University of Alberta

A Study of Energy Loss During Rock Impact Using PFC2D

by

Baoquan An

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

in

Mining Engineering

Department of Civil & Environmental Engineering

Edmonton, Alberta

Spring 2006

Your file Votre référence
ISBN: 0-494-13781-9
Our file Notre référence
ISBN: 0-494-13781-9

Dedicated to

The everlasting memory of my parents

Fusheng An

Shuyi Qiu

for their encouragement and love

and

my uncle

Yunsheng An

for his support and love

Abstract

A discrete element numerical method (PFC2D) is used to model kinetic energy loss problem during rock impact events such as those occurring during rockfall simulation. New energy-tracing functions are constructed for use within PFC2D to track energy items for different groups of particles in a model. A new User-Defined Contact Model (UDM) based on an elastic-perfectly plastic power function model is created to account for energy loss during the impact process. Two parameters are needed for the UDM: *transition force* and *exponent*. The *transition force* is the normal contact force at which the model transfers from elastic response to perfectly plastic deformation while undergoing compression. The *exponent* for the power function largely determines the damping effect of the model. The new UDM overcomes the numerical stability problems found with other UDMs and provides reasonable simulation results.

Keywords: discrete element method, energy tracing, user-defined contact model, kinetic energy, rock impact, energy loss, rockfall

ACKNOWLEDGEMENTS

I would first like to thank my advisor Dr. Dwayne D. Tannant whose kind help was very important for completion of this thesis. Many thanks go out to Dr. Qi Liu, Dr. Chan and Dr. Tim Joseph, who gave many suggestions for this thesis.

I would like to express my sincerest thanks to the following classmates: Xueqing Su, Ning Shi, Xiteng Liu, Maoxin Li, Lingen Jiang, Xiaobo Wang, Xiaowei Jia, Sibabrata Patnayak, Jovan Radmanovic, Rahul Jha and my friends: Guanglin Du, Ju Li, Hui Zhang, Yanhui Han, Cecilia Shiu, Lina Li and Chen Wang for their help during my studies in the University of Alberta.

Special thanks are given to my former advisor Renliang Shan and Quanchen Gao for their encouragement and kind help to my studies and life.

Finally I would like thank my sister, Na An, Hong An, and Zhenjie, my brothers, Baohe An, Po An and Baolin An. They provided unfaltering support to me when life is so hard to me. Without their assistance, I really don't know how to overcome numerous difficulties.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Rockfalls

Rockfalls are defined as rapid mass movements down steep slopes of one or several boulders through air by free jumping and rolling (Varnes 1978). The features of rockfalls are:

- Zero initial velocity
- Sudden movement, usually unexpectedly down a slope
- Movement by sliding, bouncing and rolling
- Human alterations to a slope or natural factors such as rain, earthquakes, freezing and thawing may be the cause
- Falling rocks may have large kinetic energy when reaching the foot of a slope.

Rockfalls result in serious hazards to roads, construction sites, working faces in mines and so on. The damage can be from direct impact or rock debris. Rockfall mitigation is thus important in protecting highways and construction sites. The following mitigation measures may be taken: use of restraining nets, rock fences, berms, rockfall attenuators, earth fills, intercepting ditches, rock sheds, and tunnels. In order to design effective rockfall mitigations, it is particularly necessary to gather data on rockfall dynamics: velocity, bounce height, and kinetic energy of boulders at various locations along a slope. Once these data are obtained, rockfall trajectories can be determined and relevant protective measures may be designed. For instance, rock fences may be set up at a position with a minimum bounce height and the lowest kinetic energy.

Since an important paper concerning rockfall problems was published by Ritchie (1963), many papers have been published to study rockfall dynamics. Generally, there are two approaches to estimate rockfall dynamics: experimental methods and numerical models (Richards et al. 1986, according to Azzoni et al. 1995).

Experimental methods include empirical studies and field tests. Scale models may be used in empirical studies (Ritchie 1963, Chan et al. 1986, Chau et al. 2002). Field tests may be carried out to determine rockfall paths and in particular, to calibrate numerical models and verify their effects in situ. Field tests are also used to check the efficiency of protective

measures (such as the use of fences, cable nets, ditches and so on). In the past, when computers were not as powerful as today and simulation methods were limited, field tests were extensively used to understand the behaviors of rockfalls. Undoubtedly, this methodology is effective, but it is expensive, time-consuming and sometimes, difficult to perform given a specific topology and environment. Therefore, it is unrealistic to do statistical and parametric analysis using experimental methods (Azzoni et al. 1995).

Rockfall simulation using numerical models is increasingly popular because of the development of computer technology and relevant software. Since an early application of numerical rockfall simulation was developed by Piteau and Peckover (1978), many papers have been published to present rockfall simulation through two-dimensional (Wu 1985, Kobayashi et al. 1990, Barret et al. 1991, Azzoni et al. 1995, Nicot et al. 2001) and three-dimensional models (Zimmermann et al. 1989, Agliardi and Crosta 2003, Yang et al. 2004).

Three-dimensional rockfall simulation is not used widely, because it is expensive to obtain high-resolution topographical models and it is more complicated to deal with lateral dispersion, i.e., the ratio between the lateral distance separating the two extreme fall paths and the slope length. According to Agliardi and Crosta (2003), lateral dispersion is the deviation of rockfall trajectories from the direction of the steepest gradient. Besides, below a specific resolution threshold, the effort/benefit ratio of 3D modeling will become unfavorable; a simpler 2D modeling will be a more cost-effective option. Agliardi and Crosta (2003), also indicates that it is hard to take a photograph with a high resolution because of specific slope complexities and modeling aims. These disadvantages make 3D modeling less popular than 2D modeling.

Most 2D rockfall simulations share similar features. For example, a boulder is considered as a lumped mass point, slopes are modeled as a concatenation of straight lines, a slope model is formed using topographic maps, a rockfall trajectory and path are in a vertical plane, the size and shape of boulders are not under consideration during simulations, and rigid-body mechanics is applied to solve impact problems. Most models provide data on average rockfall dynamics and inexact predictions of a single rockfall. The impacts of the boulder with the slope are modeled as inelastic using a restitution coefficient of tangential and normal

components of boulder velocities. Several definitions of restitution coefficients were given by Chau et al. (2002). The restitution coefficients are assumed to take all relevant values of impact characteristics into account such as sliding, deformation and the transformation of rotational moments into translational moments and vice versa (Schweigl et al. 2003).

Rockfall simulations overcome the disadvantages of experimental methods, because they are usually economic, and easy to make statistical and parametric analyses. Given such advantages, they are extensively used to study rockfall problems for designing corresponding protective works. However, most current rockfall simulations have similar limitations:

- The influence of the shape of boulders on simulation cannot be analyzed
- The interactions among different boulders cannot be simulated
- Some models adopt spherical balls, which may result in results that are too conservative, especially when boulder velocity is high.

The movements of a falling rock along a slope are affected by the following factors:

- Rock properties and discontinuities such as joints, which may cause potential fragmentation during impact
- Size and shape of rock
- Topography and inclination of slopes.

A key issue in rockfall simulations is how to model inelastic impact processes and possible rock fragmentation during impact. Unfortunately, so far there is only a limited insight into how rocks break and how energy is dissipated during impact. As well, given the very random nature of rock properties and slope topography, inelastic impact features are also hard to determine. Many scholars try to avoid discussing specific rock fragmentation events and use simple parameters such as restitution coefficients to simplify a complicated process. Such simplifications do make it easy to model rockfalls; however, the models may not be sufficiently close to physical rockfalls. For instance, a normal impact may result in a non-normal velocity component, but a method using restitution coefficient cannot simulate this case.

A numerical method developed in the last twenty years, Discrete Element Method (DEM), can overcome the many limitations of most 2D or 3D modeling. The Discrete Element Method will be discussed later.

1.2 Problem Statement

Rockfall movements include rolling, sliding and impacting. This research focuses on rock impact and the resulting bounce, which is the most important process involved in modeling rockfalls. The main concern is how to model energy losses that occur when one rock contacts another rock or the slope during rockfalls.

When a rock falls down a slope at low speeds, there is a small amount of rock powder created at the contact area during impact but the rock remains largely unbroken. The impact process is an inelastic process, in which a considerable amount of kinetic energy is consumed to create rock powder, microcracks, stress waves, and cratering of the slope, and thus the bounce velocity of the rock is lower than its initial impact velocity. At higher speeds, the rock may break into two or more pieces of rocks. This process can be viewed as a fragmentation process, in which kinetic energy is lost not only due to the above mentioned plastic impact process, but also due to the breakage of the rock.

The above two processes are roughly separated according to the macroscopic observations of rockfall phenomena. However, at the microscopic level it is impossible to determine whether there are microcracks created or not and how many microcracks are created within the rocks. Rock breakage and fragmentation results from a progressive propagation process of microcracks. In the plastic impact process, there might be microcracks created in rocks. The general tendency is that, there is relatively more kinetic energy loss during the impact process with increased impact velocity.

The total kinetic energy loss occurring during rock impact events will be accounted for by a new User-Defined Contact Constitutive Model (UDM), which is one of the main contributions of this thesis.

1.3 Objectives of PFC2D Modeling

The major objectives of this thesis are:

- Program energy tracing functions for use within PFC2D and use them to monitor the energy changes occurring during modeling
- Construct a new User-defined Contact Constitutive Model (Energy Loss Algorithm) to account for kinetic energy loss in the impact process; the new UDM will overcome the main disadvantages of existing modelling algorithms used in PFC2D
- Give guidelines for using the new UDM to simulate rock impact (rockfall) processes.

This research will focus on the simulation of impact processes or rockfalls, and will use literature data to calibrate the new model. This research will provide a solid ground for further extending the application of UDM to similar conditions such as rock milling, rock crushing, etc.

In PFC2D, all existing energy-tracing functions can only trace the energy items such as kinetic energy, friction work, body energy etc. for the whole model. In order to examine the energy distribution and demonstrate kinetic energy of components of the numerical model after impact, the energy-tracing functions were re-coded to be able to trace energy items for specified parts of a model.

Even at low impact velocities, there might be microcracks created in rocks. Since the energy lost during impact actually determines the kinetic energy available to rocks after impact. How to accurately account for kinetic energy loss is the key issue in this research.

An energy dissipation algorithm was constructed by using a User-Defined Contact Constitutive Model, which will be introduced later. The energy dissipation algorithm was used to dissipate energy due to both plastic impact process and rock fragmentation. The contact behaviors of PFC2D particles according to built-in contact models are elastic. By applying the energy dissipation algorithm, inelastic contact behavior can be introduced in modeling.

This research aims to improve the modeling of velocity and kinetic energy loss during impacts, allowing PFC2D models to simulate more accurately dynamic processes such as

rockfalls. This allows better understanding of the distribution of velocity and kinetic energy change during rockfalls or other dynamic issues such as rock crushing and rock breakage under impact loading. The aim of the research is not to calculate exactly how much energy is spent on plastic impact process and fragmentation process, but to provide a method and a numerical technique to reproduce the impact process of rocks by using PFC2D modeling. Thus, this research will aid in predicting the trajectory of real rockfalls, which can provide important data for the design of rockfall mitigation measures in engineering applications.

Implementing of the User-Defined Contact Constitutive Model (UDM) in this research will further enhance the power of PFC2D modeling to study dynamic process. The User-Defined Contact Constitutive Models is an optional feature of PFC2D. Modelers can use C^{++} to code their own contact model to adjust the force-displacement relationship of particles and simulate complex behaviors of rocks under different loading conditions. It is a very powerful feature of PFC2D to simulate rock behaviors. Given that this is an advanced technique and there are only very limited examples of UDM available for researchers to use, this research provides a new UDM for the study of rock behaviors.

In this research, an elastic-perfectly plastic power function contact model was constructed to account for kinetic energy loss due to plastic impact and fragmentation.

1.4 Methodology to Simulate Rock Impact and Energy Loss

The general research methodology includes literature studies and PFC2D modeling as shown in Figure 1.

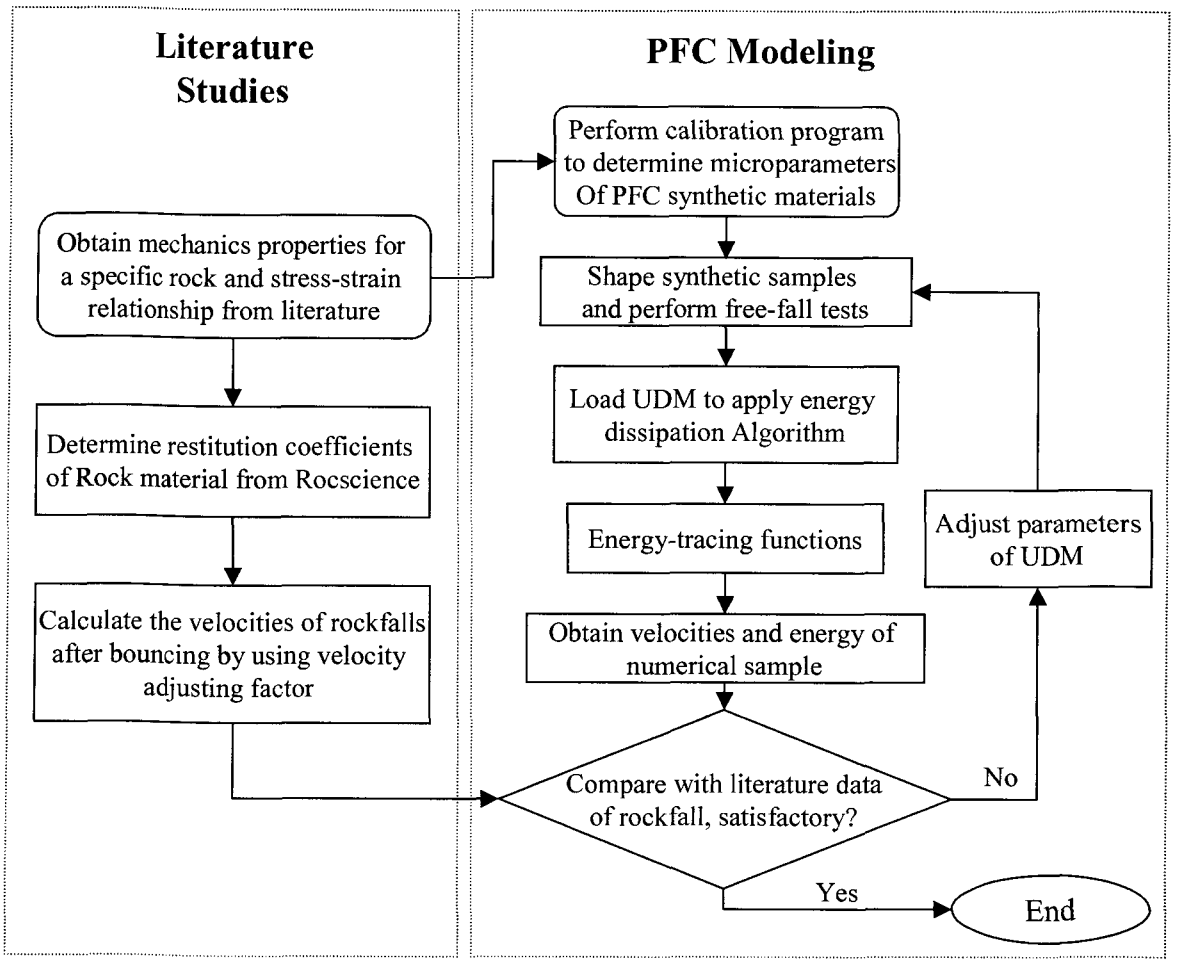


Figure 1 General research procedure

2 Discrete Element Method

The classical Discrete Element Method (DEM) was proposed by Cundall and Strack (1979). Since then, the Discrete Element Method has been generally applied to solve problems involving many discontinuous materials such as soil and rock. It is particularly suited for the simulation of granular and particle systems. To simulate complicated behaviors for rock and soil, continuum methods need very complex models, involving many parameters. Compared with continuum methods, a DEM model requires only a few material properties because the behavioral complexity such as nonlinear stress/strain response, dilation phenomena, transition from brittle to ductile behavior, nonlinear strength envelopes arises automatically (Cundall 2002).

2.1 Discontinuous Deformation Analysis (DDA)

Given its growing capability to perform powerful numerical modeling, DEM has been increasingly applied to model rockfalls. For instance, the Discontinuous Deformation Analysis (DDA) method is a technique developed recently that is already used to simulate the interactions between rock blocks, rock trajectory, and impact processes of rockfalls.

DDA is a sort of DEM method. It was first proposed by Shi and Goodman (1984) to calculate the strains and displacements of blocky system. Lin et al. (1996) first used this method in the modeling of rockfalls after providing several extensions to Shi and Goodman's original DDA method. From Lin et al.'s 1996 paper, the features of DDA method can be summarized as follows.

The formulation of blocks is similar to the definition of a finite element mesh, and a problem of a finite element type is solved in each element, which is a block divided by pre-defined discontinuities. The blocks in the DDA method may have convex or concave shapes. While two blocks are in contact, Coulomb's law applies to contact interfaces, and relevant equilibrium equations are constructed and then solved at each loading increment. Block-fracturing algorithms can be implemented in the extended DDA method. The algorithm permits rocks to be broken into smaller blocks. The failure mode may be Mode I (tensile fracturing) or Mode II (shear fracturing). Lin et al. (1996) use the Mohr-Coulomb criterion as a fracture criterion and making it possible to model fracture propagation in already-fractured rock masses.

The advantage of the DDA model is its capability to simulate the interaction and displacement between rock blocks. According to Lin et al. (1996), none of the previous modeling methods have included fracturing between rock blocks during impact.

The major limitation of DDA model is that it does not include an energy loss criterion. No energy is consumed during the shear or tensile failure of blocks, and upon breaking, each newly created block is viewed as having the same velocities as the original one. However, classical fracture mechanics has shown that strain energy will be released to create new rock

surface during fracturing. As well, there are no considerations for damping phenomena upon contact and impact in this model.

Ohnishi et al. (1996) considered the effects of dragging force and collisions between blocks, proposing that the damping coefficient due to dragging force and collisions between blocks should be calibrated by field test results. Modification of the stiffness of the contact spring in open-close iteration is used to account for the energy loss due to block collision. Koo et al. (1997) further modified the DDA method to reduce the error associated with rigid body rotation and improve program efficiency. The most important enhancement is that a mass proportional damping coefficient C is introduced to dissipate the energy due to dragging force. The damping value used by Koo et al. (1997) takes the form:

$$C = \alpha M \qquad \text{Equation 1}$$

where α is the damping coefficient and M is the mass per unit area. When applied in rockfall problems, the mass proportional damping coefficient is used in a trial and error method to match the field observations on the distribution of fallen rock blocks.

Therefore, the modified DDA method has the capability to analyze dynamic problems via the above damping procedures to make the block motion converge to a stable state.

The advantages and disadvantages of DDA method were summarized by Lin et al. (1996). Including some DDA features explained by other researchers (Ohnishi et al. 1996, and Koo et al. 1997), the advantages and disadvantages are as follows:

Advantages:

- Static and dynamic modeling can be implemented
- Sub-blocks can break under given a fracturing criterion without external intervention
- Energy loss due to collision and dragging force can be accounted for by the modification of the stiffness of the contact spring and the introduction of a mass damping coefficient
- Large displacements and interactions of blocks are allowed.

Disadvantages:

- Block geometry must be input

- Lengthy computing time to solve even simple problems
- More experience is needed to perform modeling than other methods
- The mass damping coefficient is used to account for the energy loss by dragging force, which may lack physical meaning
- The energy loss due to rock fracturing is omitted.

DDA method does not deal with energy loss very well although the introduction of the damping coefficient seems to satisfy the convergence requirement of dynamic problems. Apparently, the energy consumed in free falling, rolling, and sliding should be different. Even so, DDA method has been a milestone to model rockfalls in that the complicated impact features such as collisions and the interactions of rocks can be captured. In order to overcome the above-mentioned disadvantages, especially the last two, Particle Flow Code 2-dimension (PFC2D), another DEM method, is applied to model rock fragmentation and collision issues in this thesis.

2.2 Particle Flow Code (PFC2D)

Particle flow modeling method was first introduced by Cundall and Strack in 1979. It is applied to model the movements and interactions of spherical particles using the distinct element method. In this method, objects are represented as discrete rigid particles (balls) that can be bonded by contact bonds or parallel bonds. The bond has shear and normal strength components. Normal and shear stiffness are used to represent the contact stiffness between any two particles. Parallel bonds can transmit both forces and moments between particles, while contact bonds can transmit only forces acting at the contact point. A slip-model, defined by a friction coefficient between any two particles, governs the frictional strength characteristic of the assemblies. Fracturing can be implemented by bond breaking in either shear mode or normal mode. Assemblies of clumps or clusters, together with joint sets, can be used to simulate jointed rock block behavior.

The calculation cycle in PFC2D is a time-stepping algorithm that consists of the repeated application of the Law of Motion to each particle, a force-displacement law to each contact, and a constant updating of wall positions (Itasca 2002). Contacts, which may exist between

balls or balls and walls, are formed and broken automatically during the simulation. Newton's second Law of Motion gives the movement of a particle resulting from the action forces and moments, while the force-displacement law is used to find the contact forces that are induced by the relative motion between the contact particles.

At the start of each time step, the set of contacts is updated from the known particles and wall positions in an assembly. The force-displacement law is then applied to each contact to update the contact forces based on the relative displacement between the two entities (balls or walls) at the contact using default or user's defined contact constitutive model, which will be introduced later. Next, the Law of Motion is applied to each particle to update the velocity and position based on the resultant force and moment arising from the contact forces and any body forces acting on the particle. The wall positions are updated based on the specified wall velocities.

The users of PFC2D can use Fish, a built-in programming language, to construct their models by accounting for different sizes of particles under specified stresses. Contact behaviors can also be controlled by using built-in contact or parallel bonds. Furthermore, users can use C++ to code their own contact models (User-Defined Model) to control the behaviors of rock blocks according to different loading environments. A User-Defined Model (UDM) can be used instead of default contact models once relevant commands are carried out.

As a DEM method, the advantages and disadvantages of PFC2D are very similar to those of DDA, except that PFC2D has a more powerful dynamic modeling capacity because of UDM's function and versatility due to the fact that it is an extendable program, which can be coded by Fish or C++ by users themselves. The characteristics of PFC2D can be summarized as follows:

Advantages:

- Large displacements and interactions of blocks are allowed
- Static and dynamic modeling can be implemented
- Built-in damping functions can be used for simulations that do not involve free flights or impacting particles

- UDM can supersede built-in contact models to govern contact procedure, and dissipate energy
- Strain energy, friction energy, kinetic energy and bond energy can be traced provided corresponding programs are coded
- Bonds can break under imposed normal and shear forces without external intervention
- Clumps and clusters can be used to represent rock blocks, and random joint sets can be used to represent weakened joints in rock mass
- Typical behaviors in rock and soil tests such as dilation, strain softening, and nonlinear stress-strain relationship can be easily reproduced.

In modeling rockfalls, some specific shapes of rockfalls such as sphere-like or ellipsoid-like instead of lumped mass point can be allowed, and the complex topography of slopes can be represented by using different sizes of particles and walls.

Disadvantages:

- More experience is needed to perform modeling than with other methods, especially when some optional features such as UDM are adopted. Typically 18 months may be required to become familiar with the software and coding
- The computing time is long when a large number of particles are used
- Difficulty to calibrate with lab tests because physical parameters such as strength, Young's modulus, and Poisson ratio cannot be used directly in modeling
- Researchers have to spend much time on coding, even when solving a simple problem.

When a bond breaks, the stored strain energy at the contact will be released both as stress wave and as strain energy to form a crack. The success of the PFC2D material at reproducing the behavior of hard rock can be attributed directly to its ability to generate compression-induced tensile cracks (Itasca 2002). Potyondy and Cundall (2004) presented the formation of compression-induced cracks, as show in Figure 2, in which a group of four circular particles is forced apart by axial load, causing the restraining bond to experience tension. If angular particles replace circular particles, "wedge" and "staircases" also induce local tension. For example, "wedges" and "staircases" induce local tension if angular grains replace circular particles, shown in Figure 2a and b.

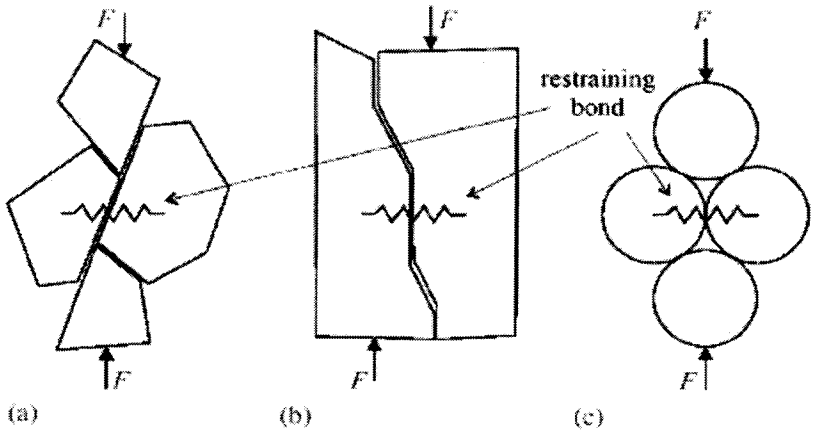


Figure 2 Physical mechanisms for compression-induced tensile cracking (a and b) and idealization as bonded assembly of circular particles (c) (Potyondy and Cundall 2004)

3 Energy-Tracing Functions

To better understand the energy losses occurring within a PFC2D model during impact events it is necessary to track or trace the energy components of specific groups of particles that can be individual particles, clumps, or clusters as well as the stored strain energy existing at contacts between particles. Bonded particles are used to represent objects such as a falling rock. Currently, the built-in energy tracing functions provided by PFC2D can only trace the total energy of the whole model. It is impossible to trace energy items such as kinetic energy, strain energy, bond energy and friction energy using these built-in functions. However, the built-in functions can be used to help validate user-written energy tracing functions.

New energy tracing functions were written and tested to trace body energy, kinetic energy, friction energy, strain energy, and bond energy of a single ball, clump and contact. The code for these functions is presented in Appendix D. The functions were validated with general built-in energy tracing functions. Since friction energy, strain energy and bond energy are stored between contact bonds, a problem arises about how to allocate energy between two balls in contact. The methodology used here is that, if two balls are in one group, then the energy is added to the total energy of the group; otherwise, each ball in different groups shares half of the strain energy. All energy-tracing functions are coded by using the Fish language.

3.1 Body Work

Body work is total accumulated work done by all body forces on the assembly. According to the Itasca (2002), body forces are defined to be gravity loading and applied forces and moments:

$$E_b \leftarrow E_b + \sum_{Np} \left((mg_i + F_i)\Delta U_i + M_3 \Delta \theta_3 \right) \qquad \text{Equation 2}$$

where, N_p, m, g_i, F_i, M_3, ΔU_i, $\Delta \theta_3$ are the number of particles, mass, gravitational acceleration, externally applied force, externally applied moment, computed displacement increment, and computed rotation increment, respectively, for a particle during current time step. The computation assume that, g_i, F_i, M_3 are constant throughout the time step.

In this research, body forces only include gravity loading, thus the above equation will be:

$$E_b \leftarrow E_b + \sum_{Np} \left(mg_i \Delta U_i \right) \qquad \text{Equation 3}$$

The flow chart to calculate the body work for each group of balls (group can be single ball, a cluster or clump) is shown in Figure 3.

The reliability of the energy tracing function for body work can be demonstrated by a simple model involving two clumps of particles to simulate a rock as shown in Figure 4. The slope was composed of several wall segments. The microparameters of balls and walls were set arbitrarily.

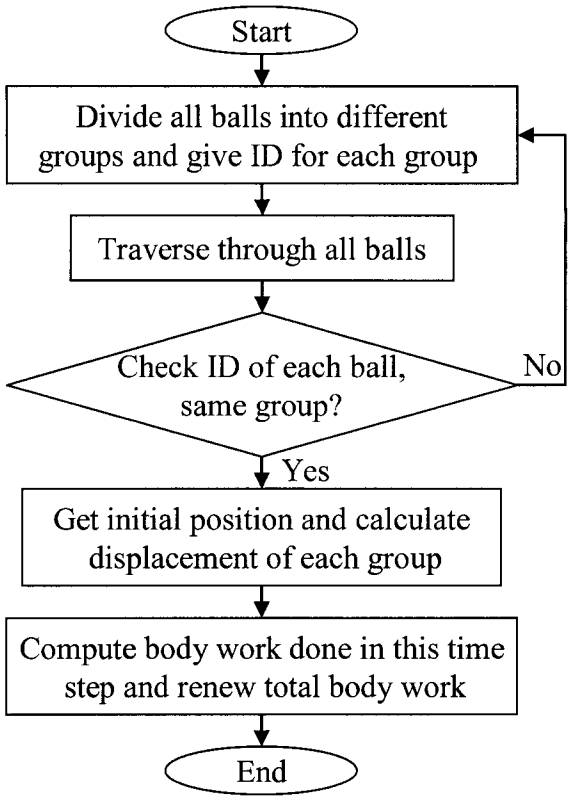


Figure 3 Schematic of the body work energy tracing function

The body energy for whole model and each clump in the rockfall simulation is shown in Figure 5. The total body energy was obtained directly from a built-in Fish function. The total body energy is exactly equal to the sum of the body energy of two clumps. This proves that the energy tracing function for body energy is valid.

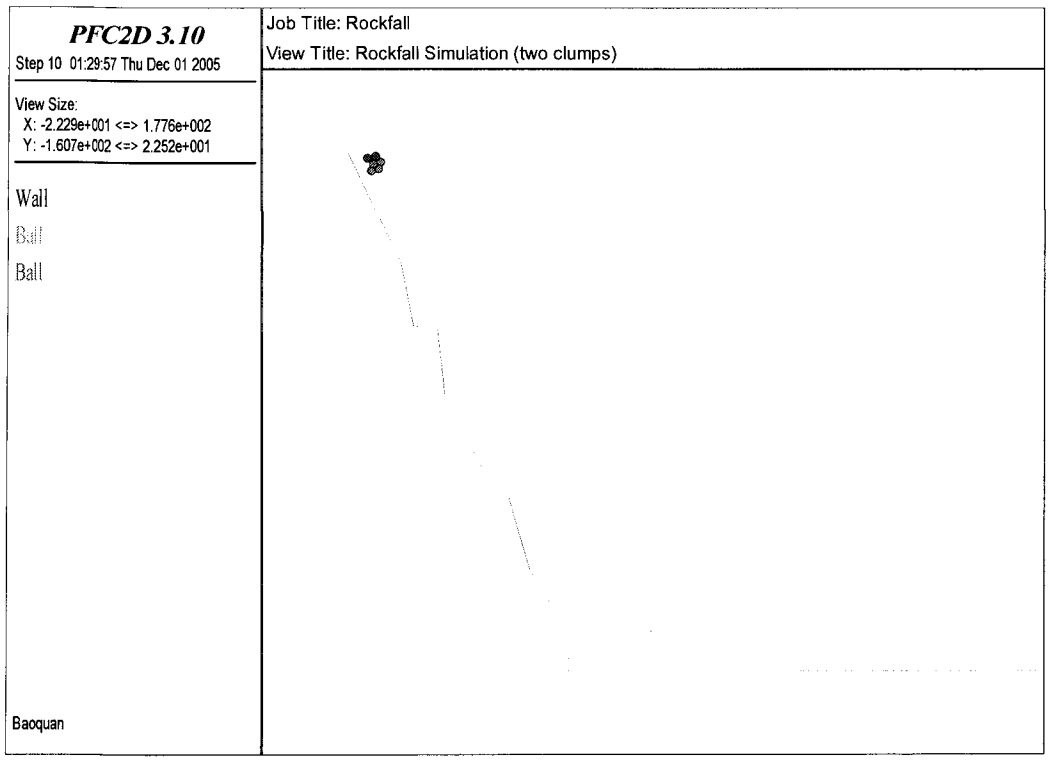


Figure 4 Rockfall simulation with the rock made by two clumps

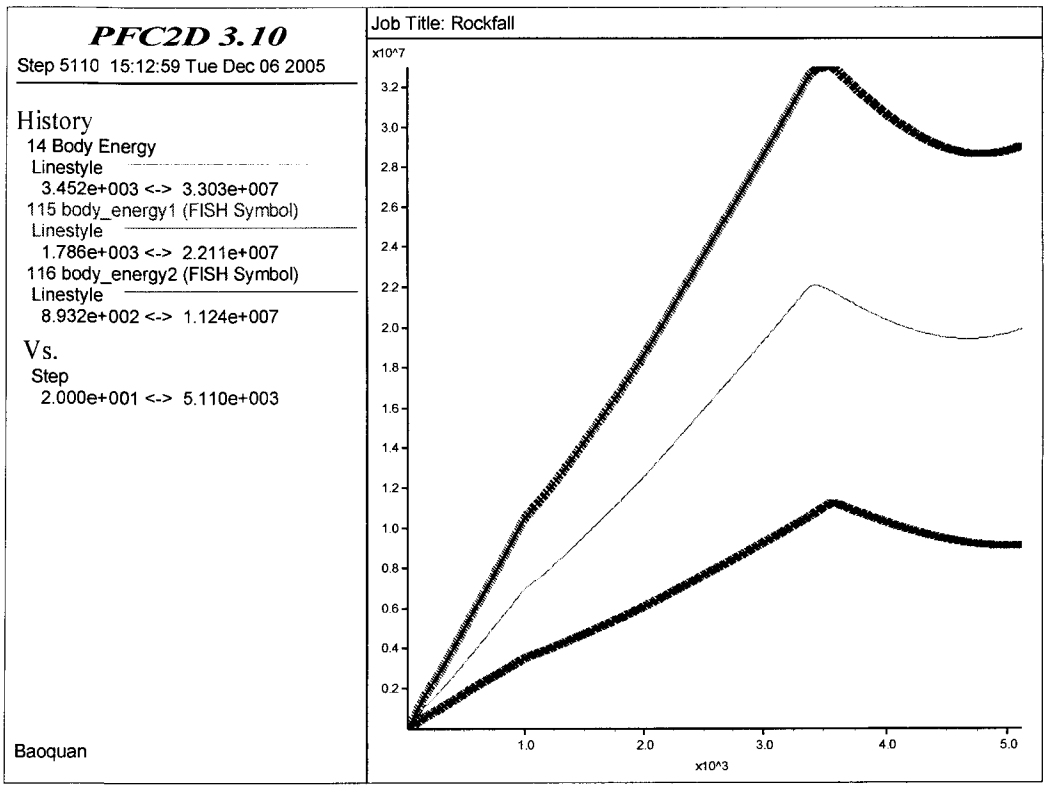


Figure 5 Body energy of all particles, clump 1 and clump 2

3.2 Kinetic Energy

Kinetic energy is total kinetic energy, E_k of all particles in one group accounting for both translational and rotational motion and can be expressed in terms of the generalized mass and velocity of each of the N_p particle as:

$$E_k = \frac{1}{2}\sum_{Np}\sum_{i=1}^{3} M_{(i)}V_{(i)}^2 \qquad \text{Equation 4}$$

where generalized mass $M_{(i)}$ includes physical mass and principal moments of inertia of particles, and generalized velocity $V_{(i)}$ includes translational velocity and angular velocity about the principal axis (the principal axis is perpendicular to 2D space).

The flow chart to calculate kinetic energy for each group of balls (group can be single ball, a cluster or clump) is shown in Figure 6.

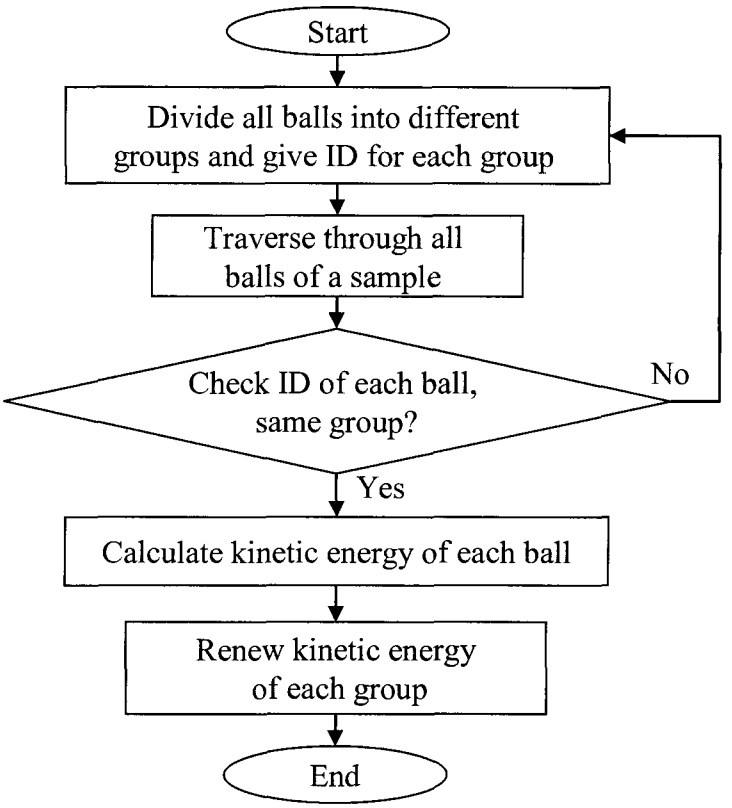


Figure 6 Schematic of the kinetic energy tracing function

The kinetic energy of the whole model and each clump in the simulation in Figure 4 is shown in Figure 7. In the figure, the total kinetic energy was obtained directly from a built-in Fish function. The total kinetic energy is exactly equal to the sum of the kinetic energy of the two clumps. This proves that the energy tracing function for kinetic energy is valid. The kinetic energy tracing function is very useful because the kinetic energy of a falling rock is one of the important parameters needed for rockfall mitigation issues.

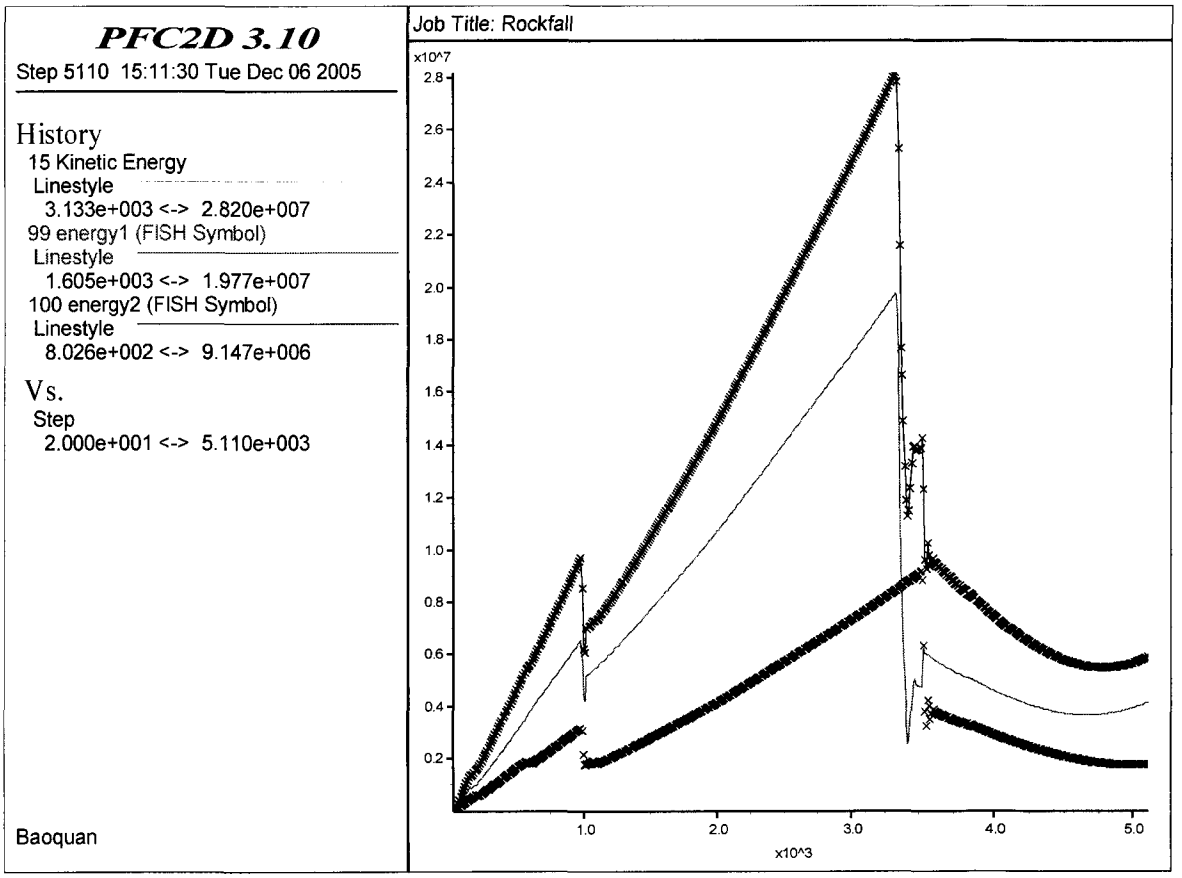


Figure 7 Kinetic energy of all particles, clump 1 and clump 2

3.3 Strain Energy

Strain energy is energy, E_c, of the group stored at all contacts assuming a linear contact-stiffness model:

$$E_c = \frac{1}{2}\sum_{N_c}\left(\frac{|F_i^n|^2}{k^n} + \frac{|F_i^s|^2}{k^s}\right) \qquad \text{Equation 5}$$

where N_c is the number of contacts; $|F_i^n|$ and $|F_i^s|$ are the magnitudes of the normal and shear components of the contact force; and k^n and k^s are the normal and shear-contact stiffnesses. Since strain energy is stored between contact bonds, how to allocate energy between two balls in contact is a problem. The methodology used in this research is that, if two balls are in one group, then the energy is simply added to total energy of the group; otherwise, each ball in different groups shares half of the strain energy. Another important point to note is that in the UDM model, a bi-linear stiffness model is used, so $|F_i^n|$ and $|F_i^s|$ must be calculated in terms of different k^n and k^s. Correspondingly, strain energy will be calculated in terms of different k^n and k^s. The flow chart is given in Figure 8.

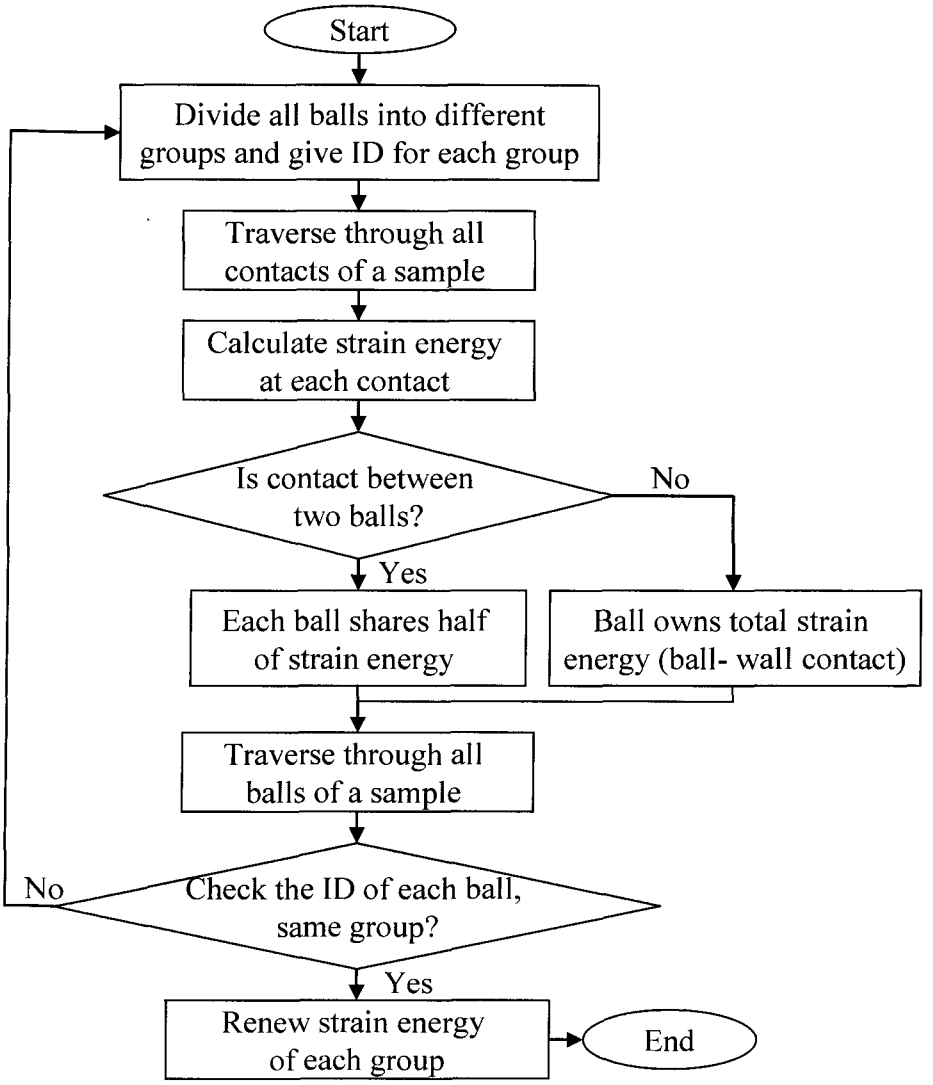


Figure 8 Schematic of the strain energy tracing function

The strain energy of the whole model and each clump in the simulation in Figure 4 is shown in Figure 9. In the figure, the total strain energy was obtained directly from a built-in Fish function. The total strain energy is exactly equal to the sum of the strain energy of the two clumps. This proves that the energy tracing function for strain energy is valid.

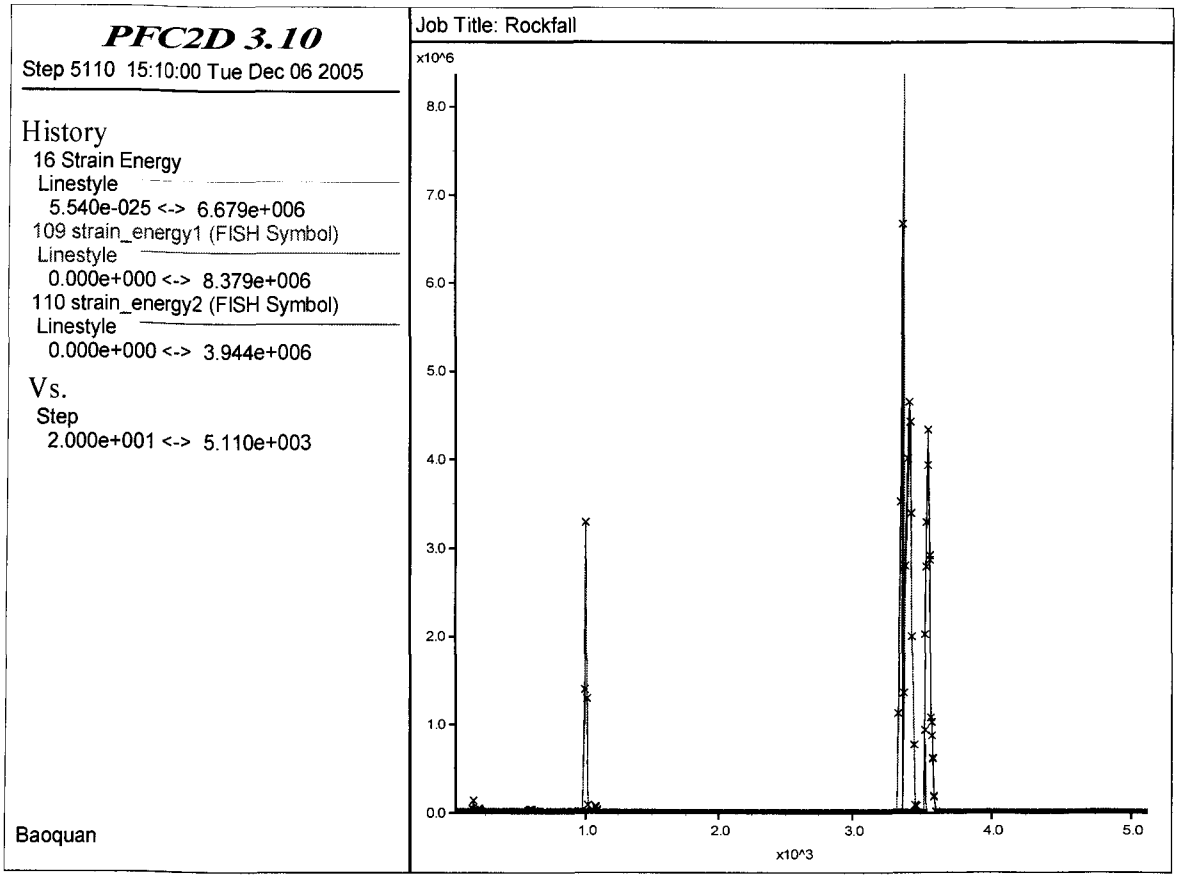


Figure 9 Strain energy of all balls, clump 1 and clump 2

3.4 Bond Energy

Bond energy is total strain energy, E_{pb}, of the assembly stored in the parallel bonds. While calculating bond energy, the same methodology used for strain energy is used, i.e., to share bond energy between different groups. The equation to compute bond energy is:

$$E_{pb} = \frac{1}{2}\sum_{N_{pb}}\left(\frac{\left|\overline{F_i^n}\right|^2}{A\bar{k}^n} + \frac{\left|\overline{F_i^s}\right|^2}{A\bar{k}^s} + \frac{\overline{M_3}}{I\bar{k}_n}\right) \qquad \text{Equation 6}$$

where N_{pb} is the number of parallel bonds, and the notation for the forces, moments, and stiffnesses associated with each parallel bond is from Section 2.3.2 of Itasca (2002). The flow chart is given in Figure 10.

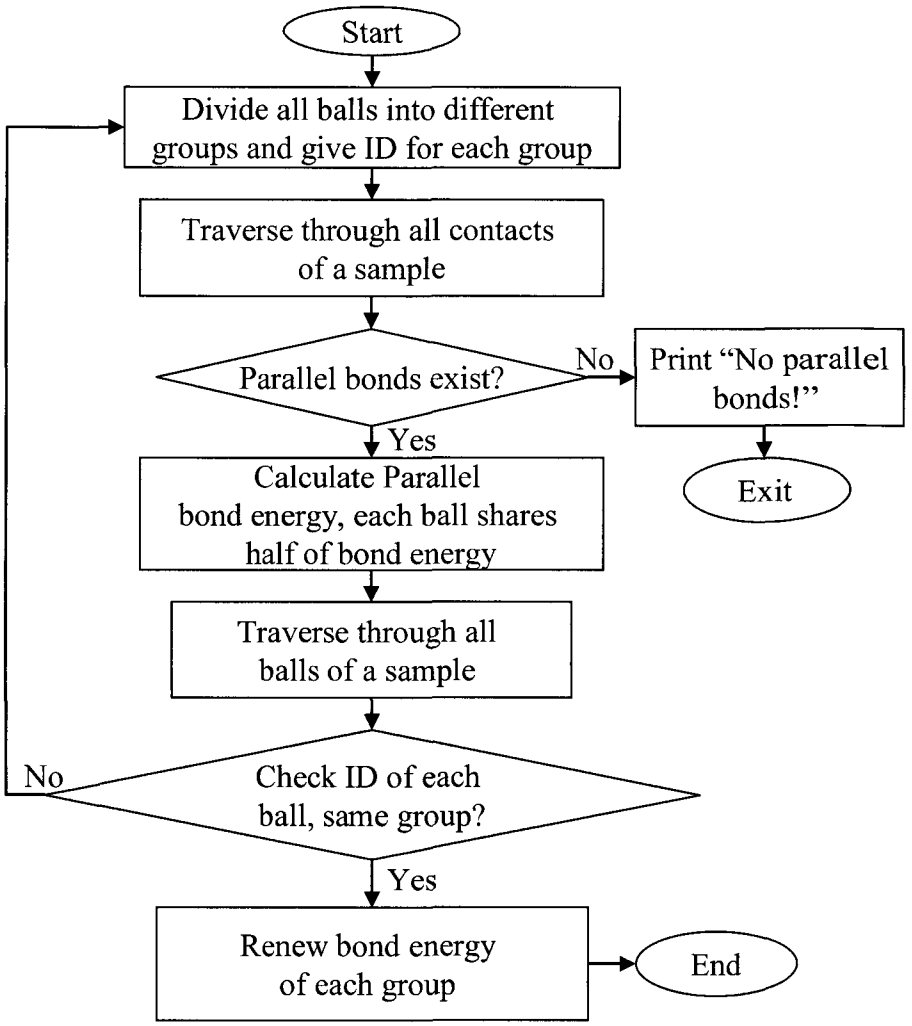


Figure 10 Schematic of the bond energy tracing function

The strain energy of the whole model and each clump in the simulation in Figure 4 is shown in Figure 11 and Figure 12. In Figure 11, the total bond energy was obtained directly from a built-in Fish function in. The total bond energy is exactly equal to the bond energy of two clumps. This proves that the bond energy tracing function is valid.

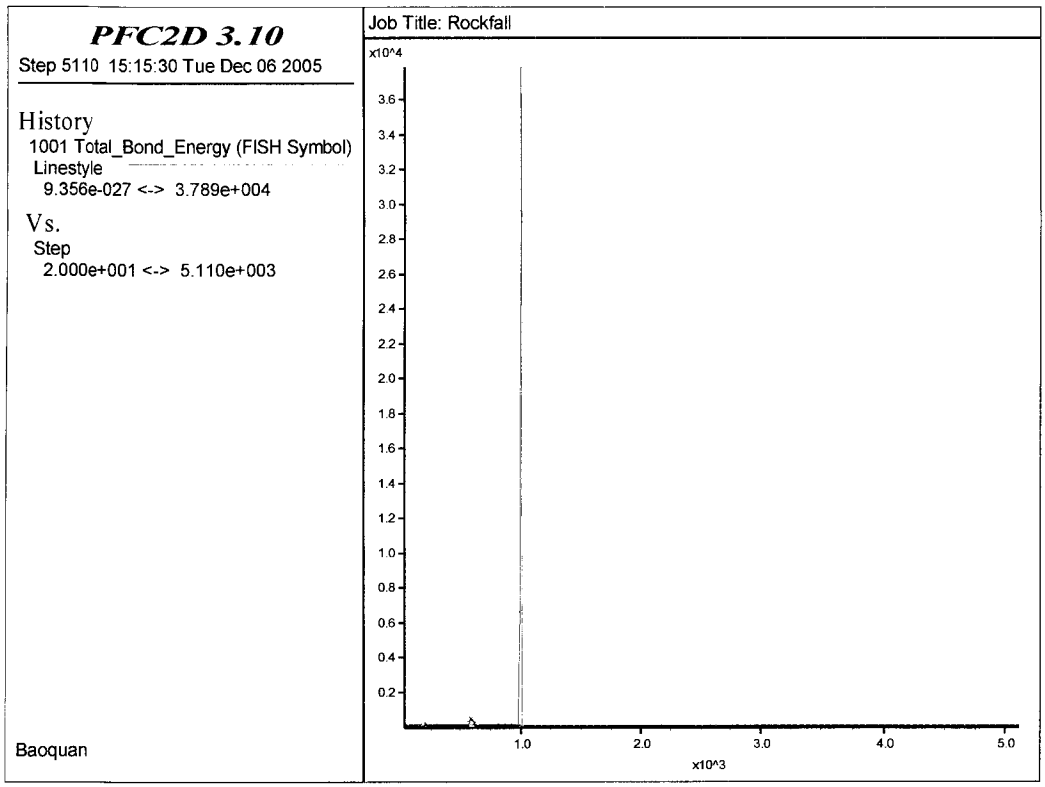


Figure 11 Bond energy of all particles

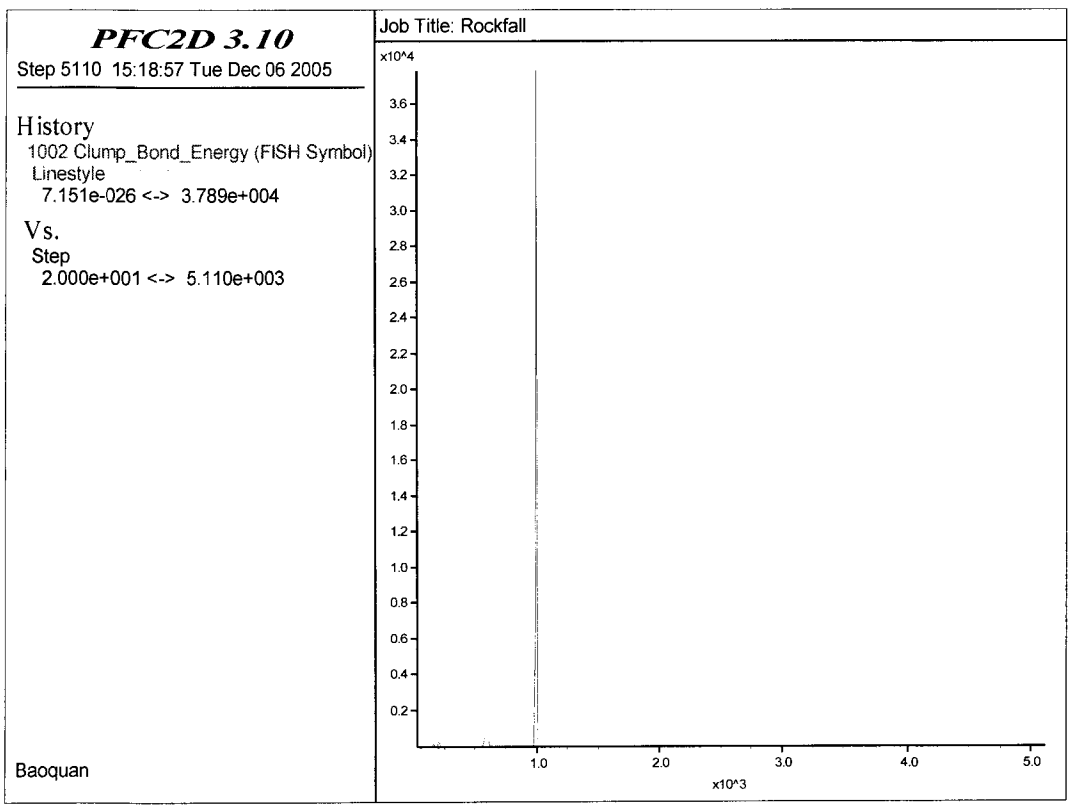


Figure 12 Bond energy between clump 1 and clump 2

3.5 Frictional Work

Frictional work is energy, E_f, dissipated by frictional sliding at all contacts. In order to calculate friction energy, we need to know slip displacement, which is given by:

$$\Delta U^s = V^s \Delta t \qquad \text{Equation 7}$$

where Δt is a time step, and V^s is relative shear velocity. According to the Itasca (2002) the shear velocity of ball B relative to ball A at the contact point for ball-ball contact, or the shear velocity of the wall relative to the ball at the contact point for ball-wall contact is given by:

$$V^s = \left(\dot{x}_i^{[\Phi^2]} - \dot{x}_i^{[\Phi^1]}\right) t_i - \omega^{[\Phi^2]} \left| x_k^{[C]} - x_k^{[\Phi^2]} \right| - \omega^{[\Phi^1]} \left| x_k^{[C]} - x_k^{[\Phi^1]} \right| \qquad \text{Equation 8}$$

where $\dot{x}_i^{[\Phi^j]}$ and $\omega^{[\Phi^j]}$ are the translational and rotational velocity, respectively, of entity Φ^j given by Equation 9.

$$\left\{\Phi^1, \Phi^2\right\} = \begin{Bmatrix} \{A, B\}, (ball - ball) \\ \{b, w\}, (ball - wall) \end{Bmatrix} \qquad \text{Equation 9}$$

and $t_i = \{-n_2, n_1\}$

Friction energy can be given by Equation 10.

$$E_f \leftarrow E_f - \sum_{N_C} \left(\left\langle F_i^s \right\rangle \left(\Delta U_i^s \right)^{slip} \right) \qquad \text{Equation 10}$$

where N_c is the number of contacts; and $\left\langle F_i^s \right\rangle$ and $\left(\Delta U_i^s\right)^{slip}$ are the average shear force and the increment of slip displacement, respectively, at the contact for the current time step. The increment of slip displacement is determined by decomposing the total displacement into a slip and elastic portion, given by Equation 7, so that

$$\begin{aligned} \left(\Delta U_i^s\right)^{slip} &= \Delta U_i^s - \left(\Delta U_i^s\right)^{elas} = \Delta U_i^s + \frac{\left(\Delta F_i^s\right)^{elas}}{k^s} \\ &= \Delta U_i^s + \frac{\left(F_i^s\right)^{(t+\Delta t)} - \left(F_i^s\right)^{(t)}}{k^s} \end{aligned} \qquad \text{Equation 11}$$

where ΔU_i^s is given by Equation 7, and F_i^s is the shear force at the contact. The increment of frictional work done at each contact is stored in the Fish variable *c_slipwork*.

The flow chart to calculate friction energy is given in Figure 13.

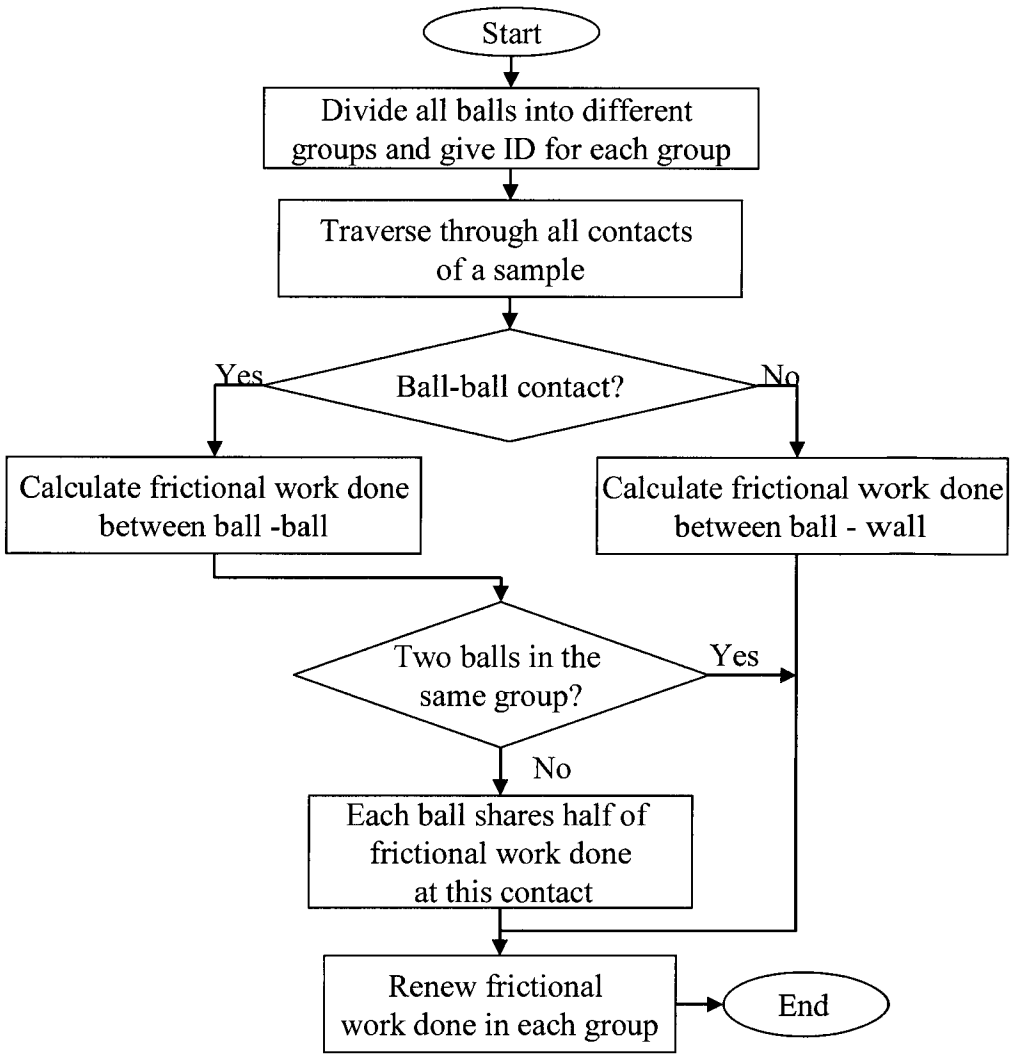


Figure 13 Schematic of the frictional work (energy) tracing function

The friction work of the whole model and each clump in the simulation in Figure 4 is shown in Figure 14. In Figure 14, the total friction work done in the impact and sliding process was obtained directly from a built-in Fish function. The total friction work is exactly equal to the friction work of the two clumps. This proves that the friction work (energy) tracing function is valid.

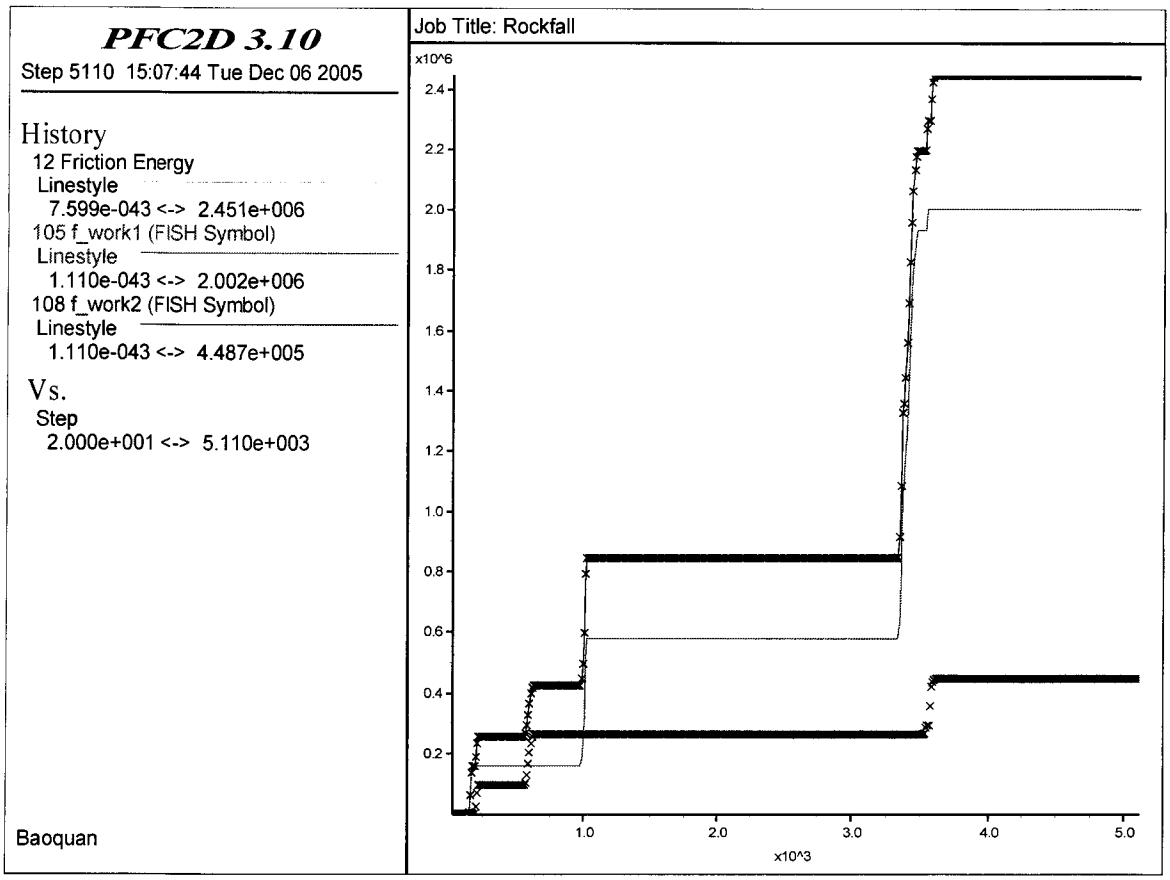


Figure 14 Friction work of all particles, and clump 1 and clump 2

3.6 Application of Energy Tracing Functions

Although the energy-tracing functions were developed for tracking energy for different groups of particles, they can also be used to track energy of a whole model. In the above descriptions of energy tracing functions, the procedure always begin with "divide all the balls into different groups and give ID for each group", that is, energy items of particles are traced in terms of groups. If energy for each particle in the whole model is traced, it will take too much computing time especially while using history function to store energy data. Then a question arises: how to divide a sample into different groups?

As the focus is the energy changes occurring during impact events, it is better to put the particles into a group that do not separate after impact so that it is easy to check energy loss during the impact and fragmentation process. To do this, a preliminarily simulation using the

model can be used to obtain the breakage pattern of the rock and then identify groups according to the fragmentation pattern.

4 Contact Constitutive Model (Contact Models)

4.1 Introduction

A force-displacement law is used in a contact model to update contact forces for each contact and then the Law of Motion will update displacement and velocity using Newton's second law. During the calculation cycle at each time step, users can introduce different contact models to control the running of a PFC2D model. There are some default contact constitutive models and several alternative contact constitutive models implemented in PFC2D for simulation of more complex contact behavior. These models are ready to use in PFC2D. If necessary, users can write their own contact model, known as a User-defined Model.

For the default contact constitutive models, users only need to us the *PROPERTY* command to define the model properties such as normal stiffness, shear stiffness, contact bond strength, parallel bond stiffness, and parallel bond strength. Once the properties of a default contact model are set up, they will act at each contact automatically. For the alternative models, users can invoke them using *MODEL* command. The *MODEL* command activates the given model only at existing contacts. If new contacts form in the future, a *FISHCALL* function must be provided for the new contacts in order to use the model.

The constitutive model working at a specific contact consists of three parts: a stiffness model, a slip model, and a bond model. The stiffness model provides an elastic relation between relative displacement and the contact forces. The slip model enforces a relation between shear and normal contact forces so that the two contacting balls may slip relative to one another. The bond model serves to limit the total normal and shear forces that the contact can carry by enforcing bond-strength limits.

4.2 Contact Stiffness Models

The contact stiffness models relate the contact forces and relative displacements in the normal and shear directions via Equation 12 and Equation 13. The normal stiffness is a secant stiffness because it relates the total normal force to the total normal displacement. The shear stiffness is a tangent stiffness because it relates the increment of shear force to the increment of shear displacement.

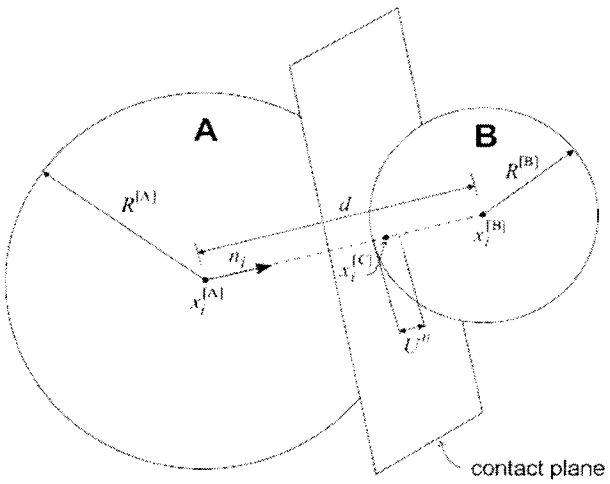
$$F_i^n = K^n U^n \qquad \text{Equation 12}$$

$$\Delta F_i^S = -k^S \Delta U^S \qquad \text{Equation 13}$$

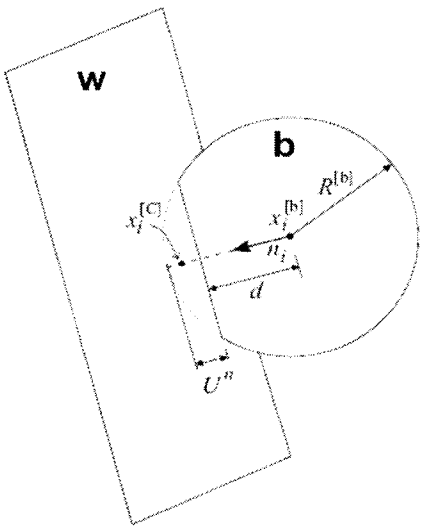
In above equations, K^n and k^s are normal and shear stiffness, U^n and ΔU^s are normal relative displacement and shear relative displacement increment.

Figure 15 shows the notation used to describe contact at two entities (ball-ball and ball-wall). The contact lies on a contact plane that is defined by a unit normal vector n_i (n_i lies in the plane of the PFC2D model.) The contact point is within the interpenetration volume of the two entities. For ball-ball contact, the normal vector is directed along the line between ball centers; for ball-wall contact, the normal vector is directed along the line defining the shortest distance between the ball center and the wall.

The contact force can be decomposed into normal component acting in the direction of the normal vector and a shear component acting in the contact plane. The stiffness model relates these two components of force to the corresponding components of the relative displacement via the normal and shear stiffness at the contact.



(a) Notation used to describe ball-ball contact



(b) Notation used to describe ball-wall contact

Figure 15 Notation used to describe PFC2D contact (after Itasca 2002)

The contact model is described for both ball-ball and ball-wall contacts. For ball-ball contact, the relevant equations are presented for the case of two spherical particles, labeled *A* and *B* in Figure 15(a). For ball-wall contact, the relevant equations are presented for the case of a spherical particle and a wall, labeled b and w, respectively, in Figure 15(b). In both cases, U_n denotes overlap.

For ball-ball contact, the unit normal n_i that defines the contact plane is given by:

$$n_i = \frac{x_i^{[B]} - x_i^{[A]}}{d} \quad \text{(ball-ball)} \qquad \text{Equation 14}$$

where $x_i^{[A]}$ and $x_i^{[B]}$ are the position vectors of the centers of balls A and B, and d is the distance between the ball centers:

$$d = \left| x_i^{[B]} - x_i^{[A]} \right| = \sqrt{(x_i^{[B]} - x_i^{[A]})(x_i^{[B]} - x_i^{[A]})} \quad \text{(ball-ball)} \qquad \text{Equation 15}$$

where n_i corresponds with position vectors at time step (t- $\Delta t/2$), which also tracks time in the equations.

For ball-wall contact, n_i is directed along the line that defines the shortest distance d between the ball center and the wall. This direction can be found by mapping the ball center into a relevant portion of space defined by the wall. For a two-dimensional wall composed of two line segments AB and BC, the determination of normal direction is shown in Figure 16. All space on the active side of this wall is separated into five regions by extending a line normal to each wall segment at its endpoints.

If the ball center lies in regions 1, 3, or 5, it will contact the wall at one of its endpoints, and n_i will be along the line connecting the endpoint and the ball center. If the ball center is in the regions 2 or 4, it will contact the wall along the length, and n_i will be normal to the corresponding wall segment.

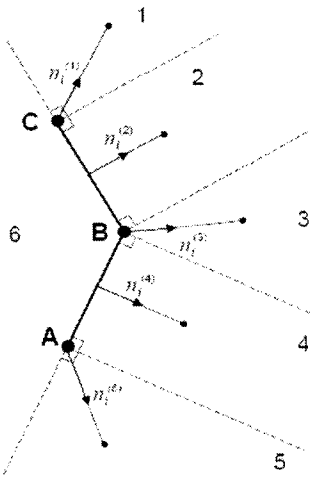


Figure 16 Determination of normal direction for ball-wall contact (after Itasca 2002)

The overlap U_n, defined to be the relative contact displacement in the normal direction, is given by

$$U^n = \begin{Bmatrix} R^{[A]} + R^{[B]} - d & (ball - ball) \\ R^{[b]} - d & (ball - wall) \end{Bmatrix} \qquad \text{Equation 16}$$

where $R[\phi]$ is the radius of ball ϕ.

The location of the contact point is given by

$$x_i^{[C]} = \begin{Bmatrix} x_i^{[A]} + (R^{[A]} - \frac{1}{2}U^n)n_i & (ball - ball) \\ x_i^{[b]} + (R^{[b]} - \frac{1}{2}U^n)n_i & (ball - wall) \end{Bmatrix} \qquad \text{Equation 17}$$

The contact force vector F_i (at the ball-ball contact or ball-wall contact) can be resolved into normal and shear components with respect to the contact plane as

$$F_i = F_i^n + F_i^s \qquad \text{Equation 18}$$

where F_i^n and F_i^s denote the normal and shear component vectors, respectively.

4.3 Slip Model

The slip model is actually an intrinsic property of the two entities (ball-ball or ball-wall) in contact. It permits slip to occur by limiting the shear force and provides no normal strength in tension. If there is no contact bond defined, this model is always active. Otherwise, the

contact bond model behavior will supersede the slip model behavior. These two models demonstrate the constitutive behavior for particle contact occurring at a point. The parallel-bond model, on the other hand, simulates the constitutive behavior for a cementing material existing between the two balls. These two behaviors can occur at the same time; so, if contact bond is not defined, the slip model is active in conjunction with the parallel-bond model.

The criterion of shear strength caused by slip model is implemented by checking whether the overlap, given by Equation 16, is less than or equal to zero. If it is, then both the normal and shear contact forces are set to zero. The contact is checked for slip conditions by calculating the maximum allowable shear contact force

$$F^s_{\max} = \mu \left| F^n_i \right| \qquad \text{Equation 19}$$

If $\left|F^s_i\right| > F^s_{\max}$, then slip is allowed to occur (during the next calculation cycle) by setting the magnitude of F^s_i equal to $F^s_{\max}$ as follows:

$$F^s_i \leftarrow F^s_i \left(F^s_{\max} / \left| F^s_i \right| \right) \qquad \text{Equation 20}$$

The some energy in a particle system can be dissipated through frictional sliding.

4.4 Contact-Bond Model

The contact bond can be viewed as a pair of elastic springs with constant normal and shear stiffness acting at the contact point. These two springs have specified shear and tensile strengths. The existence of a contact bond override the possibility of slip – i.e., the magnitude of the shear contact force is not adjusted to remain less than the allowable maximum. Instead, the magnitude of the shear contact force is limited by the shear bond strength. Contact bonds also allow tensile forces to develop at a contact. These forces arise from the application of Equation 12 when $U_n < 0$ (i.e., there is gap between two entities). In this case, the contact bond acts to bind the balls together. The magnitude of the tensile normal contact force is limited by the contact bond strength.

If the tensile normal contact force equals or exceeds the normal contact bond strength, the bond breaks, and both the normal and shear contact forces are revalued as zero. If the shear contact force equals or exceeds the shear bond strength, the bond breaks, but the contact forces are still the same, if the shear force does not exceed the friction limit, and the normal force is compressive.

The constitutive behavior relating the normal and shear components of contact force and relative displacement for particle contact is shown in Figure 17. At any time, either the contact-bond model or the slip model works. In Figure 17, F_n is the normal contact force, where $F_n > 0$ means tension force; U_n is the relative normal displacement, where $U_n > 0$ indicates overlap; F_s is the total shear contact force; and U_s is the total shear displacement measured relative to the location of the contact point when the contact bond was defined.

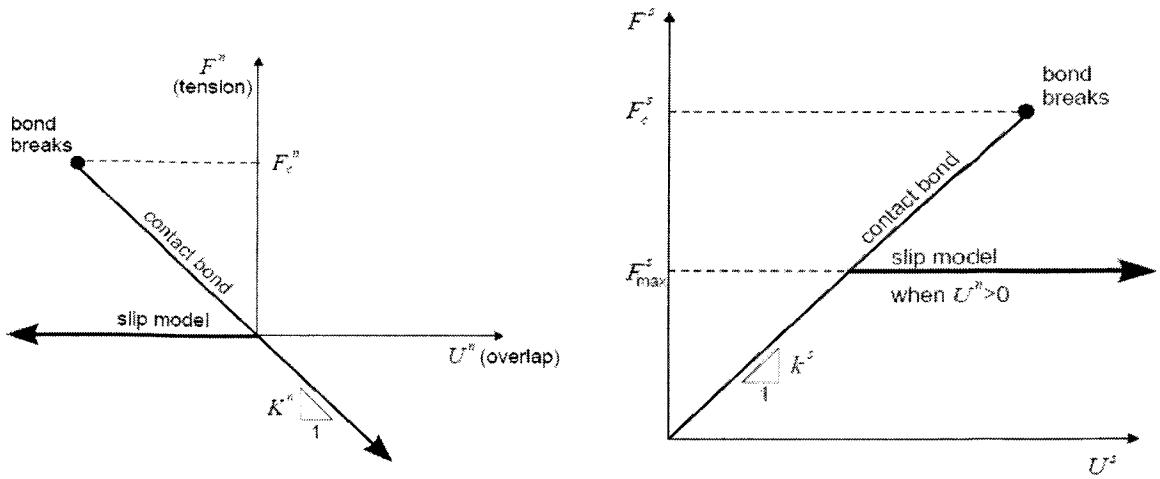


Figure 17 Normal component of contact force and shear component of contact force occurring at a point (after Itasca 2002)

4.5 Parallel-Bond Model

The parallel-bond model relates the constitutive behavior of a finite-sized piece of cementitious material deposited between two balls. The two balls are envisioned as either spheres or cylinders in the parallel-bond logic. These bonds construct an elastic interaction relationship between balls that works together with the slip or contact-bond models. Therefore, the existence of a parallel bond does not preclude the possibility of slip. Since the parallel-bond model is not used in this research, no further details are given.

4.6 Alternative Models Including Hysteretic-Damping Model

Several alternative contact models can be used with PFC2D to simulate more complicated contact behaviors. These alternative models can be called by using the *MODEL* command. Note that the *MODEL* command activates the specified model only at existing contacts. If new contacts form in the future, a *FISHCALL* function should be used for the new contacts to adopt the model. These models are given as dynamic link libraries (.dll), and it is possible for users to write their own models.

The alternative models that are ready to use are simple visco-elastic model, simple ductile model, displacement-softening model, Burger's model, and hysteretic damping model. These models can be found at Itasca (2002). In this research, one of the major objectives is to reproduce the energy losses during rock impact processes and only the hysteretic damping model is capable of simulating energy losses.

There are several numerical damping methods that essential dissipate energy such as local damping and combined damping. These are used to maintain numerical stability in PFC2D when simulating quasi-static processes. The combined damping is actually a variation of local damping for the case in which the steady-state solution includes a significant uniform motion. According to Itasca (2002), local damping is inappropriate for particle in free flight under gravity or for impact of particles. Instead, hysteretic damping model, an alternative contact model, is suggested to reproduce the type of energy dissipation occurring during impact.

The contact model *HYSDAMP* was created by Shimizu and Cundall (2001) to introduce energy dissipation by hysteretic damping to a linear contact model with frictional slip. The model has the following properties shown in Table 1.

Table 1 Properties used in Hysteretic Damping Model

Model Property	Meaning of Property
hys_ k_{nm}	Normal stiffness, the average of the normal stiffness on loading, K_{n_load}, and on unloading, K_{n_unload}
hys_ dampn	Ratio of normal stiffness, K_{n_load}, to that on unloading, K_{n_unload}, ($0.4 \leq hys\_dampn \leq 1.0$ (with tensile force); $0.05 \leq hys\_damp \leq 1.0$
hys_ k_s	Shear stiffness
hys_fric	Friction coefficient
hys_nstrp	Contact bond normal strength [force]
hys_sstr	Contact bond shear strength [force]
hys_notension	Switch (0: tensile force allowed (default); 1: no tension allowed)
hys_inheritpro	Switch (0: the model does not inherit properties from PFC2D, (default); 1: the model inherits the properties)

The normal stiffness on loading, K_{n_load}, and on unloading, K_{n_unload}, used in this model are calculated by

$$K_{n\_load} = \frac{2hys\_dampn \cdot hys\_knm}{1 + hys\_dampn} \qquad \text{Equation 21}$$

$$K_{n\_unload} = \frac{2Hys\_knm}{1 + hys\_dampn} \qquad \text{Equation 22}$$

where *hys_ knm* is taken as the average of K_{n_load} and K_{n_unload}.

In the hysteretic damping model (Figure 18), normal stiffness on unloading is larger than that on loading in order to dissipate energy. Once the property parameters are specified, the hysteretic damping is independent of the relative velocity before and after contact. It indicates that *hys_dampn*, the ratio between the two stiffnesses, should be determined with a parametric pretest to get a measurable quantity, such as the restitution coefficient.

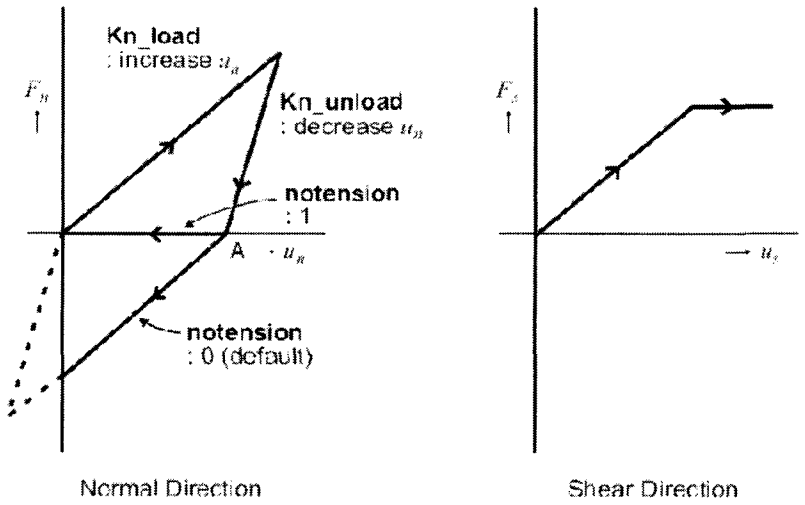


Figure 18 Hysteric-damping contact model (after Itasca 2002)

The advantage of the hysteric-damping model is that the physical meaning of damping is very straightforward and it is easy to determine the model properties. This model was successfully used in the simulation of conveying granular materials by horizontal screw by Shimizu (2002). However, the hysteric-damping model may not be appropriate to simulate the impact of rockfalls given the following disadvantages.

The hysteric-damping model has two main disadvantages. First, when normal contact force arrives at zero, relative normal displacement is at point A in the axis of U_n (see Figure 18), if the particle is loaded again, it follow the similar loading and unloading procedure as shown in Figure 18, but start from point A instead of the original point. This indicates that the "effective radius" of the particle is reduced for each loading-unloading cycle. Here the "effective radius" means the distance between the contact plane to the center of the particle. If the effective radius is equal to or less than zero, the contact force will cease to exist for a ball-wall contact or the force switches to the opposite direction destroying the contact and giving a wrong result.

The second disadvantage of the hysteric-damping model is an overloading problem. The overloading problem is caused by the PFC2D calculation cycle system. An example model was used to demonstrate this problem. In the test shown in Figure 19, the hysteric-damping model was used to simulate the free falling of two particles on a wall.

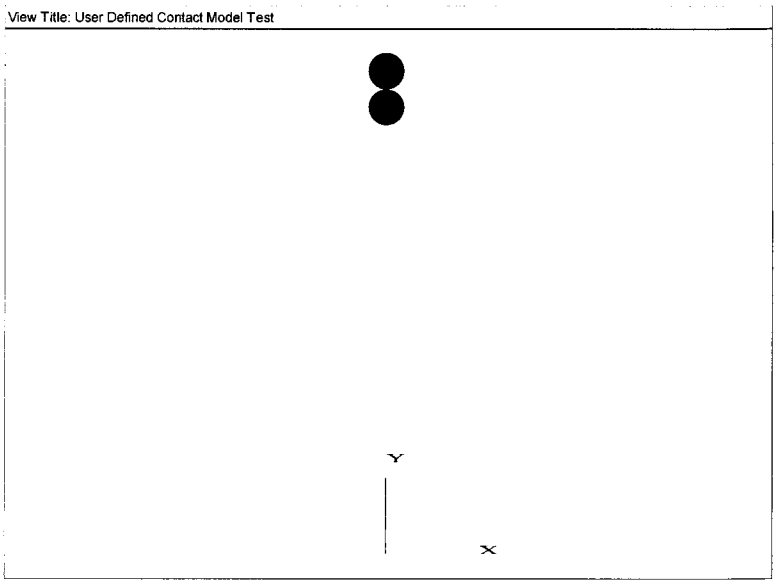


Figure 19 Free falling of two particles to test hysteric damping model (radius: 0.04 m, K_n: 6.4e10 N/m, k_s: 9.6e9 N/m, friction: 0.6)

During the test, it was observed that when bouncing back from the wall, the particles obtained a velocity that was much higher than the velocity before impact. The velocity history of each particle is shown in Figure 20 and Figure 21.

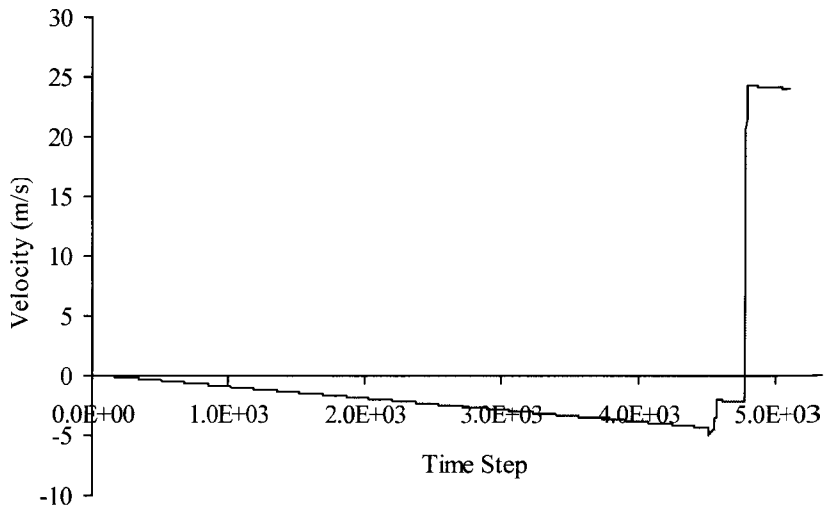


Figure 20 Relationship between velocity and time step (upper particle)

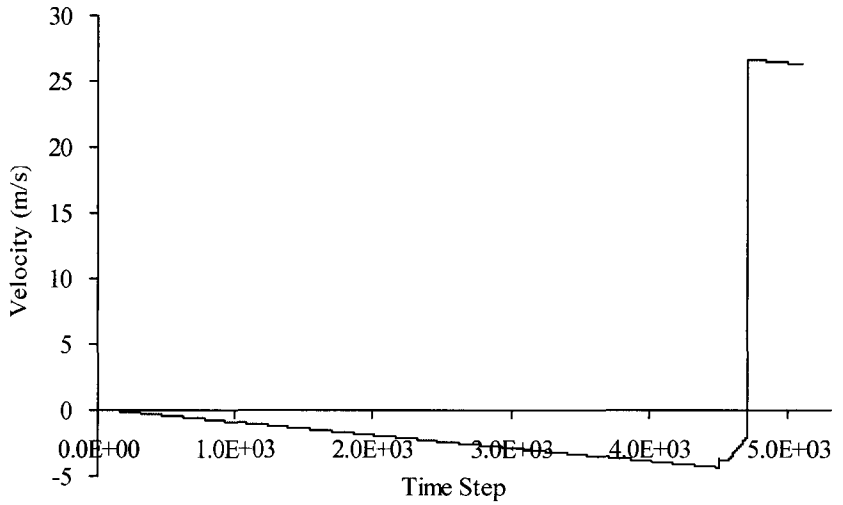


Figure 21 Relationship between velocity and time step (lower particle)

The velocity of each particle jumped from 0 to 26.7 m/s at the time step 4700, which was much larger than the velocity before impact. The total energy of the whole model after impact is much larger than that before impact. This is clearly wrong and does not happen in reality. The reason is that the initial contact force is too large (Figure 22), because of a large initial relative displacement occurs during the first calculation cycle after the ball contacts the wall. This causes an unrealistic kinetic energy increment instead of damping kinetic energy and results in numerical instability. Because the hysteretic damping model may cause solution instability problems, it cannot be used to simulate rockfalls.

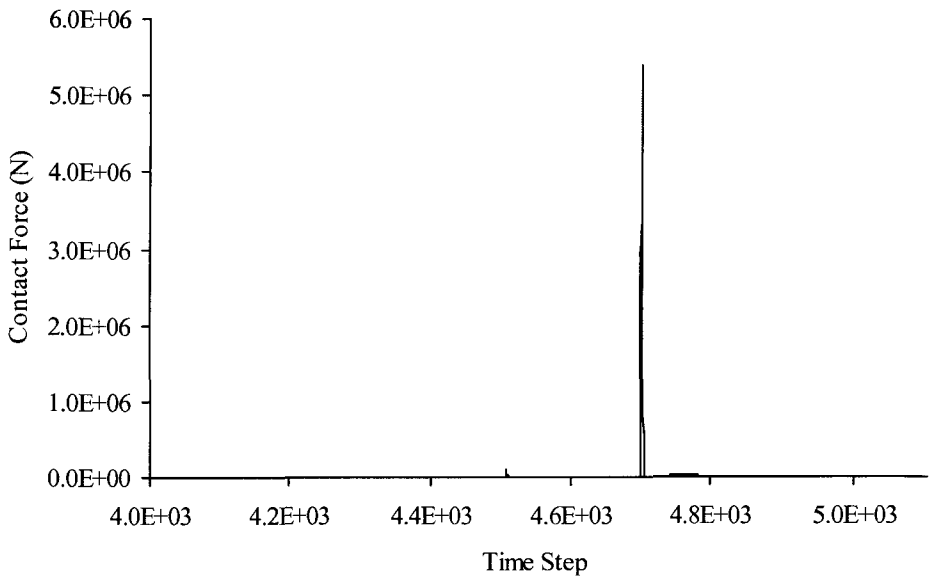
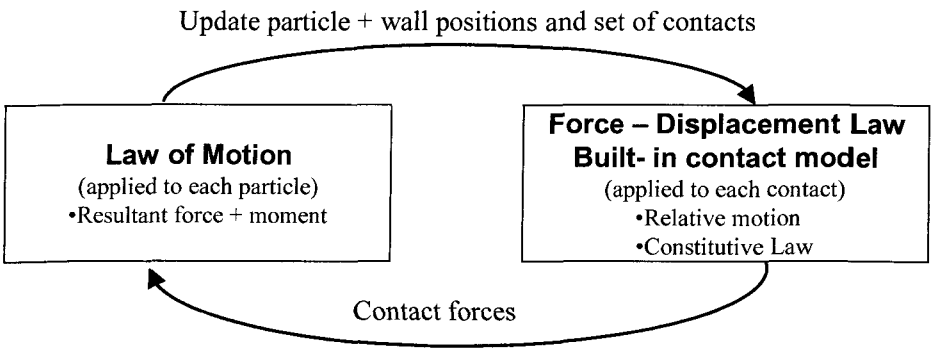


Figure 22 Resultant contact force vs. time step

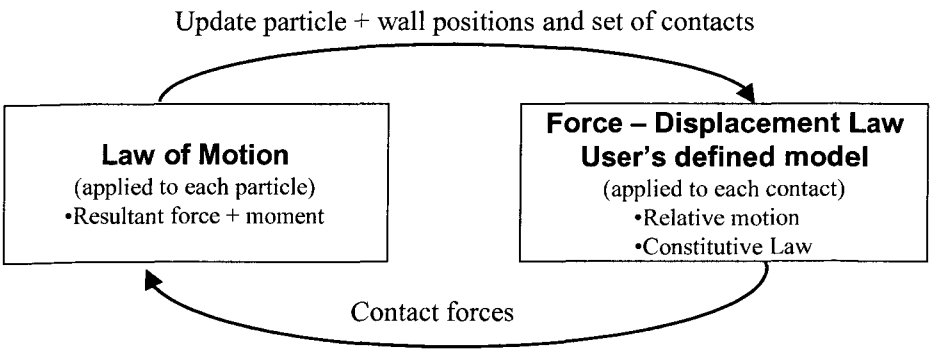
5 New User-Defined Contact Model

The User-Defined Contact Models (UDM) can directly adjust the contact stiffness and calculate contact forces according to relative particle displacements or the contact forces calculated from the last time step. Complex model behaviors can be simulated by using a UDM and they can speed up the calculation processes. User-Defined Contact Models are coded in VC++, and they are built as a dynamic link library (.dll) file. They can be loaded whenever needed.

The similarity and difference between a calculation cycle with and without UDM is shown in Figure 23. The common feature between them is that, both involve a cycle of Law of Motion and force-displacement law, and the force– displacement laws provide contact forces such as normal, shear and friction. The functions of the different force-displacement laws are the same. The difference is that the built-in contact model governing the force-displacement law is replaced by a User-Defined Contact Model.



(a) Calculation cycle using built-in contact model



(b) Calculation cycle using a User-Defined Contact Model

Figure 23 Calculation cycle in PFC

The UDM is used to calculate contact forces based on particle displacement and velocity, obtained from the built-in Law of Motion. The general structure of a dynamic link library file that constructs a UDM is shown in Figure 24.

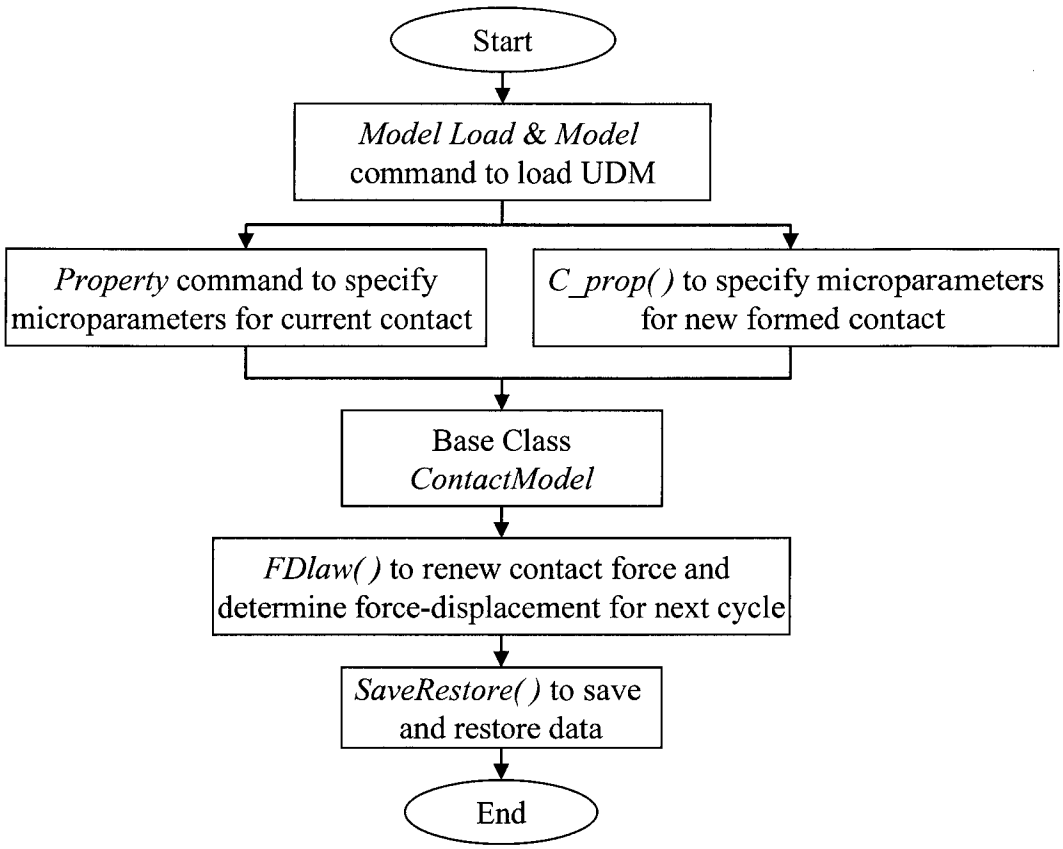


Figure 24 General structure of a dynamic link library file for UDM

The function, *FDlaw( )* is a core function that constructs the relationship between force and displacement according to contact force, relative velocities, relative distance of particles. The mathematical relationship of the desired contact model, *FDlaw( )* can be coded and then all the other functions can be programmed correspondingly.

Once a UDM is constructed within a dynamic link library file, the user does not need to understand how the file runs when it is called by PFC2D. For the user, the UDM is a 'black box'. The user only needs to know the main function of the UDM, the required input data, and how to set up the UDM parameters.

Other parameters used in the UDM control the transfer to a dynamic link library file either by automatic mode or by the user's interference if necessary (for instance, when a new contact is created). These parameters are normal and shear stiffness, friction coefficient, and bond strength if a contact bond is introduced. A schematic of a UDM is shown in Figure 25.

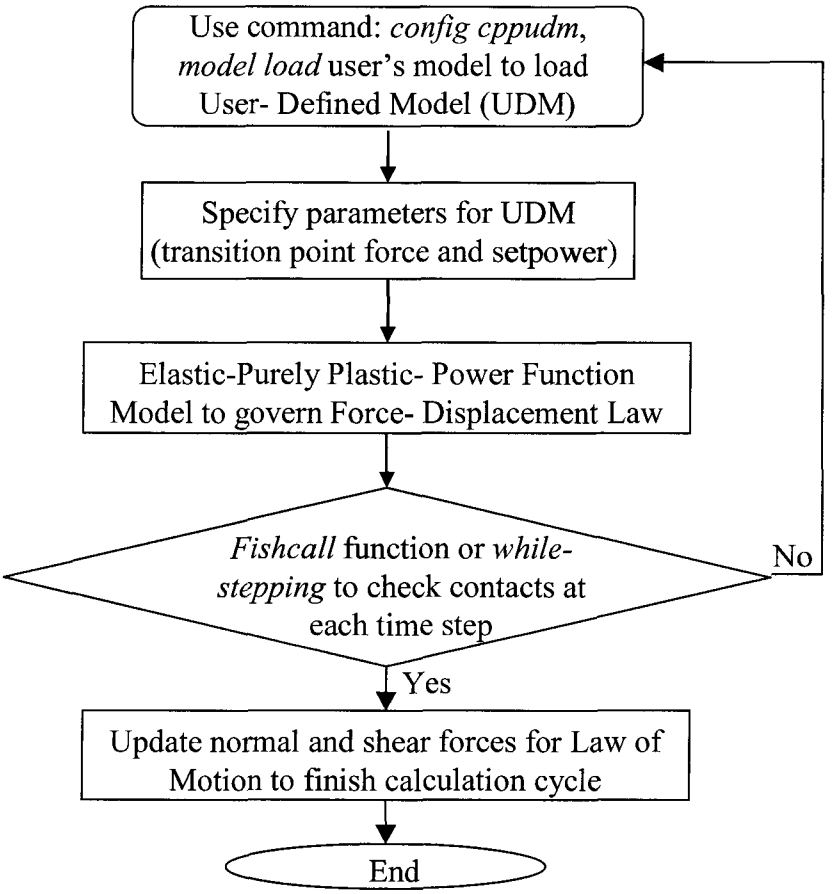


Figure 25 Schematic of the user-defined constitutive model

A major objective in this research is to model the energy dissipation during the impact process of rockfalls. The current UDMs cannot account for energy losses occurring during contact (impact) between particles. In this research, a new elastic-perfectly plastic power function model was created and used to simulate energy dissipation during rock impact events. This model is shown in Figure 26. The code for this UDM is presented in Appendix D.

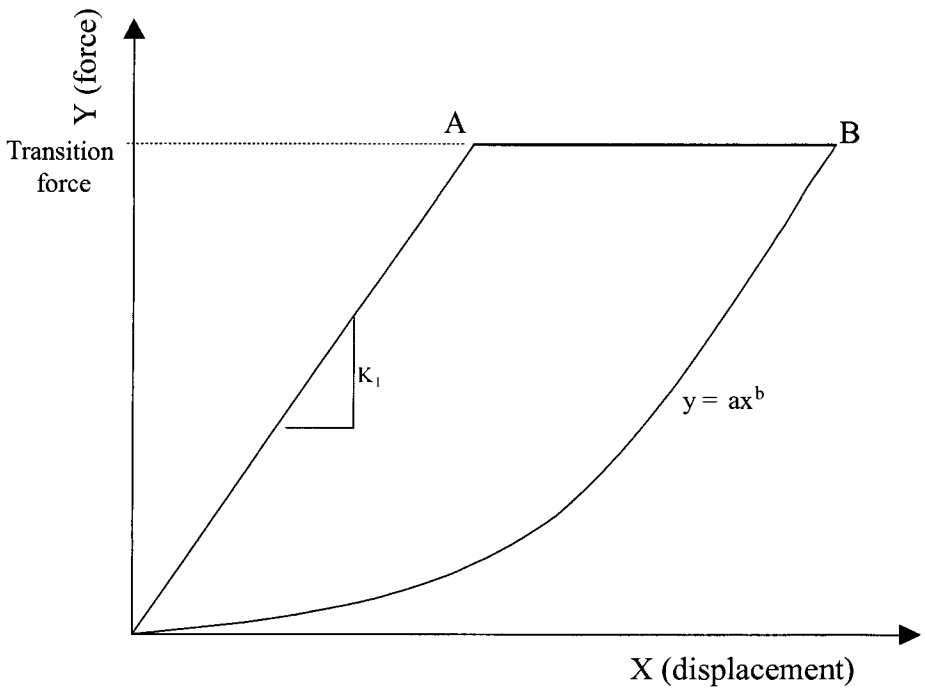


Figure 26 Elastic-perfectly plastic power function model

The main parameters needed for the elastic-perfectly plastic power function model are *transition force* and *exponent*. *Transition force* is used to control the contact force level at which a linear elastic force-displacement relationship transfers to a perfectly plastic process. The *exponent* adjusts the power for damping function. The value of *exponent*, *b* has a dominant control on the energy loss during impact and hence can be considered to control the restitution coefficient.

The normal and shear stiffness for the elastic segment adopt the value determined by the routine calibration program. The friction coefficient and particle mass and particle mode (e.g. clump) are used in the simulation of the impact process of rockfalls.

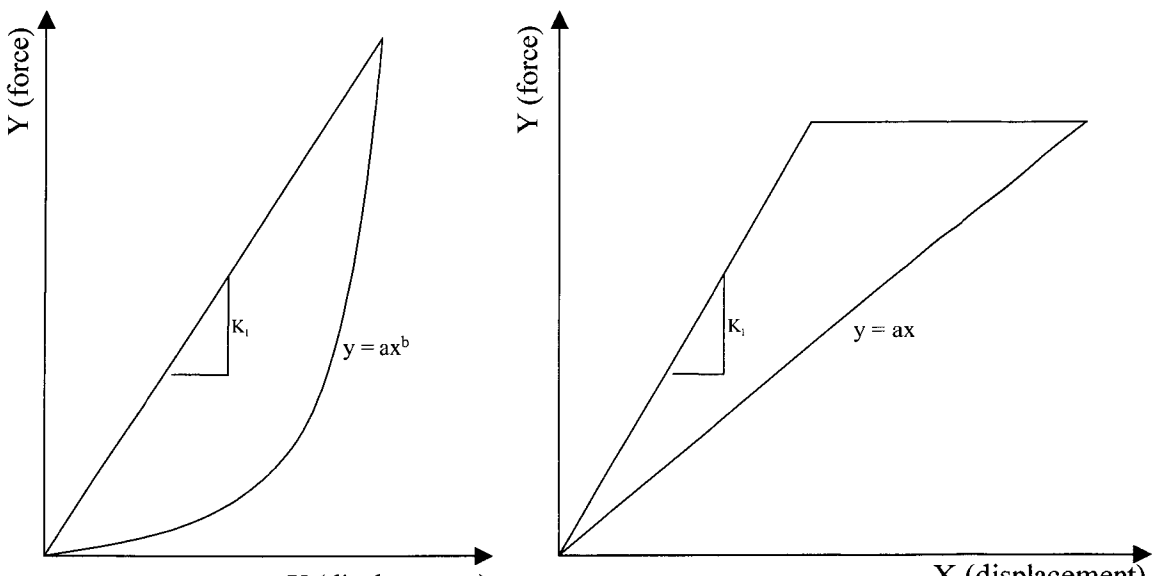
5.1 Energy Dissipation for Inelastic Contact

The concept behind the new energy dissipation algorithm is as follows: when rock falls on a slope, at low impact velocities, there is energy consumed in the plastic impact process. When a rock impacts a slope at higher velocities, there are microcracks created in the rock and

these may result in rock fragmentation. The energy consumed in the plastic impact process and fragmentation is accounted for in the UDM by using the energy dissipation algorithm.

In this research, an elastic-plastic-power normal stiffness model is constructed to account for energy dissipation during impact as shown in Figure 26. The initial normal stiffness (k_1) is the normal stiffness obtained from the routine calibration program. When the normal force reaches point A, the model enters a perfectly plastic process. The normal force at the transition point A, is set by the user. When the contacting particles begin to move away from each other at point B, the normal force follows a power function ($y = ax^b$), with an exponent b set by the user. Point B is determined by the PFC2D calculation cycle and the parameter a is automatically selected such that the power relationship extends from point B back to the origin.

If A is given a high value such that the plastic deformation is prevented, when two contacting particles begin to move away from each other, the normal force follows the power function defined by $y = ax^b$. The UDM shown in Figure 26 then becomes the elastic-power function model shown in Figure 27a. If the exponent of the power function is set to 1.0, the UDM becomes the triangular damping model shown in Figure 27b.



(a) Elastic-Power damping model (b) Triangle damping model

Figure 27 Different forms of elastic-perfectly plastic power function model

Figure 28 shows a model of one particle free falling on a wall to test the UDM. By adjusting A and b, different contact behavior can be implemented for the impact process as shown in Figure 29, Figure 30 and Figure 31.

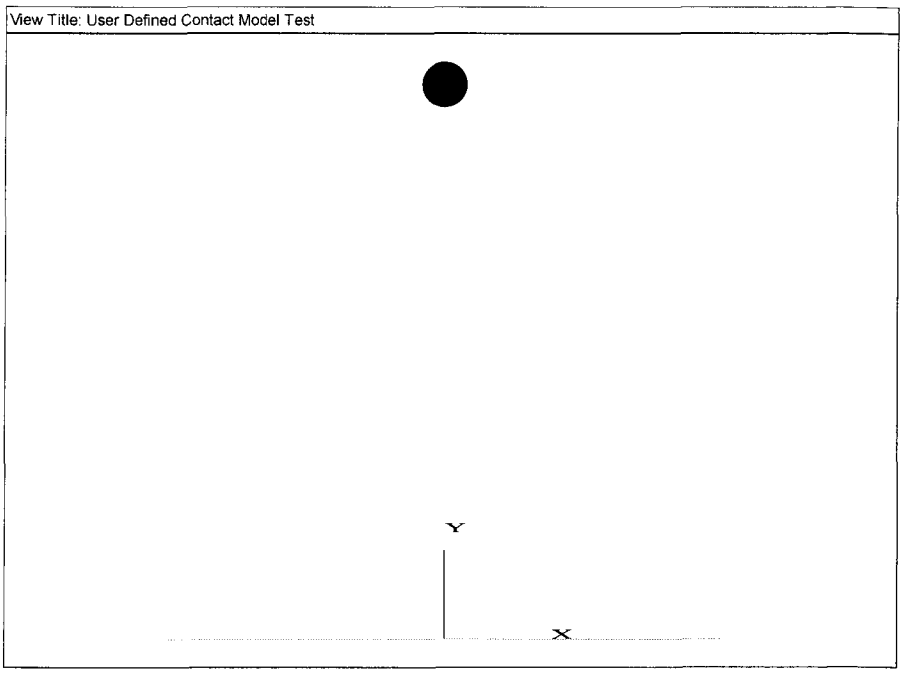


Figure 28 One particle falling on a wall to test UDM (radius: 0.04 m, K_n: 6.4e10 N/m, k_s: 9.6e9 N/m, friction: 0.6)

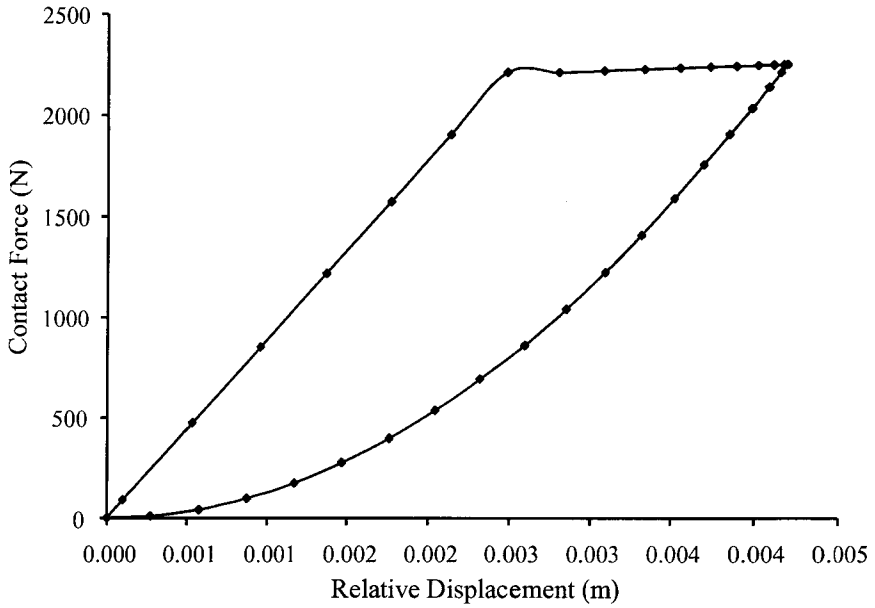


Figure 29 Particle impact on a wall using elastic-perfectly plastic power function damping model

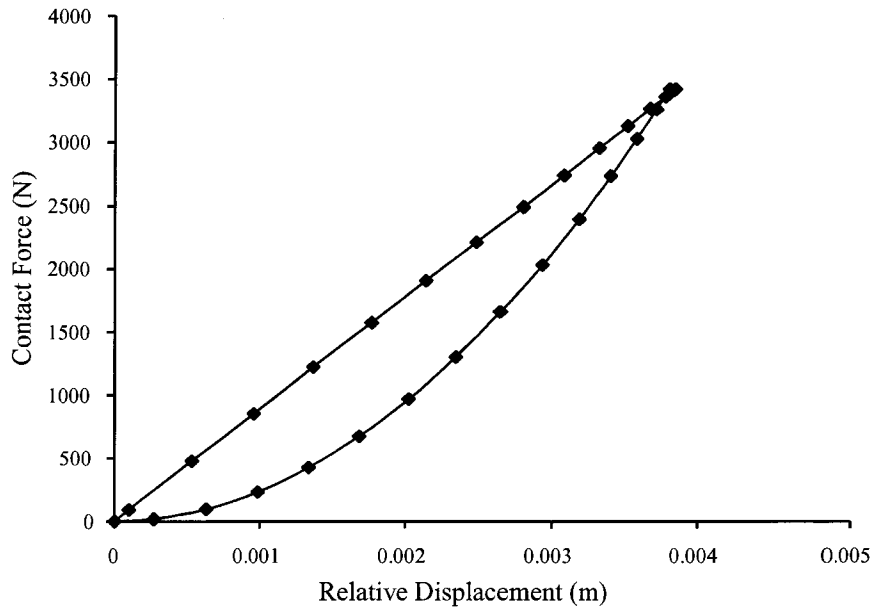


Figure 30 Particle impact on a wall using a high *transition force* and $y = ax^2$

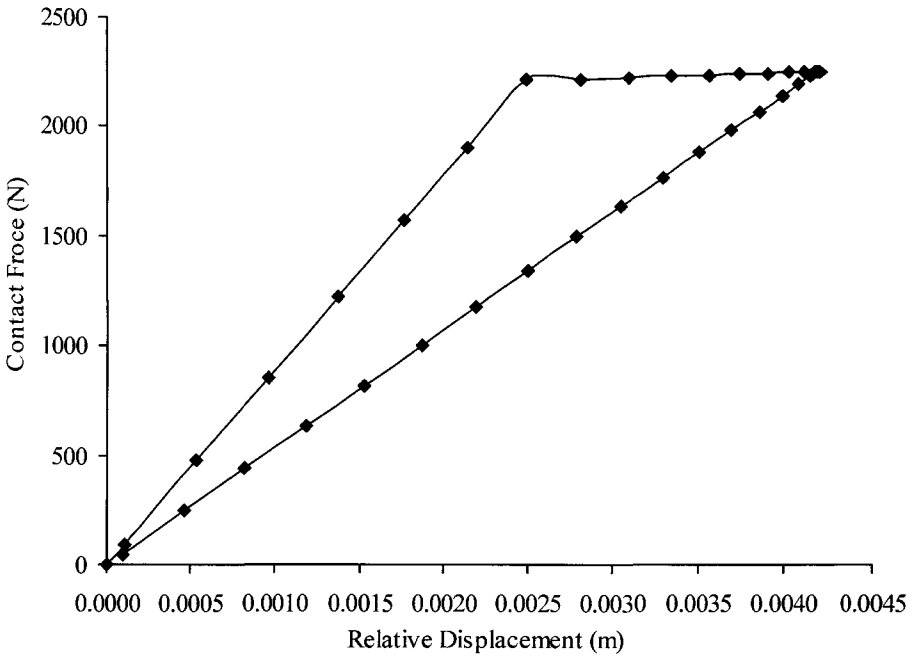


Figure 31 Particle impact on a wall using $y = ax$ (triangular model)

During modeling, it is found that the minimum normal restitution coefficient that a model with $y = ax$ (triangular model) can achieve is $\sqrt{0.5} = 0.7$ regardless of the value of A. This observation is consistent with the geometrical shape of triangle. Given that typical values for normal coefficients of restitution are less than 0.5, a triangular model cannot be used to simulate rockfall impact.

In Figure 29, Figure 30 and Figure 31 there are about 30 calculation cycles involved during the impact process as illustrated by the points corresponding to the results of each calculation cycle plotted on the curves. To follow a user-defined contact model properly it is important to ensure that at least 10 to 20 calculation cycles occur for each impact. Otherwise, the model may give unrealistic results.

For an elastic-power function model, the restitution coefficient can theoretically vary from 0 to 1 by adjusting the exponent of the power function damping model. Because the transition point from elastic to plastic behavior is set as an infinite value in the elastic-power function model and the exponent b is specified by the user, the value of a can be automatically determined by the UDM via the equation

$$a = y_m / x_m^b$$ Equation 23

Where x_m and y_m are the coordinates of the point at which the relative displacement reaches its maximum value and the relative velocity between two particles is zero. Note, a similar algorithm is used to determine the value a for the elastic-perfectly plastic power function model.

The elastic-power function model is the simplest model among these models. However, a numerical problem may arise when using this model. In the following test, an elastic-power function model ($b = 2$) was used to simulate the free falling of two particles on a wall.

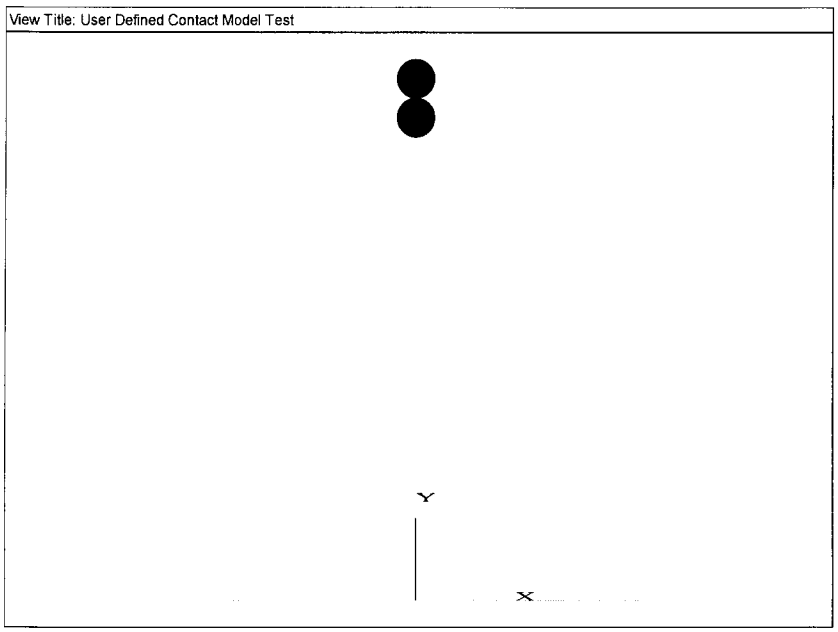


Figure 32 Free falling of two bonded particles to test elastic-power function model (Ball radius: 0.04 m, K_n: 6.4e10 N/m, k_s: 9.6e9 N/m, friction: 0.6, wall K_n: 1e6 N/m)

During the test, it was found that after bouncing back from the wall, the particles traveled at a velocity that was orders of magnitude higher than the impact velocity. The velocity history of one particle is shown in Figure 33. From this figure, it is seen that the velocity jumped from 0 to 60000 m/s at the time step 4680.

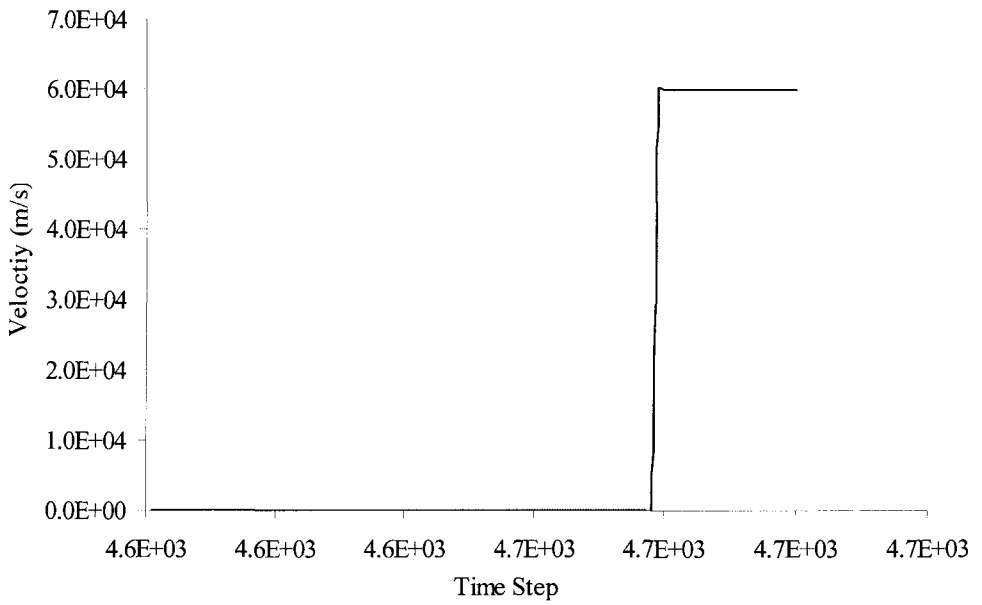


Figure 33 Relationship between velocity and time step

Figure 34 shows that the contact force jumped up to about 1.52e9 N at the time step 4680. The abrupt change of velocity and contact force occurs within one calculation cycle.

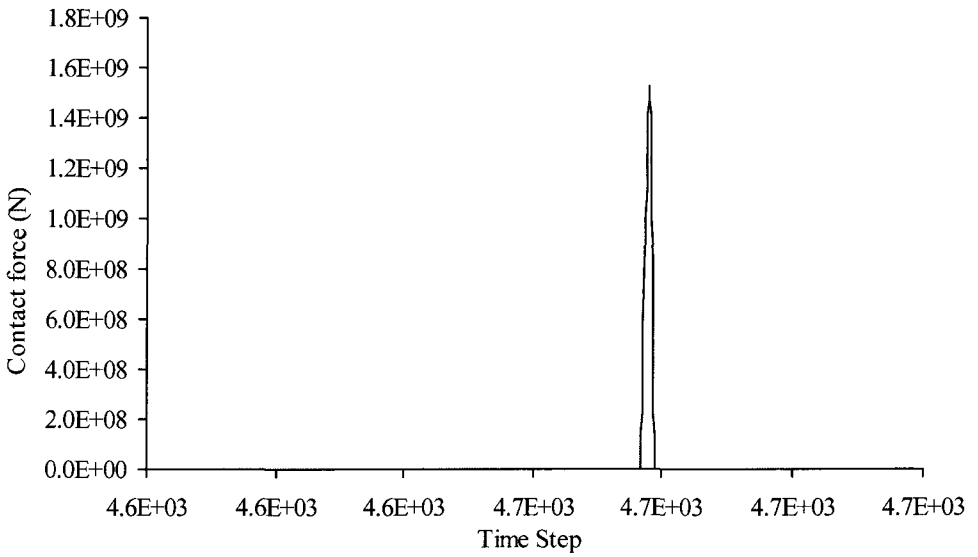


Figure 34 Relationship between contact force and time step (ball 1)

Since the UDM is constructed using Visual C++, the built-in debug function can be used to check the contact force and calculation cycle number. The debug results showed that the relative displacement of one particle at the first calculation cycle is very large (0.023m). This causes a very high contact force when applying the elastic-power function model because the

UDM adjusts the contact force according to the displacement obtained from Law of Motion. If the displacement for the calculation cycle is high, the contact force calculated by elastic-power function model will be also very high, because the loading segment of the model is actually unlimited. The results obtained by this model are unrealistic since the kinetic energy after the bounce is much higher than that before impact. This is a similar problem to that observed with the hysteric-damping model.

The introduction of the UDM that does not limit the contact force, if displacements are 'too large' in the initial time step, may cause a solution stability problem. Even though the UDM works well for a single particle or a group of particles without bonds, for instance, the tests performed by Shimizu (2002), it can cause large overlap displacement at the initial step, which results in solution stability problems. Reducing the time step does not solve this problem. For instance, increasing the time step from 1e-4 to 1e-6 gave the same results and no tendency to mitigate this problem. Further increasing the time step will do nothing but lengthen the computation time especially while using the energy tracing function and history of Fish items. The reason behind this problem is unclear. A possible explanation is that, the introduction of UDM overrides the original time step, and also makes the time step out of the control of the PFC2D command. In summary, the elastic-power function model is not suitable to simulate impact event because of the same shortcoming of hysteretic damping model.

The elastic-perfectly plastic power function model can avoid the solution instability problem. The introduction of a force limit or the *transition force* prevents unreasonably large contact forces. By using the same model (shown in Figure 32) and an elastic-perfectly plastic power function model (Figure 26) with b = 8, the solution instability problem is successfully overcome.

From Figure 35, it can be observed that the velocity change after impact is controlled within a reasonable range. For this model, the equivalent normal coefficient of restitution can be calculated using the velocity before and after impact. For this model, the coefficient of restitution is about 0.42.

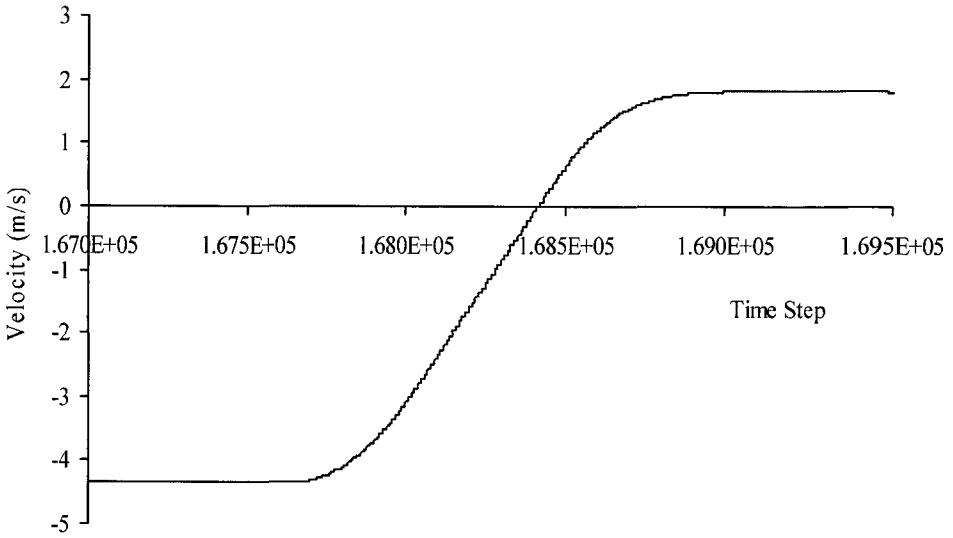


Figure 35 Velocity vs. time step (lower particle)

From Figure 36 and Figure 37, it can be seen that the contact force is limited within 4000 N during the impact process. In Figure 36, note there are thousands of calculation cycles during the impact process, and also thousands of calculation cycles before the contact force reaches the *transition force*. Compared with Figure 34, obviously, the introduction of a new UDM changes the time step. The change of time step is beyond the user's control. In some models, the initial relative displacement is so large due to the change of time step that the initial contact force can arrive at the *transition force* within one calculation cycle. In this case, the elastic-plastic-power function model can still guarantee enough calculation cycles during the impact process, because the maximum contact force is less than the *transition force*.

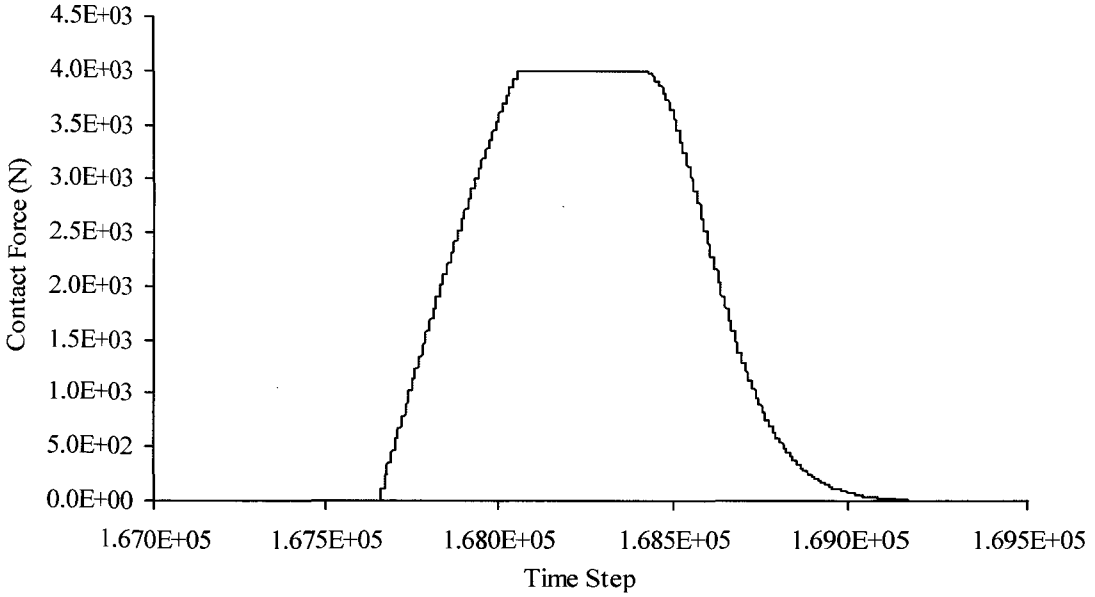


Figure 36 Wall-particle contact force vs. time step (lower particle)

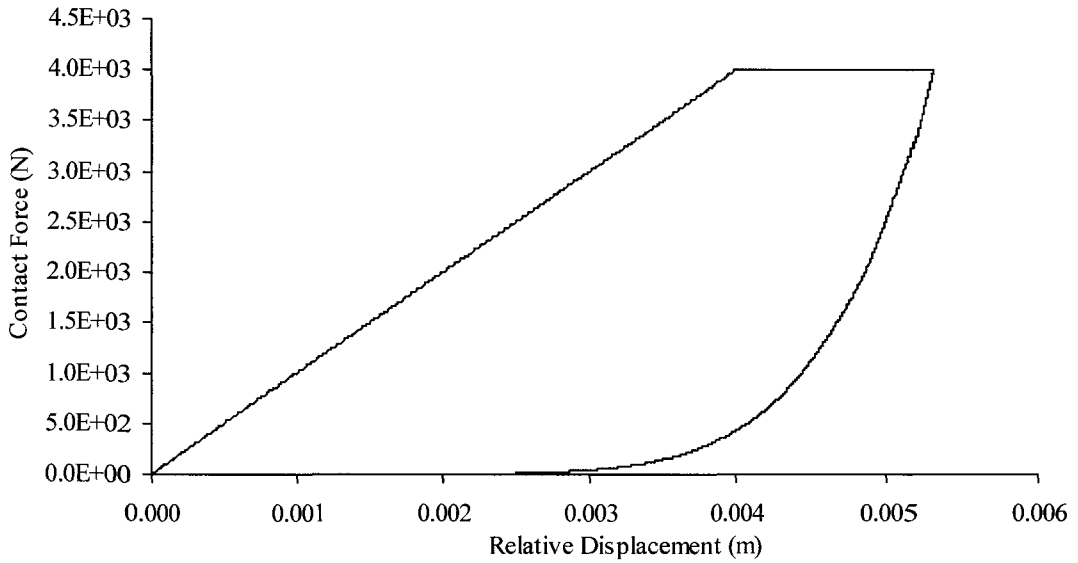


Figure 37 Wall-particle contact force vs. relative displacement with Exponent $b = 8$ (lower particle)

5.2 Damping and Normal Restitution Coefficient

A schematic plot of the energy dissipation that occurs with the elastic-perfectly plastic power function model is shown in Figure 38.

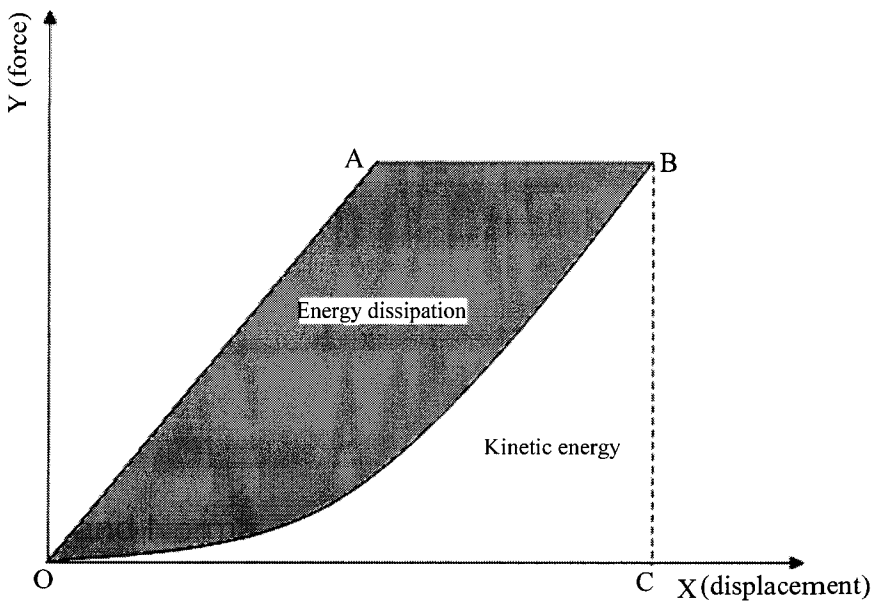


Figure 38 Schematic of energy dissipation of elastic-perfectly plastic power function model

In the Figure 38, the area of *OABC* is the total work that the contact force does during the impact process. Since the contact force follows the power function from *B* to *O* during the bounce back, the work that is transferred into kinetic energy within the particle after bouncing is the portion of the area of *OBC*. The energy represented by the area *OAB* is numerically removed from the PFC2D model by the UDM. This represents the energy loss that occurs during the impact process.

By using different power functions, the elastic-perfectly plastic power function model can represent different damping results. Therefore, the power function ($y = ax^b$) actually defines the damping results. For instance, the larger the *exponent b* of the power function, the less kinetic energy available after the bounce. The ratios of the areas *OBC* and *OABC* can be used to define the normal restitution coefficient:

$$Normal\ restitution\ coefficient = \sqrt{\frac{Area\,(OBC)}{Area\,(OABC)}} \qquad \text{Equation 24}$$

The conventional definition of restitution coefficients is based on ratios of impact and rebound velocities, not energies. Hence, the square root of the ratio of energies is used in Equation 24 to be consistent with this definition.

5.3 Relationship Between Normal Restitution Coefficient and UDM

A general relationship between the *exponent*, *transition force* and the normal restitution coefficient can be established. A schematic of the elastic-perfectly plastic power function model working at different impact velocities is shown in Figure 39. Higher impact velocities result in larger maximum relative displacements values (including more plastic deformation).

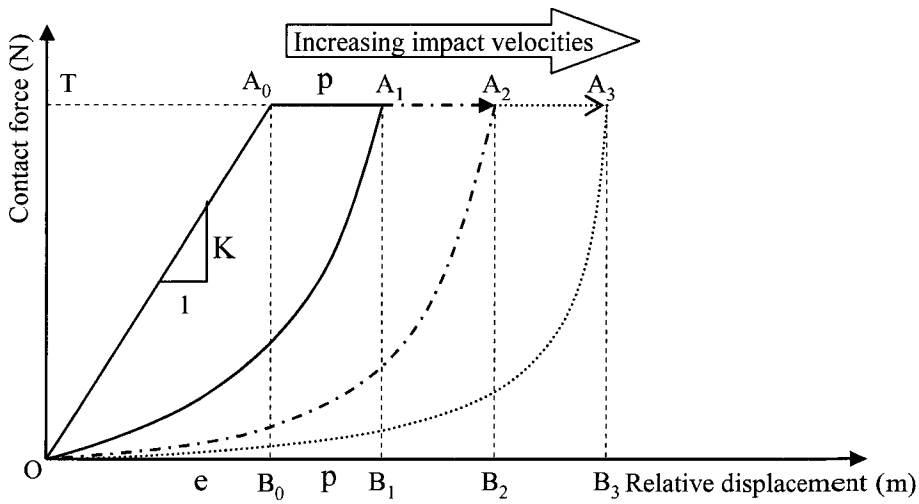


Figure 39 Schematic of the elastic-perfectly plastic power function model working at different impact velocities

In Figure 39, T is the *transition force*, and e and p are the elastic and plastic displacements (length of OB_0 and B_0B_1 or A_0A_1). The points A_1, A_2, and A_3 are the possible coordinates at which the UDM reaches its maximum relative displacement.

The area of $OA_0A_1B_1$ and OA_1B_1 in Figure 39 can be calculated as follows.

$$Area(OA_0A_1B_1) = \frac{e \cdot K}{2}(e + 2p) \qquad \text{Equation 25}$$

$$Area\,(OA_1B_1) = \int_0^{(e+p)} f(x)dx \qquad \text{Equation 26}$$

Where, $f(x)$ is the power function of the elastic-perfectly plastic power function model, $f(x) = ax^b$. In the above figure, since the coordinate of point A_1 is given, a can be determined as follows (using Equation 27):

$$a = \frac{e \cdot K}{(e+p)^b} \qquad \text{Equation 27}$$

Then

$$Area(OA_1B_1) = \int_0^{(e+p)} ax^b dx = \frac{e \cdot K(e+p)}{b+1} \qquad \text{Equation 28}$$

$$Normal\ restitution\ coefficient = \sqrt{\frac{Area(OA_1B_1)}{Area(OA_0A_1B_1)}} = \sqrt{\frac{2}{b+1} \times \frac{\frac{e}{p}+1}{\frac{e}{p}+2}} \qquad \text{Equation 29}$$

Assume g = e/p,

$$Normal\ restitution\ coefficient = \sqrt{\frac{2}{b+1}} \times \sqrt{\frac{g+1}{g+2}} \qquad \text{Equation 30}$$

From Equation 30, it is found that, the normal restitution coefficient depends on the *exponent* of the power function and the ratio of e to p. If the value of g is large, the second term in Equation 30 will be close to unity, and the normal restitution coefficient will be largely dependent only on the power function *exponent*.

Using Equation 30, the normal restitution coefficient is plotted versus *exponent* at varying amounts of plastic deformation in Figure 40 assuming K=1.0e6 N/m and *transition force* = 4.0e3 N (elastic deformation e = 0.004 m). Figure 40 shows that the normal restitution coefficient drops as the *exponent b* increases. Furthermore, the *exponent* needs to be large, in the range of say 10 to 50, to represent a typical range of normal restitution coefficients that apply for rockfall impacts. Because the normal restitution coefficient depends on the power function *exponent* and the ratio of d to e, the general trend shown in Figure 40 applies for any normal stiffness and *transition force*.

The normal restitution coefficient changes slightly depending on the amount of plastic deformation that occurs relative to the elastic deformation. This effect is more pronounced

when the *exponent* is small as illustrated in Figure 40 by the curves corresponding to varying values of g. In a PFC2D impact model with a given normal stiffness and *transition force*, the point B is determined by the particle impact velocity. If the impact velocity (or particle mass) is increased, the amount of plastic deformation will be larger, which means the ratio of e to p will decrease. Thus, the normal restitution coefficient will become smaller for higher impact velocities. The behavior is consistent with empirical evidence as discussed in Section 5.4.

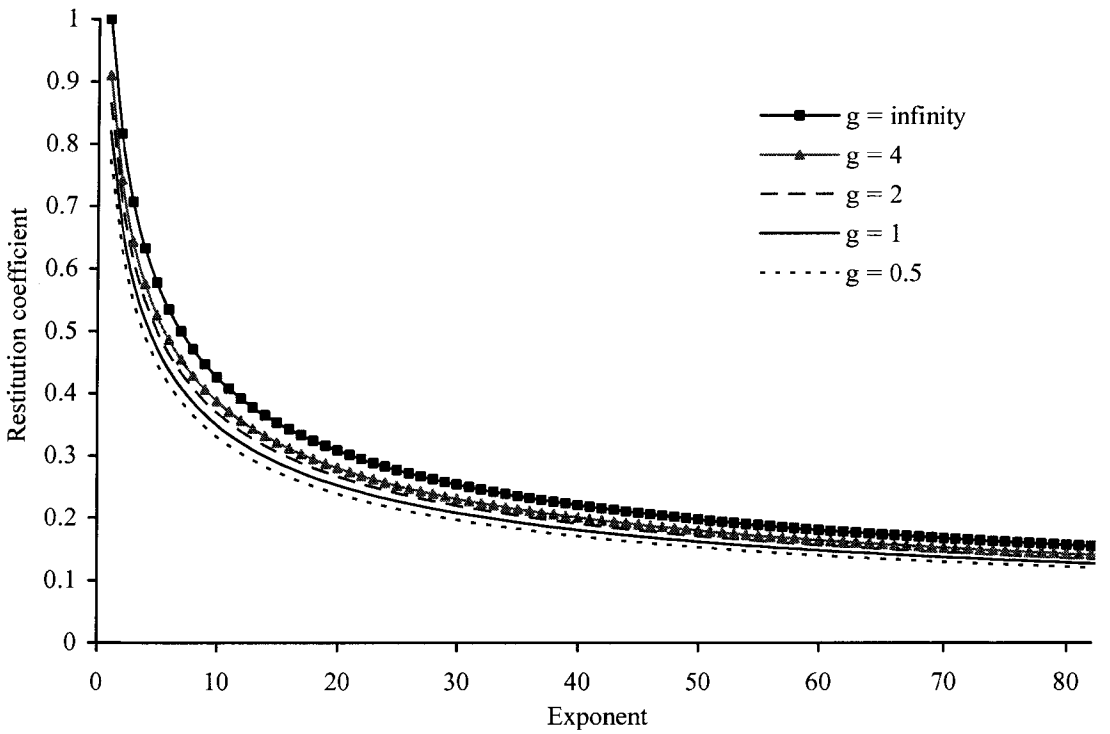


Figure 40 Normal restitution coefficient vs. *exponent* for varying amounts of plastic deformation (K = 1.0e6 N/m, *transition force* = 4.0e3 N)

By increasing the *transition force* at a given *exponent*, the plastic deformation decreases and the normal restitution coefficient increases. However, overdoing this may transfer the elastic-perfectly plastic power function model into an elastic power function model and may result in numerical stability problems. Figure 41 shows the elastic-perfectly plastic power model with different normal stiffnesses. Take a situation where the particle mass and impact velocity is the same for both contact stiffnesses, then the area under the two force-displacement curves is also the same.

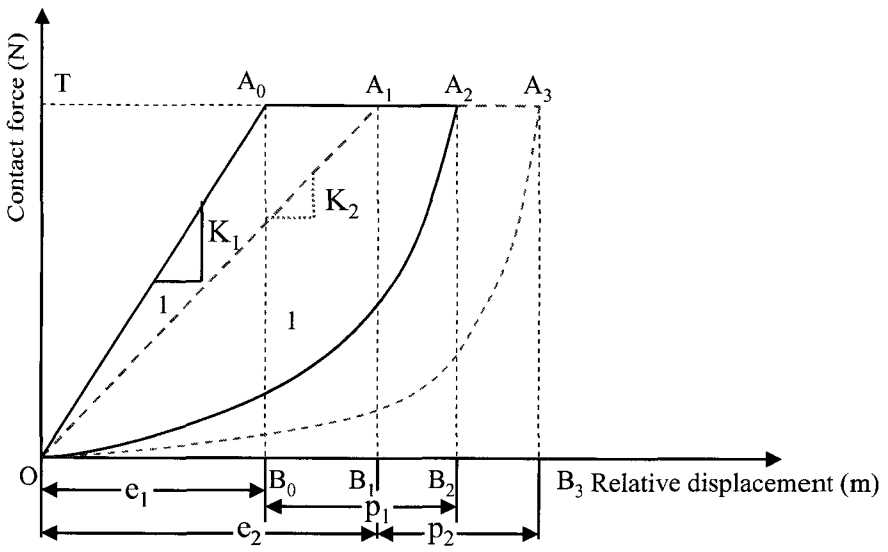


Figure 41 Elastic-perfectly plastic-power model with different contact stiffness at the same impact velocity, *transition force*, and *exponent*

In Figure 41, e_1, e_2, p_1, and p_2 represent the elastic and plastic deformations associated with the two stiffness values. The total kinetic energy is assumed the same for both models.

$$Area(OA_0A_2B_1) = Area(OA_1A_3B_3) \qquad \text{Equation 31}$$

Then,

$$Area(OA_0A_1) = Area(A_2A_3B_2B_3) \qquad \text{Equation 32}$$

After plastic deformation begins, the force is held constant (T), therefore the incremental elastic deformation that occurs with a softer contact model must equal twice the incremental plastic deformation.

$$Area(OA_0A_1) = Area(A_2A_3B_2B_3) \Rightarrow e_2 - e_1 = 2(e_2 - e_1 + p_2 - p_1) \qquad \text{Equation 33}$$

From Equation 33,

$$p_1 - p_2 = \frac{1}{2}(e_2 - e_1) \qquad \text{Equation 34}$$

In Figure 41, it can be found that $e_2 > e_1$, so $p_1 > p_2$. Taking $g_1 = e_1/p_1$ and $g_2 = e_2/p_2$ it is easy to show that $g_1 < g_2$. Therefore, according to Equation 30, the normal restitution coefficient obtained using a stiff contact is less than the normal restitution coefficient with a soft contact.

When the contact stiffness is increased without changing any other parameters, the normal restitution coefficient will decrease. To obtain the same normal restitution coefficient, the *transition force* needs to be increased to get proportionally more elastic displacement and less plastic displacement during an impact event. This effect is illustrated using Figure 42.

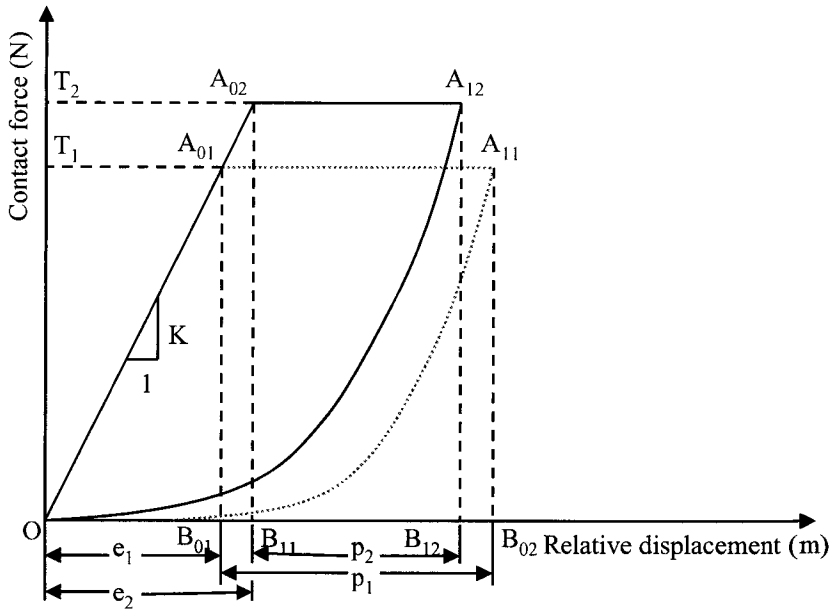


Figure 42 Elastic-perfectly plastic-power model with different *transition forces* and the same normal stiffness, *exponent*, and impact velocity

In Figure 42, the area of $OA_{01}A_{11}B_{02}$ and $OA_{02}A_{12}B_{12}$ should be the same since the impact velocity and hence total kinetic energy is the same.

In Figure 42, it can be found that $e_2 > e_1$ and $p_1 > p_2$. Taking $g_1 = e_1/p_1$ and $g_2 = e_2/p_2$ it is easy to show that $g_1 < g_2$. Therefore, according to Equation 30, the normal restitution coefficient obtained using a low *transition force* is less than the normal restitution coefficient with a high *transition force.*

5.4 Normal Restitution Coefficient and Impact Velocity

The results of the numerical simulation of rockfalls were compared with data from the RocScience (2005) website. The key data include normal restitution coefficient and scaling factor. The normal restitution coefficient depends on the rock type and slope conditions as

seen in the table of restitution coefficients found within the RocFall software (RocScience 2005). For normal impact, the normal restitution coefficient indicates how much kinetic energy is kept after bouncing. For the simple PFC2D model shown in the next chapter, the rock and the rock slope are both composed of Berea sandstone. The condition of the rock slope is clean and hard sandstone. According to the restitution coefficient data found in Rocfall (RocScience 2005), the normal restitution coefficient should be around 0.53.

The scaling factor decreases the normal coefficient of restitution as the impact velocity increases. Actually, this factor represents a transition from nearly elastic conditions at low velocities to highly inelastic conditions caused by the increased fragmentation of the rock and cratering of the slope surface at higher impact velocities (RocScience 2005).

The concept behind scaling the normal coefficient of restitution by impact velocity is that R_N depends on the impact velocity and the rock fragmentation caused by the impact. For example, at low velocities a rock may be expected to bounce, whereas at higher velocities the rock may imbed further into the ground before bouncing, or to start to break. In these cases, the value of R_N should be less at higher impact velocities. The scaling factor tries to capture these effects. The relationship between normal restitution coefficient and the scaled normal restitution coefficient is

$$Rn(scaled) = Rn \times scaling\ \ factor$$

$$Scaling\ \ factor = 1 \Big/ \left(1 + \left(\frac{V_{rock}}{V_{0.5}} \right)^2 \right) \qquad \text{Equation 35}$$

where

$V_{0.5}$ = velocity at which scaling factor = 0.5

V_{rock} = velocity of the rock, immediately before impact, measured normal to the surface.

The default value of the constant $V_{0.5}$ (9.144 m/s) is the metric equivalent of 30 ft/s, which is empirically derived (Pfeiffer and Bowen 1989)

The relationship between scaling factor and impact velocity is shown in Figure 43. The scaling factor reduces with increasing impact velocity. Using an unscaled normal restitution

coefficient of 0.53 and applying the scaling factor yields the relationship between bounce velocity and impact velocity shown in Figure 44.

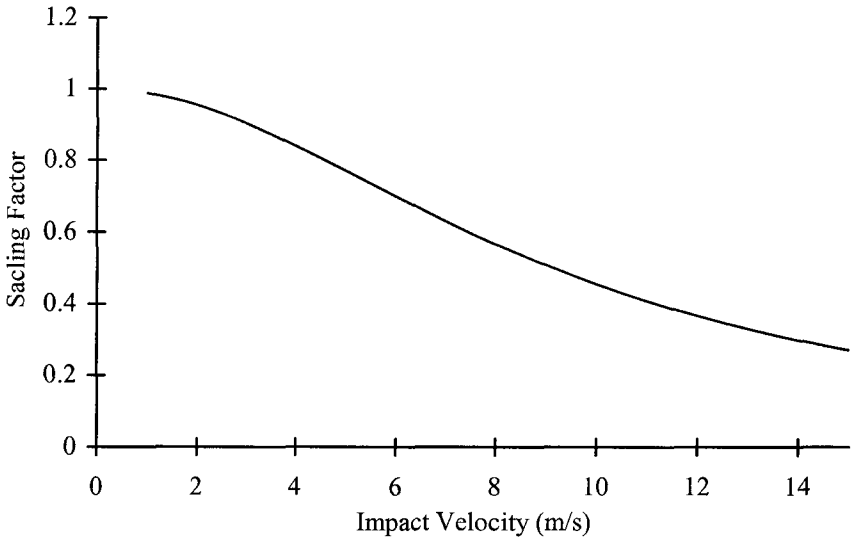


Figure 43 Relationship between scaling factor and impact velocity

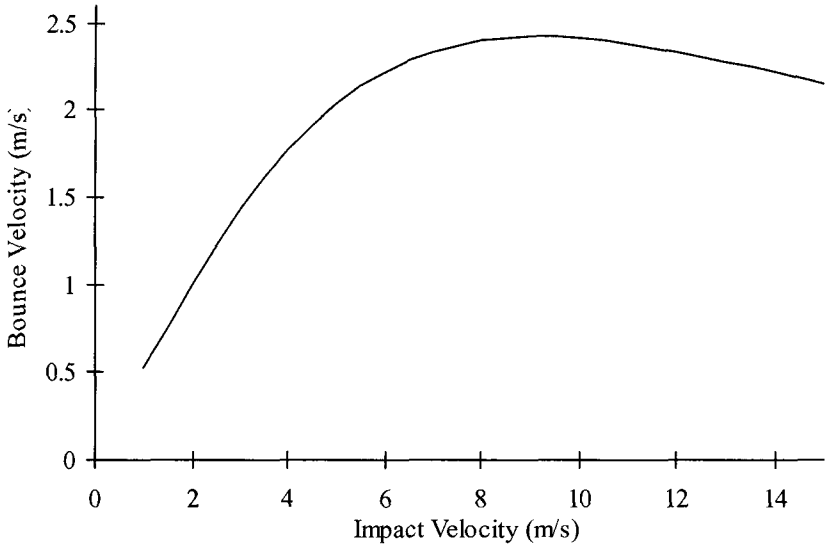


Figure 44 Relationship between bounce velocity and impact velocity

Figure 44 shows that at low impact velocity, the bounce velocity, although about one-half the impact velocity, will increase with increasing impact velocity. At higher impact velocity (around 9 m/s), the bounce velocity actually reduces with the increasing impact velocity. This reflects the influence of rock fragmentation and greater rock penetration into the slope as the impact energy gets higher.

6 PFC2D Modeling of Rock Impact

This chapter presents the results of a simple rockfall simulation to illustrate the application of the newly developed energy tracing functions and the new User-Defined Contact Model. The techniques used to construct the PFC2D model are presented and the model results are compared with literature data. Finally, some guidelines for simulating rock impact processes using the new UDM are given.

6.1 PFC2D Rock Impact Model

To illustrate the new UDM, a model of a rock free falling vertically onto a horizontal rock slope was simulated. The materials properties used to construct the model (rock and slope) were based on the published properties for Berea sandstone. The mechanical properties of the Berea sandstone according to Fakhimi (2002) are a uniaxial compressive strength of 40 MPa, Young's modulus of 16 GPa, Poisson's ratio of 0.28, and density of 2600 kg/m^3.

The PFC2D code comes with calibration programs using the Fish language. These were used to calibrate the microparameters for the PFC2D model to match the physical properties of Berea sandstone. In this research, the clump function was added into the routine calibration program. A clump is a group of particles in contact with each other. During a PFC2D calculation cycle, particles within a clump are treated as perfectly rigid bodies. The contacts within a clump are set as non-breakable contacts, which represents a non-breakable rock. The elastic-perfectly plastic power function model during the simulation only acts at contacts between the clump particles and a wall segment (not between particles within the clump).

The introduction of clump of particles to represent the falling rock eliminates the memory requirements needed to store contacts within the clump and it avoids the overhead of updating contact locations and associated vectors. The use of a clump greatly reduces computation time especially when energy-tracing functions and the history of fish items are used in the simulation. Another use of a clump during the simulation of a rockfall is to form different shapes for the falling rock. In most rockfall software, the rock treated as a lumped-mass model with an assumed circular shape. In this research, the rock was represented by a

clump with three particles. Both rotational and transitional movements can be reproduced in the simulation.

The PFC2D calibration procedure needed to determine the parameters needed for the particles is described in Appendix B. While the calibration procedure is based on a simulation of uniaxial compression tests involving thousands of particles the resulting microparameters determined from this calibration procedure were simply applied to the few particles used to make the clump representing the falling rock.

Figure 45 shows a numerical model including a clump composed of three particles. The model is designed to simulate normal impact on a horizontal surface (wall). Because the numerical rockfall is composed of three particles, a vertical line passing through the rock's center of mass may not coincide with the point(s) of contact during impact. In this case, rotational movement is created such as that shown in Figure 46.

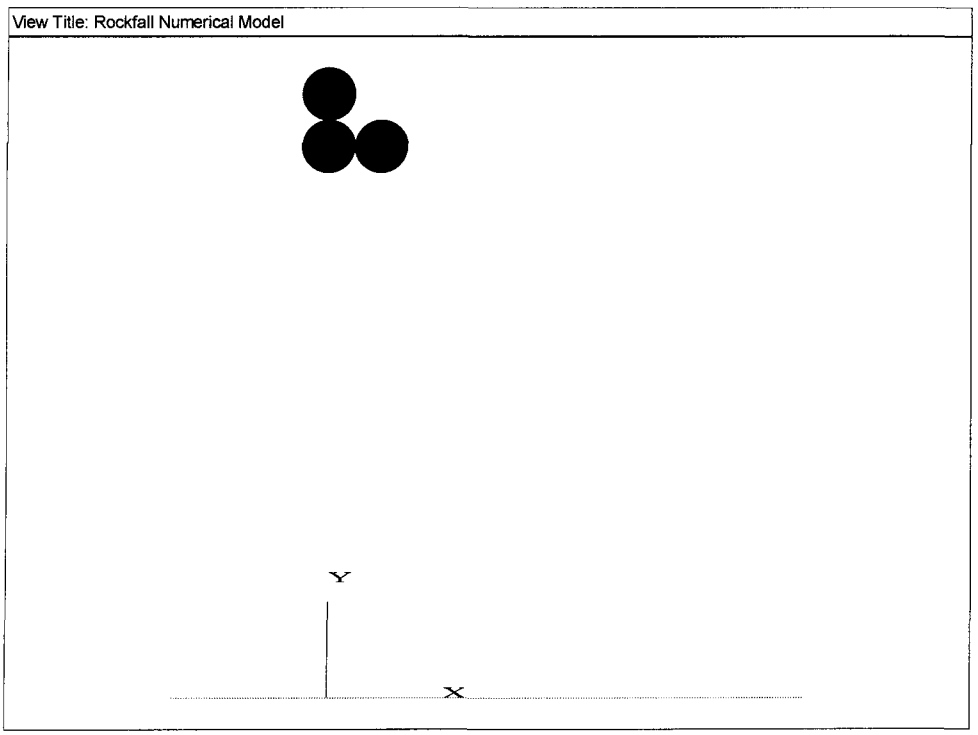


Figure 45 PFC2D model for rockfall

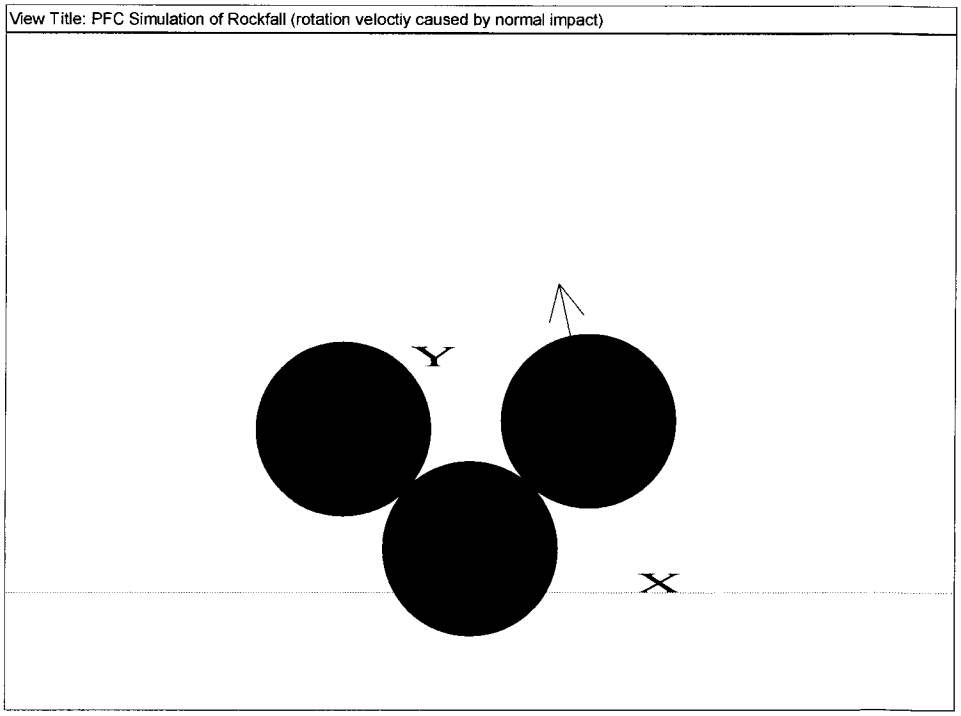


Figure 46 Rotation velocity caused by normal impact in PFC2D simulation (close-up)

The rotation movement of the numerical model cannot be calibrated with literature data from RocFall, since there is no counterpart in RocFall (RocScience 2005). The total kinetic energy including translational kinetic energy and rotational kinetic energy can be calculated, and then calibrated with the total kinetic energy obtained from Figure 44.

The microparameters used in the model are: density: 2600 kg/m^3, normal stiffness: 6.4e10 N/m, shear stiffness: 9.6e9 N/m, friction coefficient: 0.5, contact bond strength: normal 1e9 N, shear 1e10 N. The rock in the model is represented by a clump composed of three particles in contact. The microparameters normal stiffness, shear stiffness, and friction coefficient are the same as those obtained from the calibration procedure using the model of uniaxial compression (Appendix B).

The shape of the numerical model can be arbitrary and the influence of the different shapes for the falling rock is discussed later. During the modeling process, numerical rock sample was given a zero initial velocity and it was allowed to free fall under the action of gravity on the wall. To simplify interpretation of the modeling results, only a simple horizontal slope

was simulated. Energy-tracing functions were used to track kinetic energy so that the results of the simulation can be compared with literature results.

6.2 Calibration of UDM

A schematic of the PFC2D model calibration using literature data is shown in Figure 47. In the calibration process, the radius of particles determines the total mass and size of the rock and the particle radii should be set according to the desired size or mass of the rockfall to be simulated. The goal of the modeling was to match data plotted in Figure 44. The kinetic energy of the modeled rockfall was obtained from the energy-tracing functions. By converting the kinetic energy into rock velocity before and after impact, the normal restitution coefficient was determined (Equation 24). This value can then be compared to the target value or empirical values.

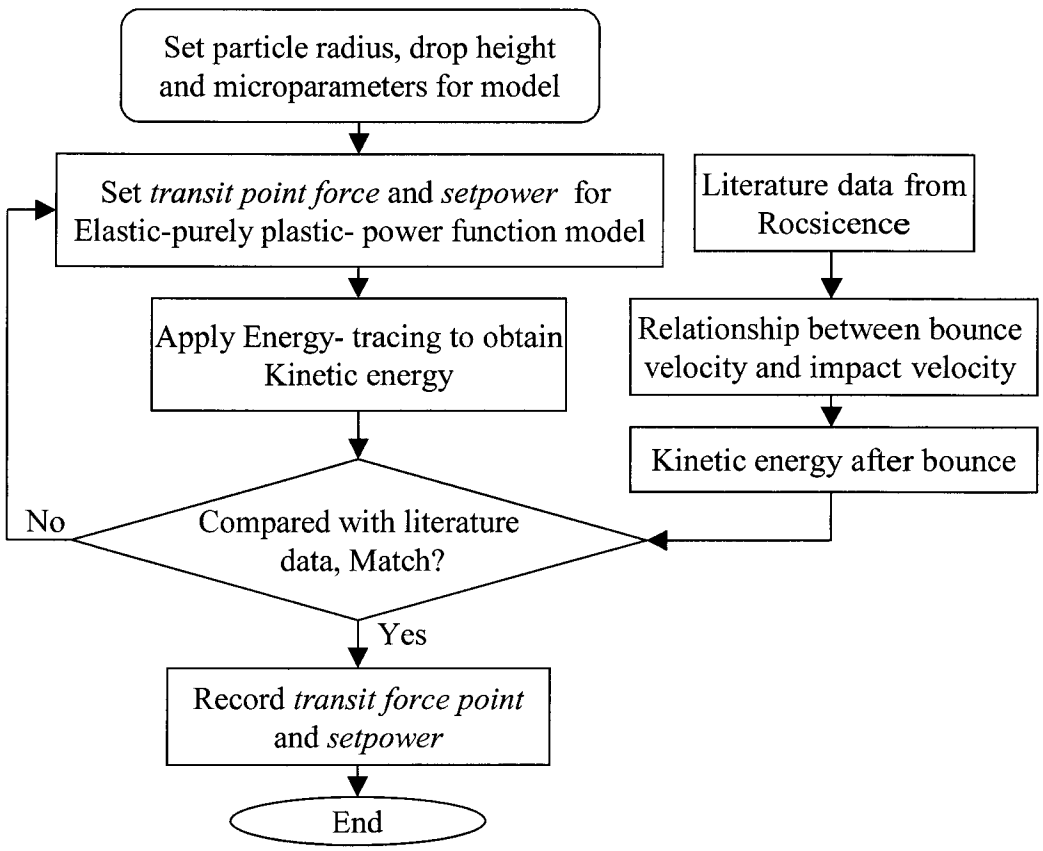


Figure 47 Schematic of PFC2D modeling calibration with literature data

In most rockfall software such as Rocfall (RocScience 2005), a restitution coefficient (normal and tangential direction) is used to calculate normal and tangent velocity of a rock after impact. The restitution coefficients are assumed to consider all relevant impact characteristics such as sliding, deformation and the transformation of rotational moments into translational moments (Schweigl et al. 2003). However, it is impossible to calculate rotational velocity by using the restitution coefficients. For instance, normal impact may result in a tangential bounce velocity, but tangential velocity cannot be calculated if the restitution coefficient is used. However, rotational velocity is important for rockfall simulation in that it affects not only the trajectory of rockfall but also run-out distance, both of which must be taken into account while designing a rockfall mitigation project.

Accurate prediction of rockfalls is practically unrealistic (Warren 1998). Variability in slope geometry, poorly defined initial conditions, and unknown material properties make accurate prediction of rockfalls very difficult. In reality, the kinetic energy of falling rock can be indirectly obtained by using a video camera to record the trajectory and speed of a rock. The advantage of the energy matching method is that it permits various modes of movement given the specific kinetic energy, which is closer to reality.

Two parameters are needed for the elastic-perfectly plastic power function model: *transition force* and *exponent*. The *transition force* is the normal contact force at which the model transfers from elastic response to perfectly plastic deformation while undergoing compression. The *exponent* for power function largely determines the damping effect of the model.

If the *transition force* is too small, some particles may penetrate the wall and the contact between the particles and wall will be destroyed. On the other hand, if the *transition force* is too large, the elastic-perfectly plastic power function model will become an elastic power function model, which may result in numerical instability as discussed before. The above cases should be avoided by giving a suitable value for the *transition force*.

The energy losses that occur during impact mainly depend on the *exponent*. However, once the *exponent* is specified, a slight adjustment of the *transition force* can still change the

damping result in a small range, thus making the model match physical rockfall more precisely.

6.2.1 Exponent of Elastic-Perfectly Plastic Power Function Model

The *exponent* was found to depend on the rock impact velocity, and is almost independent of other factors such as particle radius. The numerical model shown in Figure 45 (each particle with the same radius) was used to drop the rock from different heights to simulate different impact velocities. Different models were constructed with different particle sizes but constant normal contact stiffness (6.4e10 N/m). To match the data shown in Figure 44, the *exponent* needs to increase with higher impact velocities. The relationship between the UDM *exponent* and impact velocity (particle radius = 0.04 m) is shown in Figure 48. For the numerical samples with different radii, the relationship between UDM *exponent* and impact velocity needed to match the data in Figure 44 is shown in Figure 49, Figure 50, Figure 51, and Figure 52 (relevant data are shown in Appendix C).

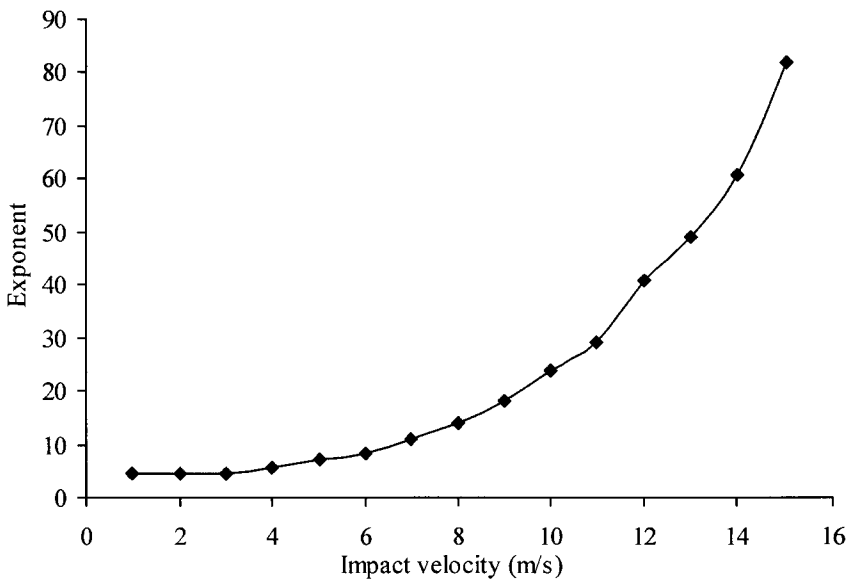


Figure 48 UDM *exponent* of vs. impact velocity (r = 0.04 m, *transition force* = 3.5e3 N)

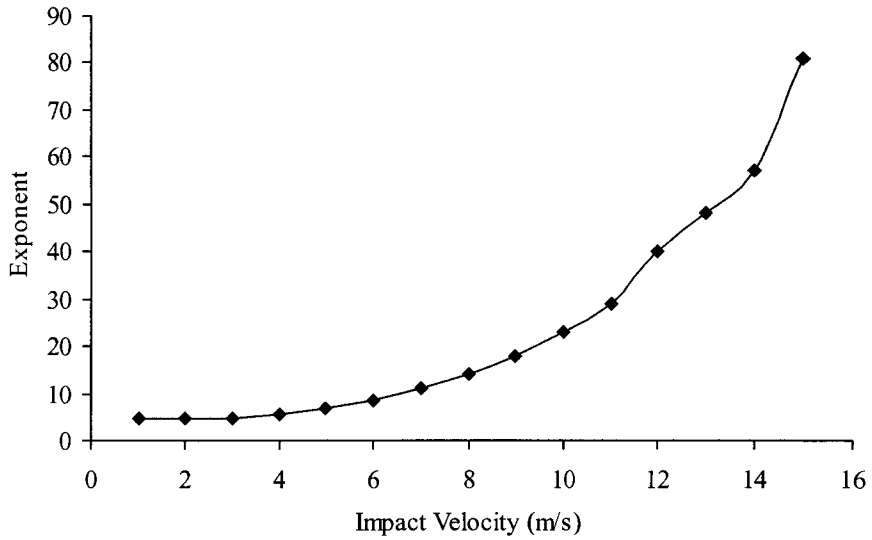


Figure 49 UDM *exponent* vs. impact velocity (r = 0.14 m, *transition force* = 4.5e4 N)

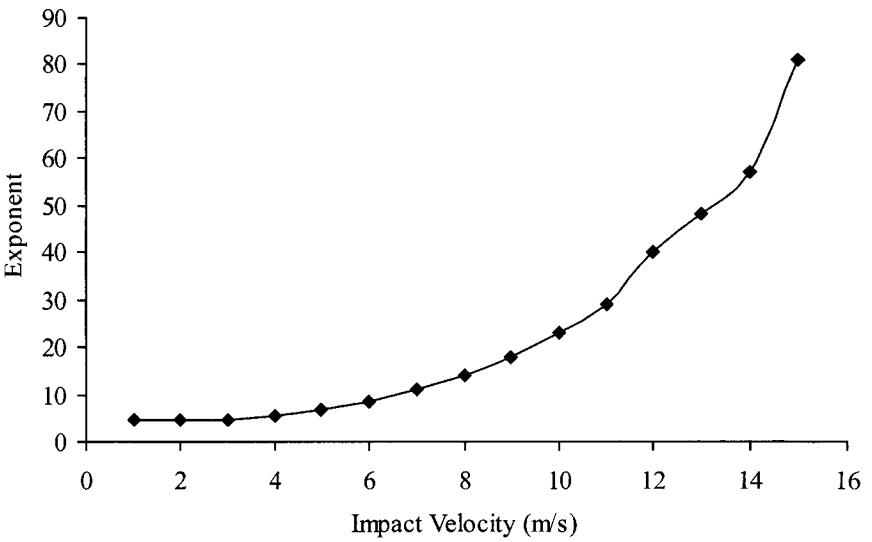


Figure 50 UDM *exponent* vs. impact velocity (r = 0.20 m, *transition force* = 2.5e5 N)

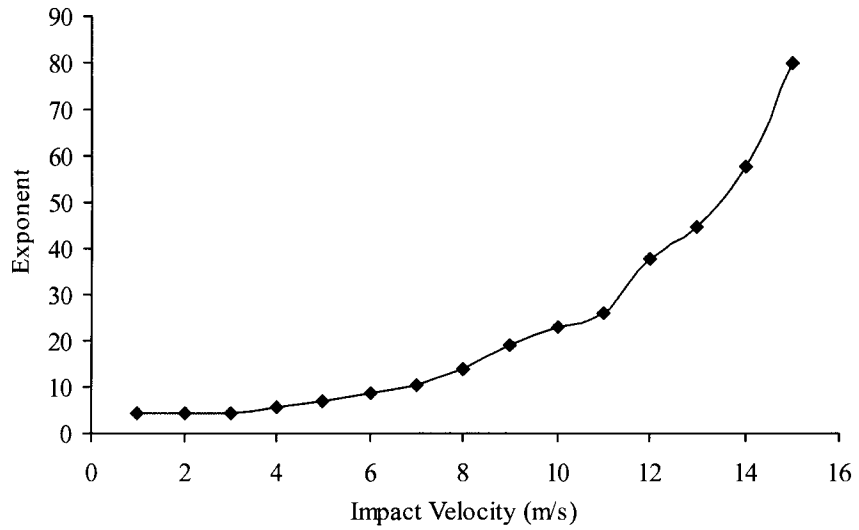


Figure 51 UDM *exponent* vs. impact velocity (r = 0.25 m, *transition force* = 5.3e5 N)

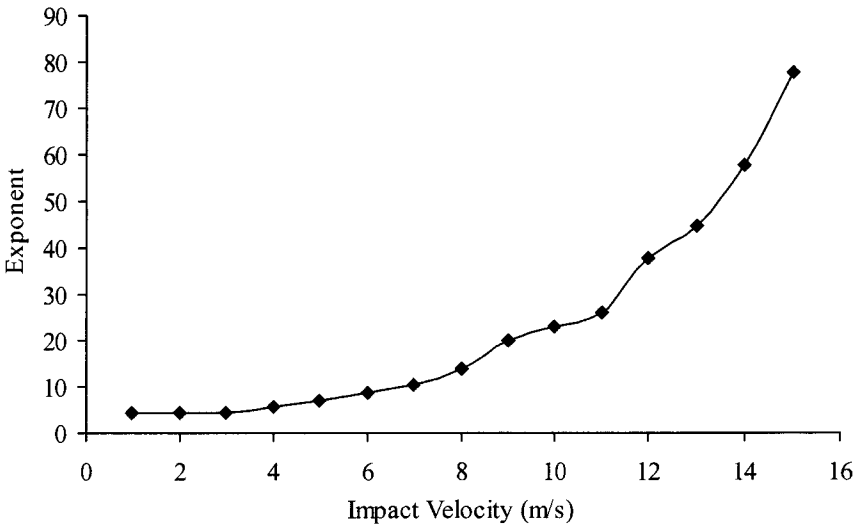


Figure 52 UDM *exponent* vs. impact velocity (r = 0.30 m, *transition force* = 8e5 N)

From the above figures, it is observed that the *exponent* follows a similar relationship with the increasing impact velocity no matter what the particle radii are. If all the data are plotted together, this tendency is more distinguished as seen in Figure 53.

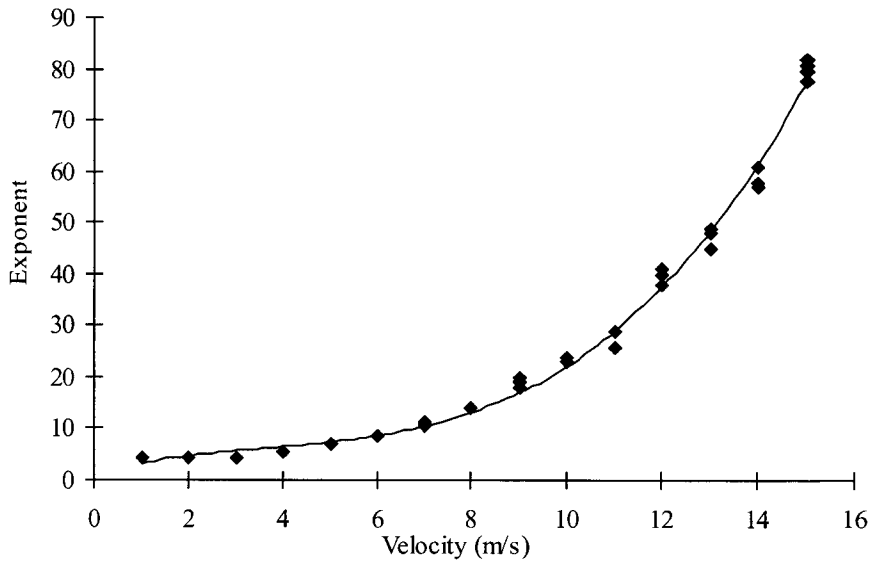


Figure 53 UDM *exponent* vs. impact velocity (various radii)

Figure 53 shows that:

- The plot of *exponent* versus impact velocity (for K_n = 6.4e10 N/m) follows the best-fit exponential function:

$$y = 0.0429x^3 - 0.4744x^2 + 2.5508x + 1.1489 \qquad \text{Equation 36}$$

- The radius of the PFC2D particles does not change this tendency.

Given prior knowledge of the desired normal restitution coefficient and the probable impact velocity, Figure 53 can be used to select an appropriate value of the *exponent b* to use in the UDM.

Actually, it would be more efficient if a renovated UDM were constructed to choose automatically the exponent based on impact velocity by using Equation 36.

A recommendation arising from this research is that an even better UDM for modeling rock impact processes during rockfalls would be to use the actual impact velocity from the model to select the exponent for each impact event. Although the normal restitution coefficient automatically drops as the impact velocity increases when using the newly developed UDM, the magnitude of the drop is not sufficient to match the empirically observed trend specified by Equation 35.

6.2.2 Transition Force of Elastic-Perfectly Plastic Power Function Model

The *transition force* dictates the transition from elastic response to perfectly plastic deformation while undergoing compression. During free-fall contact between two particles or between a particle and a wall, the rate at which compressive force develops depends on the normal stiffness. The normal stiffness assigned to the particles is meant to represent the elastic stiffness or modulus of elasticity of the rock in the model and is determined from the calibration procedure outlined in Appendix B. With a low stiffness, the contact force develops more slowly and larger relative displacements will ultimately occur during an impact event. Particle size is also important. A larger particle has higher mass and will result in generation of higher contact force (and higher relative displacement) during an impact event as it is slowed to a stop. The work or energy required to stop a large particle is more than for a small particle.

One approach to estimating an appropriate *transition force* to use in the UDM is to base the value on the uniaxial compressive strength of the rock. For a rock under compressive loading, the transition to non-elastic rock behavior begins to occur at approximately 75% of the uniaxial compressive strength σ_c. Assuming the contact force is carried over a cross-sectional area of the particle, the equivalent force at the transition to plastic response can be estimated from:

$$transition\ force, T = 0.75 * \pi(sr)^2 * \sigma_c \qquad \text{Equation 37}$$

Where r is the radius of the particle, and s is a scaling factor ($s < 1$) that reduces the radius to an effective radius carrying the contact load. Selection of s is somewhat arbitrary although by performing a number of rock impact simulations and varying the transition force to match the data shown in Figure 44 the value of effective radius that works in the PFC2D model can be estimated.

For example, Figure 54 presents the results from a number of numerical models in which the *transition force* was found for models using different particle sizes such that the rebound velocities matched the data in Figure 44. The best-fit relationship using $T = constant * r^2$ can be used to back-calculate s assuming Equation 37 is valid and assuming the model represents Berea sandstone (σ_c = 40 MPa). The best fit-relationship is plotted on Figure 54 and using

Equation 37, a value of s = 0.3 is obtained. This implies that the effective particle radius carrying the load and involved at the transition between elastic and plastic deformation is about 1/3 of the particle radius.

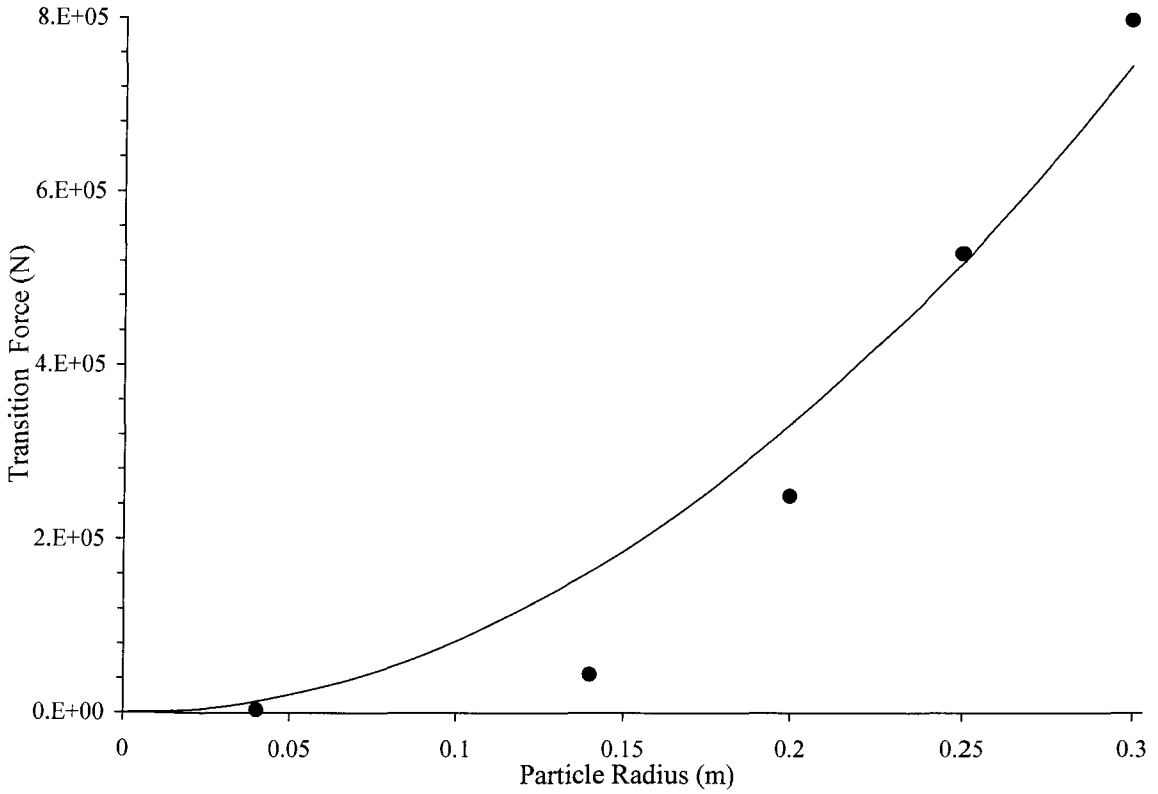


Figure 54 *Transition force* vs. particle radius (K_n = 6.4e10 N/m)

The same calibration process was used for PFC2D impact models with different normal stiffness values (representing different rock types) and the results are shown in Figure 55 with the relevant data given in Appendix C. The *transition force* increases with the increasing particle size and the numerical models with higher normal stiffness require higher *transition force* to match the empirical rockfall data plotted on Figure 44. These results are consistent with the analytical evaluation of the UDM presented in Section 5.3.

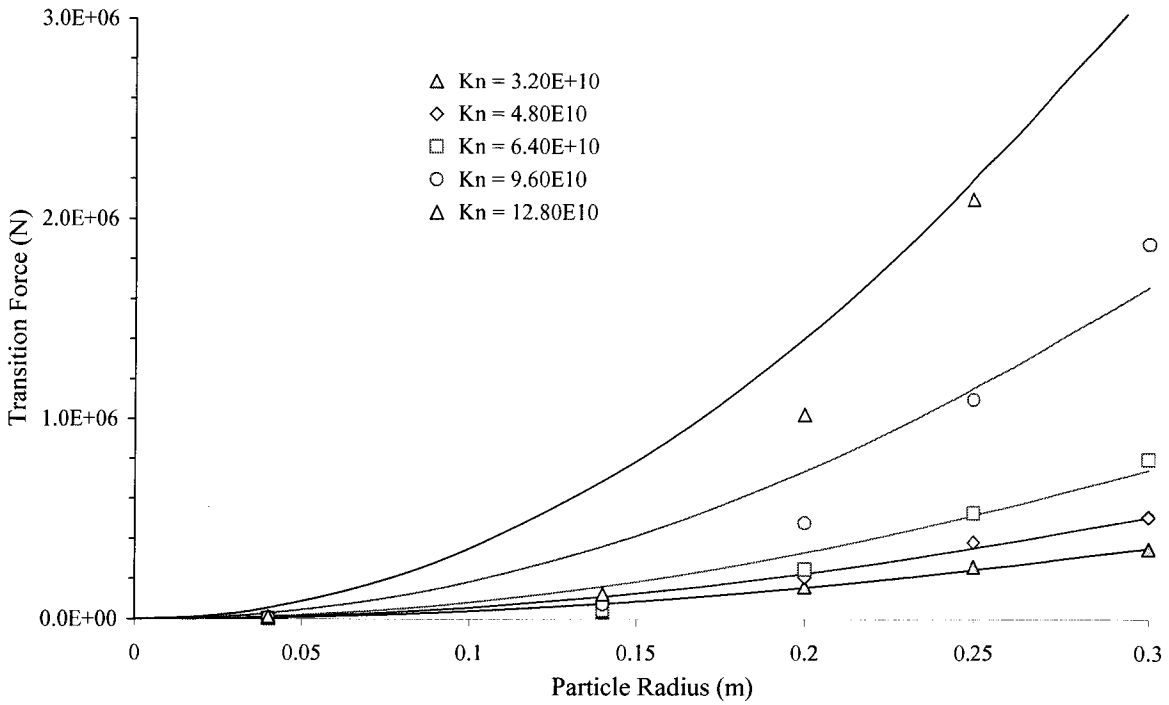


Figure 55 *Transition force* vs. particle radius (models with various normal stiffness)

6.3 Influence of Time Step on UDM

For a PFC2D model, the dynamic behavior of the assembly of particles is represented numerically by an explicit time stepping algorithm, using a central-difference scheme to integrate accelerations and velocities. It is based on the idea that each time step chosen is so small that, during a single time step, the force cannot transfer from one particle further than its immediate neighbors. Then, at all times, the forces acting on any particle are determined exclusively by its interaction with the particles with which it is in contact. Since the speed at which a disturbance can propagate is a function of the physical properties of the discrete system, the time step can be chosen to satisfy the above constraint (Itasca 2002).

The equations of particle motion are integrated in PFC2D using a central finite-difference scheme. The computed solution produced by these equations will remain stable only if the time step does not exceed a critical time step that is related to the minimum eigenperiod of the total system. The critical time step is estimated in PFC2D at the start of each cycle. The time step used in the calculation cycle is taken as a fraction of this estimated critical value.

While estimating the critical time step, the total model is viewed as a multiple mass-spring system described by a point mass, m, and spring stiffness, k. The critical time step for the whole system is expressed as

$$t_{crit} = \begin{cases} \sqrt{m/k^{tran}}, & (translation\ motion) \\ \sqrt{I/k^{rot}}, & (rotational\ motion) \end{cases} \qquad \text{Equation 38}$$

where k^{tran} and k^{rot} are the translational and rotational stiffness, respectively, and I is the moment of inertia of the particle.

Equation 38 indicates that the critical time step depends on the particle stiffness. When a UDM is introduced into an existing PFC2D model, the time step may be dramatically changed because the UDM can directly adjust contact properties such as normal and shear stiffness. Using different UDMs in a given PFC2D model may result in orders of magnitude change in the time step. For instance, while the hysteric damping model is introduced into the PFC2D model (shown in Figure 19), the time step is 1e-4, however the use of the elastic-perfectly plastic power function model in the same model results in a time step of 1e-6, even though all the microparameters other than those used in UDM are the same.

The change of time step due to use of a UDM may make a very lengthy computation process. For example, using the elastic-perfectly plastic power function model in the model shown in Figure 19 takes more than 10 hours to finish one impact process (on a computer with 2.4×2 GHz CPU and 1 GB memory). Part of the reason is that there are five history and one energy tracing functions running in the model.

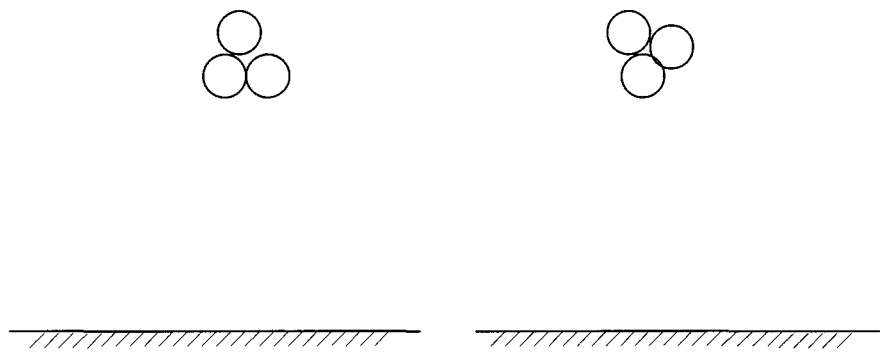
The use of a UDM not only changes the time step, but also overrides the apparent time step of the PFC2D model. Here, the apparent time step is the time step shown on the screen while the PFC2D model is running. It is found that the actual time step while invoking the UDM cannot be adjusted by a PFC2D command, although the apparent time step can be changed by the PFC2D command *SET dt dscale*. For instance, the hysteric damping model or elastic-power function models may work properly because the initial relative displacement is too large as noted earlier. The fundamental problem is that the time step is too large and the time step used by the UDM is actually not the apparent time step. Furthermore, it seems that the

time step used by the UDM cannot be adjusted by the PFC2D command. This makes the hysteric-damping and elastic-power function models unsuitable for simulating rockfalls.

The fundamental reason for these problems when using a UDM is not clear, since access to fundamental PFC2D code is impossible. However, the time step problems do set an obstacle for some models when attempting to simulate dynamic interactions between particles. The new contact model, the elastic-perfectly plastic power function can overcome this problem and runs successfully to simulate rockfalls.

6.4 Influence of Rock Shape

The shape of the falling rock can influence the simulation results. There are different modes of contact as shown in Figure 56. More than one particle may contact the wall during the impact or only one particle touches the wall during the impact process.



(a) Two particles touch together during impact (b) Single particle touches during impact

Figure 56 Two basic impact modes of numerical model

The shape of the falling rock model can influence the damping result, that is, the kinetic energy after bouncing may differ depending on whether one or more particle is involved in the impact. Without using the new UDM, when only one particle touches the wall at a given impact velocity, the contact force will be larger than that obtained if two particles were involved in the impact. When the new UDM is used, the *transition force* limits the maximum contact force between the particles and the wall. Hence, for a single particle involved in the impact, more displacement must occur compared to the displacement that results when two particles are involved in the impact. This can alter the amount of energy that is removed from the model by the UDM.

7 Conclusions

The objectives of this thesis were to improve the ability of PFC2D to simulate dynamic impact process such as those encountered during rockfall modeling. These were achieved by:

- Constructing new energy-tracing functions for use within PFC2D to enable tracking of different energy components for different parts of a numerical model. The functions ran well while performing calibration with literature data.
- Constructing a new elastic-perfectly plastic power function model (User-Defined Model). This model was used successfully to simulate a rock impact process that results in energy loss. The new model overcomes the numerical instability caused by other existing models.

The original code of the energy-tracing functions and the elastic-perfectly plastic power function model is provided in this thesis along with the methodology to use these functions. Guidelines for determining the values of the two key parameters used in the elastic-perfectly plastic power function model are also given.

An energy matching method was introduced to calibrate numerical models of rock impacts with literature data. This method allows a new way to perform calibration with physical rockfalls. The limitations of some UDMs such as the hysteric damping model and the elastic-power function model in simulating dynamic impact phenomena were analyzed. It was found that the change of time step that occurs when the UDM is invoked could cause numerical instability.

This research improves the modeling of velocity and kinetic energy loss during impacts, allowing PFC2D models to simulate more accurately dynamic processes such as rockfalls. This allows better understanding of the distribution of velocity and kinetic energy change during rockfalls or other dynamic issues such as rock crushing and rock breakage under impact loading. The new User-Defined Contact Constitutive Model (UDM) develop in this research further enhances the power of PFC2D modeling to study dynamic process.

Compared to the hysteretic damping model, the elastic-perfectly plastic power function model has the following advantages:

- The new model will not cause numerical instability because the contact force is limited by the *transition force.*
- The new model is more consistent with the physical impact process. The new model provides perfectly plastic deformation once the contact force reaches the *transition force.*
- The normal restitution coefficient will be lower with increasing impact velocity due to the increased plastic deformation. Whereas, the hysteretic damping model gives the same normal restitution coefficient as the impact velocity increases.
- The new model avoids cumulative overlap between two particles when the normal contact force returns to zero..

References

Agliardi F. and Crosta G. B. 2003. High resolution three-dimensional numerical modeling of rockfalls. Int. J. Rock Mech. Min. Sci. Vol. 40. pp. 455-471.

Andreev G. E. 1995. Brittle Failure of Rock Materials, Test Results and Constitutive Models: Balkema A. A. Rotterdam, Netherlands.

Ashby M. F. and Hallam S. D. 1986. The failure of brittle solids containing small cracks under compressive stress states. Acta. Metall. Vol. 34, pp. 497-510.

Ashby M. F. and Sammis C. G. 1990. The damage mechanics of brittle solids in compression. Pure Appl. Geophys. Vol. 133, pp. 489-521.

Atkinson B. K. and Meredith P. G. 1987. The theory of subcritical crack growth with applications to minerals and rocks. Fracture Mechanics of Rock. Atkinson B. K. (Ed.) Academic Press Geology Series. pp. 111-166.

Azzoni A. L. A., Barbera G., and Zaninetti A. 1995. Analysis and prediction of rockfalls using a mathematical model. Int. J. Rock Mech. Min. Sci. & Geomech. Abstr. Vol. 32. No. 7, pp. 109-724.

Barret R. K., Bowen R., Pfeiffer T., and Higgins J. 1991. Rockfall modeling and the Colorado rockfall simulation program – version 2.1 users manual. Colorado Trans. Dept. Rpt. CDOHDTD-ED3/CSM-89-2B, Denver, Colo.

Blair S. C. and Cook N. G. W. 1998. Analysis of compressive fracture in rock using statistical techniques, part I,, A non-linear rule-based model. Int. J. Rock Mech. Min. Sci. Vol. 35, pp. 837-848.

Brace W. F. and Bombolakis E. G. 1963. A note on brittle crack growth in compression. J. Geophys. Res. Vol. 68, pp. 3709- 3713

Brace W. F., Pauling B. W., and Sholz C. H. 1966. Dilatancy in the fracture of crystalline rocks. J. Geophys. Res. Vol 71. pp. 3939-3953.

Chan Y. C., Chan C. F., and Au W. C. 1986. Design of a boulder fence in Hong Kong. Conference on Rock Engineering in an Urban Environment, Inst. Min. Metall., Hong Kong, pp. 87-96.

Chang C. and Haimson B. 2000. True triaxial strength and deformability of the German Continental Deep Drilling Program (KTB) deep hole amphibolite. J. Geophys. Res. Vol. 105, pp. 18999-19013.

Chang C. and Haimson B. 2005. Non-dilatant deformation and failure mechanism in two Long Valley Caldera rocks under true triaxial compression. Int. J. Rock Mech. Min. Sci. Vol. 42, pp. 404-414

Chau K. T., Wong R. H. C., and Wu J. J. 2002. Coefficient of restitution and rotational motions off rockfall impacts. Int. J. Rock Mech. Min. Sci. Vol. 39. pp. 69-77.

Crech W. W. 1974. The energy balance theory and rock fracture energy measurement for uniaxial tension. Proc. 3rd Congr. ISRM, Denver, Vol. 2A, pp. 167-173.

Cundall P. A. 2002. A Discontinuous future for numerical modeling in soil and rock. Discrete Element Methods – Numerical Modeling of Discontinua. Ed. Cook B. K. and Jensen R. P. Geotechnical Special Publication No. 117.

Cundall P. A. and Strack O. D. L. 1979. A discrete numerical model for granular assemblies. Geotechnique, Vol. 29. pp. 47-65.

Escartin J., Hirth G., and Evans B. 1997. Non-dilatant brittle deformation of serpentinites: implications for Mohr-Coulomb theory and the strength of faults. J. Geophys. Res. Vol. 102, pp. 2897-2913.

Fakhimi A., Carvalho F., Ishida T., and Labuz J. F. 2002 Simulation of failure around a circular opening in rock. Int. J. Rock Mech. Min. Sci. Vol. 39, pp. 507-515

Griffith A. A. 1921. The phenomena of rupture and flow in solids. Phil Trans Royal Soc London, Ser, A. Vol. 221, pp. 163-198.

Griffith A. A. 1924. The theory of rupture. Proceeding of First International Congress Applied Mechanics, 1st Delft, pp. 55-63.

Haimson B. and Chang C. 2000. A new true triaxial cell for testing mechanical properties of rock, and its use to determine rock strength and deformability of Westerly granite. Int. J. Rock Mech. Min. Sci. Vol. 37, pp. 285-296.

Hardy M. P., Hudson J. A., and Fairhurst C. 1973. The failure of rock beams. Part I. Theoretical studies. Int. J. Rock Mech. Min. Sci. Vol. 10, pp. 53-67.

Hazzard J. F., Young R. P., and Maxwell S. C. 2000. Micromechanical modeling of cracking and failure in brittle rocks. J. Geophys. Res. Vol. 105, pp. 16,638-16,697.

Hillerborg A. 1985. The theoretical basis of a method to determine the fracture energy GF of concere. Materials and Structures. Vol. 18. No. 106 pp 291-296.

Holzhausen G. R. and Johnson A. M. 1979. Analysis of longitudinal splitting of uniaxially compressed rock cylinders. Int. J. Rock Mech. Min. Sci. & Geomech. Abstrs. Vol. 16, pp. 163-177.

Horii H. and Nemat N. S. 1985. Compression- induced microcrack growth in brittle solids : axial splitting and shear failure. J. Geophys. Res. Vol. 90, pp. 3105-3125.

Inglis C. E. 1913. Stresses in plate due to the presence of cracks and sharp corners. Trans Institute Naval Architects. LV. pp. 219-230.

Irwin G. R. 1948. Fracture Dynamics. Fracture of Metals. American Society of Metals. pp. 147-166.

Itasca Consulting Group, Inc. 2002. PFC2D Manuals.

Katz O. and Reches Z. 2000. Micro- and macro- structural analysis of small faults in a quartz-syenite intrusiona: faulting of a brittle rock without microcracking. Eos. Transations AGU. Vol. 81, pp. 1121.

Kemeny J. and Cook N. G. W. 1987. Determination of rock fracture parameters form crack models for failure in compression. 28th US Symp. on Rock Mech. Tucson. pp. 367-374.

Kobayashi Y., Harp E. L., and Kagawa T. 1990. Simulation of rockfalls triggered by earthquakes. Rock Mech. Rock Engng. Vol. 23 (1), pp. 1-20.

Labuz J. F., Shah S. P., and Dowding C. H. 1983. Post peak tensile load-displacement response and the fracture process zone in rock. Proc. 24th US Symp. on Rock Mech. Mathewson C. C. (Ed.) Texas, A. & M. Univ. pp. 421-428.

Lin C. T., Amadei B., Jung J. and Dwyer J. 1996. Extensions of discontinuous deformation analysis for jointed rock masses. Int. J. Rock Mech. Min. Sci. Vol. 33 (7), pp. 671-694.

Malan D. F. and Napier J. A. L. 1995. computer modeling of granular material microfracturing. Tectonophysics. Vol. 248, pp. 21-37.

Nicot F., Cambou B., and Mazzoleni G. 2001. Design of rockfall restraining nets from a discrete element modeling. Rock Mech. Rock Engng. Vol. 34 (2), pp. 99-118.

Orowan E. 1949. Fracture and strength of solids. Rep. Prog. Phys. Vol. 12 pp. 185

Paterson M. S. 1978. Experimental rock deformation – the brittle field. New York, Springer.

Pfeiffer, T. J. and Bowen T. D. 1989. Computer simulation of rockfalls. Bulletin of the Association of Engineering Geologists Vol. XXVI, No. 1, pp. 135-146.

Piteau, D. R., and Peckover, F. L. 1978. Engineering of rock slopes. Landslides Analysis and Control, Schuster R. L. and Krixek R. J. ed., Nat. Res. Council Spec. Rpt. 176, Washington, D. pp. 192-228.

Potyondy D. O and Cundall P. A. 2004. A bonded-particle model for rock. Int. J. Rock Mech. Min. Sci. Vol. 41, pp. 1329-1364

Ritchie R. M. 1963. Evaluation of rockfalls and its control. Highways Res. Record. Vol. 17, pp. 14-28.

RocScience 2005. http://www.rocscience.com/products/RocFall.asp (accessed Dec. 9, 2005)

Salamon M. D. G. 1993. Keynote address: some applications of geomechanical modeling in rockburst and related research , in 3rd International Symposium on Rockbursts and Seimicity in Mines, edited by Young R. P. A. A. Balkema, Brookfield, Vt. pp. 297-310.

Schmidt R. A. 1976. Fracture toughness testing of rock. Closed Loop, MTS System Corp. 5. Vol. 5, pp. 1-12.

Schweigl J., Ferretti C., and Nossing L. 2003. Geotechnical characterization and rockfall simulation of a slope: a practical case study from South Tyrol (Italy). Engineering Geology. Vol. 67, pp. 281-296.

Shi G. H. and Goodman R. E 1984. Discontinuous deformation analysis. Proc. 25th U.S. Symp. Rock Mech. pp. 269-277. Evanston, SME/AIME.

Shimizu Y. 2002. Three-Dimensional DEM simulation of conveying granular materials by horizontal screw, Discrete Element Methods –Numerical Modeling of Discontinua. Ed. Cook B. K. and Jensen R. P. Geotechnical Special Publication No. 117.

Shimizu Y. and Cundall P. A. 2001. Three-dimensional DEM simulations of bulk handling by screw conveyors. J. Engin. Mech. ASCE Vol. 127, pp. 285-292

Swan G. and Alm O. 1982. Sub-critical crack growth in Stripa granite: Direct observations. Proc 23rd U. S. Symp on Rock Mech. Univ of California, Berkeley, California. Goodman R. E. and Heuze F. E. (Ed.) pp. 542-550.

Tapponnier P. and Brace W. F. 1976. Development of stress-induced microcracks in Westerly granite. Int. J. Rock Mech. Min. Sci. Geomech. Abstr. Vol. 13, pp. 103-112.

Varnes, D. J. 1978. Slope movement types and processes. Landslides Analysis and Control, R. L. Schuster, R. J. Krized, eds., Nat. Res. Council Spec. Rpt. 176, Washington, D. C. pp. 11-33.

Wang C., Tannant D. D., and Lilly P. A. 2003. Numerical analysis of the stability of heavily jointed rock slopes using PFC2D. Int. J. Rock Mech. Min. Sci. Vol. 40, pp. 415-424

Warren D. S. 1998. RocFall: a Tool for Probabilistic Analysis, Design of Remedial Measures and Prediction of Rockfalls. Master thesis. University of Toronto.

Wawersik W. R. and Fairhust C. 1970. A study of brittle rock fracture in laboratory compression experiments. Int. J. Rock Mech. Min. Sci. Vol. 7, pp. 561-575.

Whittaker B. N., Singh R. N., and Sun G. 1992. Rock Fracture Mechanics- Principles, Design and Applications. Amsterdam: Elsevier, pp. 341-342.

Wong T. F. 1982. Micromechanics of faulting in Westerly granite. Int. J. Rock Mech. Min. Sci. Geomech. Abstr. Vol. 19, pp. 49-64.

Wong T. F. 1982. Shear fracture energy of Westerly granite from post-failure behavior. J. Geophys. Res. Vol. 87. No. B2. pp. 990-1000.

Wu S. S. 1985. Rockfall evaluation by computer simulation. Transp. Res. Rec. 1031, Nat. Res. Council, Washington, D. C., pp. 1-5.

Yang M., Fukawa T., Ohnishi Y. Nishiyama S., Miki S., Hirakawa Y., and Mori S. 2004. The application of 3-dimensional DDA with a spherical rigid block for rockfall simulation. Int. Paper 2B 25 – SINOROCK2004 Symposium, J. Rock Mech. Min. Sci. Vol. 41. No. 3 CD-ROM.

Young R. P. and Hazzard J. F. 2000. Seismic and micromechanical studies of rock fracture. Geophys. Res. Letters. Vol. 27, No. 12, pp. 1767-1770.

Zimmermann T., Robora B., Davalle E. and Descoeudres F. 1989. A three-dimensional numerical simulation model for rockfalls. Rep., Institut de Statique et Resistance des Materiaux, Lausanne, Switzerland.

Appendix A - Rock Fragmentation

Most rocks close to the Earth's surface are brittle and filled with fractures, cracks and inhomogeneities (Hazzard et al. 2000). Knowledge of the relationship between rock failure mechanisms and rock discontinuities is fundamental to solving many rock mechanics problems such as rockfalls, slope stabilities, and underground tunnel supports.

The strength of brittle rock under stress depends on the growth of microcracks and how these cracks propagate and coalesce into larger discontinuities. However, the fundamental mechanism of crack coalescence and physical failure mechanism of rock is still far from fully understood. For example, under compression load, the failure of brittle rock occurs in the form of single or a conjugate set of through-going fractures proceeded by the growth, localization and coalescence of multitude of microcracks, but it is not quite clear whether these microcracks are shear microcracks or extensile mircrocracks.

It is a predominant point of view that compressive failure of brittle rocks happens in the form of a set of through-going shear fracture preceded by the creation, propagation and coalescence of a multitude of extensile microcracks. This phenomenon has been shown by laboratory experiments (Wawersik and Fairhust 1970, Tapponnier and Brace 1976, Wong 1982). Furthermore, microscopic observations of stressed rock samples have shown that most cracks formed during compression are tensile and sub-parallel to the maximum compressive stress (Moore et al. 1995). Also, some analytical models of brittle rock failure have been developed based on these microscopic observations of the growth and interaction of stress-induced mircrocracks in rock under compressive stress (Horii and Nemat 1985, Ashby and Hallam 1986 and Ashby and Hallam 1990). These microcracks are generally aligned with the maximum principal stress direction. They are typically tensile and open in the direction of the smallest principal stress. This behavior results in a plastic volumetric expansion relative to the elastic contraction. This response was referred to as dilatancy by Brace et al. (1966).

There is evidence showing that dilatancy is a common property of most brittle rocks (Paterson 1978, Haimson and Chang 2000, Chang and Haimson 2000). Based on direct observations, Brace and Bombolakis (1963) and Horii and Nemat (1985) indicated that shear cracks cannot propagate in their own plane. Furthermore, James et al. (2000) asserted that the

eventual failure of a brittle rock must occur by the interaction of the tensile cracks to form a macro shear fault. On the other hand, there are a few studies showing that some rocks (such as Long Valley hornfels, quartz-syenites, and serpentinites) do not create extensile microcracks before final brittle failure (Chang and Haimson 2005, Katz and Reches 2000, and Escartin et al. 1997). These rocks have unique deformational properties, which cannot be easily interpreted by the above-mentioned rock failure mechanisms.

Escartin et al. (1997) indicated that two foliated serpentinite rocks, lizardite and antigorite, do not show non-dilatant brittle failure under triaxial testing. According to microscopic investigation, there is no extensile microcracks aligned with the direction of the maximum principal stress even at low confining pressures, instead, they detected inclined shear microcracks in samples that were unloaded soon before final failure. Escartin et al. (1997) inferred that shear microcracks coalesce and form a final dipping shear fracture. The crack geometry is less effective in causing dilatancy than opening of mode I (tensile) microcracks because creation of void space along shear cracks is confined by irregularities along the shear plane.

Although certain attention has been given to Mode II fracture of rock, there is still a lack of systematic research. Chang and Haimson (2005) argued that it is still too early to advance a shear failure mechanism before shear microcracks can be identified with confidence in these non-dilatant rocks.

So far, the research on rock fracture mechanism has mainly been focused on Mode I (tensile mode) fracture of the rock. The tensile fracture mechanism of rock is well accepted even though controversy concerning rock failure mechanism still exists.

In the past, most research on the failure of rock was based on experimental methods. As computer power increases, numerical models to explain cracking and failure mechanism of brittle rock have been becoming more feasible. In the numerical models, statistical methods are used to simulate rock materials that are assigned randomly distributed cracks or joint sets (Wang et al. 2003, Blair et al. 1998, Malan and Napier 1995, Salamon 1993, Lockner et al. 1991). These models have reproduced some of the observed features (e.g., the creation of tensile cracks, the dilatancy of rock before failure, the coalescence of mircrocracks into

localized shear bonds) of the fracture of rocks. But the energy released from the formation of cracks and fractures was not considered in these models. Actually, the formation of cracks will induce stress wave propagation in the rock, which may further create cracks and eventually result in the failure of rock mass (Young and Hazzard 2000, Hazzard et al. 2000).

It is generally accepted that rock is a type of discontinuous material. Microcracks such as flaws, pores greatly reduce the tensile strength of rocks. However, this point had not been understood until the work of Inglis (1913) and Griffith (1921 and 1924). Given that Griffith theory is well understood, only Griffith theory and its modifications concerning the energy needed to create new surface are introduced.

Applying energy balance approach, Griffith stated that the energy needed to overcome the cohesive molecular forces in the formation of the new surfaces is given by Equation 39.

$$U_s = 2A'\gamma_s = 2(2\alpha B)\gamma_s = 4\alpha\gamma_s \qquad \text{Equation 39}$$

where

U_s— Surface energy due to the formation of crack new surface

B — Thickness assumed to be of unit size, i.e. $B = 1$

α — ¼ length of the cross- sectional of the crack

γ_s— The specific surface energy, i.e. the energy required to create unite area of crack new surface as the crack increases in length

Because γ_s is a constant material property according to Griffith (1921), Equation 39 indicates that U_s is linearly proportional to crack length.

With some reservations, the Griffith balance theory can be viewed as valid for brittle materials with little or no preceding plastic deformation near the crack tip (Whittaker 1992). However, materials such as rock, sustain a microcracking process zone, in which the materials behaves plastically (Schmidt 1976, Labuz et al. 1983, Hillerborg 1985), and this microcracking process may be large and even be linked to subcritical crack growth (Swan and Alm 1982, Atkinson and Meredith 1987). Potential energy must be dissipated during the period of the formation of this microcracking zone and this energy transformation process is

irreversible, however, this part of energy is not included in Griffith energy balance expression.

Irwin (1948) modified Griffith's formulations to accommodate limited plastic deformation before failure and he indicated that a material's resistance to crack extension should be the sum of the elastic surface energy, γ_s, and that of plastic deformation absorbed in the fracture process, γ_p, namely

$$\gamma_{eff} = (\gamma_S + \gamma_P) \qquad \text{Equation 40}$$

Where, γ_{eff} is called as apparent specific surface energy (work of fracture) to distinguish it from specific surface energy γ_s. It is composed of two parts: elastic specific surface energy and the energy of plastic deformation.

Orowan (1949) suggested a similar modification, and in 1955, Orowan showed that the plastic energy term γ_p is about three orders of magnitude higher than the elastic surface energy term γ_s, and if a Griffith energy balance approach is appropriate, γ_s can be neglected. This point of view was also argued by Crech (1974), Hardy et al. (1973), Wong (1982), Kemeny and Cook (1987), and Holzhausen and Johnson (1979).

The limitations of Griffith theory are well stated in the literature (Andreev 1995, Wittaker 1992), and will not be presented here. The original Griffith theory and its evolution by Irwin (1948), Orowan (1949), Wong (1982), and Kemeny and Cook (1987) is more applicable to tensile than compressive conditions, because these theories refer to local failure processes only. However, under compression stresses, the process of fracture propagation and crack coalescence of the material is considerable (Whittaker 1992). Based on failure mechanisms of rock according to literature, most rock materials fail because of the creation of tensile microcracks in compression ("tensile failure is an attribute of brittle fracture", according to Andreev 1995).

In this research, the energy lost due to rock fragmentation will be accounted by an energy dissipation algorithm by using a user-defined model.

Appendix B - PFC2D Calibration to Match Berea Sandstone Properties

The general PFC2D calibration procedure is shown in Figure 57.The steps in the boxes with italic font in Figure 57 were not performed, although they could be performed when simulating a rockfall in which the falling rock is allowed to potentially break into smaller pieces upon impact.

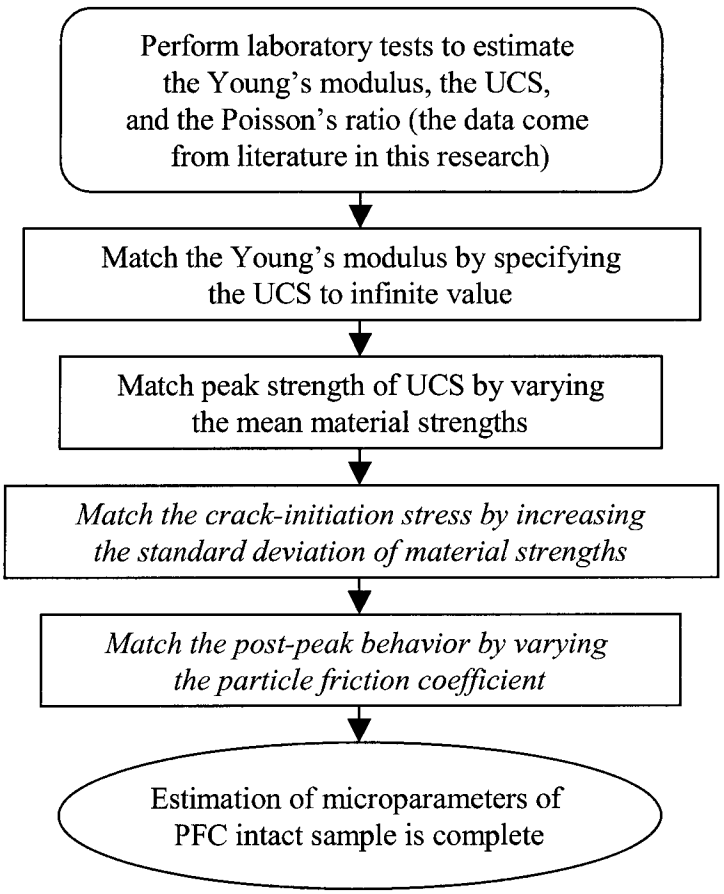


Figure 57 Routine calibration procedure of PFC

The PFC2D synthetic specimen was composed of clumps as shown in Figure 58. Each clump has three particles at maximum. The clumps are arranged randomly. Uniaxial compression tests were carried out to calibrate to model with physical properties of Berea sandstone. The specimen after peak load is shown in Figure 59, and the microcracks occurring within the specimen are shown in Figure 60.

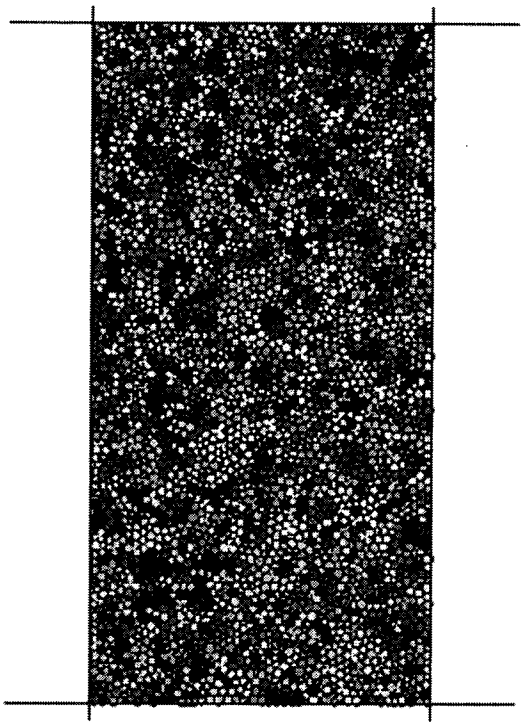


Figure 58 PFC2D synthetic specimen for calibration

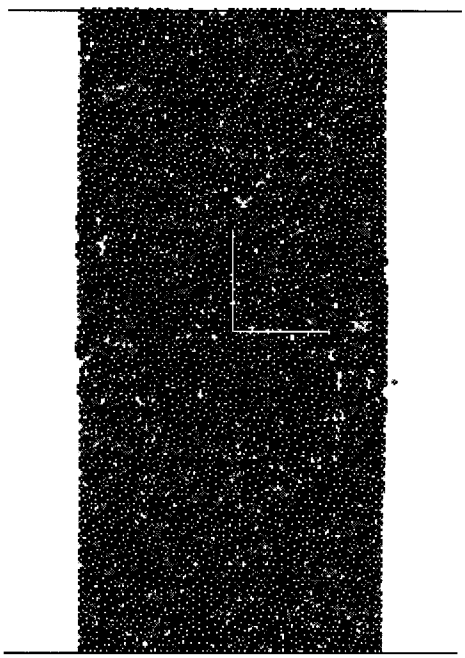


Figure 59 PFC2D synthetic specimen after peak load (at strain of 3.8e-3)

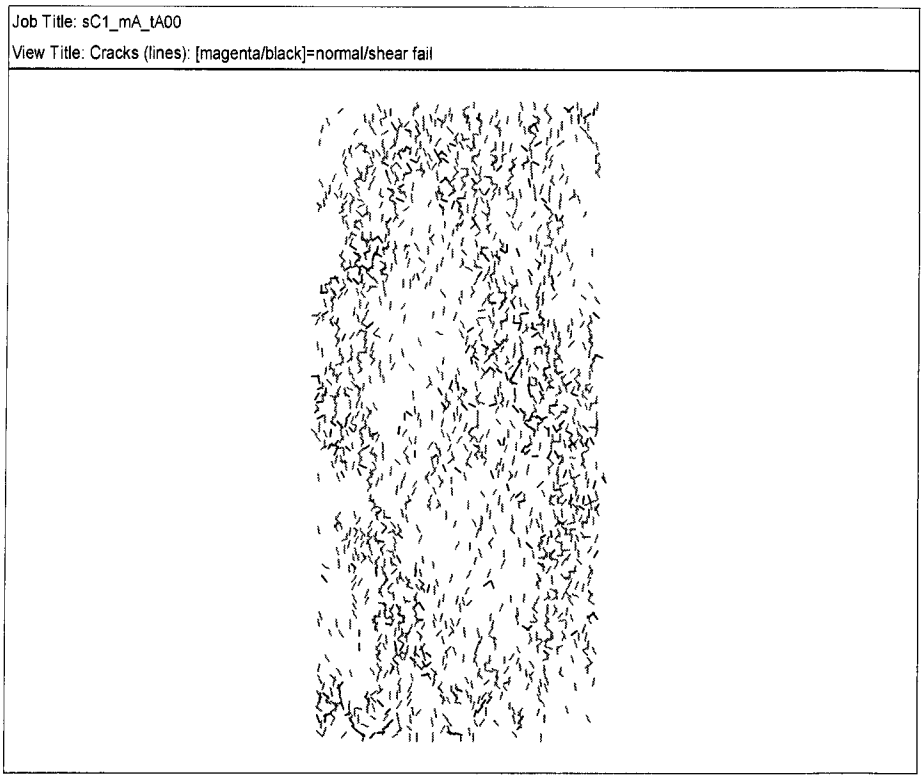


Figure 60 Microcracks in the synthetic specimen (at strain of 3.8e-3)

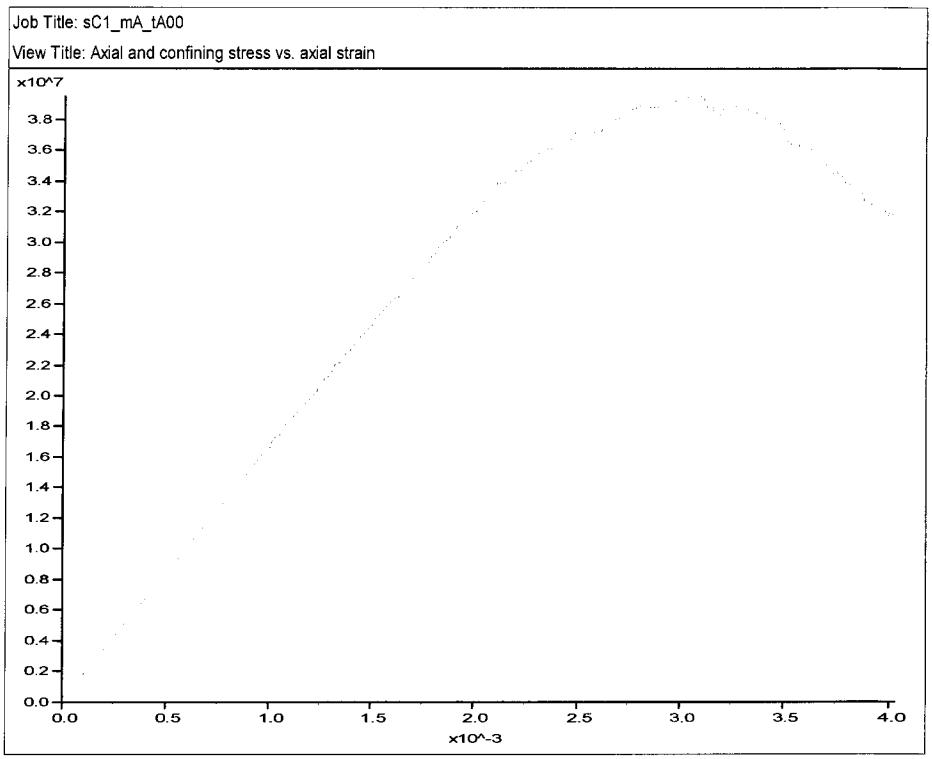


Figure 61 Axial stress vs. axial strain

The relationship between axial stress and axial stain is shown in Figure 61. The peak strength (39.6 MPa) and Young's modulus (16.1 GPa) closely matches those of Berea sandstone. The microparameters are density: 2600 kg/m^3, normal stiffness: 6.4e10 N/m, shear stiffness: 9.6e9 N/m, friction coefficient: 0.5, contact bond strength: normal 1e9 N, shear 1e10 N. The maximum clump size is three.

Appendix C – UDM Exponent and Transition Force

Table 2 UDM *exponent* at various impact velocities (Berea sandstone, *Kn* = 6.4e10 N/m)

Impact velocity	Exponent for power function at various velocities				
	R = 0.04m	R= 0.14m	R= 0.2m	R= 0.25m	R = 0.3m
1	4.5	4.5	4.5	4.5	4.5
2	4.5	4.5	4.5	4.5	4.5
3	4.5	4.5	4.5	4.5	4.5
4	5.5	5.5	5.5	5.5	5.5
5	7	7	7	7	7
6	8.5	8.5	8.5	8.5	8.5
7	11	11	11.5	10.5	10.5
8	14	14	14	14	14
9	18	18	19	19	20
10	24	23	23	23	23
11	29	29	26	26	26
12	41	40	38	38	38
13	49	48	45	45	45
14	61	57	57	58	58
15	82	81	82	80	78

Table 3 UDM *exponent* at various impact velocities (*Kn* = 3.2e10 N/m)

Impact velocity	Exponent for power function at various velocities				
	R = 0.04	R= 0.14m	R= 0.2m	R= 0.25	R = 0.3m
1	4.4	4	4	5	5
2	4.4	4.2	4.2	5	5
3	4.4	4.2	4.3	5	5
4	4.4	4.4	4.3	5	5
5	7	7	7	7	7
6	8.5	8.5	8.5	8.5	8.5
7	11	11	11	11	11
8	14	14	14	14	14
9	18	18	18	18	18
10	24	24	24	24	24
11	29	29	29	29	29
12	41	41	41	41	41
13	49	47	49	49	49
14	61	55	61	61	58
15	83	76	84	82	76

Table 4 UDM *exponent* at various impact velocities (*Kn* = 9.6e10 N/m)

Impact velocity	Exponent for power function at various velocities				
	R = 0.04	R= 0.14m	R= 0.2m	R= 0.25	R = 0.3m
1	4.4	4.3	5	5	5
2	4.4	4.3	5	5	5
3	4.4	4.3	5	5	5
4	4.4	4.3	5	5	5
5	7	7	7	7	7
6	8.5	9	8	9.5	9
7	11	11	11	11	11
8	14	14	14	14	14
9	18	18	18	18	18
10	23	24	24	24	24
11	29	31	29	29	29
12	41	42.5	41	40	41
13	49	49	49	53	49
14	60.5	61	61	61	64
15	83	84	76	82	76

Table 5 UDM *exponent* at various impact velocities (*Kn* = 12.8e10 N/m)

Impact velocity	Exponent for power function at various velocities				
	R = 0.04	R= 0.14m	R= 0.2m	R= 0.25	R = 0.3m
1	4.4	4.3	5	5	5
2	4.4	4.3	5	5	5
3	4.4	4.3	5	5	5
4	4.4	4.3	5	5	5
5	7	7	7	7	7
6	8.5	8	9	8.5	8.5
7	11	11	11	11	11
8	14.5	13.5	14	14	14
9	18	18	17.2	18	18
10	23	25	24	24	24
11	29	29	27	29	29
12	41	41	41	41	41
13	49	47	49	49	51
14	61	56	61	61	57
15	83	84	76	82	78

Table 6 Transition force for various particle size samples at different normal stiffness

Normal stiffness (N/m)	Transition force for various particle sizes				
	R = 0.04m	R = 0.14m	R = 0.2m	R = 0.25m	R = 0.3m
4.80E+10	3200	41300	210000	383500	510000
3.20E+10	3000	40000	190000	382000	350000
6.40E+10	3500	45000	250000	530000	800000
9.60E+10	8000	76000	480000	1100000	1880000
1.28E+11	12200	125000	1025000	2100000	3525000

Appendix D – Code for PFC2D

Code for UDM

Code for user-defined model (including four files: contmodel.h, HysDmodel.h, undvect3.h, and HysDmodel.cpp)

Contmodel.h

```
#ifndef __CMODEL_CONTMODEL_H
#define __CMODEL_CONTMODEL_H

#ifndef __CMODEL_UMDVECT3_H
#include "umdvect3.h"
#endif

#ifndef __UNIX__
#ifndef EXPORT
#define EXPORT __declspec(dllexport)
#endif
#else
#ifndef EXPORT
#define EXPORT
#endif
#endif

#define BALLs (unsigned char)100
// used for 3D
#define fb_u_dot_s    (*fb.ptu_dot_s)
#define fb_t_dot_rel  (*fb.ptt_dot_rel)
#define fb_s_force    (*fb.pts_force)
#define fb_moment     (*fb.ptmoment)

struct FdBlock { // Data used to communicate between PFC and Contact Models in FD law
 // --- PFC -> Contact Models ---
 double  u_n;          // overlap +, gap -
 double  u_dot_n;      // relative velocity, normal dir.
 double  u_dot_s;      // relative velocity, shear dir. ( 2D )
 double  t_dot_rel;    // relative angular velocity ( 2D )
 UMdvect *ptu_dot_s;   // relative velocity, shear dir. ( 3D )
 UMdvect *ptt_dot_rel; // relative angular velocity ( 3D )
 double  tdel;         // current time step

 // --- Contact Models -> PFC ---
```

```
  double  n_force;       // contact force, normal dir.
  double  s_force;       // contact force, shear dir. ( 2D )
  double  moment;        // contact moment ( 2D )
  UMdvect  *pts_force;   // contact force, shear dir. ( 3D )
  UMdvect  *ptmoment;    // contact moment ( 3D )
  double  knest;         // estimated kn
  double  ksest;         // estimated ks
  double  krest;         // estimated kr, just for PFB model, used FD law in Ball2D
  bool    skip;          // in case of skipping PFC FD logic after FD law of Contact Models
  bool    bfishc_brokenn; // in case of broken contact bond (normal) invoking fishcall
  bool    bfishc_brokens; // in case of broken contact bond (shear) invoking fishcall
  bool    bflag;         // bonding flag
  bool    broken;        // bonding flag
  bool    bSliding;      // sliding check flag
  EXPORT FdBlock(void);
};


struct PropBlock { // Data passed to Contact Models
  double      kn_c;      // normal stiffness of Default Contact model
  double      ks_c;      // shear  stiffness, DC model
  double      fric_c;    // friction coefficient, DC model
  double      n_strength; // normal strength of contact bond, DC model
  double      s_strength; // shear strength of contact bond, DC model
  unsigned char type_gobj2; // object type of contact_2
  bool        bIsNear;   // control flag, IsNear() for Viscous & SSB
  double      disk_thick; // ..... for SSB
  double      rad_gobj1;  // ..... for SSB
  double      rad_gobj2;  // ..... for SSB

  EXPORT PropBlock(void);
};


struct ModelSaveObject {
  bool        bWriting;   // true if writing to file
  unsigned long ulType;     // type value of contact model
  unsigned long ulVersion;  // contact model version number
  unsigned long ulNumDouble; // num. of double values that need to be saved
  unsigned long ulNumInt;   // num. of integral values that need to be saved
  unsigned long ulNumBool;  // num. of bool values that need to be saved
  double      *aDouble;   // array of doubles, will be uNumDouble in size
  int         *aInt;      // array of ints, will be uNumInt in size
  bool        *aBool;     // array of bools, will be uNumBool in size

  EXPORT ModelSaveObject(bool bwrite=true);
```

```
  EXPORT ~ModelSaveObject(void);
  EXPORT void Initialize(unsigned long ulNumD,
                unsigned long ulNumI=0, unsigned long ulNumB=0);
  EXPORT void Save(unsigned long ul, double &dVal);
  EXPORT void Save(unsigned long ul, int &iVal);
  EXPORT void Save(unsigned long ul, bool &bVal);

 private:
   ModelSaveObject(const ModelSaveObject &);
   const ModelSaveObject &operator=(const ModelSaveObject &);
};
double nnn = 0;

class ContactModel {
 private:
   unsigned long ulType;
 protected:
   bool    delete_flag;  // true if contact can be deleted
 public:
   // Creators
   EXPORT ContactModel(unsigned long ulTypeIn, bool bRegister=false);
   EXPORT virtual ~ContactModel(void);
   // Accessors
   EXPORT virtual const char *Name(void) const=0;
   EXPORT virtual const char **PropNames(void) const=0;
   EXPORT virtual ContactModel *Clone(PropBlock *pb=0) const=0;
   EXPORT virtual double ReturnProp(int n) const=0;
   EXPORT virtual double KsEstimate(void) const=0;
   EXPORT virtual double KnEstimate(void) const=0;
   EXPORT virtual unsigned long Version(void) const = 0;
   EXPORT unsigned long Type(void) const { return ulType; }
   EXPORT bool OKtoDelete(void) { return delete_flag; }
   EXPORT int NameToIndex(const char *strName) const;
   EXPORT const char *GetPropName(int) const;
   EXPORT virtual double &Property(int);
   // Manipulators
   EXPORT virtual void AcceptProp(int n,double v)=0;
   EXPORT virtual const char *PreCycle(PropBlock &pb)=0;
   EXPORT virtual void FDlaw(FdBlock &fb, char cDim)=0;
   EXPORT void SetDelete(bool d) { delete_flag = d; }
   // Save & Restore
   EXPORT virtual const char *SaveRestore(ModelSaveObject *mso);
   EXPORT virtual void remap(PropBlock &) { }
   // Static function to access all Contact Models
   EXPORT static unsigned long NumModels(void);
   EXPORT static const ContactModel *GetModel(unsigned long ulPos);
```

```
   EXPORT static const ContactModel *ContactModel::FindByType(unsigned long
ulType_in);
   EXPORT static bool IsCprop(const char *strName);
   EXPORT static const char *Load(const char *strLibPathAndName,bool &);
};

#endif
```

hysDmodel.h

```
#ifndef __CMODEL_HYSDMODEL_H
#define __CMODEL_HYSDMODEL_H

#ifndef __CMODEL_CONTMODEL_H
#include "contmodel.h"
#endif



//------------ built-in, user-defined contact model "CM_HysDamp" -----------
const unsigned long ulModelCM_HysDamp = 5;
class CM_HysDamp : public ContactModel {
 private:
   double kn_load;    // normal stiffness at laoding
   double kn_unload;  // normal stiffness at unloading
   double kn_m;       // mean normal stiffness
   double ks;         // shear stiffness
   double fric;       // frictional coefficient
   double hys_nstr;   // normal contact strength
   double hys_sstr;   // shear contact strength
   unsigned char type_gobj2;    //
   double purePlasticPoint;//sp_damp;   *****  // ratio k_load/k_unload <= 1.0, 0.8 default
resti. coef = 0.9
   double u_old;      // overlap just before
   double u_zero;     // default overlap when reloading within this contact cycle
   double u_zeroo;    // default overlap when unloading within this contact cycle
   double u_max;      // max. overlap of this contact cycle
   double powerNumber; // n_force just before to check transition + -
   bool  bNtension;  // FDlaw option, with or without tensile force, default bFalse (with
tensile)
   bool  bInheritProp;// Inherit property in contact class, default bFalse (NOT inherit)
        double kn_plastic ; //***** Changed.
        double SlopeCoefficient;
 public:
   // Creators
   CM_HysDamp(bool bRegister=false);
```

```
    ~CM_HysDamp(void) { }

    // Accessors
    EXPORT const char *Name(void) const;
    EXPORT const char **PropNames(void) const;
    EXPORT ContactModel *Clone(PropBlock *pb=0) const;
    EXPORT double  ReturnProp(int n) const;
    EXPORT double  KsEstimate(void) const { return ks; }
    EXPORT double  KnEstimate(void) const { return kn_unload; } // take high stiffness
    EXPORT unsigned long   Version(void) const { return 3; }

    // Manipulators
    EXPORT void AcceptProp(int n, double v);
    EXPORT const char *PreCycle(PropBlock &pb);
    EXPORT void FDlaw(FdBlock &fb, char cDim);
    void SetProp(PropBlock &pb);
    void SetKn(void);
    void CheckBonding(FdBlock &fb, char cDim);
        double ImpactPlastic(double ImpactForce,FdBlock& Plast_fb); // added by me to test
plastice process.
    EXPORT const char *SaveRestore(ModelSaveObject *mso);
};


#endif


Umdvect3.h

#ifndef __CMODEL_UMDVECT3_H
#define __CMODEL_UMDVECT3_H

#ifndef __UNIX__
#ifndef EXPORT
#define EXPORT __declspec(dllexport)
#endif
#else
#ifndef EXPORT
#define EXPORT
#endif
#endif


class UMdvect3 {
 public:
   double x;
```

```
  double y;
  double z;
 public:
  EXPORT UMdvect3(void);
  EXPORT UMdvect3(double x1,double y1,double z1);
  EXPORT UMdvect3(const UMdvect3 &v3);
  EXPORT const double &X(void) const;
  EXPORT const double &Y(void) const;
  EXPORT const double &Z(void) const;
  EXPORT void X(const double &t);
  EXPORT void Y(const double &t);
  EXPORT void Z(const double &t);
  EXPORT void Set(double x1,double y1,double z1);
  EXPORT void Fill(double x1);
  EXPORT UMdvect3 operator +(const UMdvect3 &t3) const;
  EXPORT void operator +=(const UMdvect3 &t3);
  EXPORT UMdvect3 operator -(const UMdvect3 &t3) const;
  EXPORT void operator -=(const UMdvect3 &t3);
  EXPORT UMdvect3 operator *(const double &t) const;
  EXPORT void operator *=(const double &t);
  EXPORT UMdvect3 operator /(const double &t) const;
  EXPORT void operator /=(const double &t);
  EXPORT UMdvect3 operator &(const UMdvect3 &t3) const;
  EXPORT double operator |(const UMdvect3 &t3) const;
  EXPORT double &operator[](int i);
  EXPORT const double &operator[](int i) const;
  EXPORT bool operator ==(const UMdvect3 &t3) const;
  EXPORT bool operator !=(const UMdvect3 &t3) const;
  EXPORT UMdvect3 &operator =(const UMdvect3 &t3);
};
typedef UMdvect3 UMdvect;


#endif
```

hysDmodel.cpp

```
#include "hysdmodel.h"
#include <math.h>
#include <stdio.h>


static CM_HysDamp cm_hysdamp(true);
```

```
CM_HysDamp::CM_HysDamp(bool bRegister) :
 ContactModel(ulModelCM_HysDamp, bRegister),
              kn_load(0.0), kn_unload(0.0), kn_m(0.0),
              ks(0.0), fric(0.0),
              hys_nstr(0.0), hys_sstr(0.0), type_gobj2(BALLs),
              purePlasticPoint (0.0), //**** sp_damp(0.8),       // rest. coeff. = 0.9
              bNtension(true),   //changed from with tensile force--(false)

bInheritProp(false), // NOT inherit
              u_old(0.0), u_zero(0.0), u_zeroo(0.0), u_max(0.0)  {
 SetDelete(true);
}


const char *CM_HysDamp::Name(void) const {
 return("TrangleMixer");
}


const char **CM_HysDamp::PropNames(void) const {
 static const char *strKey[] = {
  "hys_knm", "purePlasticPoint", "hys_ks", "hys_fric",
  "hys_nstr", "hys_sstr", "hys_notension", "hys_inheritprop", "powerNumber",0
 }; //here, ***** hys_dampn is replaced by purePlasticPoint
 return(strKey);
}


ContactModel *CM_HysDamp::Clone(PropBlock *pb) const {
 CM_HysDamp *cm_hd = new CM_HysDamp;
 if(pb) cm_hd->SetProp(*pb);
 return(cm_hd);
}


double CM_HysDamp::ReturnProp(int n) const {
 switch (n) {
  case 0 : return kn_m;
  case 1 : return purePlasticPoint; // **** sp_damp;
  case 2 : return ks;
  case 3 : return fric;
  case 4 : return hys_nstr;
  case 5 : return hys_sstr;
  case 6 : return bNtension ? 1.0 : 0.0;
  case 7 : return bInheritProp ? 1.0 : 0.0;
       case 8 : return powerNumber;
```

```
    default: return 0.0;
   }
 }


 void CM_HysDamp::AcceptProp(int n, double v) {
  switch (n) {
    case 0:  kn_m     = v;                  break;
    case 1:  purePlasticPoint =v;             break;//sp_damp   = v;    break;
    case 2:  ks       = v;                  break;
    case 3:  fric     = v;                  break;
    case 4:  hys_nstr  = v;                   break;
    case 5:  hys_sstr  = v;                   break;
    case 6:  bNtension = v!=0.0 ? true : false;    break;
    case 7:  bInheritProp = v!=0.0 ? true : false; break;
          case 8:  powerNumber =v;                                          break;
    default:                          break;
  }
  SetKn();
 }


 const char *CM_HysDamp::PreCycle(PropBlock &pb) {
          if(bInheritProp) SetProp(pb);
  // if(kn_load==0.0 || kn_unload==0.0)
    // return("zero normal contact stiffness, specify 'hys_knm' or set 'hys_inheritprop' 1");
   //if(!bNtension && (sp_damp>1.0 || sp_damp<0.4)) ****
    // return("range of damping factor : 0.4 <= damp_hysdamp <= 1");
   //if( bNtension && (sp_damp>1.0 || sp_damp<0.05)) ****
    // return("range of damping factor : 0.05 <= damp_hysdamp <= 1");
   return(0);
 }


 void CM_HysDamp::SetProp(PropBlock &pb) {
  kn_m      = pb.kn_c; //
  ks        = pb.ks_c;  // changed by me
  fric      = pb.fric_c;
  hys_nstr   = pb.n_strength;
  hys_sstr   = pb.s_strength;
  type_gobj2 = pb.type_gobj2;
  SetKn();  //call SetKn to calculate parameters
 }


 void CM_HysDamp::SetKn(void) {
```

```
        kn_load   = kn_m; //use default normal stiffness
        // kn_load   = 2.0*sp_damp*kn_m/(1.0+sp_damp);**** use 0.8 to replace sp_damp
 // kn_unload = 2.0*kn_m/(1.0+sp_damp);
}

double CM_HysDamp::ImpactPlastic(double ImpactForce,FdBlock& Plast_fb)
{
        double static  force_PlasticPoint = purePlasticPoint; //Plast_fb.n_force;
        static double  DisplacementPlastic = Plast_fb.u_n; // Take the displacement of plastic
point.
        //double kn_plastic = 0.0;
        //printf("%7.2f", kn_plastic);
        Plast_fb.n_force = force_PlasticPoint; //+ kn_plastic*(Plast_fb.u_n-
DisplacementPlastic-u_zero);
        double aaa = force_PlasticPoint;
        return(Plast_fb.n_force);
}


//------------ user-defined contact model "HysDamp" -----------
void CM_HysDamp::FDlaw(FdBlock& fb, char cDim) { // Force/displacement law,
HystereticDamping
//        if (fb.u_n <0)
//                { // for contact bond
//                        fb.n_force = kn_load*fb.u_n;}  // tensile force caused by contact bond

        if (fb.u_n<=0.0 && !fb.bflag) {
  fb.n_force = 0.0;
  if (cDim==2)
   fb.s_force  = 0.0;
              else
   fb_s_force.Fill(0.0);
  u_old = 0.0;
  u_zero = 0.0;
  u_zeroo = 0.0;
  u_max = 0.0;
 // powerNumber = 0.0;
  fb.knest = 0.0;
  fb.ksest = 0.0;
  fb.skip  = true;
 }
 else {
  if(!bNtension) {
  }
  else {  // without tensile force for damping but wiht tensile force for contact bond
              if (fb.u_n - u_old >= 0.0)
```

```
        {
                // The following is added for plastic impact process.
            // ================================================================
            //if (fb.n_force >= 2000.0)  //**** This statment is changed as follows:
            //  fb.n_force = kn_load*(fb.u_n - u_zero);// This is added to prevent force is
too big
            if (fb.n_force >= purePlasticPoint) {
                ImpactPlastic(fb.n_force,fb); // for plastic impact process.
                static int iii =1;
                iii= iii+1;

        //          static double force_PlasticPoint = fb.n_force;

        //          double kn_plastic = 30000.0;
        //          fb.n_force = force_PlasticPoint + kn_plastic*(fb.u_n-u_zero);
            } else {
                fb.n_force = kn_load*(fb.u_n - u_zero);
                if (fb.n_force >= purePlasticPoint) ImpactPlastic(fb.n_force,fb);
                fb.n_force=fb.n_force;
            }
                u_max = fb.u_n;
                u_zeroo = u_zero;
    }
                        // to cut acute initail point at plastic point.
    else
        {  // unloading
            if (fb.u_n <0)
            { // for contact bond
                fb.n_force = kn_load*fb.u_n;  // tensile force caused by contact bond
            }
            else
            {

                // the following is for determine trangle returning coefficient.
            // static double SlopeCoefficient = fb.n_force/fb.u_n;
            // fb.n_force = SlopeCoefficient*fb.u_n;
            // the above is used to calculate contact force when returning
            //since linear returning coeffieient is not too effective to dissipate energyr use
another model to dissipate
                        // the following is for determine paralic model returning
coefficient.
            //if (fb.n_force >= purePlasticPoint)
                //fb.n_force = purePlasticPoint;

            static double aaa = fb.n_force;
            static double ccc=u_max; //at the maximum displacement, the ball go back
```

```
                double xxx1 = ccc;
                double xxx2 = aaa;
                //if (fb.u_n > ccc)
                        // {
                         //static double difference = (fb.u_n - ccc); // this is used to reduce
different return point, still can't erase it
                         //fb.u_n = fb.u_n - difference;
                         //}
                //static double SlopeCoefficient = fb.n_force/(pow(fb.u_n,3));
                 SlopeCoefficient =aaa/(pow(ccc,powerNumber));
                        double bbb=SlopeCoefficient;
                fb.n_force = SlopeCoefficient*(pow(fb.u_n,powerNumber));
                // the above is used to calculate contact force when returning
               // if (fb.n_force >= purePlasticPoint)
              //        fb.n_force = purePlasticPoint;
                if (fb.u_n > ccc)
                         {
                         static double difference = (fb.u_n - ccc); // this is used to reduce
different return point, still can't erase it
                         fb.u_n = fb.u_n - difference;
                         fb.n_force = SlopeCoefficient*(pow(fb.u_n,powerNumber));
                         fb.u_n = fb.u_n + difference; // recover the displacement to use u_old
                         }

                if (fb.n_force > purePlasticPoint)
                         {
                         fb.n_force = purePlasticPoint;
                         }
    if(fb.n_force < 0.0) fb.n_force = 0.0;
    u_zero =  fb.u_n - fb.n_force/kn_load;
   }
  }
  u_old = fb.u_n;
  if (cDim==2)
   fb.s_force  -= fb.u_dot_s*ks*fb.tdel;
  else
   fb_s_force  -= fb_u_dot_s*ks*fb.tdel;
  if(fb.bflag)    // bonding check
   CheckBonding(fb, cDim);
  else {          // sliding check
   double max_s_force = fabs(fb.n_force * fric);
   if (cDim==2) {
    double sfmag = fabs(fb.s_force);
    if(sfmag > max_s_force) {
     fb.s_force = fb.s_force * max_s_force / sfmag;
                                        fb.bSliding = true;
```

```
      }
     }
                    else {
      double dX = fb.pts_force->x;
      double dY = fb.pts_force->y;
      double dZ = fb.pts_force->z;
      double sfmag = sqrt(dX*dX + dY*dY + dZ*dZ);
      if(sfmag > max_s_force) {
       fb_s_force = fb_s_force * max_s_force / sfmag;
                                          fb.bSliding = true;
      }
                    }// this is for contact bond
                          }
                    }
   fb.knest = kn_unload;
   fb.ksest = ks;
   fb.skip  = false;
  }
}


void CM_HysDamp::CheckBonding(FdBlock &fb, char cDim) {
 if (type_gobj2==BALLs) {
                    if (cDim==2) {  // 2D...
    if (-fb.n_force >= hys_nstr) {
      fb.bfishc_brokenn = true;
      fb.bflag  = false;
      fb.broken  = true;
      fb.n_force = 0.0;
      fb.s_force  = 0.0;
                          }
                          else if (fabs(fb.s_force) >= hys_sstr) {
      fb.bfishc_brokens = true;
      fb.bflag  = false;
      fb.broken  = true;
                          }
   }
                    else {  // 3D...
    if (-fb.n_force >= hys_nstr) {
      fb.bfishc_brokenn = true;
      fb.bflag  = false;
      fb.broken  = true;
      fb.n_force = 0.0;
      fb_s_force.Fill(0.0);
    }
    else {
```

```
        double dX = fb.pts_force->x;
        double dY = fb.pts_force->y;
        double dZ = fb.pts_force->z;
        double dMag = sqrt(dX*dX + dY*dY + dZ*dZ);
        if (dMag >= hys_sstr) {
          fb.bfishc_brokens = true;
          fb.bflag   = false;
          fb.broken  = true;
                                    }
                                }
    }
  }
}


const char *CM_HysDamp::SaveRestore(ModelSaveObject *mso) {
  const char *str = ContactModel::SaveRestore(mso);
  if (str) return(str);
  mso->Initialize(14,0,3); //changed from 13 to 14, increase one double invriable: kn_plastic.
  mso->Save( 0, delete_flag);
  mso->Save( 1, bNtension);
  mso->Save( 2, bInheritProp);
  mso->Save( 0, kn_load);
  mso->Save( 1, SlopeCoefficient); //kn_unload);
  mso->Save( 2, kn_m);
  mso->Save( 3, ks);
  mso->Save( 4, fric);
  mso->Save( 5, hys_nstr);
  mso->Save( 6, hys_sstr);
  mso->Save( 7, purePlasticPoint); // **** sp_damp);
  mso->Save( 8, u_old);
  mso->Save( 9, u_zero);
  mso->Save(10, u_zeroo);
  mso->Save(11, u_max);
  mso->Save(12, powerNumber);
  //mso->Save(13, kn_plastic);
  return(0);
}


/* EOF */
; ************************************************************************
```

Code for Energy-Tracing Functions

Code for Energy-tracing functions (including five programs: kinetic_tracing, friction_tracing, bond_tracing, strain_tracing, and body_tracing) and an example using the Energy-tracing functions.

```
; ********************************************************************

def kinetic_tracing
   array ball_energy(7) ; to store kinetic energy of each ball; including rotation energy.
   n = 0
   bp = ball_head
   loop while bp # null
      Vx = b_xvel(bp)
      Vy = b_yvel(bp)
      Mom_of_in = b_realmoi(bp) ; moment of ineria of ball
      Vr = b_rvel(bp)  ; rotation velocity of ball
      mass_ball = b_realmass(bp)
      n = b_id(bp)
      ball_energy(n) = 0.5*mass_ball*(Vx*Vx + Vy*Vy) + 0.5*Mom_of_in*Vr*Vr
        bp = b_next(bp)
   endloop

   energy1 = ball_energy(2)+ ball_energy(3) + ball_energy(4) + ball_energy(5) ; Kinetic
energy of clump 1
   energy2 = ball_energy(6) + ball_energy(7) ; kinetic energy of clump 2

end
; ********************************************************************

; ********************************************************************

def friction_tracing
   ;
   sign_clump ; call flag function
   ;
   f_work1 = f_work1 ; used to store friction of clump 1
   f_work2 = f_work2 ; used to store friction of clump 2
   cp = contact_head
   loop while cp # null
      if pointer_type(c_ball2(cp)) = 100 then ; indicates this is a ball_ball contact
        if b_ex(c_ball1(cp)) # b_ex(c_ball2(cp)) then
          f_work1 = f_work1 + 0.5*c_slipwork(cp)
          f_work2 = f_work2 + 0.5*c_slipwork(cp)
        endif
        ;
      else
```

```
        if b_ex(c_ball1(cp)) = 0 then ; ball belongs to clump 1
          f_work1 = f_work1 + c_slipwork(cp)
        else
          f_work2 = f_work2 +c_slipwork(cp)
        endif
      endif
      cp = c_next(cp)
    endloop
end
; ************************************************************************

; ************************************************************************

def bond_tracing
  sign_clump ; call flag function
  En = 0.0
  Es = 0.0
  Em = 0.0
  Total_Bond_Energy = 0.0
  Clump_Bond_Energy = 0.0
  ;
  cp = contact_head
  loop while cp # null
    section
      pb = c_pb(cp)
      if pb = null then
        exit section
      endif
      ; To calcualte paralell bond energy
      r1 = b_rad(c_ball1(cp))
        r2 = b_rad(c_ball2(cp))
      ball_rad = min(r1, r2); to get the small radius in the contact balls
      area = pi*(pb_rad(pb)*ball_rad)^2
      inera = 0.25*(pb_rad(pb)*pi*ball_rad)^4
      ;
      En = 0.5*((pb_nforce(pb))^2/(area*pb_kn(pb)))
      Es = 0.5*((pb_sforce(pb))^2/(area*pb_ks(pb)))
      Em = 0.5*((pb_mom(pb))^2/(inera*pb_kn(pb)))
      ; three varialbes above give energy iterms for paralell bond
      ;
     if b_ex(c_ball1(cp)) = b_ex(c_ball2(cp)) then
       if b_ex(c_ball1(cp)) = 0 then
         Total_Bond_Energy = Total_Bond_Energy + (En + Es + Em)
       else
         Clump_Bond_Energy = Clump_Bond_Energy + (En + Es + Em)
       endif
```

```
        else
          Total_Bond_Energy = Total_Bond_Energy + 0.5*(En + Es + Em)
          Clump_Bond_Energy = Clump_Bond_Energy + 0.5*(En + Es + Em)
        endif
      endsection
       cp = c_next(cp)
     endloop
 end
; ************************************************************************


; ************************************************************************

 def strain_tracing
    sign_clump ; call flag function
    strain_energy1 = 0.0 ; used to store strain energy of clump 1
    strain_energy2 = 0.0 ;                ---------of clump 2
    cp = contact_head
    loop while cp # null
      if pointer_type(c_ball2(cp)) = 100 then ; indicates this is a ball_ball contact
           ec = 0.5*(c_nforce(cp)*c_nforce(cp)/(cl_kn) + c_sforce(cp)*c_sforce(cp)/(cl_ks))
             if b_ex(c_ball1(cp)) # b_ex(c_ball2(cp)) then
                   strain_energy1 = strain_energy1 + 0.5*ec
                   strain_energy2 = strain_energy2 + 0.5*ec
             endif
                 ;
           else
             ec = 0.5*(c_nforce(cp)*c_nforce(cp)/(cl_kn) + c_sforce(cp)*c_sforce(cp)/(cl_ks))
              if b_ex(c_ball1(cp)) = 0 then ; ball belongs to clump 1
                 strain_energy1 = strain_energy1 + ec
               else
                 strain_energy2 = strain_energy2 + ec
              endif
      endif
          cp = c_next(cp)
    endloop
end

; ************************************************************************

; ************************************************************************
def body_tracing
    sign_clump ; call flag function
    bp = ball_head
    body_energy1 = 0.0
    body_energy2 = 0.0
```

```
   loop while bp # null
      mgh = b_realmass(bp)*gravy*(b_y(bp)- ball_position(b_id(bp))) ; *** study further!!!
Note positive or negative
      if b_ex(bp) = 0 then
        body_energy1 = body_energy1 + mgh
        ; to computer all body energy in clump 1
       else
        body_energy2 = body_energy2 + mgh
      endif
      bp = b_next(bp)
   endloop
end
; ***************************************************************************
```

An example using energy-tracing functions (*this program may use some code without unknown reference*) including Rock_1.dat, Rock_1.fis, and Rock_2.fis.

```
; ***************************************************************************
; FNAME: Rock_1.DAT
;
set echo on
set pin 1
new   : *** *** changed
set damp local 0.0
damp default viscous normal 0.2
set random
; An example showing debris moving down a slope.  Extensive use
; is made of FISH.  The example is set up to allow experimentation
; with both geometry and properties.
;
; Files in this set:   ROCK_1.DAT
;                      ROCK_1.FIS
;                      ROCK_2.FIS
title 'Rockfall'
;set echo off

def set_fist_env
 environment('itascaFishTank') = 'c:\\itasca\\pfc\\FisTEnv\\'
end
set_fist_env
call %itascaFishTank%\fishPfc\md\fishcall.fis ; *** ***
def setup
 ; declaring variables -- prefix cl_ = CLuster, mt_ = MounTain
 cl_pbs  = 5e5       ; parallel bond shear strength
 cl_pbn  = 5e5       ; parallel bond normal strength
 cl_pbkn  = 1e7      ; parallel bond normal stiffness
 cl_pbks  = 1e7      ; parallel bond shear stiffness
```

```
  cl_pbr  = 1          ; parallel bond radius multiplier
  cl_kn   = 1e7        ; particle normal stiffness
  cl_ks   = 1e7        ; particle shear stiffness
  cl_nb   = 0;5e5      ; contact bond normal strength
  cl_sb   = 0;5e5      ; contact bond shear strength
  cl_fric = 0.2        ; particle friction coefficient
  cl_dens = 1000       ; particle density
  cl_diam = 3.0        ; cluster diameter -- approximate
  cl_damp = 1          ; cluster internal damping factor
  cl_pack = 2; 1       ; cluster packing shape
  cl_balls = 6;  5     ; number of particles in cluster - max 7
  cl_tight = 1; 0.95 ;0.99    ; tightness of packing in cluster
  mt_ks   = 1e8; 1e7   ; mountain wall shear stiffness
  mt_kn   = 1e8; 1e7   ; mountain wall normal stiffness
  mt_fric = 0.1        ; mountain wall friction coefficient
  mt_steep = 2.0       ; mountain average steepness ratio
  mt_nodes = 10        ; number of wall segments to make slope
end

setup ; initializing variables
trace energy on
call rock_1.fis
;*** *** *** ***
; pl create Slope      ; show plot view, setting up a plot view

; *** *** ***
call rock_2.fis
call crk.fis
mountain             ; making the mountain
crk_init
set g 0 -9.8111
def bunch
 command
  set cl_damp 10
  ; del ball  ; *** *** changed ***
  gen id 1 10 rad .1 .2 x 1 3 y 1 3
  prop dens 2000 kn 1e6 ks 1e6
  range name original id 1 10
  set start_id 3 end_id 5
  make_cluster
  prop c_index 1 range name first id mc_start_id mc_end_id
  set start_id 7 end_id 9  cl_balls 3 cl_pack 2
  make_cluster
  prop c_index 2 range name second id mc_start_id mc_end_id
  damp_clusters
 endcommand
```

```
  setup
end
boulder
set damp local 0.0
range name clump1 id 2 5
clump id 1 range clump1
range name clump2 id 6 7
clump id 2 range clump2
pl add clump id on skin on
pl sh
; ***************************************************************************
def kinetic_tracing
   array ball_energy(7) ; to store kinetic energy of each ball; including rotation energy.
   n = 0
   bp = ball_head
   loop while bp # null
      Vx = b_xvel(bp)
      Vy = b_yvel(bp)
      Mom_of_in = b_realmoi(bp) ; moment of ineria of ball
      Vr = b_rvel(bp)  ; rotation velocity of ball
      mass_ball = b_realmass(bp)
      n = b_id(bp)
      ball_energy(n) = 0.5*mass_ball*(Vx*Vx + Vy*Vy) + 0.5*Mom_of_in*Vr*Vr
        bp = b_next(bp)
   endloop

   energy1 = ball_energy(2)+ ball_energy(3) + ball_energy(4) + ball_energy(5) ; Kinetic
energy of clump 1
   energy2 = ball_energy(6) + ball_energy(7) ; kinetic energy of clump 2

end
; ***************************************************************************
;
def sign_clump ; *** To give a flag to each ball belonging to different clump
   bp = ball_head
   loop while bp # null
      if b_id(bp) > 5 then
        b_ex(bp) = 1
       else
        b_ex(bp) = 0
      endif
      bp = b_next(bp)
   endloop
end

def friction_tracing
```

```
    ;
    sign_clump ; call flag function
    ;
    f_work1 = f_work1 ; used to store friction of clump 1
    f_work2 = f_work2 ; used to store friction of clump 2
    cp = contact_head
    loop while cp # null
        if pointer_type(c_ball2(cp)) = 100 then ; indicates this is a ball_ball contact
          if b_ex(c_ball1(cp)) # b_ex(c_ball2(cp)) then
            f_work1 = f_work1 + 0.5*c_slipwork(cp)
            f_work2 = f_work2 + 0.5*c_slipwork(cp)
          endif
          ;
        else
          if b_ex(c_ball1(cp)) = 0 then ; ball belongs to clump 1
            f_work1 = f_work1 + c_slipwork(cp)
          else
            f_work2 = f_work2 +c_slipwork(cp)
          endif
        endif
        cp = c_next(cp)
      endloop
end
; ************************************************************************
; This function is used to trace bond energy

def bond_tracing
    sign_clump ; call flag function
    En = 0.0
    Es = 0.0
    Em = 0.0
    Total_Bond_Energy = 0.0
    Clump_Bond_Energy = 0.0
    ;
    cp = contact_head
    loop while cp # null
      section
        pb = c_pb(cp)
        if pb = null then
          exit section
        endif
        ; To calcualte paralell bond energy
        r1 = b_rad(c_ball1(cp))
          r2 = b_rad(c_ball2(cp))
        ball_rad = min(r1, r2); to get the small radius in the contact balls
        area = pi*(pb_rad(pb)*ball_rad)^2
```

```
        inera = 0.25*(pb_rad(pb)*pi*ball_rad)^4
        ;
        En = 0.5*((pb_nforce(pb))^2/(area*pb_kn(pb)))
        Es = 0.5*((pb_sforce(pb))^2/(area*pb_ks(pb)))
        Em = 0.5*((pb_mom(pb))^2/(inera*pb_kn(pb)))
        ; three varialbes above give energy iterms for paralell bond
        ;
       if b_ex(c_ball1(cp)) = b_ex(c_ball2(cp)) then
         if b_ex(c_ball1(cp)) = 0 then
           Total_Bond_Energy = Total_Bond_Energy + (En + Es + Em)
         else
           Clump_Bond_Energy = Clump_Bond_Energy + (En + Es + Em)
         endif
       else
         Total_Bond_Energy = Total_Bond_Energy + 0.5*(En + Es + Em)
         Clump_Bond_Energy = Clump_Bond_Energy + 0.5*(En + Es + Em)
       endif
     endsection
      cp = c_next(cp)
    endloop
end

;*************************************************************************
; ==================this function is used to trace friction of each ball *** =======
; ========================================================================
def strain_tracing
   sign_clump ; call flag function
   strain_energy1 = 0.0 ; used to store strain energy of clump 1
   strain_energy2 = 0.0 ;                ---------of clump 2
   cp = contact_head
   loop while cp # null
     if pointer_type(c_ball2(cp)) = 100 then ; indicates this is a ball_ball contact
          ec = 0.5*(c_nforce(cp)*c_nforce(cp)/(cl_kn) + c_sforce(cp)*c_sforce(cp)/(cl_ks))
            if b_ex(c_ball1(cp)) # b_ex(c_ball2(cp)) then
                  strain_energy1 = strain_energy1 + 0.5*ec
                  strain_energy2 = strain_energy2 + 0.5*ec
            endif
                ;
          else
            ec = 0.5*(c_nforce(cp)*c_nforce(cp)/(cl_kn) + c_sforce(cp)*c_sforce(cp)/(cl_ks))
            if b_ex(c_ball1(cp)) = 0 then ; ball belongs to clump 1
                strain_energy1 = strain_energy1 + ec
              else
                strain_energy2 = strain_energy2 + ec
            endif
     endif
```

```
        cp = c_next(cp)
    endloop
end
;
==========
==========
;
;
**********
**********
; the following functions are used to trace body energy, first get the initial positions of balls
def get_position
    ;n=7; *** attention. varialbles are not suitable to define array?????
    array ball_position(7)

    bp = ball_head
    loop while bp # null
        ball_position(b_id(bp)) = b_y(bp)
        bp = b_next(bp)
    endloop
end
    ;
def body_tracing
    sign_clump ; call flag function
    bp = ball_head
    body_energy1 = 0.0
    body_energy2 = 0.0
    loop while bp # null
        mgh = b_realmass(bp)*gravy*(b_y(bp)- ball_position(b_id(bp))) ; *** study further!!!
Note positive or negative
        if b_ex(bp) = 0 then
          body_energy1 = body_energy1 + mgh
          ; to computer all body energy in clump 1
         else
          body_energy2 = body_energy2 + mgh
        endif
        bp = b_next(bp)
    endloop
end
;
==========
==========

;hist id 1 diag muf
;hist id 2 diag mcf
;
```

```
get_position ; *** to get initial position of balls;
;
set fishcall 0 kinetic_tracing ; Trace kinetic energy ***
set fishcall FC_CYC_MOT friction_tracing ; *** to calculate the friction at each step.
set fishcall 0 strain_tracing ; Trace strain energy of each clump
set fishcall 0 body_tracing ; Trace body energy of each clump
set fishcall 0 bond_tracing ; Trace paralell bond energy of each clump
;
;pl create Slope        ; setting up a plot view
;pl add ball lblue lred lgree cyan
;pl add wall
;pl add cbond mag
;pl add pbond yell
;pl add vel bla
;pl set win po 0 0 size .5 .8

; ----- FISH function to create rubble at the top
;  of the slope - shows how to make multiple calls to the
;  make_cluster function and how the ranges can be utilized.


set echo on
; You may change any of the SETtings with the SET command, then enter
;   bunch     to create a bunch of particles and clusters
;   boulder   to create a single cluster
;   mountain  to create a different (random) slope
;
; Then type   plot show   to see the model
;
; EOF: Rock_1.DAT
; **********added ********** *** ***

;his id 11 energy bond
his id 12 energy fric
;his id 13 energy boundary
his id 14 energy body
his id 15 energy kinetic
his id 16 energy strain
;hist id 1000 e1
his id 99 energy1
his id 100 energy2
his id 105 f_work1
his id 108 f_work2
his id 109 strain_energy1
his id 110 strain_energy2
his id 115 body_energy1
```

```
his id 116 body_energy2
his id 1001 Total_Bond_Energy
his id 1002 Clump_Bond_Energy
;
;pl create hist_view
;pl add hist 11 green ma 12 mag 13 bl both 14 yel 15 blue 16
;pl add hist 12 15 99 100 105 108
;pl add his 15 mark 99 100 bo
;pl add his 1001 1002

;pl add his 14 bo 115 116
;pl set win po 0.5 0 size .5 .8
;pl sh

;step 6767

;************************* Added for Movie function ***************

pl add ball lblue lred lgree cyan
pl create Rock_fall
pl set title text "Kinetic Energy-Tracing Program"
pl add wall bl
pl add cbond mag
pl add bal green range clump1
pl add ball mag range clump2
pl add pbond yell
pl add his 15 both 99 mag 100 bo
;pl add vel bla
pl set win po 0 0 size 1.0 1.0
pl set cap size 25
pl sh
def cycle_plot
 loop i(1,100)
   command
   cycle 1
   ;pause
     ;cycle 50
     plot current 1
     movie snap 1 file rock_snap1'.avi
     ;
     cycle 50
     ;plot current 2
     ;movie snap 2 file rock_snap2.avi
   end_command
 end_loop
end
```

```
;

set plot avi size 640 480
movie avi_open file rock_snap1'.avi
;movie avi_open file rock_snap2.avi
; start cycle
pause
cycle_plot
movie avi_close file rock_snap1'.avi
;movie avi_close file rock_snap2.avi

; FNAME: Rock_1.FIS  (Job-specific FISH file for ROCK_1.DAT)

; ----- create an arbitrary mountain slope ----------------------
def mountain
 ; check for existing hill, kill it if it exists
 if mt_start_id > 0
  _start_id = mt_start_id
  _end_id   = mt_end_id
  zap_walls
 endif
 if mt_nodes < 2              ; keep it tidy
  mt_nodes = 2
 endif
 wall_id = max_wid + 1   ; get starting id for slope
 mt_start_id = wall_id         ; save it so we can kill the walls
 x0 = 0.0                  ; we start at the origin and go right
 y0 = 0.0                  ;   and down
 xtable(1,1) = x0             ; table 1 contains x,y coord pairs
 ytable(1,1) = y0             ; table 2 contains slope/convexity data
 loop i (2, mt_nodes+1)
  xtable(1,i)=xtable(1,i-1)+urand*10  ; changed from urad*10, *** ***
  ytable(1,i)=ytable(1,i-1)-urand*10*mt_steep
  _temp=float(xtable(1,i)-xtable(1,i-1))
  xtable(2,i)=(ytable(1,i)-ytable(1,i-1))/_temp
 endloop
 loop i (2, mt_nodes+1)
  if xtable(2,i+1) <= xtable(2,i) then
     ytable(2,i) = 0         ; convex at node
   else
     ytable(2,i) = 1         ; concave at node
  endif
 endloop
  x1 = xtable(1,1)  ; *** ****
  y1 = ytable(1,1)
  x2 = xtable(1,2)
```

```
    y2 = ytable(1,2)
    command
      wall id wall_id kn mt_kn ks mt_ks fric mt_fric node x1 y1 x2 y2
    endcommand
  loop i (2, mt_nodes)
    if ytable(2,i) = 0          ; convex, so add segment
       x1 = xtable(1,i+1)
       y1 = ytable(1,i+1)
       command
        wall id wall_id kn mt_kn ks mt_ks fric mt_fric node x1 y1
       endcommand
     else                       ; concave, so add new wall
       x1 = xtable(1,i)
       y1 = ytable(1,i)
       x2 = xtable(1,i+1)
       y2 = ytable(1,i+1)
       wall_id = wall_id + 1
        command
         wall id wall_id kn mt_kn ks mt_ks fric mt_fric node x1 y1 x2 y2
        endcommand
    endif
  endloop
  x1 = xtable(1,mt_nodes+1)     ; make a landing zone
  y1 = ytable(1,mt_nodes+1)
  x2 = x1 * 5.0 ; *** *** original is x1*2.0
  y2 = y1
  wall_id = wall_id + 1
  command
    wall id wall_id kn mt_kn ks mt_ks fric mt_fric node x1 y1 x2 y2
  endcommand
  mt_end_id = wall_id
end

;----- make a new boulder to roll down the hill ---------
def boulder
  rr = cl_diam / 3.0            ; radius of main article - approx!
  xx = (xtable(1,2)-xtable(1,1))/2.0 + 3.5 * rr  ; make the boulder near
  yy = (ytable(1,2)-ytable(1,1))/2.0 + 3.5 * rr  ;  the top of the slope
  start_id = 1                  ; we know we're creating just one
  end_id = 1                    ;  cluster, so we know the id
  command
   ; del ball                   ; delete all old particles ; *** *** changed
    ball id 1 rad rr x xx y yy   ; add the new main particle
  end_command
  make_cluster                  ; fire up the clustering routine
end
```

```
return
; EOF: Rock_1.FIS

; FNAME: Rock_2.FIS
;------------------ zap_walls -----------------------------------
; FUNCTION : zap_walls
; PURPOSE  : This FISH function deletes all walls within a
;          given ID range.
; VARIABLES: _start_id -- starting id for range
;          _end_id   -- ending id for range
;          msg      -- contains return message string
; USE      : a) assign values to _start_id and _end_id
;          b) execute zapwalls
;
def zap_walls
  _start_id = int(_start_id)
  _end_id   = int(_end_id)
  if _start_id > 0
    if _end_id >= _start_id
      loop zw_i (_start_id, _end_id)
        command
          del wall zw_i
        endcommand
      endloop
      msg = 'Walls ID ' + string(_start_id) + ' to '
      msg = msg + string(_end_id) + ' deleted.'
    else
      msg = '_end_id specified less than _start_id'
    endif
  else
      msg = '_start_id must be greater than zero'
  endif
  zw_i = out(msg)
end
;------------ Cluster creation functions--------------------------
; FUNCTIONS: make_cluster and glom_balls
; PURPOSE  : These functions allow the user to easily create clusters
;          of particles.
; VARIABLES: copy the following block to the start of your data file
;          where you will then easily be able to initialize various
;          options.
;  cl_pbs  = 0          ; parallel bond shear strength
;  cl_pbn  = 0          ; parallel bond normal strength
;  cl_pbkn = 0          ; parallel bond normal stiffness
;  cl_pbks = 0          ; parallel bond shear stiffness
;  cl_pbr  = 0          ; parallel bond radius
```

```
;  cl_kn   = 1e7       ; particle normal stiffness
;  cl_ks   = 1e7       ; particle shear stiffness
;  cl_nb   = 1e6       ; contact bond normal strength
;  cl_sb   = 1e6       ; contact bond shear strength
;  cl_fric = 0.2       ; particle friction coefficient
;  cl_dens = 2000      ; particle density
;  cl_diam = 2.0       ; cluster diameter -- approximate
;  cl_damp = 5.0       ; cluster internal damping factor
;  cl_pack = 1         ; cluster packing shape 1=close 2=spread
;  cl_balls = 6        ; number of particles in cluster - max 7
;  cl_tight = 0.99     ; tightness of packing in cluster
;
;        There are two return variables of specific value:
;        mc_start_id -> starting id for current group of clusters
;        mc_end_id   -> ending id for current group of clusters
;
; USE    : a) set the above variables
;        b) create some particles
;        c) identify a range of these particles which are to be
;           turned into clusters eg SET start_id = 6 end_id = 17
;        d) run make_cluster
;        e) the clusters created may be grouped into a range
;           using the mc_start_id and mc_end_id variables.
;           eg RANGE NAME myname ID mc_start_id mc_end_id
;        f) make fresh SETtings and go again if needed.
;
; ----- finds the place and sets up the data for the cluster -------
def make_cluster
 mcbp = ball_head            ; local ball pointer storage
 b_max_id = max_bid
 mc_start_id = b_max_id + 1    ; start id for this group of clusters
 num_range = end_id-start_id+1  ; number of main particles involved
 num_done  = 0               ; number of main particles completed
 section
  loop while mcbp # null      ; this way we step thru balls once
   next_mcbp = b_next(mcbp)   ; need to collect now before ball deleted
   if b_id(mcbp) >= start_id  ; is current ball in required range?
    if b_id(mcbp) <= end_id
     glom_balls          ; yes, so glom us a cluster
     num_done = num_done + 1
    endif
   endif
   if num_done = num_range    ; jump off the train
    exit section
   endif
   mcbp = next_mcbp
```

```
  endloop
 endsection
 mc_end_id = b_max_id          ; last id for this collection
 damp_clusters                 ; damp out internal motion
end

;----- given basic data, create particles in a cluster ----------
def glom_balls
 b_max_id = b_max_id + 1         ; will be id for new main ball
 clust_start_id = b_max_id       ; start id of cluster balls
 mb_id  = b_id(mcbp)             ; main ball id
 mb_rad = b_rad(mcbp)            ; main ball radius
 mb_x   = b_x(mcbp)              ; main ball x coord
 mb_y   = b_y(mcbp)              ; main ball y coord
 command
  del ball range id mb_id mb_id   ; make contiguous id range for cluster
  ball id clust_start_id x mb_x y mb_y rad mb_rad
 endcommand
 start_ang = urand * 2.0 * pi     ; random orientation for cluster
 if cl_tight <= 0.0               ; governs overlap of cluster particles
  cl_tight = 1.0
 endif
 pc_rad    = 2.0 * mb_rad * cl_tight
 if cl_balls >= 7                 ; limit of 7 particles
  cl_balls = 7
 endif
 if cl_pack = 1
    inc_ang = 2.0 * pi / (float(cl_balls) - 1.0)
  else
    inc_ang = pi / 3.0
 endif
 loop j (1, cl_balls - 1)
  b_max_id = b_max_id + 1
  xx = pc_rad * cos(start_ang) + mb_x
  yy = pc_rad * sin(start_ang) + mb_y
  command
   ball id b_max_id x xx y yy rad mb_rad
  endcommand
  start_ang = start_ang + inc_ang
 endloop
 command
  range name temp id clust_start_id b_max_id
  prop pb_s cl_pbs pb_n cl_pbn range temp
  prop pb_kn cl_pbkn pb_ks cl_pbks pb_r cl_pbr range temp
  prop kn cl_kn ks cl_ks n_b cl_nb s_b cl_sb range temp
  prop fric cl_fric dens cl_dens range temp
```

```
  endcommand
end

;----- trick to damp out initial velocities ----------
def damp_clusters              ; use cl_damp
  xgrav = gravx  ; *** changed by me from grav_x to gravxt      ; capture user gravity
settings...
  ygrav = gravy
  gravx = 0                  ; ... so we can turn it off....
  gravy = 0
  loop i (1, cl_damp)          ; ... while we let the cluster settle
    command                  ;  down a bit (user sets iterations) ...
      pause key ;added
      cycle 10
      ini xvel 0 yvel 0
      pause key
    endcommand
  endloop
  gravx = xgrav ; *** corrected from xgrax to xgrav         ; ... and finally we reinstate the
user's
  gravy = ygrav              ;  gravitational conditions
end
return
; EOF: Rock_2.FIS

;*************************************************************************
```