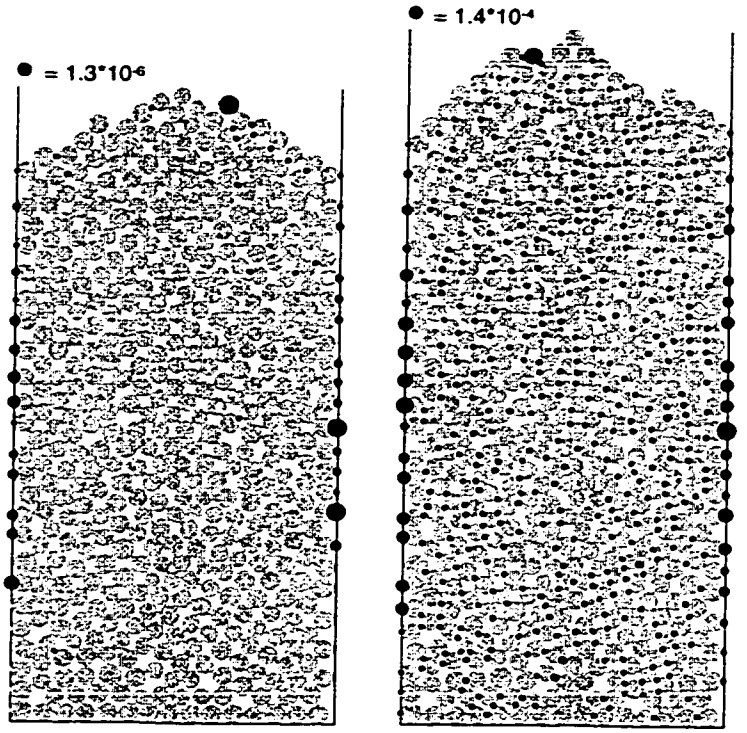


Figure 4.9. Irregular configuration.

The example in the Figure 4.9 represents an irregular configuration because the disk A is supported from below by two almost parallel links. As the pressure on the disk A from the top is increased, the disk will keep sliding down.

One of the parameters measured during the simulation is the energy transformed to heat due to slides (see Equation (4.36)). The energy values for each slid link are recorded on each iteration, as well as the total energy dissipated during the filling of the silo is computed. The energy dissipated in the links after the 463th disk was dropped is presented in the Figure 4.10a (compare with the Figure 4.8). As one can see, the most energy is dissipated along the silo walls.



(a) (b)

Figure 4.10. Energy dissipated due to slides.

The total energy dissipated during the filling of the silo is presented in the Figure 4.10b. It can be clearly seen that the energy is dissipated uniformly throughout the volume of the silo with the exception of the wall links, where a significant amount of slides were occurring.

4.7.3. Validation

The relationship between the pressure on a silo wall[1] and the height of a contact was investigated in order to validate the macro-mechanical properties of the model. The dependence of the contact stresses on the height is presented in the Figure 4.11. The close correspondence with both experimental results [Rong et. al. 97] and the Janssen theory [Janssen 1895] can be clearly seen.

The evident presence of arches and stress paths (see Figure 4.7) is also a validating result. The global distribution of slides (Figure 4.10b) seems to correspond to the general understanding of a granular material being progressively compressed in vertical direction, which explains the reoccurring of slides along the side walls of the silo.

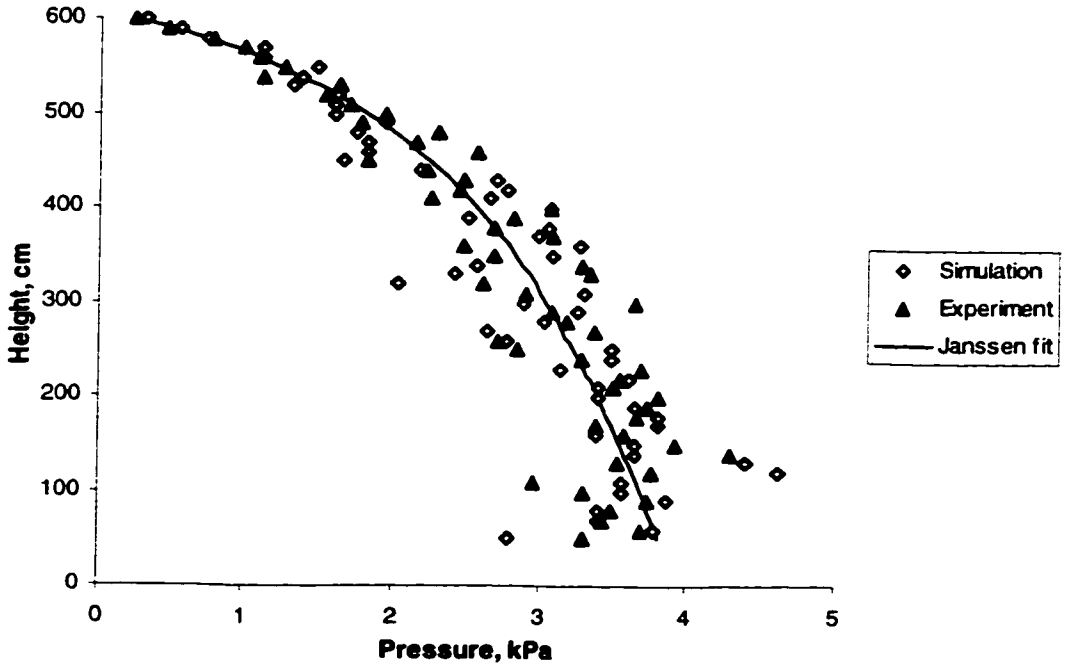


Figure 4.11. Silo wall pressure.

[1] The pressure on the wall was computed over an area of the wall of the height equal to 10 ball radii.

The micro-mechanical properties of the model are more difficult to validate, because it is very hard to measure them experimentally. It is known [Jarrett et. al. 96], that the presence of imperfections and inserts in silos leads to large stress fluctuations.

The following can be considered as a validation of the micro-mechanical properties. The forces acting on a single particle are always balanced. This is guaranteed by the RIMA algorithm, which computes the exact inverse stiffness matrix for the system. The application of the algorithm handling slides and topology changes guarantees that only compressive contacts are present in the system and there are no links, for which the friction law is violated, i.e. the system is maintained in static equilibrium after each step.

4.8. Parallel implementation of RIMA

The structure of the Recursive Inverse Matrix Algorithm (RIMA) allows for a natural parallel implementation. The state of the system is defined by a large symmetric matrix Z, which has to be updated every time a new particle or link is added to the system. These updates are defined by equations (4.19) – (4.28). Clearly, these updates can be performed in parallel. For the cases 0 and 1 (equations (4.19) and (4.20)) the matrix is bordered by an additional row and column. For the cases 2 and 3 (equations (4.21) – (4.28)) the whole matrix is updated, however, each matrix element is updated independently from others, which allows for effective parallelization.

The algorithm was implemented in C++ on a cluster of 30 DEC Alphas connected by a fast Myrinet switch. The Message Passing Interface (MPI) [Gropp et. al. 94] was used in implementation. MPI has become a de facto standard for implementing portable parallel algorithms. It defines a set of generic routines for inter-process communication and synchronization. Implementations of MPI are generally available for most parallel computers, as well as for many sequential machines working in a multi-tasking mode.

The parallel implementation of RIMA assumes that one process is designated as *master* and the rest of the processes are designated as *slaves*. The master process is responsible for data input and output, as well as for coordinating the work of slave processes. The

slave processes are responsible for storing and updating the inverse matrix, as well as for computing the displacements of particles given the vector of external forces.

4.8.1. Data storage

It is sufficient to store and update only a half of the matrix Z, since it is symmetric. The lower left triangle of the matrix is distributed among the slave processes as illustrated in the Figure 4.12, where n is the total number of slave processes.

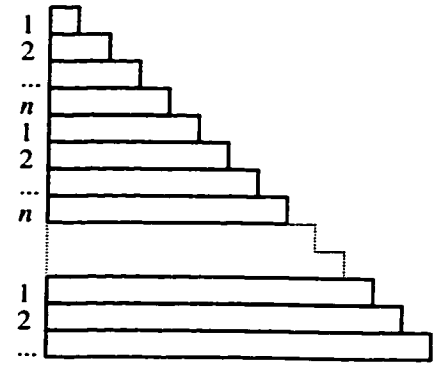


Figure 4.12. Distribution of rows among slave processes.

The first row is stored at the first process, the second row at the second, the n-th row is stored at the n-th process, the $(n+1)$-th at the first, the $(n+2)$-th at the second, and so on. Such distribution scheme ensures that the processes are responsible for storing and updating approximately the same amount of data. Note that the size of the matrix is equal to the total number of particles in the system, which varies in time. In other words, the matrix grows as more particles are added to the system.

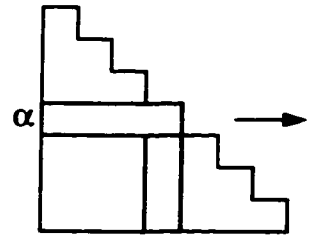


Figure 4.13. *sendRow*(α, P) primitive.

The implementation uses one basic primitive called *sendRow*(α, P), which sends the contents of the full α-th row of the matrix to the process number P. The elements of the α-th row are distributed among several processes in the general case (see Figure 4.13). Note that the horizontal segment of the row belongs to a single process, while the vertical segment is distributed among several slave processes. The elements of the vertical segment have to be transposed before being sent to the process P. Each process gathers all elements belonging to the α-th row, encodes them into a single message and sends it to the process P. The process P reconstructs the row after receiving messages from all other slave processes.

4.8.2. Case 0

Case 0, when a new node is attached to a fixed node (see Section 4.2.1), involves the least amount of processing. A new row is created at the corresponding process, filled with zeros and the last element is assigned $\mathbf{C}^{-1}$ (see Figure 4.14).

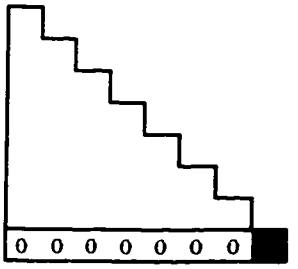


Figure 4.14. Case 0 matrix update.

4.8.3. Case 1

Case 1 (see Section 4.2.2) is also relatively computationally inexpensive. The α-th row is sent to the process P_n, which is responsible for storing the newly created row (see Figure 4.15). The last element of the row is computed according to formula (4.22).

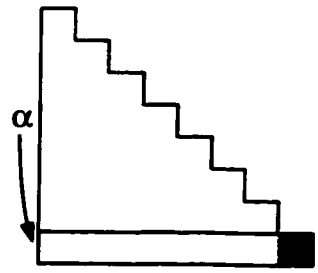


Figure 4.15. Case 1 matrix update.

4.8.4. Case 2

Case 2, when an existing node is connected to a fixed node (see Section 4.2.3) requires the full matrix update. The α-th row is sent to the process P_α, which stores the horizontal segment of this row (see Figure 4.13). Note that this minimizes the inter-process communication. Then the process P_α sends the full α-th row to each of the other slave processes, which use the row to update stored matrix elements (see Figure 4.16).

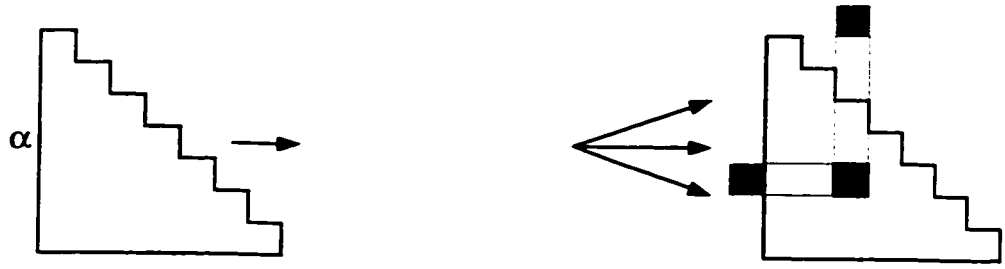


Figure 4.16. Case 2 matrix update.

4.8.5. Case 3

Case 3, like the previous case, requires a full matrix update (see Section 4.2.4). However, now two rows are needed for the update. Both α-th row and β-th row ($\alpha < \beta$) are sent to the process P_β (this minimizes the inter-process communication). The process P_β deducts one row from the other and then sends the resulting row to the rest of the slave processes, which use the row to update stored matrix elements (see Figure 4.17).

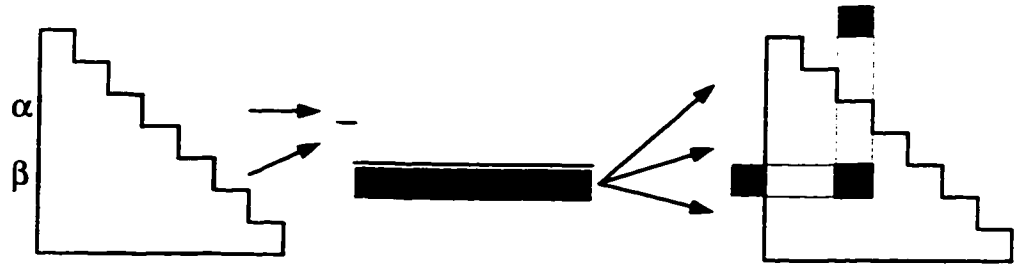


Figure 4.17. Case 3 matrix update.

4.8.6. Numerical experiments

A series of numerical experiments was performed in order to estimate the performance of the parallel implementation of RIMA. A planar silo with 1000 particles and 2000 links was simulated. The results of the parallel runs are presented in the Figure 4.18.

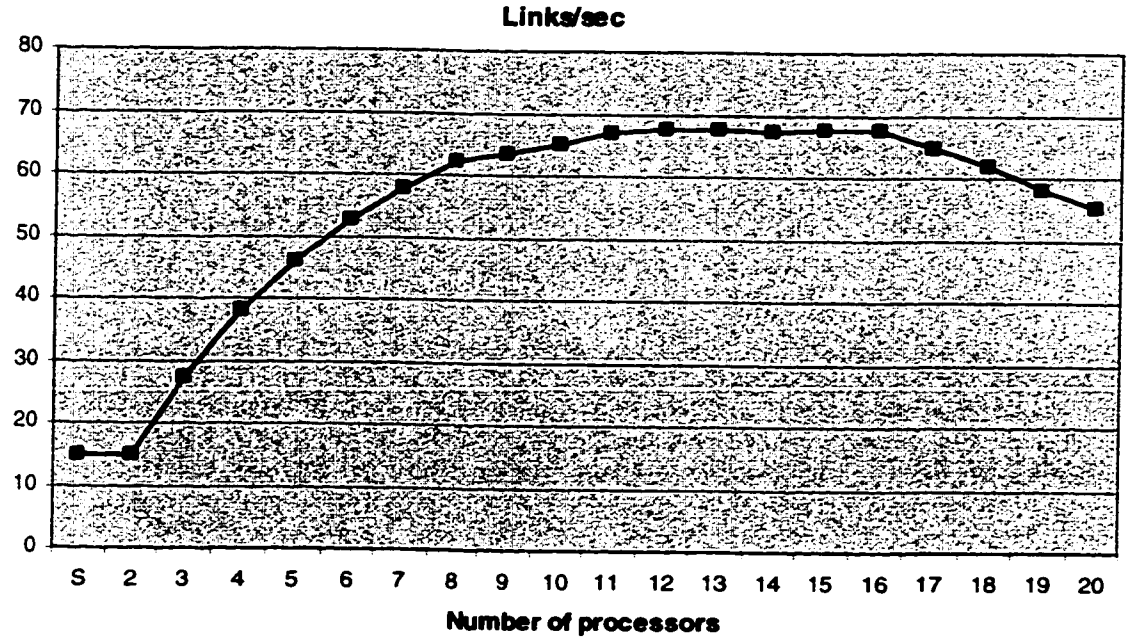


Figure 4.18. Parallel RIMA performance.

In the Figure 4.18, the one-processor version (marked with S) is, indeed, a sequential implementation of the algorithm. The 2-processor version shows the same performance as the sequential version because one of the processors hosts the master process and the

other one hosts the only slave process. As seen from the figure, the parallelism of this implementation is about 12, i.e. the performance does not improve any more when the number of processors reaches 12. This phenomenon can be explained by the observation that for cases 2 and 3 a row is sent to all processes. So, the more processors are available, the more communication is taking place. This causes an overflow of communications and, when the number of processors exceeds 16, even slows down the computation.

4.9. Conclusions

The Quasi-Static (QS) Model was introduced in this chapter. This model is best suited for modeling granular systems in steady or near-to-steady state, when no global topology changes take place. The model is based upon the Recursive Inverse Matrix Algorithm (RIMA), which maintains the inverse stiffness matrix of a truss (beam) system as the topology of the system changes. The algorithm for handling slips and other micro-changes in system's topology was introduced.

The QS model use was demonstrated on the applied problem of stress distribution simulations during the filling of a silo. The model can be used to investigate macro- and micro-properties of the granular matter, which is practically impossible in physical experimentation. Finally, a parallel implementation of RIMA was presented together with the results of numerical experiments.

CHAPTER 5. MULTIBODY DYNAMICS MODEL

The Multibody Dynamics (MBD) model is the most complete and precise model for simulating granular matter. Both dynamic behaviors of granular matter in high-energy systems and steady-state granular problems can be investigated using this model. Essentially, the MBD model includes both molecular dynamics (MD) model and quasi-static (QS) model as submodels.

The MBD model can be considered as an extension of the MD model, which allows for long lasting contacts between particles and considers forces in these contacts. If a quasi-static subsystem is detected in a MBD simulation, then the QS model is invoked to simulate the subsystem in steady or near-to-steady state. The internal properties of such subsystem are constantly monitored and, if a global-scale dynamic change is detected (which can not be handled by the QS model), the QS subsystem is disintegrated and the simulation is carried on using the MBD model.

The model is constantly morphing depending on the current system's parameters, thus automatically adjusting to the system being simulated. This, in turn, achieves a better simulation efficiency and improves the accuracy of simulation.

The MBD model is based upon the discrete-event simulation scheme. A hierarchy of objects is introduced, which represents the subsystems at all levels. The hierarchy consists of such objects as system, clusters, quasi-rigid-bodies (QRB), boundaries and bodies. The topology of the system is represented by a hierarchical graph, which is maintained throughout the simulation.

In the general case, the MBD simulations are more computationally expensive than MD and QS simulations. MBD simulations can involve numerical integration of systems of ordinary differential equations (ODE) in addition to the tasks associated with the MD and QS models and other general simulation tasks, such as collision detection. Since, however, the model adjusts automatically to the system being simulated, the simulations are always carried out in a most efficient manner.

The MBD model overcomes the limitations of both MD and QS models and, as such, can be used to simulate any granular system without arriving to a deadlock or halt. This makes the model an invaluable tool for evaluating and comparing other simulation models and approaches. The MBD model can simulate granular matter with a greater degree of accuracy than any other granular model known to the author.

The Distinct Element Method (DEM) [Cundall and Struck 79] has become a dominant tool for granulates simulations due to its simplicity and the ease of implementation. In this model, the particles are subjected to the restoring forces in the points of contact. The magnitude of these forces is calculated based on the relative overlap between particles. This scheme is roughly equivalent to solving a system of ODE using a first-order numerical integration method (Euler's method) [Burden et. al. 78]. The main limitation of this approach is the lack of error-control mechanisms and the need to select many simulation parameters (such as the timestep or the damping value) basing on some heuristics.

This chapter is organized as follows. First, the notion of hierarchical graph is introduced together with some basic primitives for manipulating the hierarchical graphs. Then, the general simulation algorithm used by MBD model is presented, followed by the detailed consideration of system components on all levels: bodies, links, clusters, quasi-rigid bodies (QRBs) and the system. Finally, an application of the MBD model to the simulation of hopper flow is considered.

5.1. Hierarchical graphs

A hierarchical graph (H-graph) is a generic topological data structure, which can be used to represent a system constituting a hierarchy of objects connected by links. The hierarchical graphs can be used naturally to represent the structure of granular systems. An example of a granular system together with the corresponding hierarchical graph is presented in the Figure 5.1.

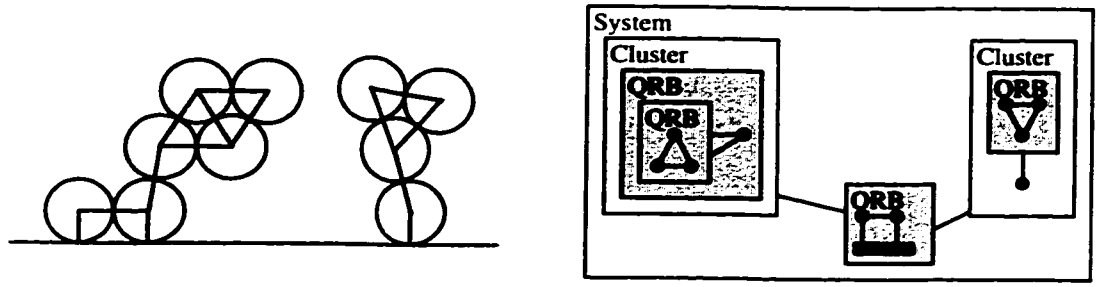


Figure 5.1. Example of a hierarchical graph.

The hierarchical graph presented in the Figure 5.1 consists of the clusters, Quasi-Rigid Bodies (QRBs) and disks. A *QRB* is an interconnected set of disks that has no internal degrees of freedom (see Section 5.7 below for the detailed description). A QRB can contain other QRBs as its members. A QRB is called fixed if it contains a boundary as one of its members. A *cluster* is an interconnected set of disks and/or QRBs, that has internal degrees of freedom (see Section 5.6 below for a detailed description).

The top-level graph represents the whole granular system. It contains two clusters and one QRB. The cluster on the left contains a single QRB, which, in turn, consists of a 3-disk QRB and a disk. The cluster on the right consists of a 3-disk QRB and a single disk. The QRB (in the middle) is fixed, it contains a boundary and two disks. Both clusters are connected to the fixed QRB.

A hierarchical graph essentially represents a tree of graphs. Indeed, nodes of a hierarchical graph can be hierarchical graphs themselves. Such a node can be considered as a "group" of nodes of a regular graph. A hierarchical graph can be converted to a regular graph by "ungrouping" the nodes, represented by hierarchical graphs. A more precise definition of a hierarchical graph is given in the following section.

5.1.1. Hierarchical graph definition

A *hierarchical graph* $H = (V, E)$ is defined by a set of nodes $V = \{v_1, v_2, ..., v_k\}$, $k \geq 1$ and a set of links $E = \{e_1, e_2, ..., e_m\}$, $m \geq 0$.

A *node* v_i, $i = 1..k$ is represented by either an atomic element or a hierarchical graph. The hierarchical graph H is called the *owner* of the node v_i, $H = owner(v_i)$, and v_i is a *member* of H, $v_i \in H$. The function $owner(v)$ is defined uniquely, i.e. any node has only one owner. If H is a top-level graph, then the value of the *owner* function is **null**.

A node v is called a *descendant* of a hierarchical graph H ($v \subset H$), if either $v \in H$ or $owner(v) \subset H$. Loops in owner chains are not allowed, i.e. a hierarchical graph can never be a descendant of itself. A node v *belongs to* a hierarchical graph H ($v \subseteq H$) if either $v = H$ or $v \subset H$.

A *link* $e_i \in E$, $i = 1..m$ is defined as $e_i = \{(v_{1i}, v_{2i}), (a_{1i}, a_{2i})\}$, where $v_{1i}, v_{2i} \in V$ are two distinct members of H and a_{1i} and a_{2i} are two atomic elements such that $a_{1i} \subseteq v_{1i}$ and $a_{2i} \subseteq v_{2i}$. v_{1i} and v_{2i} are called *members* of e_i, and a_{1i} and a_{2i} are called *nodes* of e_i. The node v_{2i} is called a *peer* of the node v_{1i} (and vice versa) in H, and the link e_i is called a *peer link* of the node v_{1i} in H. The hierarchical graph H is called the *owner* of the link e_i, $H = owner(e_i)$, and the link e_i is called an *internal link* of H.

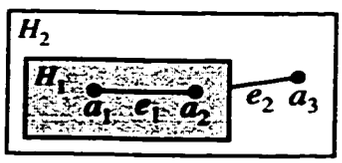


Figure 5.2. Illustration to the definition of a hierarchical graph.

The relationships between members of a hierarchical graph are illustrated in the Figure 5.2. The top-level graph is H_2, it has two members: a hierarchical graph H_1 and an atomic element a_3. H_2 contains one link e_2. The hierarchical graph H_1 has two atomic members a_1 and a_2 and one link e_1.

The link e_1 is defined as $e_1 = \{(a_1, a_2), (a_1, a_2)\}$, i.e., a_1 and a_2 are both members and nodes of e_1. a_1 and a_2 are peers, and e_1 is a peer link of both a_1 and a_2. H_1 is the owner of a_1, a_2 and e_1.

The link e_2 is defined as $e_2 = \{(H_1, a_3), (a_2, a_3)\}$, i.e. it connects H_1 and a_3 in H_2. The link e_2 is a peer link of H_1, and a_3 is a peer of H_1 in H_2.

Note. The owner H and members v_1 and v_2 of a link e are uniquely identified by the nodes a_1 and a_2 of the link. Consider the owner chains of a_1 and a_2: $a_1 \in H_{11} \in H_{12} \in ... \in H_{root}$ and $a_2 \in H_{21} \in H_{22} \in ... \in H_{root}$. Obviously, both chains begin at different elements a_1 and a_2 and end at the same top-level graph H_{root}. Then there has to exist a common element H^* in the chains such as $a_1 \subseteq H_{1i} \in H^* \subseteq H_{root}$ and $a_2 \subseteq H_{2j} \in H^* \subseteq H_{root}$, where $H_{1i} \neq H_{2j}$ (see Figure 5.3). Then the element H^* is necessarily the owner of the link e and the graphs H_{1i} and H_{2j} are the members of this link because $a_1 \subseteq H_{1i} \in H^*$ and $a_2 \subseteq H_{2j} \in H^*$. The element H^* is called the *lowest common owner* of a_1 and a_2.

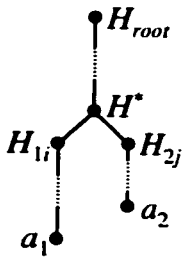


Figure 5.3. The lowest common owner.

5.1.2. Basic primitives on hierarchical graphs

So far, only one basic primitive was introduced. It is the *owner*(v) function, which returns the owner of a graph v. In addition, the following basic primitives can be defined on hierarchical graphs.

5.1.2.1. FindMemberContaining

This primitive is defined as *FindMemberContaining*(H,v). It returns a reference to a member of the graph H, which contains the graph v, or **null** if v does not belong to H. The primitive can be implemented by the following recursive function[1].

```
Node FindMemberContaining(Node H, Node v) {
   if (v == null) return null;
   else if (H == owner(v)) return v;
   else return FindMemberContaining(H, owner(v));
}
```

Note that this algorithm will never enter an infinite loop because owner chains can never have loops and any owner chain will end at the top-level graph, whose owner is **null**.

5.1.2.2. Depth

This primitive returns the depth of a graph v. If v is a top-level graph then it is assumed that *depth*(v) = 0. The primitive can be defined recursively as

```
int depth(v) {
   if (owner(v) == null) return 0;
   else return (depth(owner(v)) + 1);
}
```

Again, this primitive will never enter an infinite loop because all owner chains end at the top-level graph, whose owner is **null**. A simple non-recursive version of the algorithm can also be written.

---

[1] A Java-like syntax is used to define the algorithm.

5.1.2.3. FindLowestCommonOwner

This primitive is used to find the lowest common owner (see Figure 5.3) for a pair of graphs (or atomic elements). It takes two parameters: references to graphs v_1 and v_2 and returns their lowest common owner. Note that the lowest common owner always exists when v_1 and v_2 belong to the same top-level graph. Also note that if $v_1 \subseteq v_2$ then $FindLowestCommonOwner(v_1, v_2) = v_1$ (and vice versa). The primitive can be implemented by the following recursive function.

```
Node FindLowestCommonOwner(Node v1, Node v2) {
   if (v1 == v2) return v1;
   if (depth(v1) < depth(v2))
      return FindLowestCommonOwner(v1, owner(v2));
   else
      return FindLowestCommonOwner(owner(v1), v2);
}
```

Note that a more efficient (but longer) non-recursive implementation of the primitive can be written.

5.1.2.4. InsertLink

This primitive is used when a new link has to be inserted into a hierarchical graph. Usually, only the nodes of the link (i.e. the atomic elements which are connected by the link) are known when a link is created. It is required to determine the owner and members of the link. This primitive takes two parameters a_1 and a_2 and returns a triple containing references to the owner of the link and two members of the link. The *InsertLink* primitive can be implemented using the primitives already introduced.

```
LinkData InsertLink(Node a1, Node a2) {
   owner = FindLowestCommonOwner(a1, a2);
   m1 = FindMemberContaining(owner, a1);
   m2 = FindMemberContaining(owner, a2);
   return new LinkData(owner, m1, m2);
}
```

Again, it is possible to write a more efficient single-pass routine, which will determine the owner and members of a link.

5.1.2.5. Group

This is a high-level primitive, which groups a set $S=(v_1, v_2, ..., v_k)$ of members of a graph H into a new graph $H_1 \in H$ (see Figure 5.4). The primitive returns a reference to the newly created graph H_1.

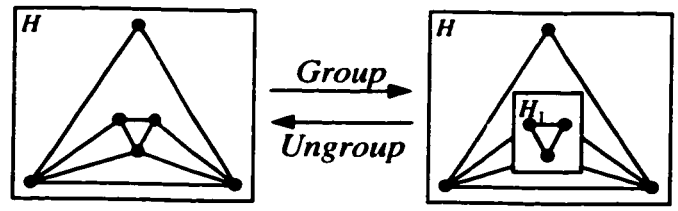


Figure 5.4. *Group* and *Ungroup* primitives.

First, a new graph H_1 is created inside H and all nodes in S are moved into H_1. A primitive $setOwner(v, H)$ is used to set the owner of a node v as a graph H. Then, a set of links E is computed, such that each link in E contains at least one member in S. Then each of the links in the set is analyzed and reattached to the updated owner or members. A primitive $setOwner(l, H)$ is used to set the owner of a link l as a graph H. The primitive $attach(l, member_1, member_2)$ is used to attach a link l to a new pair of members.

```
Node Group(NodeList S) {
   Node H1 = new Node(); // create a new graph H1
   setOwner(H1, H);  // set the owner of H1
   for each (v in S) // move all members in S into H1
      setOwner(v, H1);
   // gather peer links of all elements of S into E
   E = [];
   for each (v in S)
      for each (l in peerLinks(v))
         E = E ∪ {l};
```

```
    // analyze each of the peer links
    for each (l in E) {
       m1 = l.member1; // retrieve the first member of l
       m2 = l.member2; // retrieve the second member of l
       if (m1 ∈ H1 and m2 ∈ H1)
          setOwner(l, H1); // no need to reattach members
       else if (m1 ∉ H1)
          attach(l, m1, H1); // l is a peer link of m1 and H1
       else // m2 ∉ H1
          attach(l, H1, m2); // l is a peer link of H1 and m2
    }
    return H1;
 }
```

5.1.2.6. Ungroup

The *Ungroup* primitive is the inverse of the *Group* primitive. It is passed a reference to a graph H_1. The primitive moves all the members of H_1 into the owner H of H_1.

First, all internal links of H_1 are moved into H. Then, the peer links of H_1 are analyzed and reattached to the updated members. Finally, all members of H_1 are moved into H and H_1 is released. The primitive returns the list of members of H_1 so that it is possible to notify H that a set of new members has appeared in it.

```
NodeList Ungroup(Node H1) {
   NodeList updatedMembers = new NodeList();
   Node H = owner(H1);
   // move the internal links of H1 into H
   for each (l in internalLinks(H1))
      setOwner(l, H);
   // analyze each of the peer links
   for each (l in peerLinks(H1)) {
      m1 = l.member1; // retrieve the first member of l
      m2 = l.member2; // retrieve the second member of l
      if (m1 == H1)
         m1 = FindMemberContaining(H1, l.node1);
      else // m2 == H1
         m2 = FindMemberContaining(H1, l.node2);
```

```
        attach(l, m1, m2); // reattach l
    }
    // move the members of H1 into H, add to the result
    for each (v in members(H1)) {
        setOwner(v, H);
        append(updatedMembers, v);
    }
    free(H1); // release H1
    return updatedMembers;
}
```

5.2. General system structure

The MBD model represents the simulated granular system by a hierarchical graph. The nodes on different levels of the hierarchical graph are represented by different objects. The top-level object is called *System* and it is responsible for global-level functions, such as maintaining the event queue, event routing, interaction with the collision optimization engine, as well as other global-level functions.

The objects on the first level (members of *System* object) are called *Clusters*. Each *Cluster* object represents an interconnected group of bodies. The *System* can contain several *Clusters*, representing disconnected groups of bodies. The motion of each cluster is defined by a coupled system of ODE, therefore, it is required to consider the motion of a cluster as whole. It is possible, however, to consider each cluster separately from other clusters. Thus, the most appropriate timestep can be selected when integrating each particular ODE system. As the topology of the system changes when links between bodies appear and disappear, new clusters can appear and existing clusters can merge or disintegrate.

The ODE system defining the motion of a cluster is never formulated explicitly, but rather the equations are "embedded" into the topology of the cluster, represented by the links in the hierarchical graph representing the cluster. This allows for a natural "distributed" representation of the ODE system, as well as saves time by not requiring to regenerate the equations of motion each time the topology of the cluster changes.

Each cluster contains one or more member nodes representing *Bodies* in the system. The main property of a *Body* object is that it represents a rigid body without internal degrees of freedom. There are true bodies in the system, namely, the particles (they are represented by objects of type *Disk*), and there are transient bodies, called *Quasi-Rigid Bodies* (QRBs), which are formed by groups of interconnected particles and/or boundaries (see Figure 5.5). While a QRB exists, it does not have internal degrees of freedom and, as a result, it can be considered a rigid body. Note that as the external conditions change, some of the links inside a QRB can break or become unstable. In this case, the QRB is disintegrated and its members are merged into a higher-level cluster. Conversely, if a group of bodies without internal degrees of freedom is detected inside a cluster, then it is converted to a QRB.

A *Body* in the system can be either *free* or *fixed*. An example of a free body is a *Disk* object, and an example of a fixed body is a *Boundary* object. A QRB is a body, which can be either free or fixed (see Figure 5.5). A fixed QRB always contains a fixed body as one of its members and, as a result, the whole QRB is fixed. Thus, a fixed QRB is not considered a member of any cluster, but, rather, it is stored as a member of the *System* object. Conversely, a free QRB is always a member of some cluster.

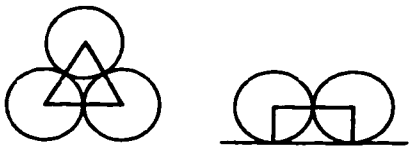


Figure 5.5. An example of a free and a fixed QRB.

Since a QRB does not have internal degrees of freedom, it can be defined by a static model (note that the motion of a free QRB as a rigid body is handled by its owner, the cluster object). Thus, the Quasi-Static model is used to define the interactions between bodies inside a QRB. However, if a slip or break is detected in an internal link, then the QRB is disintegrated and either converted to a cluster (if it was a fixed QRB) or its members are merged into the owner cluster (if it was a free QRB).

The atomic elements in the hierarchy are represented by *Disks* and *Boundaries*. Another important object in the system structure is the *Link* object, which is used to represent links between objects in the hierarchical graph representing the system. A *Link* is always attached to a couple of atomic bodies as its nodes (i.e. a link either connects a couple of *Disks* or a *Disk* and a *Boundary*). Links can appear at all levels of the system hierarchy: a link belonging to the *System* object connects a cluster to a boundary or to a fixed QRB, links are also used in clusters as well as in QRBs.

A link is defined by a special model, which is used to compute the internal stress in the link based on the state of the link's nodes (see Section 5.5.2 below). This stress computation routine is used both when integrating an ODE system defining the motion of a cluster as well as when computing the internal stresses in links inside a QRB. The link model also defines the conditions for detecting a break or a slip in the link (see Section 5.5.2 below).

Each of the objects constituting the system hierarchy will be discussed in detail in the following sections. But first, the general simulation algorithm is presented.

5.3. General simulation algorithm

The general simulation algorithm used in MBD model is an extended version of the algorithm used in the Molecular Dynamics model (see Section 3.5.1). The main difference is related to the fact that the system is now represented by a set of different objects responsible for different tasks. When an event has to be handled, it is routed to the object, which is responsible for handling this event. Then, depending on the type of object handling the event, different actions can be taken. For example, when a *slip-link* event is handled, it is forwarded to the owner of this link. If the owner is a cluster object, then the cluster updates its state and nothing is changed in the system's structure. If the owner is a QRB, then this QRB is disintegrated.

Also, the set of events is different: for example, instead of a single *collision* event type, there are two distinct events types in MBD model: *insert-link* event and *remove-link* event.

The following sections define the event types used by the MBD model. But first, several important routines are defined.

5.3.1. FindConnectedGroup routine

The *FindConnectedGroup* routine is used to find a connected portion of a graph starting from some node. The routine is passed a reference to the starting node and returns a list of nodes connected to the starting node. The routine has an additional parameter – a flag, which controls whether it is required to include fixed nodes into the resulting list. The routine can be defined recursively as follows. The resulting list is collected in the variable *result*.

```
void FindConnectedGroup(Node startNode, NodeList result,
                        boolean includeFixed)
{
   Node peer;
   append(result, startNode); // include the startNode
   if (startNode.fixed) return; // don't grow fixed nodes
   // scan through the list of peers
   for each (peer in peers(startNode)) {
      if (contains(result, peer))
         continue; // visited this node already
      if (!peer.fixed || includeFixed)
         // make the recursive call
         FindConnectedGroup(peer, result, includeFixed);
   }
}
```

5.3.2. FindClusters routine

The *FindClusters* routine is used to detect clusters in a set of nodes. The routine is passed a reference to a hierarchical graph, containing the set of nodes (this graph can be

considered just as a temporary container, it is usually ungrouped right after *FindClusters* has returned the result). The routine searches for clusters inside the container using the *FindConnectedGroup* routine and creates them (using the *Group* primitive).

```
void FindClusters(Node container) {
   NodeList connectedGroup = new NodeList();
   Node node;
   boolean found;
   while(true) {
      // try to find a free body inside the container
      found = false;
      for each (node in members(container))
         if (!node.fixed && node instanceof Body) {
               found = true;
               break;
         }
      if (!found) break; // no more free bodies left, leave
      // find a connected group starting from the node
      FindConnectedGroup(node, connectedGroup, false);
      // group the result into a cluster
      Group(connectedGroup);
      connectedGroup.clear(); // clear the group
   }
}
```

5.3.3. FindQRBs routine

The *FindQRBs* routine is used to detect QRBs inside a container. The QRBs are detected by a pattern-matching process to the set of QRB templates. The QRB templates are the basic QRB building "blocks" (for a detailed description of QRB templates, see the Section 5.7.4 below). See the Figure 5.6 for the examples of basic QRB templates.

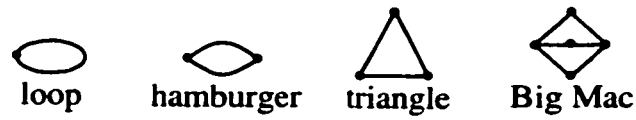


Figure 5.6. Basic QRB templates.

The algorithm proceeds by trying to find a subgraph, which is congruent to one of the QRB templates. If such a subgraph is found, then it is converted to a QRB (using the *Group* primitive). The process stops when no more matching templates can be found. An example of the QRB detection process is presented in the Figure 5.7.

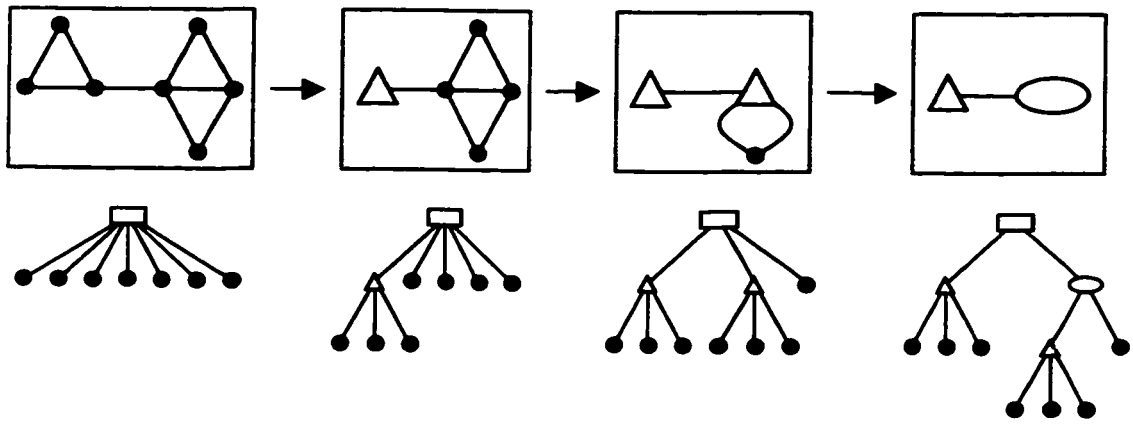


Figure 5.7. QRB detection process and the corresponding H-graph transformations.

Note that if the original set of nodes contains several fixed nodes, then all of them are considered as a single fixed node during the pattern-matching process. Also, the *FindQRBs* routine only takes into consideration the links which are stable (see Section 5.5.2 below for the detailed explanation of this term).

The MBD model is based on the discrete-event simulation approach. The following sections define the types of events recognized by the simulation engine.

5.3.4. Predict-trajectory type events

Several types of events somewhat related to the predict-trajectory event are used in the system.

5.3.4.1. Predict-trajectory event for clusters

The *predict-trajectory* events are used as a means of the timestep-based integration of ODE describing the motion of a cluster. Thus, a predict-trajectory event is scheduled for a cluster when the solution of the ODE has to be computed on the next timestep. Unlike

in the MD model, the equations of motion of a cluster can not be resolved analytically in most cases (except, perhaps, for the case, when a cluster consists of a single body and there are no internal or peer links – this is roughly equivalent to a single particle in the MD model).

When a predict-trajectory event is handled by a cluster, the current states of the cluster's members and links are used to predict the trajectory of each member for the next timestep. Different integration schemes can be used in order to compute the solution at the next grid point. The single-step Euler's method [Burden et. al. 78] is the simplest method, however, it is quite inaccurate. The predictor-corrector method [Burden et. al. 78] is another choice, which offers a greater degree of accuracy at a slightly higher computational cost.

After the solution at the next grid point is obtained, the trajectories of cluster's members are approximated by linear splines (see Section 3.2.2). Unlike in the MD model, the state of a cluster in MBD model also includes the values of reaction forces acting in the links of this cluster. The values of these forces are obtained in the course of numerical integration. These values are also approximated by linear splines during the timestep.

Similarly to the MD method, a predict-trajectory event is also scheduled at any moment of time, when the state of a cluster has changed for some reason. For example, a predict-trajectory event is scheduled immediately after any *insert-link* or *remove-link* event.

Since handling of a predict-trajectory event involves updating the state of the cluster and its members, all events pending for the cluster or any of its descendants (direct or indirect) have to be cancelled. This includes collision events scheduled for any of the disks inside the cluster, as well as any miscellaneous events (such as collision-detection optimizer events) scheduled for the descendants of the cluster.

After a predict-trajectory event has been handled for a cluster, the *analyze-internal-forces* events (see the next section) are scheduled for all QRBs inside the cluster as well as for all peer QRBs of the cluster.

5.3.4.2. Analyze-internal-forces event for QRBs

Events of this type of event are scheduled for QRBs. They are roughly equivalent to predict-trajectory events for clusters. Since the QRBs are not concerned about their motion (this is taken care of by clusters), they do not have to store their trajectories as such. The QRBs do, however, keep track of the values of the stresses in their internal links, because they have to detect breaks and slips in these links. Obviously, the internal forces in a QRB depend on the values of the external forces acting on this QRB. Thus, the *analyze-internal-forces* event is scheduled for a QRB when it was determined that the external forces acting on the bodies in this QRB have changed. Usually, this event is scheduled by a cluster for a QRB once the cluster has completed the handling of a predict-trajectory event and the forces in cluster's links were computed.

When an analyze-internal-forces event is being handled, the QRB uses the values of external forces to compute the values of internal forces acting in the internal links of the QRB. After this has been done, each link is examined and, if instability was detected in a link, a *remove-link* or a *slip-link* event is scheduled. Note that the event is scheduled immediately, because the QRB model does not have a time-scale. In other words, it is assumed that once the external forces change, the redistribution of internal forces happens immediately. Thus, if it is determined that some links have broken or slipped, the corresponding events are scheduled at the current time.

5.3.4.3. Predict-trajectory event for moving boundaries

If the system contains moving boundaries, whose motion is defined by a piece-wise function, predict-trajectory events are scheduled for such boundaries at times when the trajectory of the boundary has to be updated. Note that the motion of boundaries is assumed to be predefined, i.e. it is unaffected by the motion of particles in the system.

5.3.5. Link events

There are several types of link events recognized by the system. All of them (with the exception of the *insert-link* event) relate to an existing link and are handled by the object, which owns the link.

5.3.5.1. Insert-link event

The *insert-link* event is scheduled when a collision between two atomic bodies (two disks or a disk and a boundary) was detected. Since the trajectories of the bodies approximated by linear splines are known, the time of the collision can be computed using the formulas presented in the Section 3.3.

When an *insert-link* event is handled, a new link connecting the colliding bodies is created. The *InsertLink* primitive (see Section 5.1.2.4) is used to determine the owner and the members of the link. Then the link is attached to the owner and members and the owner is notified of the change.

Note that the owner of the new link must be either the *System* object or a *Cluster*. A new link can never appear in a QRB[1], because the members of a QRB are considered to be rigidly linked together and can not move relatively to one another.

If a link appears inside an existing cluster, then a predict-trajectory event is scheduled for the updated cluster. If the link's owner was determined as the *System* object, then the members of the link are either two clusters, or a cluster and a fixed body (a fixed QRB or a boundary). In either case, the cluster topology in the system is changed and, therefore, the *FindClusters* routine is invoked to update the topology. Then, the *predict-trajectory* event is scheduled for the updated cluster.

---

[1] There are only three different types of compound objects (i.e. the objects, that can contain other objects as members): the *System* object, *Clusters* and *QRBs*.

The newly created link is assumed to be slipping and unstable (see Section 5.5.2 below for the detailed explanation of the terms), and, therefore, it can not be a part of a QRB. Thus, the *FindQRBs* routine is not invoked as a part of handling of an *insert-link* event.

5.3.5.2. Remove-link event

The *remove-link* event is scheduled for an existing link when a negative normal stress is detected in the link (i.e. the link is being "stretched"). The event is handled by the current owner of the link.

If the link belongs to the *System* object, then it is necessarily a link between a cluster and a fixed body. In this case, the link is simply removed and a *predict-trajectory* event is scheduled for the updated cluster. If the link was connecting the cluster to a fixed QRB, then an *analyze-internal-forces* event is also scheduled for the QRB.

If the link belongs to a cluster, then there are two possibilities: the cluster might remain intact after the removal of the link or the cluster may be split into two independent clusters (see Figure 5.8). It is generally impossible to determine which one is the case without analyzing the full topology of the cluster. Therefore, the general *FindClusters* routine is invoked in order to update the topology.

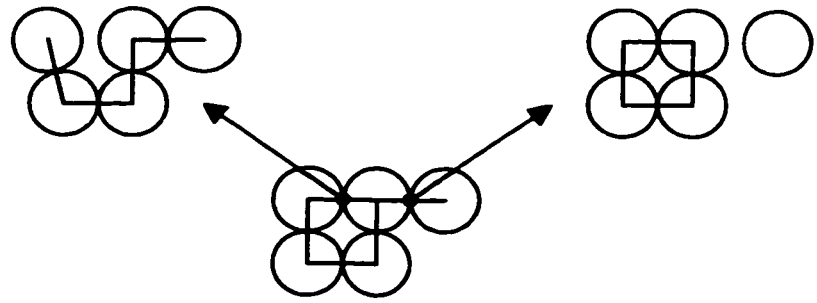


Figure 5.8. Removing a link from a cluster.

The *FindClusters* routine returns a list of found clusters. A *predict-trajectory* event is scheduled for each of the returned clusters.

If the link belongs to a QRB, then this QRB has to be disintegrated. If this QRB was a part of another QRB, then the owner is also disintegrated, until a cluster or the *System* object is encountered in the owner chain. As a result, a set of bodies (disks, boundaries and possibly other QRBs) is obtained. Then, the *FindQRBs* routine is run on this set followed by the *FindClusters* routine. As a result, a set of new clusters and/or QRBs is obtained (see Figure 5.9, where the link being removed is marked in bold).

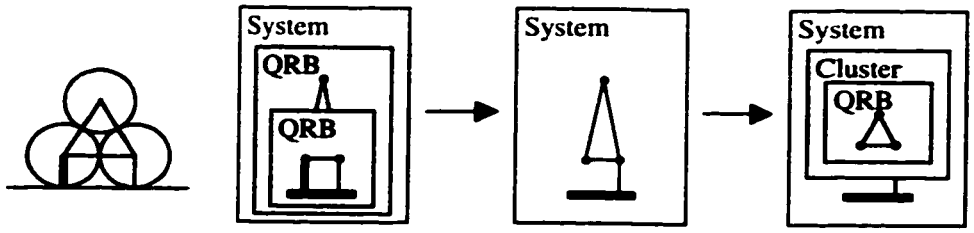


Figure 5.9. Removing a link from a QRB.

The *predict-trajectory* or *analyze-internal-forces* events are scheduled for each of the returned objects.

5.3.5.3. Slip-link event

The *slip-link* event occurs when a slip has been detected in a link. A slip occurs when the ratio between the tangential and the normal stresses acting in the link exceeds the friction coefficient. The event is handled by the current owner of the link. When the event is handled, the link is marked as unstable and slipping.

If the link is owned by a cluster then a *predict-trajectory* event is scheduled for this cluster. If the link's owner is the *System* object, then it's a link connecting a cluster to a fixed body. In this case a *predict-trajectory* event is also scheduled for the cluster. If the link is owned by a QRB, then the QRB has to be disintegrated. The event is handled in exactly the same manner as the *remove-link* event in a QRB.

5.3.5.4. Grip-link event

A *grip-link* event is scheduled for a slipping link when it is detected that the link has stopped slipping. The owner of the link handles the event. Note that a slipping link is considered to be unstable; therefore, it can never belong to a QRB. Thus, the link necessarily belongs to a cluster or it is a peer link of a cluster. Then, a *predict-trajectory* event is scheduled for this cluster.

5.3.5.5. Stabilize-link event

A *stabilize-link* event is scheduled when it was detected that a link has come to a stable state. The owner of a stabilized link can only be a cluster or the *System* object because QRBs can only contain links, which are already stable.

When a *stabilize-link* event is handled, the *FindQRBs* routine is run followed by the *FindClusters* call. Note that stabilization of a link may or may not lead to the formation of QRBs. This is because for a QRB to be created it is required that all links in it are stable.

The stability condition (see Section 5.5.2 below for the definition) requires that the two bodies in contact almost do not move relatively to each other and that the link is not slipping. Another important condition of stability is the temporal parameter. It is required that the link stays in a stable state for a certain period of time before a *stabilize-link* event is fired. This prevents considering links in transitional states and cuts down an overhead related to creating a QRB and destroying it almost instantly.

5.3.6. System events

The MBD model also recognizes several types of system events.

5.3.6.1. Visualize event

The *visualize* events are fired with a specified time interval. Scheduling these events over a fixed time period allows making snapshots of the system, which can be later converted

to an animation, that can run in a real time scale. If the visualize time interval is set to zero, then no visualization is performed.

5.3.6.2. Log-state event

The *log-state* events are also fired with a specific time interval so that the application program could make periodic snapshots of the system state. The application program gets an opportunity to output some system parameters (such as velocities of bodies, energy levels, etc.), compute necessary statistics, and keep the simulation log.

5.3.6.3. Finish event

The *finish* event is scheduled to signal the simulation engine that the simulation run has to be stopped after the specified period of time.

5.3.6.4. Add-body event

The *add-body* events are used to provide a random feed of bodies into the system (such as in ice-flow simulations). The handling of such an event involves creating a new body and inserting it into the system. The *add-body* events can be all scheduled up front, or added one-by-one as they are being handled.

5.3.6.5. Collision-detection-optimizer event

The *collision-detection-optimizer* events are scheduled and handled by the Collision-Detection-Optimization (CDO) engine. A particular implementation of CDO algorithm introduces specific CDO events. These events are usually linked to bodies. Thus, when a body changes its trajectory (e.g. due to a *predict-trajectory* event), all events related to this body, including the CDO events, are cancelled.

5.4. Bodies

The *Body* class represents an object without internal degrees of freedom. The examples of body objects are *Boundary*, *Disk* and *QRB* objects. The class diagram representing the relationships between different *Body* objects is presented in the Figure 5.10.

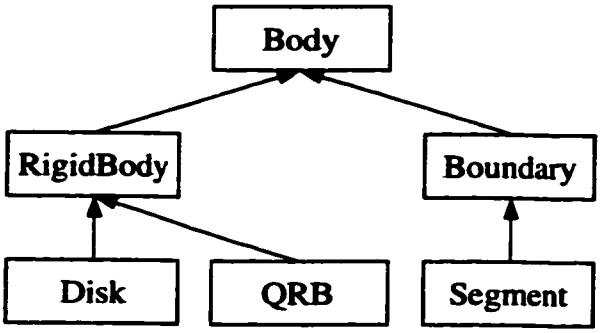


Figure 5.10. *Body* class diagram.

The *Cluster* object assumes that it contains *body* objects, without making assumptions about each element being a particular subclass of *Body*. Note that in the MBD simulation approach, a cluster can never contain a *Boundary* object as a member, but it can only be connected to a boundary (or a fixed QRB) by a peer link. Thus, the main function of a body object is to be able to operate as a part of a cluster. A detailed description of body's properties is given in the following sections.

5.4.1. Body properties

Each instance of the *Body* class has the following properties:

- a flag, indicating if it is a fixed body or a free body;
- the mass of the body;
- the moment of inertia of the body;
- the current position of the center of gravity of the body.

Each of the properties can be implemented differently in the descendants of the *Body* class. For example, *Boundary* bodies are always fixed and have an infinite mass.

5.4.2. RigidBody properties

In addition to the above properties, the instances of *RigidBody* class include the following properties. Each *RigidBody* maintains a list of *states*, each state representing a time snapshot of the body's dynamic attributes. The states are used by the cluster for integrating the ODE describing the motion of the cluster, as well as for collision detection and other tasks. Usually, a body only maintains several states, including the *current state* (the state at the beginning of the current timestep), the *predicted state* (the state at the end of the current timestep) and the *previous state* (the state at the beginning of the previous timestep).

Each state is attributed with the following parameters:

- the time of the state snapshot;
- the position of the center of mass;
- the velocity of the center of mass;
- the angle of rotation;
- the angular velocity;
- the force acting on the body and
- the moment of the force acting on the body.

It is assumed that the body's motion between two consecutive timesteps is interpolated by linear splines (see Section 3.2.2). Indeed, during a timestep, all of the state parameters are assumed to be changing according to the formulas similar to equation (3.3), except for the velocity, which is kept constant during the timestep.

5.4.3. Disk properties

The *Disk* class is a concrete class, which is used to represent particles. In addition to the properties inherited from the *Body* and *RigidBody* classes, it also has the *radius* property.

It also implements the properties for computing the position of the center of mass (which is the center of the disk) and the moment of inertia ($\frac{mr^2}{2}$ in 2D, where m is the mass of the disk and r is the radius of the disk).

The *Disk* class also contains the routines necessary for detecting collisions with other disks and boundaries. The formulas (3.8) and (3.12) are used in computations. Note that it is assumed that the disk moves along a straight-line trajectory between two consecutive timesteps. Therefore, the collision detection equations can be resolved analytically (see Section 3.3 for the discussion).

5.5. Links

The *Link* class is used to represent a link in the hierarchical graph of the simulated system. Physically, a link represents a contact between a pair of particles or a particle and a boundary. As noted above (see Section 5.1.1), a link is defined by four objects: two nodes (the atomic elements, which are connected by the link) and two members (the two hierarchical graphs connected by the link within the same owner graph).

Links play an important role in a MBD model. They are used for computing the RHS (right-hand side) of the ODE system defining the motion of a cluster and for computing stresses inside a QRB. Thus, the main function of the *link* object is to compute the internal stress in the link based on the states of its nodes. Note that since bodies maintain a list of available states, a link can compute the stresses in any of these states. Therefore, the links also maintain a list of states corresponding to the states of their nodes.

5.5.1. Link properties

A *Link* object maintains the following properties (see the following Section 5.5.2 for the detailed explanation of the properties):

- the normal and tangential stiffnesses K_η and K_τ;

- the normal and tangential damping coefficients C_η and C_τ;
- the friction coefficient μ.

In addition to these static properties, the link maintains two flags: the *slipping* flag and the *stable* flag. Also, the link maintains a list of states, each state attributed with the following parameters:

- the time of the state snapshot;
- the contact point coordinates;
- the normal and tangent vectors defining the link coordinate system (η,τ) (see Figure 5.11)
- the relative displacement of link nodes in the link coordinate system $(\delta_\eta,\delta_\tau)$;
- the stress in the link coordinate system (s_η,s_τ);
- the stress in the global coordinate system (s_x,s_y);
- the slip indicator value (see Section 5.5.2 below);
- the stability indicator value (see Section 5.5.2 below).

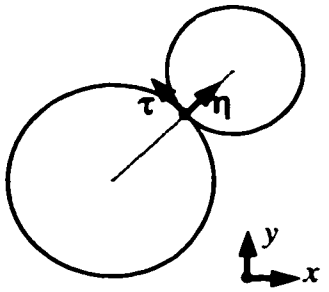


Figure 5.11. The link coordinate system and the global coordinate system.

5.5.2. Link model

All of the attributes in a link state are obtained from the attributes of the states of the links' nodes. The link model defines the rules for computing these values.

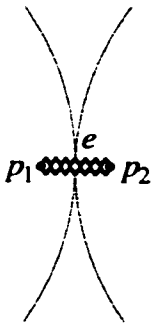


Figure 5.12. The link model.

The link model considers the relative displacement of two points p_1 and p_2 belonging to the nodes of the link (see Figure 5.12). The points are assumed to be in the close vicinity of the contact point. It is assumed that the points are connected by an element e, which is equivalent to two damped springs: one normal and another tangential.

The stress in the link connecting two disks is computed using the following formulas. The normal displacement δ_η and the normal stress s_η in the link are computed as

$$\boldsymbol{\delta} = \mathbf{x}_2 - \mathbf{x}_1,\ \boldsymbol{\delta}_\mathbf{v} = \mathbf{v}_2 - \mathbf{v}_1, \tag{5.1}$$

$$\delta_\eta = (\boldsymbol{\delta}, \boldsymbol{\eta}) - (r_1 + r_2), \tag{5.2}$$

$$s_\eta = \delta_\eta K_\eta + v_\eta C_\eta,\ v_\eta = (\boldsymbol{\delta}_\mathbf{v}, \boldsymbol{\eta}) \tag{5.3}$$

where $\mathbf{x}_i$, $\mathbf{v}_i$ and r_i, $i = 1..2$ are the coordinates of centers, velocities of centers and the radii of two disks, correspondingly. Note that the normal stress should never assume positive values, because the link is one-sided (i.e., it can not be stretched). If the contribution to the normal stress due to damping exceeds the contribution due to displacement, the normal stress is reset to zero. The physical meaning of this is that a

compressed contact is getting relaxed so fast that the contact is lost even though the value δ_η is still negative (i.e. the disks would be in contact if they were not moving).

When computing the tangential stress, it is necessary to take into account the rotation of disks and rolling or sliding. Thus, it is necessary to consider the relative displacement of the disks at the previous state and compute how the displacement at the current state changed relatively to the previous value.

Consider the relative displacement of the contact points p_1 and p_2 in the tangential direction (see Figure 5.13).

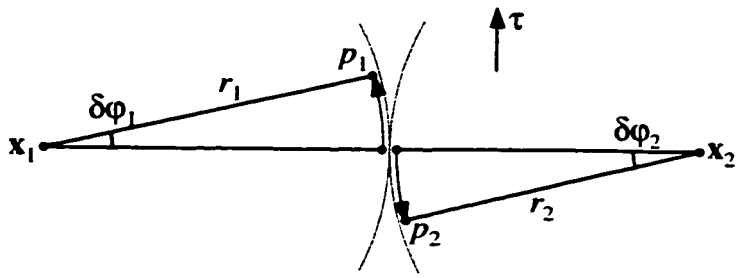


Figure 5.13. Relative tangential displacement computation.

The tangential displacements of the points p_1 and p_2 can be written as

$$\delta_{\tau,1} = (\mathbf{x}_1 - \mathbf{x}_{1,prev}, \boldsymbol{\tau}) + \delta\varphi_1 r_1, \ \delta\varphi_1 = \varphi_1 - \varphi_{1,prev}, \tag{5.4}$$

$$\delta_{\tau,2} = (\mathbf{x}_2 - \mathbf{x}_{2,prev}, \boldsymbol{\tau}) - \delta\varphi_2 r_2, \ \delta\varphi_2 = \varphi_2 - \varphi_{2,prev}, \tag{5.5}$$

where φ_i, $i = 1..2$ is the angles of rotation of disks, and the *prev* subscript denotes the values from the previous state. Thus, the following formula is used to compute the relative tangential displacement.

$$\delta_\tau = \delta_{\tau,2} - \delta_{\tau,1} = (\boldsymbol{\delta} - \boldsymbol{\delta}_{prev}, \boldsymbol{\tau}) - \delta\varphi_2 r_2 - \delta\varphi_1 r_1 + \delta_{\tau,prev}, \tag{5.6}$$

where $\delta_{\tau,prev}$ is the relative tangential displacement taken from the previous timestep. It is assumed that when the contact is formed the initial tangential displacement δ_τ is zero.

Assume that there is no slipping in the link. Then, the tangential stress can be computed by the following formula.

$$s_\tau = \delta_\tau K_\tau + (v_\tau - \omega_1 r_1 - \omega_2 r_2) C_\tau,\ v_\tau = (\delta_v, \tau), \tag{5.7}$$

where ω_i, $i = 1..2$ are the angular velocities of disks. Same as for the normal stress, the contribution due to damping can not exceed the contribution due to displacement. If it does, then the tangential stress is reset to zero.

Next, the model computes the *SlipIndicator* value and checks for possible slips. The *SlipIndicator* value is computed as

$$SlipIndicator = |s_\tau| - \mu |s_\eta|. \tag{5.8}$$

If $SlipIndicator > 0$ then the link is slipping and the tangential stress and tangential displacement are updated as

$$s_{\tau,slipping} = \mathrm{sgn}(s_\tau)\mu |s_\eta|,\ \delta_{\tau,slipping} = \frac{s_{\tau,slipping}}{K_\tau}, \tag{5.9}$$

i.e. so that they have the maximum absolute values possible (note that the sign of the values is kept).

The use of the angle of rotation φ_i for defining the current state of the disks yields more accurate results than using angular velocity ω_i only (the technique used in the DEM approach [Cundall & Strack 79]). This is because the quantities $\delta\varphi_i r_i$ representing the relative rotation of the disk during the timestep are more accurate than the expression involving the angular velocity: $\omega_i r_i h$, where h is the timestep. The latter expression is equivalent to a single-step integration of the equation $\dot{\varphi}_i = \omega_i$, i.e. $\delta\varphi_i = \omega_i h$, while the former represents the actual values of the angle of rotation, which can be obtained using the selected integration method (e.g. predictor-corrector), which will yield a result with the desired accuracy.

Note that the *slip-link* and *grip-link* events are scheduled based on the current value of *SlipIndicator*. Indeed, a *slip-link* event is scheduled when it is detected that $SlipIndicator > C_{tolerance}$, where $C_{tolerance}$ is some small positive tolerance value. Likewise, a *grip-link* event is scheduled when $SlipIndicator < -C_{tolerance}$. This approach prevents infinite loops related to round-off errors.

Finally, the model computes the *StabilityIndicator* value using the following formula.

$$StabilityIndicator = 1 - \frac{|v_\eta| + |v_\tau|}{C_{stability}}, \tag{5.10}$$

where v_η and v_τ are the relative normal and tangential velocities of the contact points, respectively, and $C_{stability}$ is some small constant. The link is considered to be *stable* when $StabilityIndicator > 0$ (i.e. when $|v_\eta| + |v_\tau| < C_{stability}$). The physical meaning of the *StabilityIndicator* value is that the link is assumed to be stable when the relative velocity of two disks is small.

As it was noted above (see Section 5.3.5), a *stabilize-link* event is scheduled only after the link has been in a stable state for some time. Thus, if it is detected that the link's *StabilityIndicator* has changed sign from negative to positive (this is done by comparing the values at the current and previous states) then the time of the event is recorded. Then, if the link stays in a stable state for a specified period of time (say, 10 timesteps), then a *stabilize-link* event is fired and the QRB detection is performed.

Similar formulas are obtained for a disk-to-boundary link. Indeed, a boundary segment can be considered as a fixed disk with an infinite radius.

As was noted above, the main function of the link object is to compute the internal stress in the link given the states of the link's nodes. Indeed, this is a function that takes the values of nodes states as parameters and returns the reaction force acting on the nodes as a result. Note that the link object is able to compute and store the value of this function at

different states. The function computing the reaction forces acting on nodes is used by the cluster object in computations of the right-hand-side of the ODE defining the motion of the cluster. Also note that both positions and velocities of the link's nodes are used in computations along with the angle of rotation and angular velocities of nodes. The values from the previous state are also used as reference points.

5.6. Clusters

This section considers the *Cluster* object in detail. The cluster objects are used at the first level of the system hierarchy. They represent an interconnected group of particles with internal degrees of freedom, or a single free particle. A cluster can contain disks and/or QRBs, connected by links. The motion of a cluster is defined by a system of ODEs.

5.6.1. Definition

A *Cluster* is defined by a connected graph of *bodies*, not containing any fixed bodies. If a cluster contains more than one body, then it has internal degrees of freedom. A cluster's graph can not contain a subgraph, which can be classified as a QRB (see Section 5.7 below for a detailed definition of a QRB). A cluster can have peer links connecting it to fixed bodies. Examples of clusters are presented in the Figure 5.14.

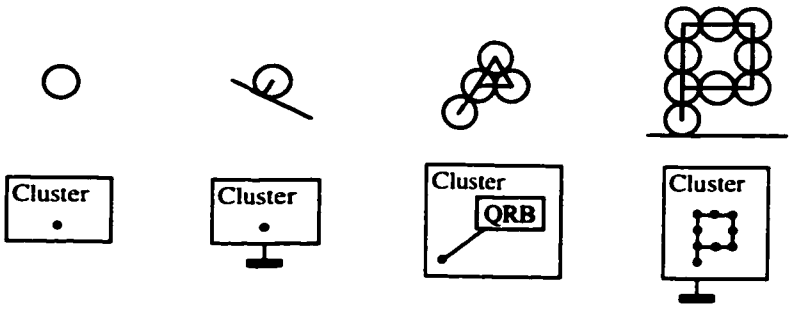


Figure 5.14. Examples of clusters.

Note that a cluster can not contain other clusters as its members (as opposed to QRBs, which can contain other QRBs). When a link disappears in a cluster, the cluster can be

split into two clusters, and, conversely, two clusters can be merged into one if a link connecting their members appears in the system.

Two or more clusters can be connected to the same fixed body, but they still will be considered separately from each other (see Figure 5.15).

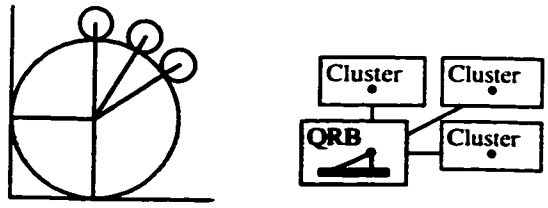


Figure 5.15. Multiple clusters connected to the same fixed body.

The decision not to include fixed bodies as a part of the cluster allows treating each of the clusters individually, so that in each case appropriate integration timestep and error-control scheme are selected.

The main property of a cluster is that it has three or more degrees of freedom. One can argue that if a cluster is connected by a link to a fixed body (see, for example, the second cluster in the Figure 5.14), then the number of degrees of freedom is decreased. Note, however, that the links in a cluster are not considered as rigid elements restricting the motion of the nodes they connect, but rather as flexible elements, which impose additional reaction forces on their nodes. Thus, if a cluster contains n bodies, then the total number of degrees of freedom in the cluster is exactly $3n$, because each body has two translational and one rotational degree of freedom. The links in the cluster are not considered as constraints, but rather are used to compute the reaction forces acting upon the bodies in the cluster.

5.6.2. Laws of motion

The motion of a cluster can be defined in terms of rigid body dynamics. Thus, the problem can be represented as an initial-value Cauchy problem. The governing equations can be obtained using different methods. One possibility, which yields a system with a

minimal number of unknown functions, is the Lagrange method [Vinogradov 92a, Vinogradov 92b, Vinogradov 93]. In this approach, it is assumed that the links in the cluster are, indeed, rigid and restricting the relative motion of the connected nodes.

The equations are obtained as follows. A spanning tree is selected in the cluster's graph. If the cluster is connected to a fixed body, then the tree is rooted at this node (see Figure 5.16). Note that if the graph contains loops, then the spanning tree does not cover the whole graph and the additional links are regarded as "cuts" (see the link marked by a dotted line in the Figure 5.16).

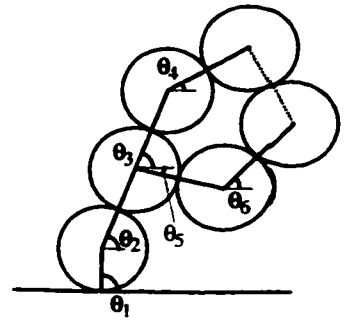


Figure 5.16. Lagrangian formulation of the equation of motion.

The rotational angles of links θ_i, $i = 1..n$ are selected as independent variables. The resulting differential system contains n second-order differential equations plus $k \geq 0$ additional algebraic equations representing the constraints, which correspond to the "cut" links. The system can be linearized and solved using an appropriate method [Vinogradov and Sun 97]. However, the matrix of the system is quite dense and has to be updated every time the topology of the cluster changes, which is a quite time consuming procedure. Also, the matrix has to be computed and stored in an explicit form.

An alternative method, which yields a more natural representation of equations, is based on the Newton-Euler formulation of the equations of motion. The resulting system of ODE is sparse and can be efficiently stored in an implicit form right inside the data structure (the graph representing the cluster). Another advantage of the approach is that the resulting system is purely differential, as opposed to a differential-algebraic system

obtained using the Lagrangian approach. The disadvantage of the method is that it yields a system with more unknown functions.

Consider a single member of a cluster. Its motion in 2D is defined by the following differential equations.

$$\ddot{\mathbf{x}}_i = \frac{1}{m_i}\left(\mathbf{F}_i\left(\mathbf{x}_i, \dot{\mathbf{x}}_i, \theta_i, \dot{\theta}_i, t\right) + \sum_{j=1}^{p_i} \mathbf{R}_{i,k_j}\left(\mathbf{x}_i, \dot{\mathbf{x}}_i, \theta_i, \dot{\theta}_i, \mathbf{x}_{k_j}, \dot{\mathbf{x}}_{k_j}, \theta_{k_j}, \dot{\theta}_{k_j}\right)\right), \qquad (5.11)$$

$$\ddot{\theta}_i = \frac{1}{I_i}\left(M_i^F\left(\mathbf{x}_i, \dot{\mathbf{x}}_i, \theta_i, \dot{\theta}_i, t\right) + \sum_{j=1}^{p_i} M_{i,k_j}^R\left(\mathbf{x}_i, \dot{\mathbf{x}}_i, \theta_i, \dot{\theta}_i, \mathbf{x}_{k_j}, \dot{\mathbf{x}}_{k_j}, \theta_{k_j}, \dot{\theta}_{k_j}\right)\right), \qquad (5.12)$$

where $\mathbf{x}_i$ is the position of the center of mass of the i-th body, θ_i is the rotational angle of the body, $i = 1..n$, m_i and I_i are the mass and the moment of inertia of the body, $\mathbf{F}_i$ and M_i^F are the external force and its moment acting on the body, $\mathbf{R}_{i,k_j}$ and M_{i,k_j}^R are the reaction force and its moment acting in a peer link of the body and p_i is the number of peers of the body.

Note that the equations depend on both coordinates of the body and the coordinates of its peers. Thus, the differential equations defining the motion of all members of a cluster have to be solved simultaneously. The differential system contains $3n$ unknown functions and $3n$ equations. The initial state of the system is given and there are no algebraic constraints, since every link is represented by the reaction force $\mathbf{R}_{ij}$ acting in it.

The reaction force R_{i,k_j} and its moment M_{i,k_j}^R are computed using equations (5.1) – (5.10). Note that the expressions are linear in their arguments, except for the case when the link is crossing the "slipping" boundary, where the function is continuous, but not smooth. Also note that $\mathbf{R}_{ij} = -\mathbf{R}_{ji}$, i.e. the system is symmetric. If the external force is

given by a linear function (e.g. the gravitational force $\mathbf{F}_i = m_i\mathbf{g}$), then the differential system is linear.

The 3D equations are the same, except that both $\mathbf{x}_i$ and $\boldsymbol{\theta}_i$ are 3D vectors and thus the system has the order of $6n$.

Another important property of the system is that it is diagonally dominant. Consider the reaction forces acting on the i-th body. Essentially, $\mathbf{R}_{ij}$ is proportional to $(\mathbf{x}_i - \mathbf{x}_j)$[1]. Thus, a single equation in the system's matrix can be written as[2]

$$\ddot{\mathbf{x}}_i = \frac{1}{m_i}\left(\mathbf{F}_i + \left(K_{j_1} + K_{j_2} + \ldots + K_{j_{p_i}}\right)\mathbf{x}_i - K_{j_1}\mathbf{x}_{j_1} - K_{j_2}\mathbf{x}_{j_2} - \ldots - K_{j_{p_i}}\mathbf{x}_{j_{p_i}}\right), \quad (5.13)$$

where K_{j_i} is the stiffness of the j_i-th link. If the body is connected to a fixed body, then there is an additional element in the $\mathbf{x}_i$ multiplier. So, the matrix of the system is diagonally dominant (or at least each row's diagonal element is not smaller than the sum of the off-diagonal elements). This has an important effect on the stability of the solution of the system.

5.6.3. Numerical solution of the ODE system

The motion of a cluster is defined by a relatively simple system of ODEs. In most cases, the system can not be resolved analytically (except, maybe, for clusters consisting of a free single body). A numerical method has to be used to obtain the solution of the system [Burden et. al. 78].

Generically, an ODE system can be written in the canonical form as

---

[1] It is also proportional to $(\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j)$.

[2] The equation also contains the velocity member of a similar form, which has been omitted for clarity.

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, t), \tag{5.14}$$

where $\mathbf{x}$ is the vector of unknown functions and the $\mathbf{F}(\mathbf{x},t)$ is the a vector-function, which is computed based on the value of $\mathbf{x}$. Note that most numerical methods do not require that $\mathbf{F}(\mathbf{x},t)$ is given in an explicit form, rather, they assume that a subroutine is available, which computes the value of $\mathbf{F}(\mathbf{x},t)$ for a given $\mathbf{x}$ and t.

The equations (5.11) and (5.12) can be converted to the canonical form. Then, a system containing $6n$ unknown functions is obtained. Note that half of the unknowns are just derivatives of other half: the velocity $\mathbf{v}_i = \dot{\mathbf{x}}_i$ and the angular velocity ω_i are the derivatives of $\mathbf{x}_i$ and θ_i, respectively.

Most numerical methods obtain the solution on a time-grid, i.e. the unknown functions are represented by a sequence of values $\mathbf{x}(t_0)$, $\mathbf{x}(t_1)$, $\mathbf{x}(t_2)$, etc. The difference between two consecutive values of time is called the *timestep*, $h_i = t_{i+1} - t_i$, $i \geq 0$. The majority of methods use a fixed timestep throughout the integration.

The simplest numerical method, which can be used to solve the system (5.14), is the Euler's method [Burden et. al. 78]. The method approximates the value of $\mathbf{F}(\mathbf{x})$ during a timestep $t \in [t_i, t_{i+1}]$ by a constant $\mathbf{F}_i = \mathbf{F}(\mathbf{x}(t_i))$, which is calculated at the beginning the timestep using the known value of $\mathbf{x}(t_i)$. Then, the value of $\mathbf{x}(t_{i+1})$ is obtained as

$$\mathbf{x}(t_{i+1}) = \mathbf{x}(t_i) + \mathbf{F}_i h_i . \tag{5.15}$$

It is possible to show that the popular Distinct Element method (DEM) [Cundall & Strack 79] is equivalent to integrating the equations of motion using Euler's method. Indeed, the forces acting on bodies are computed at the beginning of each timestep and assumed to be constant during the timestep. Then, the position and velocity of each particle are computed using the following formulas.

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{v}_i h_i + \frac{\mathbf{F}_i h_i^2}{2m_i},\ \mathbf{v}_{i+1} = \mathbf{v}_i + \frac{\mathbf{F}_i h_i}{m_i}, \tag{5.16}$$

where $\mathbf{F}_i$ is the total force acting on the particle at the beginning of the timestep. Then, then value of $\mathbf{F}_{i+1}$ is computed using the obtained values for $\mathbf{x}_{i+1}$ and $\mathbf{v}_{i+1}$, and the procedure is repeated.

The Euler's method can be used to carry out the simulation at the minimal computational cost. Also, no additional storage is required to record the intermediate states. However, the method is not too accurate [Burden et. al. 78]. There are many alternative methods, which provide a better accuracy at a slightly higher computational and storage cost. One of these methods is the 2-step predictor-corrector method [Burden et. al. 78]. The integration is performed using the following formulas.

$$\mathbf{x}_{i+1}^p = \mathbf{x}_i + \mathbf{F}(\mathbf{x}_i)h_i \text{ (predictor step)}, \tag{5.17}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{\mathbf{F}(\mathbf{x}_i) + \mathbf{F}\left(\mathbf{x}_{i+1}^p\right)}{2} h_i \text{ (corrector step)}. \tag{5.18}$$

The predictor-corrector integration scheme achieves a higher accuracy (second order, as compared to the first order in Euler's method) at the cost of two RHS calculations per timestep (instead of one in the Euler's method). Note that no extra storage is required, since the reaction force computation requires that two consecutive states are stored.

Note that the structure of the MBD model can incorporate any integration method with a desired accuracy. This is achieved by storing several states per body or link. For a detailed review of various numerical integration methods, including some very accurate iterative methods, see [Vinogradov and Sun 97]. The error-control scheme based on the energy-conservation law can be used to control the accuracy of the integration [Vinogradov and Sun 97].

5.6.4. Timestep selection

Varying the timestep in integrating ODE can be used in order to reduce the computational effort when the system goes through a stable state by increasing the timestep and to maintain the desired accuracy when the system undergoes rapid dynamic changes by decreasing the timestep. There exist various techniques for adjusting the timestep based on either the system parameters or by analyzing the behavior of the solution itself [Burden et. al. 78]. These methods, however, assume that a single system of ODEs is being solved. The MBD model has to deal with a transforming set of ODE systems, as the topology of the simulated system changes. Consider an example presented in the Figure 5.17.

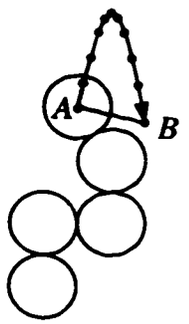


Figure 5.17. Timestep selection.

The cluster has detected that the disk A (shaded in the picture) is disconnecting from the cluster (it is moving upwards). So, the cluster has to disintegrate into two clusters, one containing the disk A and the other containing the rest of the disks. The original cluster was large and the selected timestep was quite small. Thus, if the integration continued with the same timestep, then it would take the disk A 10 timesteps to arrive at the point B. Note that when the disk arrives to the point B, the two clusters will have to merge.

When the disk A is disconnected from the large cluster, a new cluster containing only this disk is formed. This cluster has its own system of ODE, so an appropriate timestep should be selected. Since the cluster contains a single disk only, the ODE has an analytical solution (assume that the disk moves in a gravitational field), and, thus, a much

larger timestep can be selected without sacrificing the accuracy. Assume that a timestep is selected, which is 10 times larger than the timestep of the original cluster. Then, the disk will arrive at the point B in a single timestep.

Thus, once the clusters are separated, and the new cluster has predicted the trajectory of the disk A, it is detected that the disk A is moving *towards* the cluster. An *insert-link* event is scheduled at the current time and, thus, the system has arrived to the original state. This leads to an infinite sequence of events (*remove-link*, *insert-link*, *remove-link*, *insert-link*, etc.). Note that this situation is quite similar to the deadlock described in the Section 3.5.4.

The situation can be dealt with by conserving the timestep for a period of time after the original cluster has disintegrated. Therefore, the disk A physically separates from the cluster so that when the timestep is increased, the collision with the cluster is not scheduled right away.

5.7. Quasi-Rigid Bodies

The *Quasi-Rigid Body* (QRB) object is a special object, which has both properties of a rigid body and, as such, can be a part of a cluster, and also the properties of a compound object, which consists of other objects. Another important property of a QRB is that it can contain other QRBs as its members. A QRB can be either free or fixed.

The QRBs are not simulated on a dynamic scene, but rather considered as static objects, which can respond to changing external conditions.

5.7.1. Definition of Quasi-Rigid Body

A *Quasi-Rigid Body* is a group of $n \geq 2$ objects (disks and/or boundaries) without internal degrees of freedom. This means that if a QRB contains a boundary, then the whole group is fixed, and if a QRB only contains disks, then the whole group has only 3 degrees of freedom (6 degrees of freedom in 3D), i.e. it behaves as a single rigid body.

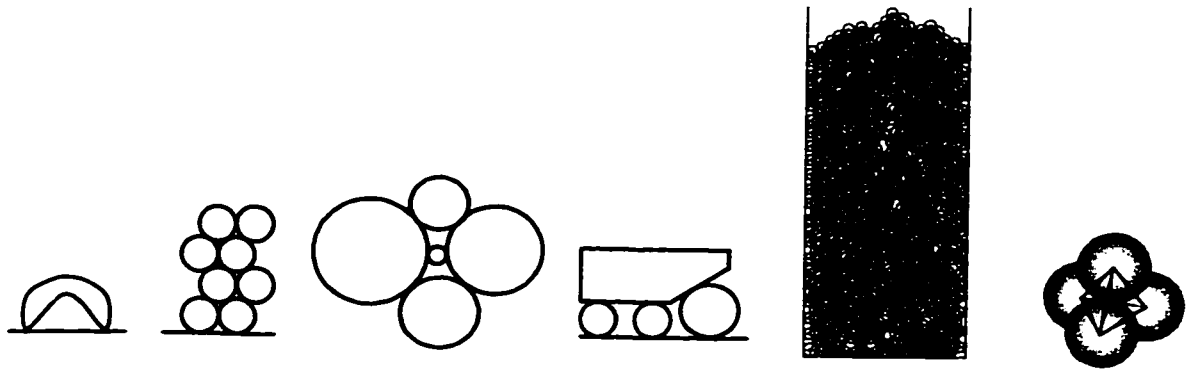


Figure 5.18. Examples of Quasi-Rigid Bodies.

Several examples of QRBs are presented in the Figure 5.18.

Since a QRB can not have internal degrees of freedom, it can not contain any links that are slipping or rolling. The main property of a QRB is that no part of it can move without breaking, slipping or rolling one or more links.

5.7.2. Non-slipping link model

Consider a non-slipping link connecting two bodies B_1 and B_2. Assume that the point of contact is P (see Figure 5.19).

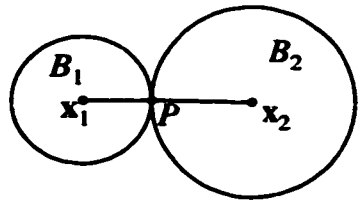


Figure 5.19. Non-slipping link.

The motion of the point P can be expressed as a function of coordinates of either B_1 or B_2. Clearly, the basic condition imposed by the link is that the velocity of the point P expressed either way has to be the same.

This can be formalized as follows. Consider displacements of bodies $(\delta\mathbf{x}_1, \delta\theta_1)$ and $(\delta\mathbf{x}_2, \delta\theta_2)$. The following kinematic conditions must be satisfied:

$$|\mathbf{d}+\boldsymbol{\delta}| = r_1 + r_2, \tag{5.19}$$

where $\mathbf{d} = \mathbf{x}_2 - \mathbf{x}_1$, $\boldsymbol{\delta} = \delta\mathbf{x}_2 - \delta\mathbf{x}_1$, and

$$\delta\theta_1 r_1 + \delta\theta_2 r_2 = \delta\theta(r_1 + r_2) \text{ [Vinogradov and Sun 97]}, \tag{5.20}$$

where $\delta\theta$ is the relative rotation of the vector $\mathbf{x}_2 - \mathbf{x}_1$, i.e.

$$\delta\theta = \arccos\frac{(\mathbf{d}, \mathbf{d}+\boldsymbol{\delta})}{|\mathbf{d}||\mathbf{d}+\boldsymbol{\delta}|} = \arccos\left(1 + \frac{(\mathbf{d}, \boldsymbol{\delta})}{(r_1 + r_2)^2}\right). \tag{5.21}$$

When the displacements are very small, the equations can be linearized:

$$(\boldsymbol{\delta}, \boldsymbol{\eta}) = 0, \tag{5.22}$$

$$\delta\theta_1 r_1 + \delta\theta_2 r_2 = (\boldsymbol{\delta}, \boldsymbol{\tau}), \tag{5.23}$$

where $\boldsymbol{\eta}$ and $\boldsymbol{\tau}$ are the normal and tangent unit vectors of the original link.

Thus, the condition can be represented by 2 linear equations relating the displacements of B_1 and B_2 (in 3D a non-slipping link is represented by 3 equations).

5.7.3. Graph-theoretical properties of QRBs

Each QRB satisfies the following important property.

Property 5.1. For a planar QRB with n nodes and e links the following inequality is satisfied: $3n - 2e \leq 3$.

Proof. Assume that the QRB is fixed. If it is not fixed, then we can impose an artificial condition that the first body is fixed, this will not affect the topology of a QRB. Consider a system of equations representing the system of bodies. Clearly, each link is represented by 2 linear equations. Also since the first body is fixed, there are three additional equations restricting the motion of that body. Thus, the system has $2e+3$ linear equations and $3n$ variables.

Since the QRB does not have internal degrees of freedom, the linear system has only one solution: $\delta \mathbf{x}_i = 0$, $\delta \theta_i = 0$, i.e. the bodies can not move. Therefore, the rank of the system is $3n$ (otherwise, the solution would contain free variables and would not be unique). So, the total number of equations in the system has to be is at least $3n$, i.e. $3n - 2e \leq 3$.

■

Note 1. The above property is a custom form of the Gruebler's equation [Shigley and Uicker 95].

Note 2. For a QRB in 3D $6n - 3e \leq 6$ (or $2n - e \leq 2$).

Note 3. The condition is in the form of an inequality, because some of the equations in the system can be linearly dependent. The physical meaning of this is that some links can impose redundant conditions, i.e. the QRB would still be intact if these links were allowed to slip. Consider an example presented in the Figure 5.20.

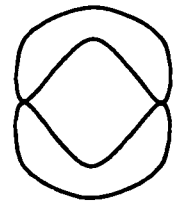


Figure 5.20. Overconstrained system.

Clearly, it is enough that just one link is not slipping, this will guarantee that the other one will not slip. Note that for this example $3n - 2e = 3 \cdot 2 - 2 \cdot 2 = 2$. This does not mean that the QRB has just 2 degrees of freedom (it is still 3), but only indicates that one of the links imposes a redundant condition, i.e. the system is overconstrained.

Note 4. The Property 5.1 is a necessary but not sufficient condition. If for some graph $3n - 2e \leq 3$, it does not necessarily mean that the graph represents a QRB. Consider an example presented in the Figure 5.21.

Figure 5.21. Property 5.1 is not a sufficient condition.

Clearly, $n=3$ and $e=3$, so $3n-2e=3$. However, the system has one degree of freedom: the ball can roll on top of the boomerang.

The following condition is conjectured. If for a graph with n nodes in general positions and e links the inequality $3n-2e \le 3$ is satisfied, then the graph contains a QRB.

There exist certain examples, where this condition does not apply. These examples, however, represent improbable cases, since they only work for very special arrangements of bodies of specific sizes. Several such examples are presented in the Figure 5.22.

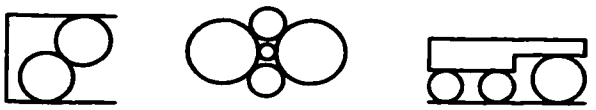


Figure 5.22. Examples of irregular configurations.

The body configurations in the examples turn into regular QRBs if any small perturbation is applied to disk sizes.

Property 5.2. If a graph does not contain loops shorter than 6 (i.e. each face of the graph contains at least 6 edges), then the graph does not contain a QRB.

Proof. Since the graph is planar, Euler's formula is satisfied: $f-e+n=2$, where f is the total number of faces in the graph (including the external face), e is the number of edges and n is the number of nodes. Walking around each face, count every edge. Then the total number of edges counted will be at least $6f$ (because each face has at least 6 edges). Also, each edge will be counted exactly twice (because it belongs to two faces). Thus, $2e \ge 6f$ or $f \le \frac{e}{3}$. Note that this inequality is also true for any subgraph of the original graph because any subgraph is also a planar graph, which does not contain loops

shorter than 6. Then, for any subgraph the Euler's formula can be written as $\frac{e}{3} - e + n \geq 2$. Then, $3n - 2e \geq 6$ for any subgraph of the original graph. Therefore, the original graph could not contain a QRB, because due to the Property 5.1 in a QRB the following inequality must be satisfied: $3n - 2e \leq 3$.

■

5.7.4. QRB detection

The abstract problem of QRB detection is as follows. Given a graph, it is assumed that all links in the graph are non-slipping. It is required to detect all subgraphs of the original graph, which represent QRBs, and are maximal, i.e. a QRB is not a subgraph of some other QRB.

The problem of QRB detection is not easy to solve. A straightforward approach would be to consider all possible subgraphs of the original graph. Note that it is not enough to check that $3n - 2e \leq 3$ for a subgraph, since this does not guarantee that the subgraph in consideration is a QRB. Indeed, the system of equations representing the system of links has to be considered and the groups of linearly dependent equations have to be separated. Such an approach would require an exponential time to complete the search.

An alternative approach based on pattern matching is suggested. The algorithm is correct, i.e., it returns a list of true QRBs present in the original graph. It is conjectured that the algorithm is complete, i.e., that it returns the full set of QRBs.

The approach is based on the following observation.

Property 5.3. Assume that a graph G with m nodes is a subgraph of a graph H with $n > m$ nodes and that both G and H represent QRBs. Then, if the graph G is collapsed to a single node in H obtaining the reduced graph H_1, then H_1 is also a QRB[1].

Proof. Since H is a QRB, the system of linear equations S_H representing the displacements of nodes in H has a single trivial solution. Consider a subset S_G of S_H, which corresponds to the links in G. S_G is a linear system of $3m$ variables $(\delta x_i, \delta y_i, \delta\theta_i)$, $i = 1..m$. Since G is a QRB, S_G also has a unique trivial solution. Compute the coordinates of the center of mass of G (x_G, y_G, θ_G) and perform a substitution in the original system such that the displacements $(\delta x_i, \delta y_i, \delta\theta_i)$ are expressed through the displacement of the center of mass of G $(\delta x_G, \delta y_G, \delta\theta_G)$. Then, the resulting system containing $3n - 3m + 3$ variables still has a unique trivial solution, i.e. H_1 is a QRB.

■

The physical analogue of the proof is that since G is a QRB, it is virtually replaced with a real rigid body (or, equivalently, every link in G is "welded"). This will not add any degrees of freedom to H; thus, the resulting graph H_1 will remain a QRB.

The property enables detecting QRBs by looking for basic "QRB templates" in the original graph and collapsing them to single nodes (by applying the *Group* primitive). The QRB detection process is terminated when no more templates can be found in the graph. An example of QRB detection process is presented in the Figure 5.7. Note that for the QRB detection purposes, all fixed nodes in the graph are represented by a single fixed node.

---

[1] In other words, if *Group* primitive is applied to G then the resulting hierarchical graph H_1 is also a QRB.

5.7.5. QRB templates

The *QRB template* is defined as a minimal graph representing a QRB, i.e. a graph that does not contain another QRB template as a subgraph.

The three basic QRB templates are the "loop", "hamburger" and "triangle" (see Figure 5.23).

Figure 5.23. Basic QRB templates.

Note that it is quite easy to detect the first three templates by simply analyzing the faces of the original graph. Each template corresponds to a single face with either 1, 2 or 3 edges. Note that the "loop" template does not appear in practice (although it is listed here for completeness), because a configuration when a link appears between two nodes belonging to the same QRB is improbable. The "hamburger" and "triangle" templates are the most common ones and the majority of QRBs are constructed using only these two templates.

Note that to continue the list of QRB templates, it is necessary to look for graphs that do not contain loops, double edges and triangles, because a QRB template can not contain another QRB template as a subgraph. Note that due to the Property 5.2 a QRB template must contain a face with 4 or 5 edges. Thus, it does not make sense to search for QRB templates among graphs only containing 6-faces or higher.

It will be shown that there exist an infinite number of different QRB templates. Consider a graph shown in the Figure 5.24.

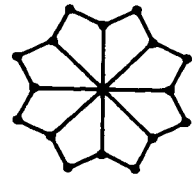


Figure 5.24. A Q_{f_4} template, $f_4 = 8$.

The graph represents a QRB template Q_{f_4}, which consists of f_4 4-faces, $f_4 \geq 2$. The graph has $2f_4 + 1$ nodes and $3f_4$ edges. Thus, the total number of degrees of freedom is $3(2f_4 + 1) - 2 \cdot 3f_4 = 3$.

Are there QRB templates containing 5-faces or higher? Consider a graph consisting of f_4 4-faces and f_5 5-faces around a common central point (see Figure 5.25).

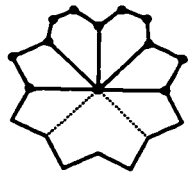


Figure 5.25. Building a template with 5-faces.

The graph contains $1 + 2f_4 + 3f_5$ nodes and $3f_4 + 4f_5$ edges. Obviously, the graph has $3(1 + 2f_4 + 3f_5) - 2(3f_4 + 4f_5) - 3 = f_5$ internal degrees of freedom. It is possible, however, to obtain a QRB template from the graph by adding several 4-faces (see Figure 5.26).

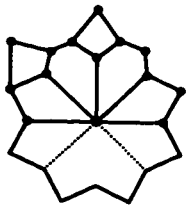


Figure 5.26. A QRB template obtained from a graph with 5-faces.

By adding a 4-face on the outside boundary of the graph one additional node and two edges are introduced, i.e. the total number of degrees of freedom is decreased by one. So,

by adding f_5 4-faces, a new QRB template can be obtained. Note that it is not allowed to attach the 4-faces on top of each other (see Figure 5.27), because the resulting graph would contain a subgraph congruent to Q_{f_4}, $f_4 = 2$.

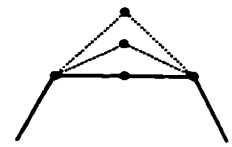


Figure 5.27. Attaching 4-faces on top of each other.

Adding a 5-face to the graph can either add 1 node and 2 edges, 2 nodes and 3 edges or 3 nodes and 4 edges (see Figure 5.28). Correspondingly, the total number of degrees of freedom is either decreased by one, not changed, or increased by one.

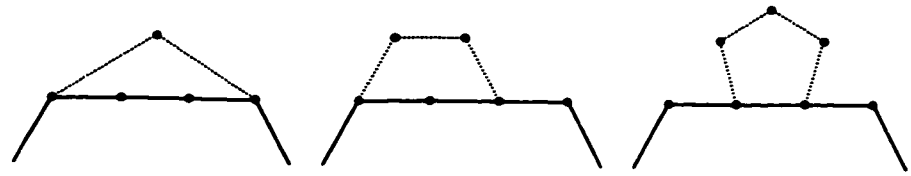


Figure 5.28. Adding a 5-face.

Likewise, adding a 6-face may change the total number of degrees of freedom from minus one to plus two, and so on. Thus, it is possible to build different QRB templates using various faces. Note that while building a QRB template, one must keep in mind that it is not allowed to obtain graphs that contain templates, which were already obtained. Thus, it is not very straightforward to build a template, which does not contain 4-faces. It is possible, though, and the example of such template is presented in the Figure 5.29. The template contains 11 nodes and 15 edges.

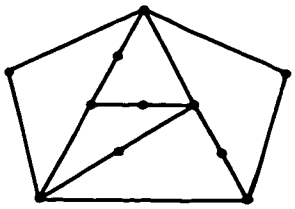


Figure 5.29. A 5-face only QRB template.

A generic formula can be obtained as follows. Assume that the graph contains f_1 loops (1-faces), f_2 double edges (2-faces), f_3 triangles (3-faces) and so on. Then, counting faces and edges, one can obtain the following relationships: $f = \sum_i f_i$, $2e = \sum_i i f_i$. From Euler's formula, $n = 2 + e - f$. Thus, the following formula is obtained.

$$3n - 2e = 6 + e - 3f = 6 + \sum_i \left(\frac{i}{2} - 3 \right) f_i . \tag{5.24}$$

Property 5.2 follows from equation (5.24). Since $f_i = 0$ for $i = 1..5$, all negative multipliers are absent and, consequently, $3n - 2e \geq 6$.

Note that it is quite difficult to perform pattern matching with a large template. In the general case, the problem can be as difficult as the original straightforward QRB detection approach, which checks all possible subgraphs. Thus, it makes sense to perform pattern matching with small templates only. This might potentially leave some QRBs undetected, but the probability that such QRBs will appear is very low. It is usually sufficient to perform matching only with two basic templates: "hamburger" and "triangle" (see Figure 5.23).

5.7.6. Dynamic maintenance algorithms

An algorithm maintaining a QRB must watch the forces in the QRBs links and detect slips and disconnections. Note that a connection can not occur inside a QRB because it is assumed that the bodies in a QRB do not move relatively to each other. Likewise, a grip event can not occur in a link in a QRB because all links in the QRB are already stable.

If a slip or a disconnection is detected then the fundamental property of a QRB, which states that the QRB does not have internal degrees of freedom, is violated. Therefore, such QRB has to be disintegrated. If the owner of the link is a QRB, which is a member of another QRB, then this parent QRB has to be disintegrated too, and so on. As a result of this disintegration process, a set of QRBs, representing the intact subtrees of the original tree, is obtained (see Figure 5.30, where the original disintegrated QRB is marked with a larger dot).

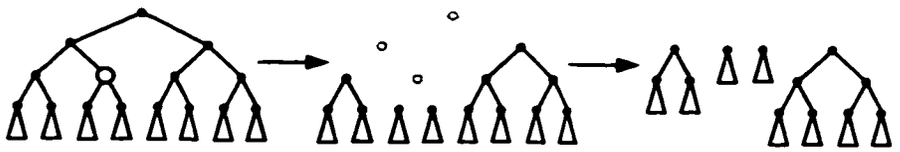


Figure 5.30. Disintegrating a QRB.

Then, the QRB detection algorithm *FindQRBs* is applied to the obtained set of bodies and their neighbors, and, possibly, a different QRB configuration is constructed. Note that it is possible that the new QRB contains all the bodies of the QRB, which just got disintegrated. The reason for this is that many links in the original QRB might be redundant. Consider an example presented in the Figure 5.31, where $n = 7$, $e = 11$ and thus $3n - 2e = 1$.

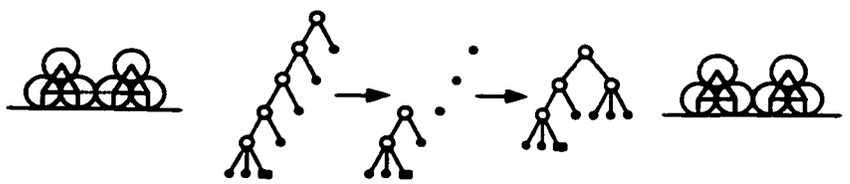


Figure 5.31. Removing a redundant link from a QRB.

Note that the original tree was built as the disks were settling one-by-one, such that the left triangle was formed first, and then the remaining three disks were added one after another. Assume that the link marked as dotted line in the left picture is removed. Then the tree is disintegrated; however, once the *FindQRBs* routine is run, all six disks are connected into a single QRB once again. Note that the new QRB has a different topology.

The *FindQRBs* routine first detected a triangle consisting of three disks. Then, the two remaining QRBs were connected into a single fixed QRB, because the triangle is connected to the first QRB (the one which did not get disintegrated) by two links. The QRB detection process is not defined uniquely, because at any moment of time in the detection process it is possible that several QRB templates are found in the graph. Then, the resulting structure might depend on the order, in which these templates are *Group*ed (collapsed into single nodes).

Generally speaking, it is more efficient to build wide and shallow QRB trees (as opposed to narrow and deep trees), because when a QRB is disintegrated, the resulting set of bodies, which is passed to the *FindQRBs* routine, is smaller. When a link is removed, every node on the path from the QRB being disintegrated to the root of the QRB tree has to be disintegrated as well. Thus, if the length of the path from a QRB to the root of the tree is minimized, then the total number of disintegrated QRBs is also minimized.

The goal of "balancing" the QRB tree can be achieved by analyzing the total number of disks in each of the found templates. A template with the least total number of disks has to be *Group*ed first. The alternative approach is minimizing the maximum depth of the trees comprising the template.

5.7.7. Forces in QRBs

The primary function of the QRB object is computing the internal stresses in the links of the structure. One of the possible approaches is to maintain the inverse stiffness matrix of the QRB as it is done in the QS model (see Chapter 4). This allows computing the internal forces in links by directly multiplying the inverse matrix by the vector of external forces, acting on the members of the QRB. Note that for this approach, the tree-like structure of the QRB is not used. Rather, the hierarchical graph represented the QRB is "flattened" into a regular graph. The methods presented in this section assume that the QRB is fixed. The case, when the QRB is free, will be considered in the next section.

5.7.7.1. RIMA-based force computation

The Recursive Inverse Matrix Algorithm (RIMA, see Chapter 4) can be used to compute the internal stresses in a QRB. The method is based on maintaining the inverse stiffness matrix of the system. The internal stresses are computed by multiplying the inverse matrix by the vector of external forces.

The inverse stiffness matrix has to be maintained as the topology of QRB changes. Obviously, the only way a new QRB is formed is by a QRB template. Thus, two or more QRBs are merged into one.

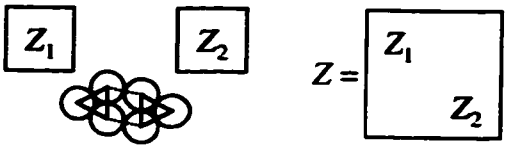


Figure 5.32. Combining two inverse matrices.

Assume that the inverse stiffness matrices are maintained for each of the smaller QRBs. When the smaller QRBs are merged into a larger QRB, their inverse matrices are used as the blocks of the resulting inverse matrix. This matrix is then updated as the links connecting the smaller QRBs are added to the system (see Figure 5.32).

5.7.7.2. Conjugate Gradient iterative method

The alternative approach is not to maintain the inverse matrix, but rather to solve the linear system describing the stresses in QRB links each time the external forces or the QRB topology changes. The linear system possesses nice properties: it is symmetric and diagonally dominated. Some efficient iterative algorithms can be employed, which can obtain the solution in a time comparable with the RIMA approach. Moreover, the values from the previously obtained solution can serve as good approximations of the new solution in the case, when the external forces do not change too much.

One of the iterative methods, which can be used to solve the system, is the Conjugate Gradient method [Burden et. al. 78]. This method uses the fact that the system is symmetric and positive definite.

Assume that the linear system is defined as $\mathbf{Ax} = \mathbf{b}$. The solution $\mathbf{x}^{(i)}$ and the residual $\mathbf{r}^{(i)} = \mathbf{b} - \mathbf{Ax}^{(i)}$ are updated on each iteration as follows.

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)},\ \mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)} \tag{5.25}$$

where $\mathbf{p}^{(i)} = \mathbf{r}^{(i)} + \beta_{i-1}\mathbf{p}^{(i-1)}$ is the search direction vector, $\beta_i = \mathbf{r}^{(i)T}\mathbf{r}^{(i)} / \mathbf{r}^{(i-1)T}\mathbf{r}^{(i-1)}$ and $\alpha_i = \mathbf{r}^{(i)T}\mathbf{r}^{(i)} / \mathbf{p}^{(i)T}\mathbf{A}\mathbf{p}^{(i)}$, $\mathbf{p}^{(1)} = \mathbf{r}^{(0)}$. The iterations are stopped when the norm of the residual becomes smaller than some given tolerance value.

A preconditioner can be used to improve the convergence of the iterative method. The preconditioner $\mathbf{M}$ is usually computed somehow from the original matrix $\mathbf{A}$ before the iterations begin. Then, the search vector is computed based on the value of $\mathbf{z}^{(i)} = \mathbf{M}^{-1}\mathbf{r}^{(i)}$. The simplest preconditioner is $\mathbf{M} = \mathbf{I}$, i.e. $\mathbf{z}^{(i)} = \mathbf{r}^{(i)}$. In this particular implementation, the preconditioner was selected as the main diagonal of the matrix $\mathbf{A}$.

5.7.7.3. Comparison of RIMA and CG methods

Both methods have their advantages and disadvantages. The RIMA-based method computes the forces very quickly by simple matrix-vector multiplication. However, it has the overhead of updating the inverse matrix each time the topology of the QRB changes. Also, when the QRB is large, the matrix occupies a lot of memory.

The iterative method does not require any additional storage. It can be as efficient as RIMA when the external conditions don't change too much, thus, the previous solution provides a good approximation to the current solution. In some cases, however, the stresses in a QRB are redistributed significantly. An illustrative example of such a situation is the "closing of a bridge" configuration (see Figure 5.33).

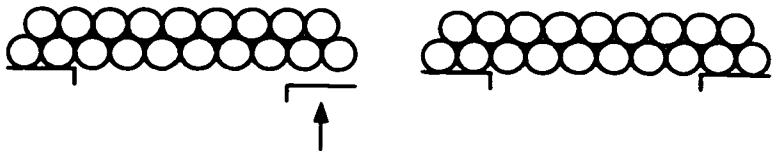


Figure 5.33. "Closing of a bridge" configuration.

The stresses in the QRB links change considerably once the right end of the bridge comes into contact with the support. This is because originally the whole weight of the bridge was supported by the left end only, while after the right end of the bridge comes into contact with the support, the weight becomes redistributed evenly between the two supports. Note that even though this configuration rarely appears in the simulation, similar stress redistributions can occur in real QRBs.

Each of the methods has its own applicability domain. A series of computational experiments was performed to compare the models. A single QRB representing a silo configuration (see Chapter 4) was simulated using both methods (note that micro-topological changes, such as slips, were not considered). The experiments were conducted for silo configurations containing up to 1000 disks. The results of the experiments are presented in the following figures.

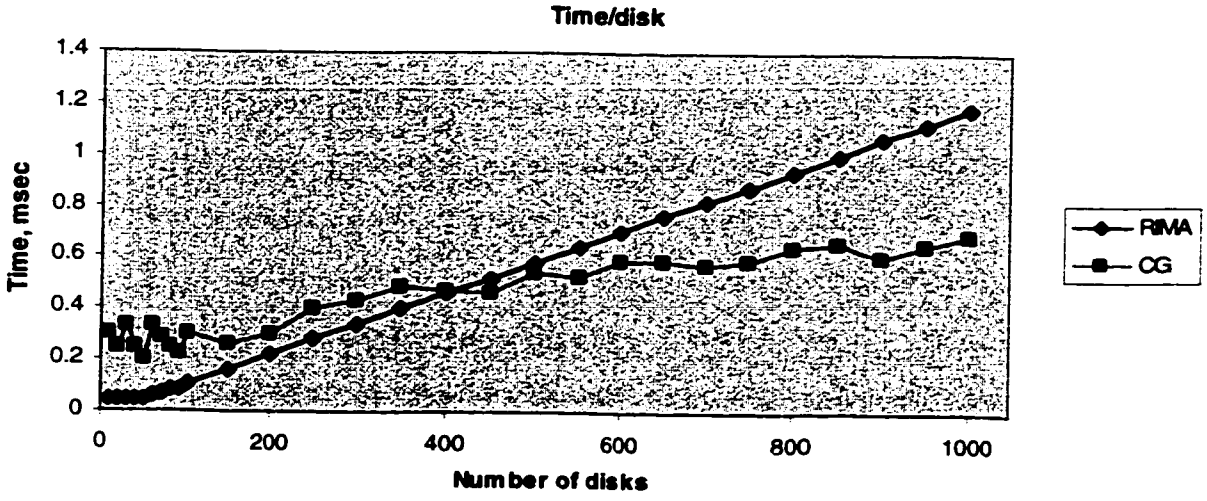


Figure 5.34. Internal stress computation time.

The comparison of the two methods in respect to the computing the internal stresses in QRB links from the external forces is presented in the Figure 5.34. Note that the time was scaled by the number of disks in the system. It can be clearly seen that the RIMA time per disk grows linearly (note that the total time, therefore, grows as a quadratic function). This is in correspondence to the fact that to compute internal stresses RIMA multiplies an $n \times n$ inverse stiffness matrix by a $1 \times n$ vector of external forces, i.e. $O(n^2)$ operations is performed in total.

The Conjugate Gradient (CG) iterative method run time does not grow as fast as RIMA's. Note that for small QRBs the computation time can be almost 6 times larger than the RIMA time. However, as the size of QRB grows, the method shows a good performance and it clearly outperforms RIMA for configurations containing 500 disks or more.

Another important factor is the overhead related to maintaining the inverse stiffness matrix by RIMA. Consider the Figure 5.35, which show the time spent for updating the inverse matrix after a single link is added to the system.

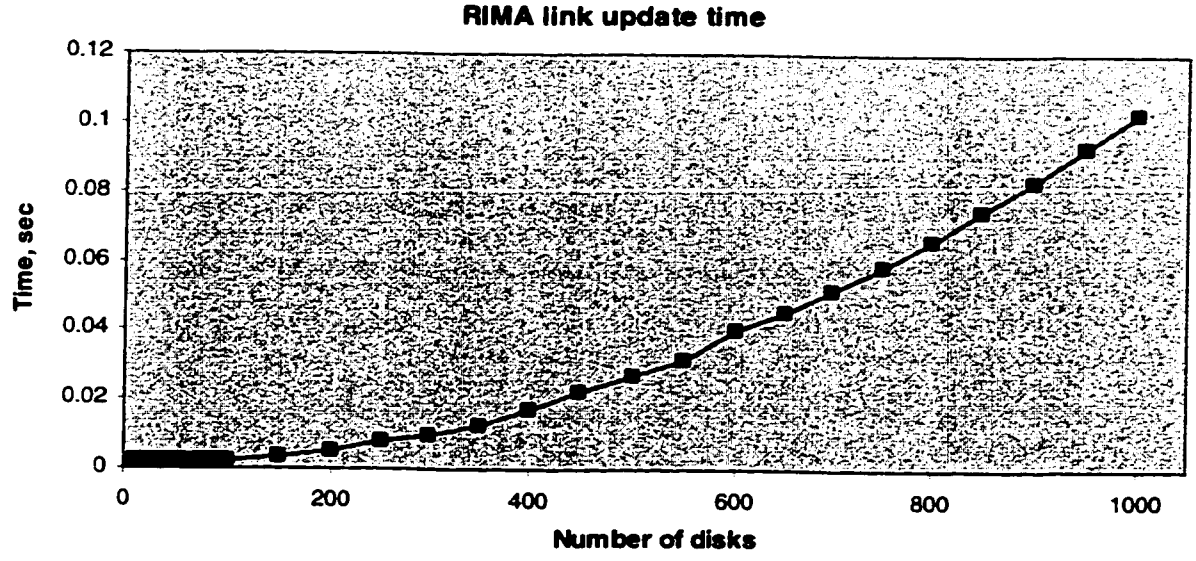


Figure 5.35. RIMA matrix maintenance overhead.

Such updates need to be performed each time the topology of a QRB changes. Note that usually only a few links need to be added to the system, because the QRB matrix is created from several known smaller blocks corresponding to QRB members (see Figure 5.32). Still, the updates are quite computationally expensive. Note that there is no maintenance overhead associated with the CG iterative method.

Another interesting point is demonstrated in the Figure 5.36, which shows the number of iterations the method required to arrive to a stable solution.

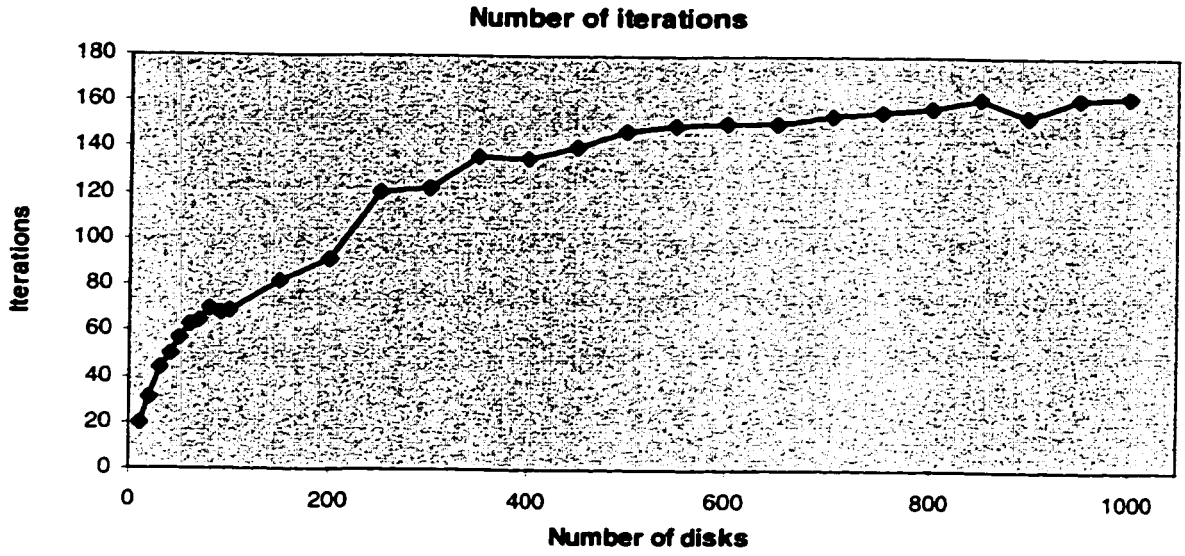


Figure 5.36. Number of iterations in CG method.

Clearly, the number of iterations almost does not increase as the number of disks grows over 500. Apparently, this is the reason for the good performance of the CG method for larger QRBs. This behavior can probably be explained by the fact that in the particular QRB configuration tested (silo) each node affects only a small number of neighbors[1]. Thus, the particles at the top and the bottom of the silo almost do not affect each other, thus, the iterative method converges faster. Such phenomenon is, however, related to the

[1] The average number of neighbors is usually referred to as the cardinality of the system.

particular topology of the QRB. The results probably would not be as good for the "closing bridge" configuration (see Figure 5.33).

5.7.7.4. Selecting which method to use

In each particular case, a decision must be made on which of the two methods has to be used based on the anticipated properties of the simulated system. RIMA is preferable for smaller QRBs and stable systems (so that the inverse matrix is not updated too often). CG method is better for large QRBs and variable systems.

It is possible to use the two methods interchangeably in a simulation. One could set up a threshold value for the number of disks in a QRB that would control which of the models is used. Another possibility is to control the number of iterations the CG method takes to achieve the desired accuracy.

It would not be efficient, if the RIMA inverse matrices were computed from scratch each time the method is switched. Thus, it makes sense to store the inverse matrices of smaller QRBs for future reuse, if a decision is made to switch to RIMA method. It is conjectured that these small inverse matrices can serve a certain purpose even in the iterative method. Indeed, it would probably improve the convergence if they were used as preconditioners in iterations. Indeed, each of these matrices represents the inverse matrix of a part of the larger system.

5.7.8. Free-flowing QRBs

Both stress computation methods defined in the previous section require that the QRB be fixed. Indeed, it is impossible to resolve the linear system defining the displacements of the nodes when the QRB is free, because the matrix of the system is indefinite (in other words, the system's equations are linearly dependent). The physical meaning of this is as follows. Assume that a set of external forces is applied to the disks of a QRB. Then, if the total sum of forces is non-zero, then the QRB will start moving. In a static declaration this means that the displacements of the nodes are infinite. If the total sum of forces is

zero, then the QRB will not move. However, in this case the linear system has an infinite number of solutions, because the whole QRB can be essentially translated as whole into any point in space, which gives an infinite number of valid values for node displacements.

This can be formulated algebraically as follows. The stress in each link is represented by the equation $\mathbf{K}_{ij}(\mathbf{d}_i - \mathbf{d}_j) = \mathbf{R}_{ij}$, where $\mathbf{d}_i$ are the displacements of nodes, $\mathbf{K}_{ij}$ is the stiffness of the link and $\mathbf{R}_{ij}$ is the stress in the link. Thus, a row in the matrix corresponding to an element i has $\sum \mathbf{K}_{ij}$ as the diagonal element and $-\mathbf{K}_{ij}$ as off-diagonal elements[1]. Note that the sum of all rows is zero, i.e. the equations are linearly dependent. Assume that $\mathbf{A}$ is the $n \times n$ block matrix of the system (where n is the number of nodes). Clearly, $rank(\mathbf{A}) < n$ because the equations are linearly dependent. Denote the vector of external forces by $\mathbf{b}$. Assume that the total sum of external forces is zero. Then, the sum of all equations in the system $\mathbf{Ax} = \mathbf{b}$ is $0 = 0$. Thus, $rank(\mathbf{A}) = rank(\mathbf{A} \mid \mathbf{b}) < n$ and, thus, the system has an infinite number of solutions. Note that every solution can be obtained as a translation of some selected solution: $\mathbf{x} = \mathbf{x}_0 + \mathbf{C}$. If the sum of external forces is not zero, then $rank(\mathbf{A}) < rank(\mathbf{A} \mid \mathbf{b})$ and the system does not have a solution.

Despite the fact that the straightforward approach does not yield an acceptable result, the problem can, indeed, be solved in a static formulation. A free-flowing QRB is a part of some cluster. Thus, its motion is defined in terms of the motion of its center of mass and the rotational displacement. Associate a non-inertial system of coordinates with the QRB. By D'Alembert's principle, it is necessary to apply the inertia force $-m_i\ddot{\mathbf{x}}$ and the

---

[1] When the QRB is fixed, the diagonal element has additional members in the sum, corresponding to the links to fixed nodes.

centrifugal force $m_i \mathbf{y}_i \dot{\theta}^2$ to each disk, where m_i is the mass of disk, $\mathbf{y}_i$ is the coordinate of the disk in the local coordinate system, $\ddot{\mathbf{x}}$ is the acceleration of the QRB, and $\dot{\theta}$ is the angular velocity of the QRB [Shigley and Uicker 95]. Note that since the QRB is stable in the local coordinate system, the total sum of forces acting on the QRB is zero.

Although the linear system can be solved, the solution is not unique. Note that it is not required to determine the actual values of node displacements. Rather, relative node displacements are of interest (because they define the stresses in the links). Thus, it is enough to find *some* solution of the system, that can be used to obtain the reaction forces in the links.

The system can be solved using the regular approach with the technique called "grounding". Consider a free QRB such that the total sum of forces acting on its nodes is zero (see Figure 5.37).

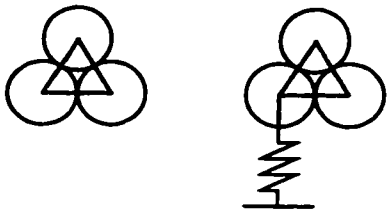


Figure 5.37. "Grounding" a free QRB.

It is possible to attach one of the nodes of the QRB to a fixed node (ground) by a "fictional link" (spring). Then, the system becomes determinate and a unique solution can be found using either RIMA or CG method. Note that since the total sum of forces acting on QRB is zero, in the resulting QRB the stress acting in the fictional link is always zero.

Algebraically, the attachment of the fictional link introduces a missing equation into the system, so that now a unique solution can be found. Also note that this solution is also a solution of the original system, thus it can be used to determine the correct stresses in QRB links. The stiffness of the fictional link can be selected arbitrarily, it will not affect the solution. It is possible to require that the link is "absolutely stiff", in other words, to

impose an exact condition on the displacement of the "grounded" disk ($\mathbf{d}_{grounded} = 0$). This approach works well for the CG method, however, it is not acceptable for the RIMA method, because RIMA requires that the stiffness matrix of a link must be explicitly given.

If two free QRBs are getting merged into a single QRB then one of the fictional links has to be disconnected so that the results of computations are not falsified.

5.7.9. QRB stiffness computation

The cluster object uses the link model to compute the stresses in the links connecting its members. If a QRB is a member of a cluster, then it is necessary to define how to compute the stress in a link between a QRB and another body.

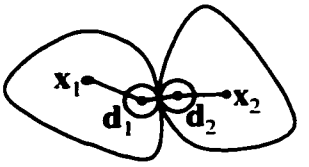


Figure 5.38. Link between two QRBs.

Consider two QRBs connected by a link (see Figure 5.38). The centers of mass of the two QRBs are $\mathbf{x}_1$ and $\mathbf{x}_2$, and the two disks in contact are $\mathbf{d}_1$ and $\mathbf{d}_2$. Then the link between the QRBs can be modeled as three sequential elements: $\mathbf{x}_1\mathbf{d}_1$, $\mathbf{d}_1\mathbf{d}_2$ and $\mathbf{d}_2\mathbf{x}_2$. The stiffness of the combined link can be computed as $\mathbf{K}_{\mathbf{x}_1\mathbf{x}_2} = \left(\mathbf{K}^{-1}_{\mathbf{x}_1\mathbf{d}_1} + \mathbf{K}^{-1}_{\mathbf{d}_1\mathbf{d}_2} + \mathbf{K}^{-1}_{\mathbf{d}_2\mathbf{x}_2}\right)^{-1}$. The stiffness $\mathbf{K}_{\mathbf{d}_1\mathbf{d}_2}$ of the link connecting two disks is known. It is required to be able to compute the "internal stiffness" of a QRB, specifically, the stiffness of a "virtual link" connecting the center of mass of the QRB to any disk in this QRB.

This can be done as follows. Assume that the QRB is considered separately from the rest of the system (i.e. it is singled out as a rigid body with its own internal stiffness properties). The stiffness is defined as the displacement of a spring under the influence of

a unity force. Consider a "grounded" QRB. Apply a unity force to the disk in question. Note that it is required to apply the inertia and centrifugal forces to all disks the QRB, so that the total force is zero. Under the influence of external forces all nodes will be displaced according to the applied force (note that if RIMA is used, then it is sufficient to take a single row of the inverse matrix). Given the displacements of nodes, it is possible to compute the "virtual displacement" of the center of mass of the QRB. Then, the difference between the displacements of the center of mass and the disk in question will give the desired stiffness value.

5.8. System

This section defines the *System* object, which is used as the root of the hierarchical graph representing the simulated granular system. The main functions of the *System* object include maintenance of the event queue, routing events, dynamic creation and deletion of bodies, statistics and logs maintenance, as well as interfacing with the Collision Detection Optimizer (CDO) module and the External Forces Computation (EFC) module.

5.8.1. Description of the System object

The *System* object is the top object in the hierarchical graph of the granular system. This compound object is persistent throughout the simulation, i.e. it is not dynamically created or deleted during the simulation (unlike the other compound objects: *Cluster* and *QRB*). The *System* object is responsible for driving the simulation and coordinating the actions of its members and the communication between them.

The *System* object is also responsible for maintenance and storage of global-scope data. This includes the event queue, which is shared among all objects. The event queue is the means of synchronizing all objects in the simulation and making sure that the events are happening in the proper order. The *System* object is also responsible for writing simulation logs and for input/output of data.

Being the top-level object in the hierarchy, the *System* object contains all of the other objects as its members or descendants. The *System* object's immediate members can include clusters and fixed bodies (boundaries or QRBs). The object can also be the owner of some links. These are the links, which connect clusters to fixed bodies. The *System* object does not maintain these links; this is done by the clusters.

The *System* object implements the main simulation loop, which retrieves the events from the head of the event queue and routes them to the objects, which are responsible for their processing. The *System* object itself is capable of handling certain types of events, including *insert-link* and *remove-link* events, as well as all of the system events (see Section 5.3.6).

5.8.2. Dynamic creation and deletion of bodies

One of the *System* object's responsibilities is dynamic creation and deletion of bodies. This is required in certain cases, for example, when the flow of particles through a pipe is simulated (see Figure 5.39).

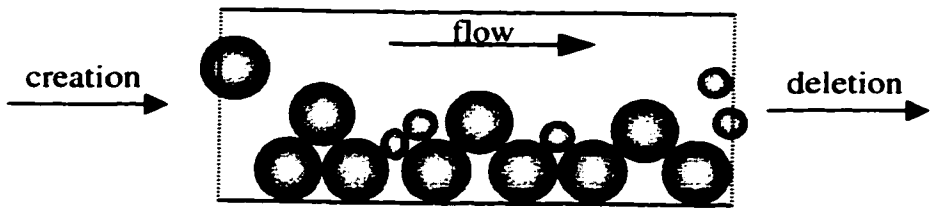


Figure 5.39. Pipe flow simulation.

The particles are created dynamically and fed into the pipe from the left end. The particles, which reach the right end of the pipe, are dynamically deleted, as they leave the simulation space.

Note that in some cases, no dynamic creation and deletion of particles is required. For example, if a ball mill is simulated, then all particles are placed into the simulation space in the beginning of the simulation, and no particles leave the simulation space during the simulation. In other cases, only dynamic deletion is required. For example, in hopper

simulations, all particles are loaded into the system in the beginning of the simulation, and then they are allowed to leave the simulation space as they flow down the hopper.

5.8.2.1. Dynamic creation of bodies

The dynamic creation of particles is managed via the *new-body* events, which are scheduled periodically. The *System* object handles the event and creates a new body with parameters in the specified ranges. Note that the new body is created at a random position, such that it does not intersect any of the existing bodies. This is achieved by random "shooting" into a specified "particle creation" area of the simulation space. If such a position can not be found, then the *new-body* event is postponed for a specified period of time (in other words, a new *new-body* event is scheduled after that period of time). If the particle can not be placed again, the creation is postponed, etc. If the particle can not be placed after a specified number of tries, a *jam* condition is reported and the simulation is halted.

After a successful placement of a new particle, the next *new-body* event is scheduled. There are several approaches for dynamic scheduling of new particle appearances. One simple approach is to schedule them at fixed periods of time. The flow will still be random because the position and radius of each new particle is selected randomly.

The alternative approach is to model the flow using the Poisson distribution [Burden et. al. 78]. Thus, a random stream with an average frequency λ can be modeled by generating new bodies at random intervals Δ_i using the following formula.

$$\Delta_i = -\frac{\ln \xi}{\lambda}, \tag{5.26}$$

where ξ is a random variable uniformly distributed in $[0,1]$.

Note that all *new-body* events can be scheduled at once at the beginning of the simulation. It is also possible to schedule the next *new-body* event only when a current

new-body event is being processed. The latter approach is more efficient, since the event queue length is smaller in this case.

5.8.2.2. Dynamic deletion of bodies

The dynamic deletion of bodies is achieved by scheduling *delete-bodies* events at specified fixed periods of time. The *System* object scans all bodies in the system and determines if they have left the simulation space. Those bodies that have left the simulation space are marked for deletion. Note, however, that the bodies are not deleted right away. Rather, the *System* object is looking for the whole clusters of bodies, such that all their members are marked for deletion. If such a cluster is found, then it is deleted from the system as a whole. This approach ensures that there are no sudden changes in system's topology, which would lead to the falsification of the simulation results.

5.8.3. Input/output, statistics and state logging

The *System* object is also responsible for loading the simulation model initially into memory. The simulation model is defined by the set of fixed boundaries, the set of particles, which are initially loaded into the simulation space, as well as definitions of the various system parameters, such as stiffness coefficients, external force field definitions, etc. Before the main simulation loop is started, the *System* object loads all model definitions and creates the initial model. Then it schedules the initial set of events (such as the initial *predict-trajectory* events for clusters and the *finish-simulation* event) and then runs the main simulation loop.

The output is performed via two mechanisms: *state logging* and *event logging*. The *state logging* is achieved by firing the *log-state* events periodically at a specified fixed rate (such as every second of virtual time). The application program can handle the *log-state* event and output selected system state parameters, such as total kinetic or potential energy of the system, average velocities of particles, pressures on the boundaries, events rate during the log interval, and many other parameters. The state logging mechanism

obtains periodic snapshots of the system state, which can be used, for example, to create an animation.

The *event logging* provides a more granular approach to monitor the changes in system state. Thus, two *log-event* application events are fired for each simulation event, as it is being handled. One *log-event* event is fired just before the simulation event is handled, and the other one is fired after the simulation event is handled. This allows tracking micro-changes in the system state due to each simulation event. The application program must be careful with processing *log-event* events and not to include any time-expensive operations, such as writing to a file, since it would have a significant effect on the simulation performance. Note that hundreds of thousands of events are processed during a usual simulation run.

5.8.4. External forces

The computation of external forces acting on the bodies in the simulated system is performed by a separate External Force Computation (EFC) module. The *System* object has a handle to the EFC module, thus enabling clusters and all other objects to query it and compute the external forces acting on each particle.

The EFC module is implemented as a separate component, which can be easily exchanged for a different one. The EFC module must implement a single function *GetExternalForce*(*Body, State*), which must compute the external force and the moment acting on the *Body*. Since a body can maintain several different states, the index of the state is also passed to the function. Note that a body state contains all information about the body, including its position, velocity, angular displacement and velocity, and the time of the state snapshot. Also, *Body* contains static information, such as the mass and the radius of the particle. Thus, by querying *Body*, the EFC module can perform complex computations involving the current state of the body, if needed.

The simplest EFC module is the *GravitationEFC*, which models the gravitational field. Thus, the external force acting on a body is computed as $\mathbf{F}_{ext} = m\mathbf{g}$, where m is the mass

of the body and $\mathbf{g}$ is the gravitation constant. The moment of the gravitational force is always zero. Note that the *GravitationEFC* can be initialized with $\mathbf{g} = 0$, which can be used to represent weightless models.

Another simple EFC models the friction force based on the current velocity of a body. This EFC can, perhaps, be used in billiard simulations. The external force is computed as $\mathbf{F}_{ext} = -k\mathbf{v}$, where $\mathbf{v}$ is the velocity of the body and k is the viscosity coefficient.

A number of more complex EFCs can be introduced. For example, an EFC can be used to model dragging forces in a pipe flow simulation (see Figure 5.39). Thus, the position of the body will be taken into account when computing the force acting on the body (this will be used to compute the flow velocity, which can be different in different points of the simulation space). Then, the difference between the velocity of the flow and the body velocity will be used to compute the drag force. A more complex flow EFC might consider the influence of the bodies on the flow, i.e. introduce the "feedback" into the model. Such a model requires maintaining the internal state.

A more complex EFC might need to evolve over time. To achieve this, the EFC is given access to the global event queue, which is maintained by the *System* object. Thus, the EFC can define its own events and schedule them in order to update the internal state of its model. The *System* object will route the EFC events to the EFC module. Note that using a common event queue is crucial to ensure that the simulation objects (including the EFC module) are synchronized.

Various EFC modules can be developed and "plugged-in" into the system without changing the rest of the model. This is achieved by using a standard interface into an EFC module. Thus, an EFC module has to be able to compute the external force acting a body, and the MBD model does not need to know how this is done. The EFC module, in turn, has access to the event queue, as well as to the states of all bodies in the system, which it can use if needed. For example, in a simulation of a planetary system the EFC module

will compute the gravitation force acting on a planet based on the current positions of other planets in the system.

5.8.5. Collision detection optimization

The collision detection optimization is a crucial task in almost any granular system simulation. It has been shown [Gavrilova 98] that collision detection can take up to 90% of the processing time, when collision detection optimization is not used. Indeed, collision detection has to be performed each time when a state of a body changes (usually, after a *predict-trajectory* event). If collision detection is not optimized then the body has to be checked for collisions with the rest of the bodies, which results in $O(n)$ collision checks, where n is the total number of particles in the system. Note that a collision check usually involves solving a quadratic equation (see Section 3.3 for a more detailed discussion). Thus, the collision detection procedure can be very computationally expensive, taking into account that the *predict-trajectory* events are very frequent.

A Collision Detection Optimizing (CDO) algorithm implicitly maintains a list of neighbors for each body, thus reducing the total number of collision checks that need to be performed. For an overview and comparison of different CDO methods see [Gavrilova 98]. Note that the CDO algorithm has to maintain the neighborhood information as the bodies move, which introduces a certain computational overhead. This overhead, however, is usually not too high and in some cases the simulation efficiency can be improved by an order of magnitude by employing a CDO algorithm.

The MBD model defines a standard CDO interface, which can be used to plug in different CDO modules, depending on the type of granular system being simulated. This interface is defined as follows.

First of all, the CDO module must be able to return a list of neighbors of a particular body on a request. These requests are usually done by clusters once they have computed the trajectories of their bodies (after a *predict-trajectory* event has been handled) and are

detecting collisions. Thus, a CDO module must implement a function *GetNeighbors*, which passed a reference to a body and has to return the list of neighbors of the body.

The CDO module also gets notified of dynamic additions and deletions of particles so that it could update its internal data structures. Thus, a CDO module must implement two functions *AddObject(Body)* and *RemoveObject(Body)*, which are getting called by the *System* object once a body is added or removed from the system.

The CDO module needs to keep track of current positions and velocities of particles. Thus, it is notified when a cluster has computed the trajectories of its members for the next timestep. This notification is implemented by the function *TrajectoriesUpdated*, which is passed the list of the bodies, whose trajectories were updated.

Finally, a CDO module may define its own events, which are scheduled into the global event queue. Most CDO algorithms do define their own events, which are usually called *topological events* (because the topology of the system changes when such an event occurs). When the *System* object retrieves a topological event from the event queue, the event is routed to the CDO module, which has to handle it. For this purpose, the CDO module must implement a method called *ProcessEvent*.

Usually, when a topological event occurs, the topology of the neighboring graph changes. This results in new neighbor pairs being created. The CDO algorithm must notify the MBD model that the set of neighbors has been updated. The notification is implemented via an event, which is fired by the CDO module and has to be caught by the *System* object. The event contains a reference to an object and a list of updated neighbors of this object. The *System* object handles the event by detecting the collisions in the newly created neighbor pairs.

5.9. Program architecture

This section describes the internal architecture of the MBD model simulation software. The program was developed in a form of object-oriented library of software components,

representing various components of the model at different abstraction levels. The program was written in object-oriented Pascal, in Borland Delphi 3.02 environment. The executable runs on a Windows 95/98/NT platform.

The MBDS (Multi-Body Dynamics Simulation) library consists of three abstraction layers: the base layer, the model layer and the implementation layer (see Figure 5.40). A particular application must also include the application layer.

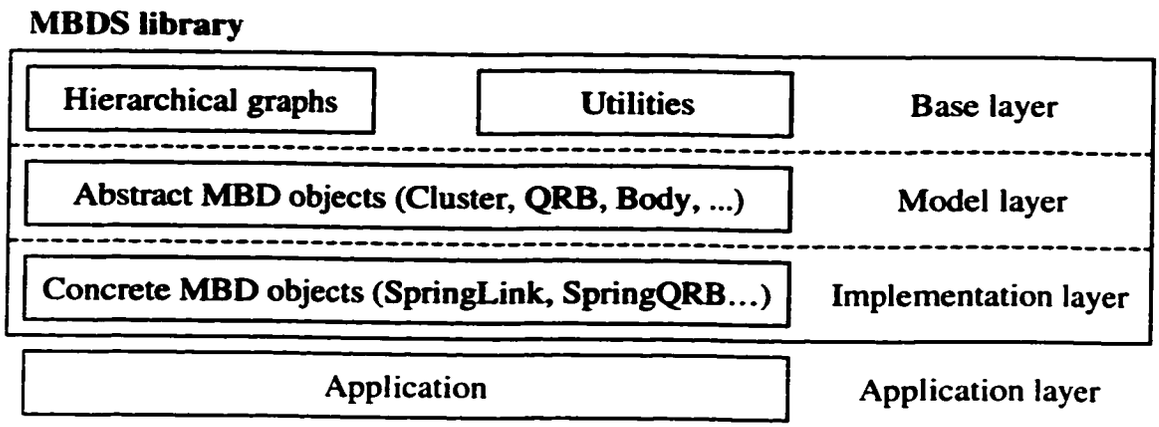


Figure 5.40. Abstraction layers.

The objects at the *base layer* include *SimObject* and *Link* objects, which are used to represent generic hierarchical graphs, as well as several utility objects, such as *Vector* object responsible for vector operations and *IDList* object, which is used to store indexed lists of bodies or links.

The objects at the *model layer* represent various model components found in the MBD model. These objects include *Cluster*, *QRB*, *Body*, *Boundary* and *System*. Note that the objects of this layer do not implement any numerical algorithms (such as a particular integration method used by the *Cluster* object). Rather, they are only concerned about graph-theoretical aspects of MBD algorithms, such as the transformation of clusters, QRB detection and so on. Also, the event processing is not implemented at this layer, i.e. the objects are not tied to any particular event set. This functionality is open for implementation at a lower layer.

The objects at the *implementation layer* implement a particular set of numerical algorithms as well as define a set of events and rules for their processing. Thus, various approaches, such as different integration methods, can be implemented. The library includes an implementation of MBD objects, based on "spring-like" links, i.e. the links represented by elastic elements. This, in particular, determines the system of ODE defining the motion of a cluster, as well as the stiffness matrix of a QRB. Note that a different approach can be implemented without affecting the objects on higher layers. For example, a "hard-body" approach can be implemented, which is based on the Lagrange formulation, which yields an algebraic-differential system of equations (see Section 5.6.2).

The implementation layer also includes the objects representing the "plug-in" components, such as various *ForceField* objects (see Section 5.8.4) and the CDO (Collision Detection Optimizer) implementations (see Section 5.8.5).

Finally, a particular application has to include implementations of various input/output, visualization and logging functions. The *System* object exposes a set of events, which can be handled by the application, such as *visualize*, *log-state*, *log-event*, *new-body* and *delete-bodies* events. The application also is responsible for supplying the initial system configuration, as well as handling the statistics and logging.

5.10. Application – hopper flow

The model of hopper flow was implemented based on the MBD model. A set of numerical experiments was conducted to validate the developed approach.

Hoppers (see Figure 5.41) are used in many industries, including mining, chemical, agricultural and many others. They are commonly used as a device for storage and discharge of granular materials. Hoppers have been a subject of study for many years, starting from a pioneering work by Janssen [Janssen 1895].

It is quite difficult to investigate the internal properties of granular matter in a hopper because this granular system is quite sensitive to the presence of external bodies. Thus, it is only practical to investigate such parameters as the discharge rate, velocities of particles moving inside the hopper or the stresses on the walls.

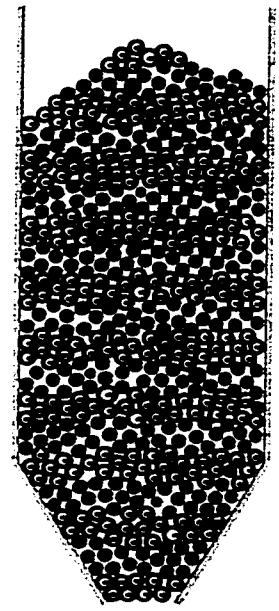


Figure 5.41. A hopper with 500 particles.

Many simulation models have been introduced to investigate the micro-properties of the granular matter in a hopper. Most of the models are based on the continuum approach [Jenike and Shield 59, Pariseau 69, Davidson and Nedderman 73, Brennen and Pearce 78, Jackson 83, Savage 84, Nedderman 92]. Discrete models include cellular-automata based methods [Claudin and Bouchaud 97], molecular-dynamics methods [Ristow and Herrmann 95, Sakaguchi et. al. 93, Hirshfeld et. al. 97, Rong et. al. 97] and the distinct element method [Potapov and Campbell 96].

5.10.1. Problem description

A two-dimensional hopper is simulated. The hopper geometry is defined by the height of the hopper H, the width W, the orifice W_1 and the angle α (see Figure 5.42). The

hopper is loaded with n disks with normally distributed radii. Then, the orifice at the bottom of the hopper is opened and the disks flow down under the influence of the gravitation force.

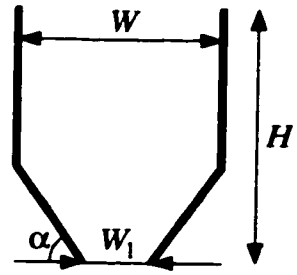


Figure 5.42. Hopper geometry.

The contacts between the disks and hopper walls are modeled by links with friction coefficient μ, normal stiffness K_η, tangential stiffness K_τ, normal damping C_η and tangential damping C_τ.

The Quasi-Static model (see Chapter 4) is used to initially place the disks into the hopper and to compute the micro-displacements that bring the system to an equilibrium state. This configuration is then loaded into the MBD model as an input dataset. The disks start to flow down the orifice. They are removed from the simulation space if their top point is below the orifice. The simulation is stopped once there are no more disks left in the hopper.

5.10.2. Numerical experiments

A series of numerical experiments were performed. The following parameters were used in the simulations: $W = 0.3m$, $H = 0.8m$, $W_1 = 0.08m$, $\alpha = 60°$, planar disk density $\rho = 1190kg/m^2$ and the gravitation constant $g = 9.8m/sec^2$. For a link between two disks the stiffness, damping and friction coefficients were defined as $K_\eta = 1.0\times10^6 Nm^{-1}$, $K_\tau = 1.0\times10^5 Nm^{-1}$, $C_\eta = C_\tau = 1.0\times10^3 N(m/sec)^{-1}$, $\mu = 0.43$. For a link between a disk and a boundary the coefficients were defined as

$K_\eta = 1.5\times10^6 Nm^{-1}$, $K_\tau = 1.5\times10^5 Nm^{-1}$, $C_\eta = C_\tau = 1.5\times10^3 N(m/sec)^{-1}$, $\mu = 0.33$. In the first experiment, a monosized set of 500 disks was used; all disks had the radius of $0.01m$. In the second experiment, a polysized set of 500 disks was used; the radii had mean of $0.01m$ and variance of 5%. The timestep for cluster ODE integration was selected as $10^{-5} sec$. The original configurations are shown in the Figure 5.43. The disks are "dyed" in layers to better illustrate the flow process.

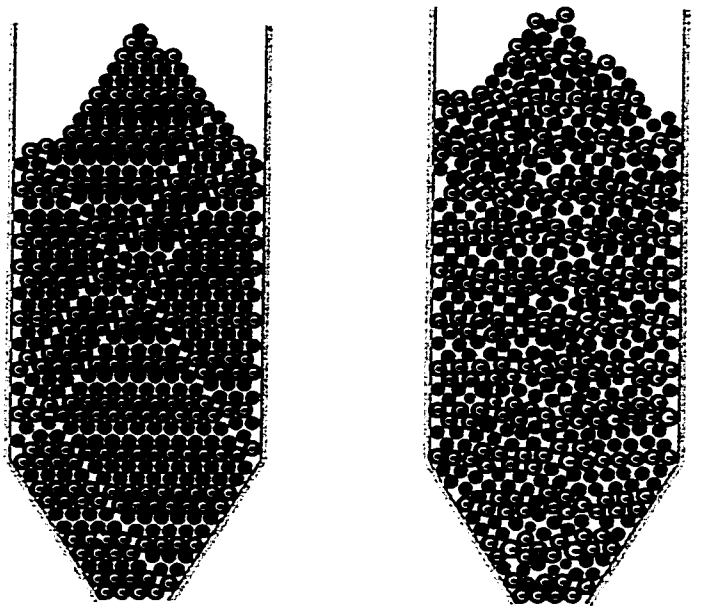


Figure 5.43. Initial configurations for the monosized and polysized datasets.

In both experiments, the total simulation time was set to 5 seconds. The elapsed computation time was about 8 hours in both cases. The experiments were conducted on a PC with a 350 MHz Pentium-II CPU and 128 MB of RAM running Windows NT 4.0.

The full system states were logged every 0.005 seconds of virtual time. A state included the coordinates of each disk (including position, velocity, angular displacement and velocity, the force and moment acting on the disk), the state of every link in the system (including contact point, direction of the link, internal stresses and displacements). A state also included the information about system topology (i.e. the current state of the

hierarchical graph representing the system) as well as energy parameters. The logs were then used to produce real-time animations of hopper flow.

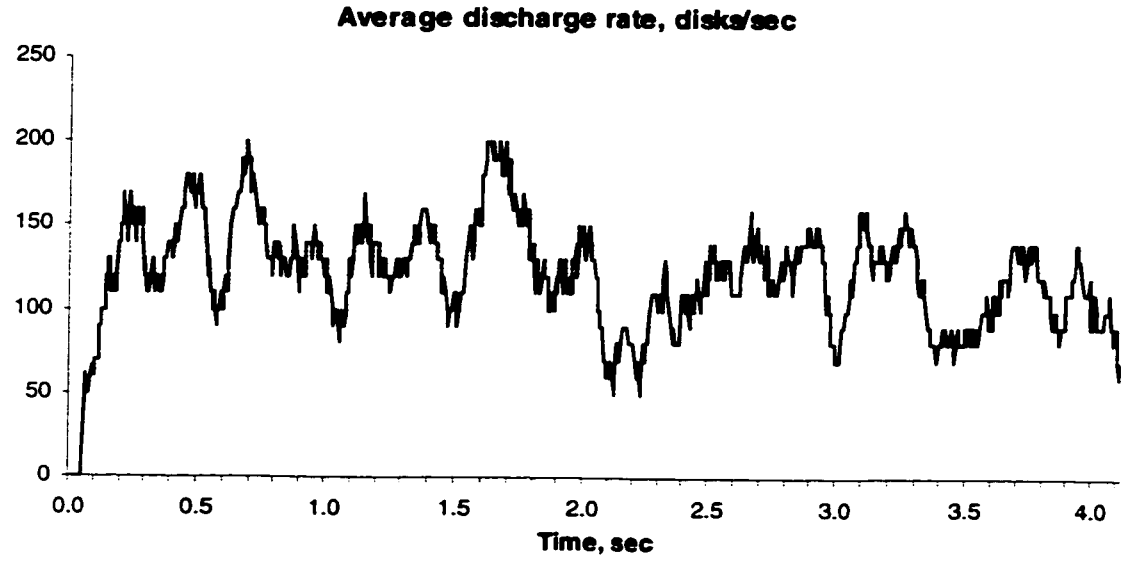


Figure 5.44. Average discharge rate, monosized configuration.

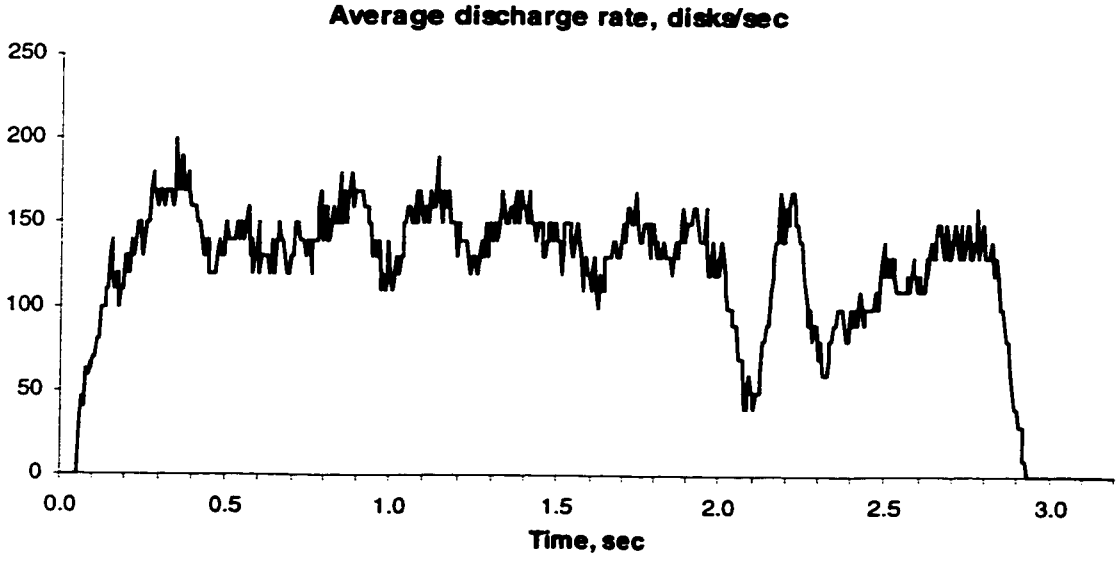


Figure 5.45. Average discharge rate, polysized configuration.

The average discharge rate graphs for the monosized and polysized hoppers are presented in the Figure 5.44 and Figure 5.45, respectively. The value for the discharge rate was

computed as a moving average over a time interval of $0.1sec$. The average discharge rate stayed approximately in 100-200 range for both configurations.

As it can be clearly seen from the figures, the flow rate was not uniform in both cases. An oscillating pattern is seen in both graphs. Also, "near-jams" were experienced at around $2.1sec$ in both cases. While the monosized configuration managed to flow out of the hopper entirely, the polysized configuration jammed completely at approximately $2.8sec$. The jam occurred when the disks formed an "arc" across the orifice (see Figure 5.46).

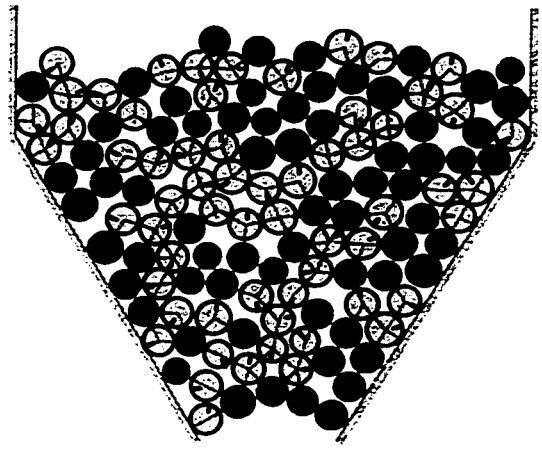


Figure 5.46. Jam in the polysized configuration.

The next two figures show the effective event rate occurring in the simulation. As can be seen from the figures, a very large amount of events is being processed, which explains why it takes so long to do a simulation run. Note that the total number of events processed was about 14 million for the monosized hopper and over 12 million for the polysized hopper. The oscillating pattern of the flow also appears in the event rate graphs. Some increases in the discharge rate correspond to the increases in the event rate (consider, for example, the humps in the monosized graphs at 0.4, 0.7 and 1.7 seconds). Apparently, as the discharge rate increases, the disks are moving faster, and more topological changes take place, which result in an increased event rate. However, some event rate increases (such as the $1.5sec$ one in the polysized configuration) do not correspond to changes in the discharge rate.

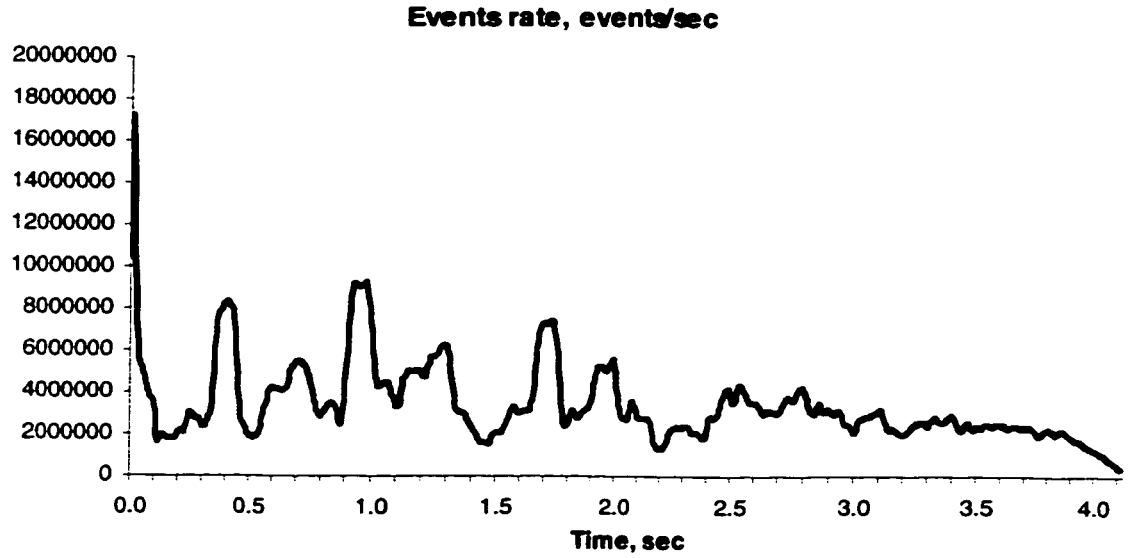


Figure 5.47. Effective event rate, monosized configuration.

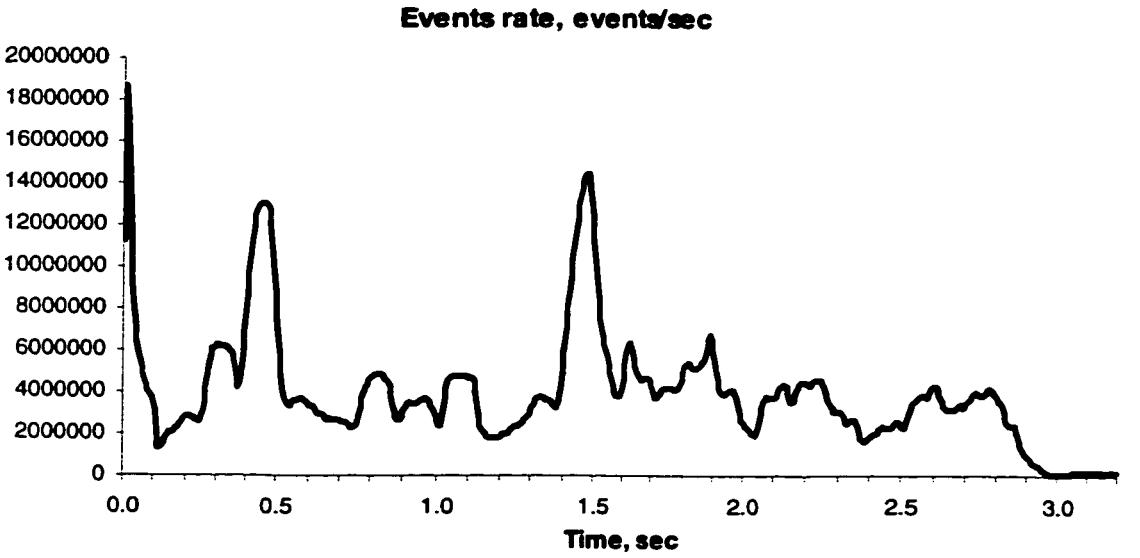


Figure 5.48. Effective event rate, polysized configuration.

The following two figures show the event processing rate, or, in other words, how fast the program was processing events. Unlike the previous two figures, which used the virtual time for computing the rate, these two figures are related to the actual elapsed time.

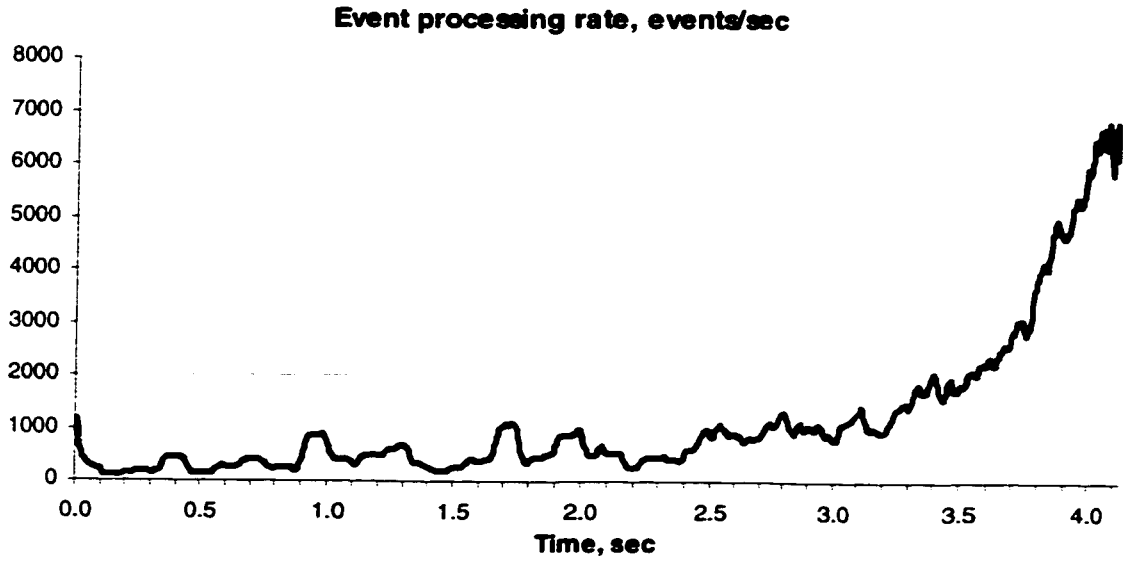


Figure 5.49. Event processing rate, monosized configuration.

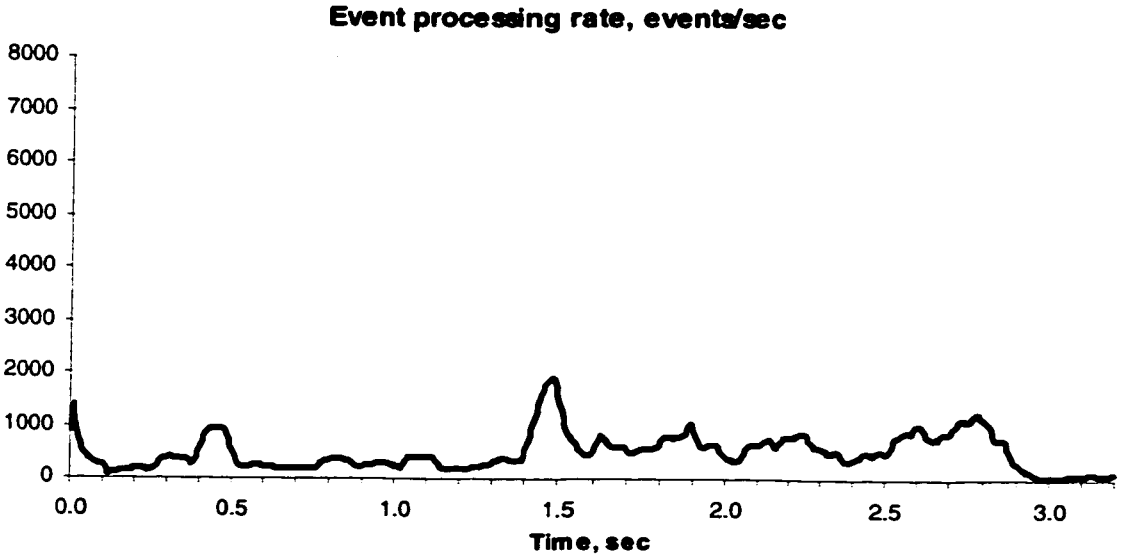


Figure 5.50. Event processing rate, polysized configuration.

As it can be seen from the figures, the event processing rate was fluctuating in a pattern almost identical to that of the effective event rate. Near the end of the simulation, the event processing rate of the monosized configuration significantly increased. The sudden increase in the event processing rate corresponds to the stage when the hopper only

contains disks in its lower part (see Figure 5.51, monosized configuration at $t = 3.53$). In this configuration, most disks move independently from others and only small clusters are formed. This speeds up the simulation significantly.

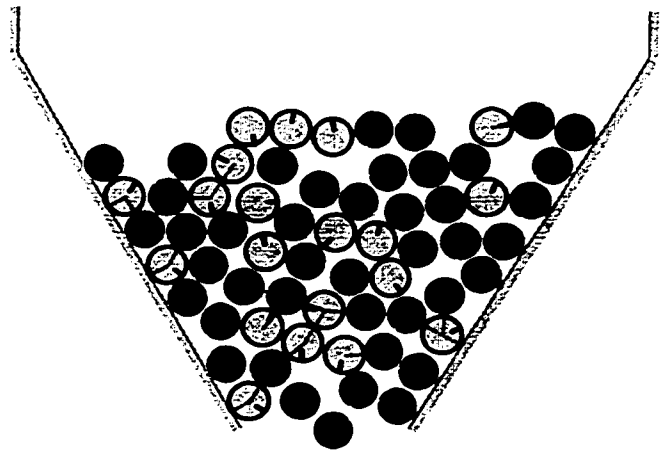


Figure 5.51. Last stages of the simulation, monosized configuration.

In the polysized configuration, at the end of the simulation the event processing rate is considerably decreased. This corresponds to the formation of a jam (see Figure 5.46). The jammed system is represented by a single QRB. Since a QRB does not have to schedule frequent *predict-trajectory* events (as opposed to a cluster), most event processed by the system were *log-state* and *visualize* events, which are quite expensive in terms of time. This explains the drop in the event-processing rate for the jammed polysized configuration.

Finally, Figure 5.52 and Figure 5.53 show the simulation efficiency, measured as the ratio between the elapsed time and the virtual time. Note the logarithmic scale of the vertical axis. Clearly, the efficiency is lowest at the beginning of the simulation, when the total number of disks in the system is the largest. It starts as low as 0.0001, i.e., 10000 seconds of computation time is required to process a single second of simulation.

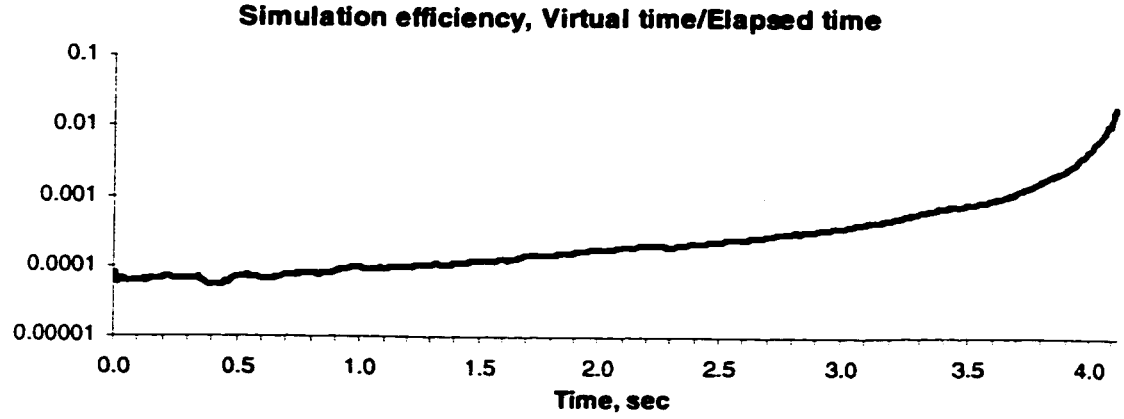


Figure 5.52. Simulation efficiency, monosized configuration.

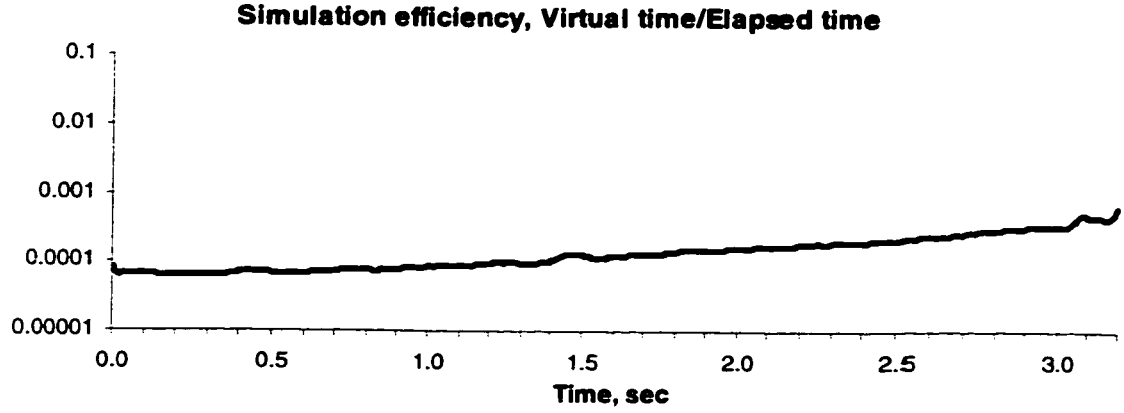


Figure 5.53. Simulation efficiency, polysized configuration.

Also note that the effective event rate is at its highest at the beginning of the simulation. As the amount of disks in the system decreases and the effective event rate decreases, the simulation efficiency increases. In fact, the efficiency can reach 1 or even exceed it (i.e., it can run in real-time or faster) when the number of disks in the system is small and there is no overhead related to visualizing the system and logging the state. In these experiments, the system state was logged 200 times a second, plus the system was visualized 200 times a second, which affected the maximum achievable efficiency of the simulation. Note that a too detailed visualization rate or log-state rate can bring the simulation efficiency down.

5.10.3. Validation

To validate the simulation, the energy conservation law was checked. The "energy flow" in the system is shown in the following figures.

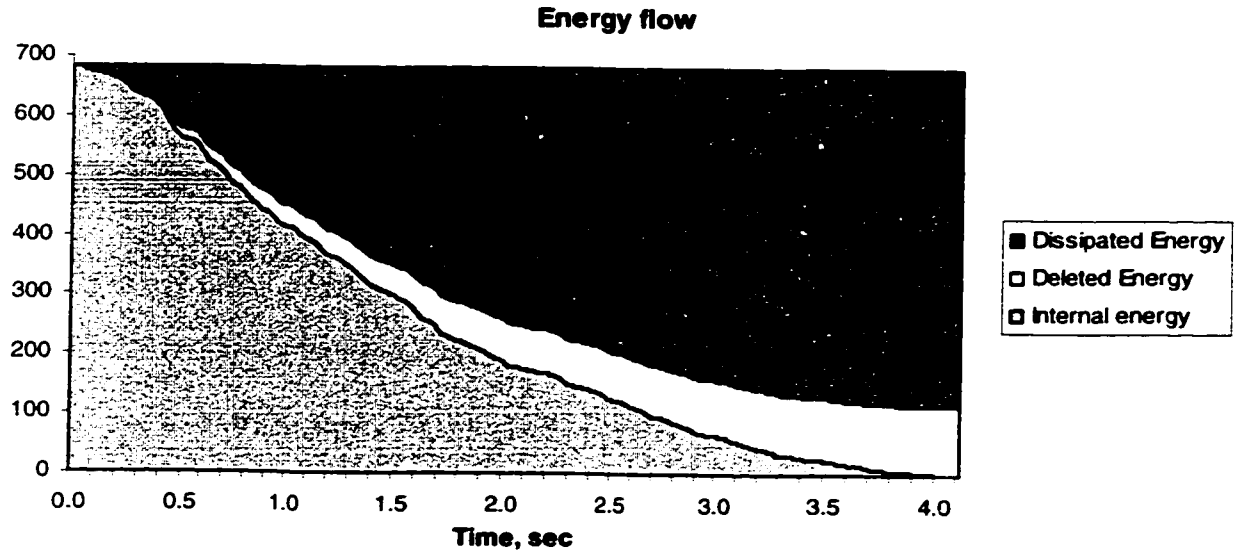


Figure 5.54. Energy flow, monosized configuration.

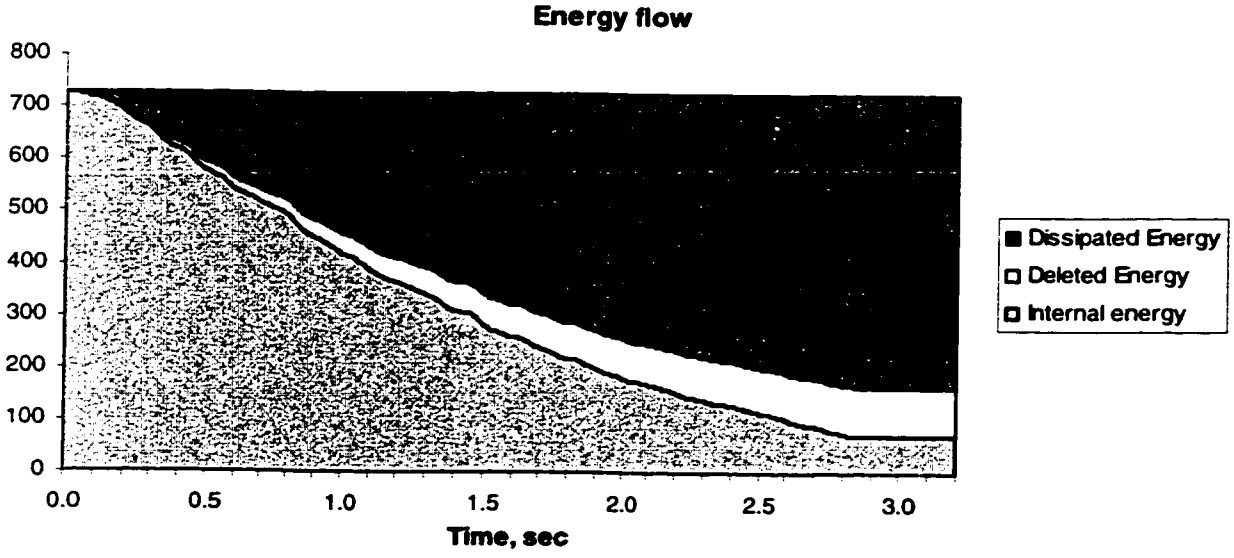


Figure 5.55. Energy flow, polysized configuration.

The figures illustrate how the energy is redistributed in the system during the simulation run. Three main components of energy value are the internal energy of the system, the energy of deleted bodies, and the dissipated energy. The internal energy of the system consists of the kinetic energy of disks, their potential energy, and the potential energy of compressed links. The deleted bodies energy consists of the kinetic energy of the disks that were removed from the system (note that the snapshot of their state is made at the time of their removal). The dissipated energy consists of the energy spent during slips in links, as well as the energy dissipated due to damping in links.

Another validating result is related to the patterns of discharge in planar hoppers. It has been reported in both experimental papers [Blair-Fish and Bransby 73, Lee et. al. 74] and computer simulations [Potapov and Campbell 96] that hoppers exhibit an interesting pattern of discharge, when the flow alternates from the left and right sides of the hopper. A similar pattern was observed in our simulations. Note that [Potapov and Campbell 96] in their DEM simulation only observed the phenomenon in the monosized hopper, while in our simulations it was clearly seen in both monosized and polysized hoppers.

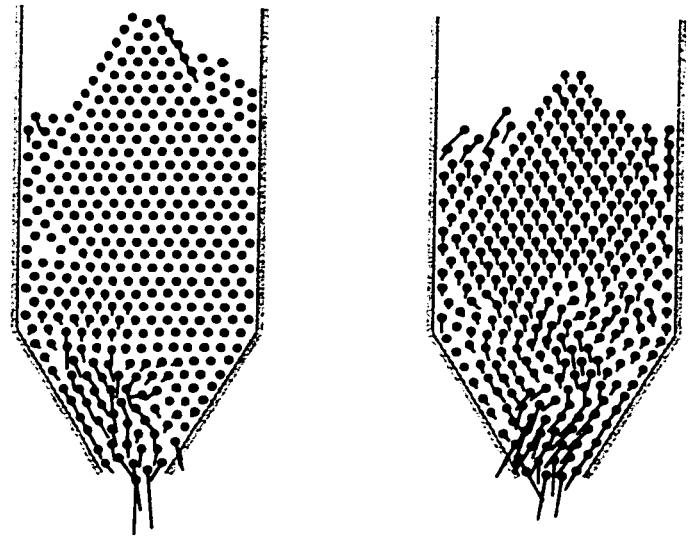


Figure 5.56. Alternating flow, monosized configuration.

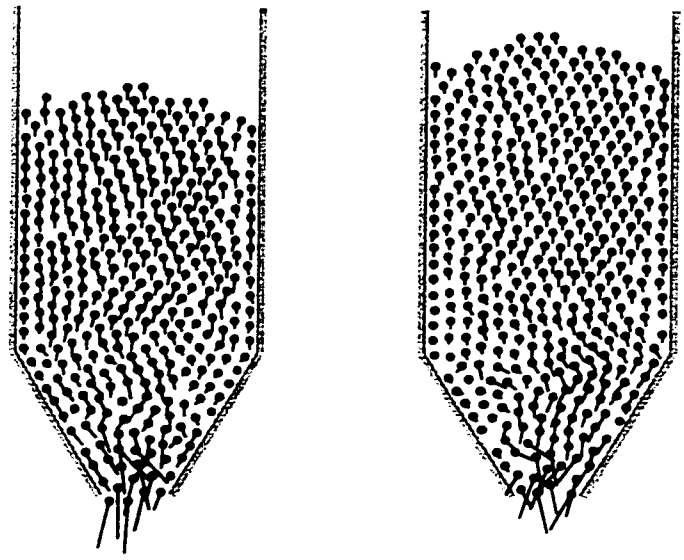


Figure 5.57. Alternating flow, polysized configuration.

In Figure 5.56 and Figure 5.57, the instantaneous velocity fields for disks in a hopper are presented. The flow paths can be clearly seen.

5.11. Conclusions

The Multi-Body Dynamics (MBD) model was presented in this chapter. This model is the most complete model and it can be used to simulate any granular system. The MBD model comprises both MD (Molecular Dynamics) and QS (Quasi-Static) models as its submodels. Switching between models is performed automatically based on the current properties of the system.

The MBD model uses the notion of hierarchical graph, which was introduced together with some basic operations for transforming hierarchical graph. The core objects of the MBD model – *Body*, *Cluster*, *QRB* and *System* – were then discussed in detail. The general architecture of the object-oriented implementation of the model was discussed next. Finally, the application of the model was demonstrated on the hopper flow example.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK DIRECTIONS

A unique methodology for simulating granular systems has been developed and presented in this thesis. The currently available approaches to granulate materials simulations are limited in their applicability domains or are being misused. The main property of the developed methodology is that the simulation model contains three submodels with different applicability limits. The model dynamically adjusts to the system being simulated and switches from one submodel to another based on the current properties of the system.

The core of the model is the MBD (Multi-Body Dynamics) simulation approach, which was originally developed by Vinogradov [Vinogradov 92a, Vinogradov 92b, Vinogradov 93, Vinogradov and Sun 97]. This approach provides a numerically stable alternative to the popular DEM (Distinct Element Method) approach [Cundall and Strack 79]. Unlike the DEM, the MBD method provides explicit means of controlling the numerical error, as well as allowing the implementation specify the most appropriate integration method.

In this research, the MBD model was extended with a comprehensive link model, which introduced rotational parameters to equations of motion and provided a better means of modeling friction. Also, an extensible software object model was developed and implemented for the MBD approach.

The MBD model was extended with two additional submodels: the Molecular Dynamics model, which is used to simulate high energy systems, and the Quasi-Static Model, which is best suited for simulating static or nearly-static systems. Each of these models can be used as a stand-alone simulation model, with certain applicability limits.

The stand-alone MD model was implemented and used for simulating shaker ball mills. In addition to the basic MD model, two models, specific to the milling application, were developed. They were used to represent milling properties of a shaker-ball mill. The use of the developed application increased the milling efficiency of an experimental shaker ball mill by an order of magnitude.

A unique model for simulating quasi-static granular systems was developed. It is based on the original idea by Vinogradov [Vinogradov 86b], which was extended and applied to the incremental construction of the inverse stiffness matrix for a granular system. The Recursive Inverse Matrix Algorithm (RIMA) [Gavrilov and Vinogradov 97a] was developed for dynamic maintenance of this matrix. A special algorithm dealing with micro-topological changes occurring in the granular system as it is being built was developed. Thus, the process of filling of a silo was modeled in an efficient manner while providing an accurate tool for measuring the stress field in the silo and for investigating the topological properties of the granular system (such as micro-avalanches and slides). With the application of the QS model to simulating the silo filling process, it has been shown that the granular material in a silo possesses many interesting properties, such as that the stress field in the silo is heavily dependent on the history of filling. Indeed, the pressures on the silo walls can vary in a quite large range for two consecutive fillings of a silo. The developed model also was used to investigate the self-organized criticality (SOC) properties of the granular matter in a silo.

Finally, the three models were integrated into a unified simulation environment, which adjusts to the system being simulated and selects the most appropriate model depending on the system's parameters. Moreover, various parts of the system can be simulated using different models. For example, in a sand pile simulation, the topmost layer of the pile, which is subjected to wind drag, would be simulated with the MD model; the core of the pile, which is relatively stable, would be simulated using the QS model; and the transient layer near to the surface of the pile would be simulated using the MBD model. To the best of author's knowledge, this represents the most flexible granulates simulation model currently available. Also, the model can dynamically adjust the accuracy of the simulation and employ some strong error-control mechanisms. This makes the developed model an invaluable tool for measuring the performance of other models.

The following can be outlined as the future directions of research. One area, which has yet to be investigated thoroughly, is the conditions that control the switching between

models. It is not very clear how to select the parameters that are used for determining the stability of a link (this ultimately controls the switching between MBD and QS models). Similarly, it is not very clear how to determine whether a collision between two objects can be considered using the MD model (i.e. as an instantaneous collision) or the MBD model (i.e. the dynamics of the collision is taken into consideration).

Another very interesting area of research is the study of the properties of nearly-static granular systems, such as sand piles or silos. Various results are being reported regarding whether such systems can be classified as self-organized criticality systems. The most currently available tools for simulating such systems are very basic (which perhaps explains the range of the reported results). The QS model is perfectly suited for simulating this class of granular systems and can provide valuable insights into the physics and mechanics of these systems.

Other directions of research may include investigating various integration methods in application to MBD simulations and utilizing different iterative and error-control schemas and comparing their performance. Another very important area of research is developing effective collision-detection optimization (CDO) algorithms and determining their applicability limits and perhaps developing a dynamic approach, which would switch between the CDO methods depending on the current conditions (similarly to the way the switching between the three simulation models is done).

REFERENCES

Baezner, D., Lomow, G. and Unger, B. 1994. "Parallel Simulation Environment Based on Time Warp," *International Journal in Computer Simulation*, vol. 4, no. 2, pp. 183-193.

Bak, P., Tang, C. and Wiesenfeld, K., 1988. "Self-organized criticality", *Physical Review A*, vol.38, no.1, pp. 364-374.

Bazant, Z.P. and Ozbolt, J., 1990. "Nonlocal Microplane Model for Fracture, Damage, and Size Effect in Structures," *Journal of Engineering Mechanics*, vol. 116, no. 11, pp. 2485-2505.

Benjamin, J.S., 1970. "Dispersion Strengthened Superalloys by Mechanical Alloying", *Metallurgical Transactions*, vol. 1, pp. 2943-2951.

Bickford, W.B., 1994. *Finite Element Methods*, Irwin Publishing, Homewood, IL.

Blair-Fish, P.M. and Bransby, P.L., 1973. "Flow Patterns and Wall Stresses in a Mass-Flow Bunker," *Journal of Engineering for Industry*, vol. 95, pp. 17-25.

Bouchaud, J.-P. and Cates, M. E., 1998. "Triangular and uphill avalanches of a tilted sandpile," *Granular Matter*, vol. 1, no. 2, pp. 101-103.

Bouchaud, J.-P., Cates, M.E. and Claudin, P., 1995. "Stress Distribution in Granular Media and Nonlinear Wave Equation." *Journal de Physique I (France)*, vol. 5, no. 6, pp. 639-656.

Bouchaud, J.-P., Claudin, P., Cates, M.E. and Wittmer, J.P. 1998. "Models of Stress Propagation in Granular Media," in *Physics of Dry Granular Media*, eds. Herrmann, H.J., Hovi, J.P. and Luding, S., NATO Advanced Study Institute, Kluwer, pp. 97-122.

Brennen, C. and Pearce, J.C., 1978. "Granular Material Flow in Two-dimensional Hoppers," *Journal of Applied Mechanics*, vol. 43, pp. 43-58.

Briscoe, B.J., Luckham, P.F. and Ren, S.R., 1993. "Settling of Spheres in Clay Suspensions," *Powder Technology*, vol. 76, no. 2, pp. 165-174.

Brown, R.L. and Richard, J.C., 1966. *Principles of Powder Mechanics*, Pergamon, New York, NY.

Burden, R.L., Faires, J.D. and Reynolds, A.C., 1978. *Numerical Analysis*, Prindle, Weber and Schmidt.

Carreras, B. A., Newman, D. and Diamond, P. H., 1996. "Self-Organized Criticality as a Paradigm for Transport in Magnetically Confined Plasmas." *Plasma physics reports*, vol. 22, no. 9, pp. 740-751.

Chen, J.F., Ooi, J.Y. and Rotter J.M., 1996. "A Rigorous Statistical Technique for Inferring Circular Silo Wall Pressures from Wall Strain Measurements." *Engineering Structures*, vol. 18, no. 4, pp. 231-331.

Claudin, P. and Bouchaud, J.-P., 1997. "Static Avalanches and Giant Stress Fluctuations in Silos." *Physical Review Letters*, vol. 78, no. 2, pp. 231-234.

Claudin, P. and Bouchaud, J.-P., 1998. "Stick-slip transition in the Scalar Arching Model," *Granular Matter*, vol. 1, no. 2, pp. 71-74.

Connelly, L.M., 1983. "Wall–Pressure and Material–Velocity Measurements for the Flow of Granular Material under Plane–Strain Conditions." In *Mechanics Applied to the Transport of Bulk Materials*, ASME, pp. 35-59.

Cundall, P.A. and Strack, O.D.L., 1979. "A Discrete Numerical Model for Granular Assemblies," *Geotechnique*, vol. 29, pp. 47-65.

Davidson, J.K. and Nedderman, R.M., 1973. "The Hour-Glass Theory of Hopper Flow," Transactions of the Institute of Chemical Engineering, vol. 29, pp. 29-39.

Dendy, R.O. and Helander, P., 1997. "Sandpiles, Silos and Tokamak Phenomenology: a Brief Review." *Plasma Physics and Controlled Fusion*, vol. 39, no. 12, pp. 1947-1961.

Drake, T.G., 1990. "Structural Features in Granular Flows," *Journal of Geophysical Research*, vol. 95, no. B6, pp. 8681-8696.

Drescher, A. and de Josselin de Jong, G. 1972. "Photoelastic Verification of a Mechanical Model for the Flow of a Granular Material," *Journal of Mechanics and Physics of Solids*, vol. 20, pp. 337-351.

Eibl, J., 1984. "Design of Silos – Pressures and Explosions." *The Structural Engineer*, vol. 62A, pp. 169-175.

Fuji, M., Ueno, S., Takei, T., Watanabe, T. and Chikazawa, M., 1998. "Conformation study of normal alkoxy groups introduced on to a silica surface," *Advanced Powder Technology*, vol. 9, no. 3, pp. 261-272.

Gavrilov, D. and Vinogradov O., 1994. "Object-Oriented Library for Simulation of Granular-Type Materials," in *Proceedings of the 1994 Summer Computer Simulation Conference*, San Diego, CA., pp. 51-56.

Gavrilov, D. and Vinogradov, O., 1995. "Object-Oriented Programming and Object-Oriented Simulation," in *Proceedings of 1995 Summer Computer Simulation Conference*, Ottawa, ON, 1995, pp.143-148.

Gavrilov, D. and Vinogradov, O., 1996. "On Analysis of Discrete Systems With Variable Structure," in *Recent advances in solids/structures and application of metallic materials*, ASME, eds. Kwon, Y. W., Davis, D. C. and Chung, H. H., pp. 63-66.

Gavrilov, D. and Vinogradov, O., 1997a. "Recursive Inverse Matrix Algorithm in Granular Mechanics Applications," *Computational Mechanics*, vol. 20, pp. 407-411.

Gavrilov, D. and Vinogradov, O., 1997b. "A Cluster in Granular Systems as a Topologically Variable Structure," in *Mechanics of deformation and flow of particulate materials*, ASME, ed. Chang, C.S., pp. 299-307.

Gavrilov, D. and Vinogradov, O., 1998a. "Simulation of Grinding of Particles in a Shaker Ball Mill," in *Engineering Mechanics: A Force for the 21st Century*, eds. Murakami, H. and Luco, J.E., ASCE, pp. 1716-1719.

Gavrilov, D. and Vinogradov, O., 1998b. "Micro Instabilities in a System of Particles in Silos During Filling Process," in *Recent advances in solids and structures*, ASME, vol. 381, pp. 47-54.

Gavrilov, D. and Vinogradov, O., 1999. "Micro Instabilities in a System of Particles in Silos during Filling Process," *Computer Modeling and Simulation in Engineering*, in press.

Gavrilov, D., Vinogradov, O. and Shaw, W.J.D., 1995. "Computer Simulation of Mechanical Alloying in a Shaker Ball Mill." In *Proceedings of the 10th International Conference on Composite Materials*, Whistler, BC, 1995, pp. 11-17.

Gavrilov, D., Vinogradov, O. and Shaw, W.J.D., 1997. "Simulation of Mechanical Alloying in a Shaker Ball Mill with Variable Size Particles," in *Proceedings of 11th International conference on composite materials*, Gold Coast, Australia, vol. 4, pp. 370-378.

Gavrilov, D., Vinogradov, O. and Shaw, W.J.D., 1999. "Simulation of Grinding in a Shaker Ball Mill," *Powder Technology*, vol. 101, no. 1, pp. 63-72.

Gavrilova, M., 1998. *Proximity and Applications*, Ph.D. dissertation, University of Calgary, Calgary, AB.

Gera, D., Gautam, M., Tsuji, Y., Kawaguchi, T. and Tanaka, T., 1998. "Computer Simulation of Bubbles in Large-Particle Fluidized Beds," *Powder Technology*, vol. 98, no. 1, pp. 38-47.

Goles, E., Gonzalez, G., Herrmann, H. and Martrinez, S., 1998. "Simple Lattice Model with Inertia for Sand Piles," *Granular Matter*, vol. 1, no. 3, pp. 137-140.

Gropp, W., Lusk, E. and Skjellum, A., 1994. *Using MPI*, The MIT Press.

Gudehus, G., 1996. "Constituive Relations for Granulate-Liquid Mixtures with a Pectic Constituent," *Mechanics of Materials*, vol. 22, pp. 93-103.

Gustafson L. and Gustafson, P., 1996. "Studying Mixed Granular Flows by Image Analysis," in *Proceedings of the 11th Conference on Engineering Mechanics*, Fort Lauderdale, FL, vol. 1, pp. 100-103.

Herrmann, H. J., Hovi, J.-P. and Luding, S., 1998. *Physics of dry granular media*, NATO ASI Series, Vol. E 350, Kluwer Academic Publishers, Dordrecht.

Hirshfeld, D., Padzyner, Y and Rapaport, D.C., 1997. "Molecular Dynamics Studies of Granular Flow through an Aperture," *Physical Review E*, vol. 56, no. 4, pp. 4404-4415.

Jackson, R., 1983. "Some Mathematical and Physical Alpects of Continuum Models for the Motion of Granular Materials," in *Theory of Dispersed Multiphase Flow*, ed. Meyer, R.E., Academic Press, N.Y.

Janssen, H.A., 1895. "Versuche ueber Getreidedruck in Silozellen," *Zeitschr. Vereines deutscher Ingenieure*, vol. 39, no. 35, pp. 1045-1049.

Jarrett, N.D., Brown, C.J. and Moore, D.B., 1996. "Pressure Measurements in a Rectangular Silo," *Geotechnique*, vol. 45, no. 1, pp. 95-104.

Jenike, A.W. and Shield, R.T., 1959. "On the Plastic Flow of Coulomb Solids beyond Failure," *Journal of Applied Mechanics*, vol. 26, pp. 599-608.

Kano, J. and Saito, F., 1998. "Correlation of Powder Characteristics of Talc during Planetary Ball Milling with the Impact Energy of the Balls Simulated by the Particle Element Method,"

Karlsson, T., Klisinski, M. and Runesson, K., 1998. "Finite Element Simulation of Granular Material Flow in Plane Silos with Complicated Geometry," *Powder Technology*, vol. 99, no. 1, pp. 29-39.

Kawaguchi, T., Tanaka, T. and Tsuji, Y., 1998. "Numerical Simulation of Two-dimensional Fluidized Beds Using the Discrete Element Method (Comparison Between the Two- and Three-dimensional Models)," *Powder Technology*, vol. 96, no. 2, pp. 129-138.

Ktitarev, D.V. and Wolf, D.E., 1998. "Stratification of Granular Matter in a Rotating Drum: Cellular Automaton Modelling," *Granular Matter*, vol. 1, no. 3, pp. 141-144.

Lee, J., Cowin, S.C and Templeton, J.S. III, 1974. "An Expermiental Study of the Kinematics of Low through Hoppers," *Transactions of Society of Rheology*, vol. 18, pp. 247-257.

Li, E. and Bagster, D.F., 1993. "Idealized Three-dimensional Model of Heaped Granular Materials," *Powder Technology*, vol. 74, no. 3, pp. 271-278.

Liu, C.-H. and Nagel, S.R., 1992. "Sound in Sand." *Physical Review Letters*, vol. 68, no. 15, pp. 2301-2304.

Luding, S. and McNamara, S., 1998. "How to Handle the Inelastic Collapse of a Dissipative Hard-Sphere Gas with the TC Model," *Granular Matter*, vol. 1, no. 3, pp. 113-128.

Matuttis, H.-G. and Schinner, A., 1999. "Influence of the Geometry on the Pressure Distribution of Granular Heaps," *Granular Matter*, vol. 1, no. 4, pp. 195-201.

McCarthy, J.J. and Ottino, J.M., 1998. "Particle Dynamics Simulation: a Hybrid Technique Applied to Granular Mixing," *Powder Technology*, vol. 97, no. 2, pp. 91-99.

Meng, Q.G., Jofriet, J.C. and Negi. S.C., 1997. "Finite Element Analysis of Bulk Solids Flow. 1. Development of a Model Based on a Secant Constitutive Relationship." *Journal of Agricultural Engineering Research*, vol. 67, no. 2, pp. 141-150.

Milling of Brittle and Ductile Materials, 1984. *Metals Handbook*, Ninth Edition, Powder Metallurgy, vol. 7, ASM, 1984, pp. 57-70.

Mindlin, B.R. and Deresiewicz, H., 1953. "Elastic Spheres in Contact under Varying Oblique Forces," *Journal of Applied Mechanics*, vol. 20, no. 1, pp. 327-344.

Nedderman, R.M., 1992. *Statics and Kinematics of Granular Materials*, Cambridge University Press, Cambridge.

Nuebel, K. and Karcher, C., 1998. "FE Simulations of Granular Material with a Given Frequency Distribution of Voids as Initial Condition," *Granular Matter*, vol. 1, no. 3, pp. 105-112.

Ooi, J.Y. and She, K.M., 1997. "Finite Element Analysis of Wall Pressure in Imperfect Silos." *International Journal of Solids and Structures*, vol. 34, no. 16, pp. 2061-2072.

Pariseau, W.G., 1969. "Gravity Flows of Ideally Plastic Materials through Slots," *Transactions of ASME*, vol. 91, pp. 414-421.

Potapov, A.V. and Campbell, C.S., 1996. "Computer Simulation of Hopper Flow," *Physics of Fluids*, vol. 8, no. 11, pp. 2884-2894.

Potapov, A.V. and Campbell, C.S., 1998. "A Fast Model for the Simulation of Non-Round Particles," *Granular Matter*, vol. 1, no. 1, pp. 9-14.

Ragneau, E. and Aribert, J.M., 1993. "Analytical Solutions for the Prediction of Loads in Silos during Filling and Emptying Stages." In *Powder and Grains 93*, Balkema, Rotterdam, pp. 469-475.

Rapaport, D.C., 1995. *The Art of Molecular Dynamics Simulation*, Cambridge University Press.

Ristow, G.H. and Herrmann, H.J., 1995. "Forces on the Walls and Stagnation Zones in a Hopper Filled with Granular Material." *Physica A*, vol. 213, no. 4, pp. 474-481.

Rong, J., Ooi, J.Y. and Rotter, J.M., 1997. "Discrete Element Modeling of Particulate Solids in Silos," in *Mechanics of Deformation and Flow of Particulate Materials*, eds. Chang, C.S., Misra, A., Liang, R.Y. and Babic, M., ASCE, pp. 321-334.

Rotter, J.M., Rong, G.H., Ooi, J.Y. and Holst, J.M.F.G., 1995. "CA-SILO Collaborative Action: WG5. Comparative evaluation of numerical methods for predicting flow and stress fields in silos," http://www.civ.ed.ac.uk/research/silo/demfem/

Sakaguchi, H., Ozaki, E. and Igarashi, T., 1993. "Plugging of the Flow of Granular Materials during the Discharge from a Silo." *International Journal of Modern Physics*, vol. 7, no. 9-10, pp. 1949-1963.

Savage, S.B., 1984. "The mechanics of rapid granular flows," *Advanced Applied Mechanics*, vol. 24, pp. 289-304.

Schwedes, J. and Feise, H., 1995. "Modelling of Pressures and Flow in Silos," *Chemical Engineering and Technology*, vol. 18, no. 2, pp. 96-109.

Shaw, W.J.D., Pan J. and Gowler, M.A., 1993. "Property Relationships of Some New MA Polymers," in *Proceedings of the 2nd International Conference on Structural Applications of Mechanical Alloying*, Vancouver, B.C., Canada, September 1993, pp. 431-437.

Shigley, J.E. and Uicker J.J., Jr., 1995. *Theory of Machines and Mechanisms*, 2nd ed., McGraw-Hill.

Stadler, J., Mikulla, R. and Trebin, H.-R., 1997. "IMD: A Software Package for Molecular Dynamics Studies on Parallel Computers," *International Journal of Modern Physics*, vol. 8, pp. 1131-1140.

Streletskii, A.N., 1993. "Measurements and Calculation of Main Parameters of Powder Mechanical Treatment in Different Mills," in *Proceedings of the 2nd International Conference on Structural Application of Mechanical Alloying*, Vancouver, B.C., September 20-22, 1993, pp. 51-58.

Sun, Y., Vinogradov, O., Gavrilova, M. and Rokne, J., 1994. "An Algorithm of Updating System State in Simulation of Dynamics of Granular-Type Materials," In *Proceedings of 1994 Summer Computer Simulation Conference*, pp. 45-50.

Takeuchi, N. and Kawai, T., 1988. "A Discrete Limit Analysis of Granular Materials Including Effects of the Solid Contact," *Micromechanics of Granular Materials*, eds. Satake, M. and Jenkins, J.T., Elsevier, pp. 103-112.

Tejchman, J., 1998. "Numerical Simulation of Filling in Silos with a Polar Hypoplastic Constitutive Model," *Powder Technology*, vol. 96, no. 3, pp. 227-239.

Tejchman, J. and Gudehus, G., 1993. "Silo Music and Silo-quake Experiments and a Numerical Cosserat Approach," *Powder Technology*, vol. 76, pp. 201-209.

Timoshenko, S. and Goodier, J.N., 1951. *Theory of Elasticity*, McGraw-Hill, N.Y. p.372.

Vinogradov, O.G., 1986a. "Simulation Methodology for a Flow of Interacting Floes Around an Obstacle," *International Journal of Modeling and Simulation*, vol. 7, no. 1, pp. 28-31.

Vinogradov, O.G., 1986b. "Algorithm of Stiffness Matrix Inversion Based on Substructuring Concept," *Computers and Structures*, vol. 22, no. 3, pp. 253-259.

Vinogradov, O.G., 1992a. "Explicit Equations of Motion of a Discrete System of Disks in 2-D," *Journal of Engineering Mechanics*, ASCE, vol. 118, no. 9, pp. 1850-1858.

Vinogradov, O.G., 1992b. "Explicit Equations of Motion of Interacting Spherical Particles," *Recent Advances in Structural Mechanics*, PVP – vol. 248, eds. Kwon, Y.W. and Chung, H.H., book no. G00775, pp. 111-115.

Vinogradov, O.G., 1993. "Dynamic Equations for System of Irregularly Shaped Plane Bodies," *Journal of Engineering Mechanics*, ASCE, vol. 119, no. 11, pp. 2226-2237.

Vinogradov, O.G., 1999. "On the Accuracy and Efficiency of Simulation of an Impact Problem in Granular Mechanics," *Computer Modeling and Simulation in Engineering*, in press.

Vinogradov, O. and Springer, A., 1990. "Simulation of Motion of Multibody System with Interactions," in *Proceedings of 1990 Summer Computer Simulation Conference*, Calgary, AB, pp. 51-55.

Vinogradov, O.G. and Sun, Y., 1997. "A Multibody Approach in Granular Dynamics Simulations," *Computational Mechanics*, vol. 19, no. 4, pp. 287-296.

Wan, R.G. and Guo, P.J., 1998. "Simple Constitutive Model for Granular Soils: Modified Stress-Dilatancy Approach," *Computers and Geotechnics*, vol. 22, no. 2, pp. 109-133.

Wan, R.G., Chan, D.H. and Morgenstern, N.R., 1990. "Finite element method for the analysis of shear bands in geomaterials," *Finite Elements in Analysis and Design*, vol. 7, no. 2, pp. 129-143.

Wang, Y. and Hutter, K., 1999. "A Constitutive Model of Multiphase Mixtures and Its Application in Shearing Flows of Saturated Solid-Fluid Mixtures," *Granular Matter*, vol. 1, no. 4, pp. 163-181.

Wierzba, P. and Vinogradov, O.G., 1991. "Simulation of Topologically Variable Multibody System in Plane Motion," in *Proceedings of 1991 European Simulation Multiconference on Modeling and Simulation*, SCS, San Diego, CA, pp. 935-940.

Wittmer, J.P., Cates, M.E. and Claudin, P., 1997. "Stress Propagation and Arching in Static Sandpiles," *Journal de Physique I (France)*, vol. 7, no. 1, pp. 39-80.

Wittmer, J.P., Claudin, P., Cates, M.E. and Bouchaud, J.-P., 1996. "An Explanation for the Central Stress Minimum in Sand Piles," *Nature*, vol. 382, no. 6589, pp. 336-338.

Xie, H.-Y., 1997. "Role of Interparticle Forces in the Fluidization of Fine Particles," *Powder Technology*, vol. 94, no. 2, pp. 99-108.

Zhang, D. and Whiten, W.J., 1998. "Efficient Calculation Method for Particle Motion in Discrete Element Simulations," *Powder Technology*, vol. 98, no. 3, pp. 223-230.