Multiscale Modeling of Friction Mechanisms with Hybrid Methods

Xinfei Wang

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Civil Engineering

9/18/2014

Blacksburg, VA

Keywords: Newtonian dynamics, finite element method, friction, multiscale modeling

Multiscale Modeling of Friction Mechanisms with Hybrid Methods

Xinfei Wang

ABSTRACT

This thesis presents a simulation model of sliding process of friction, which combines Newtonian particle dynamics and finite element method to study friction mechanisms that bridging micro and macro scales. In the thesis, it first reviews the importance of studying pavement friction that is associated to safety of drivers, society economics and environmental impact. Then, the hybrid numerical methods of Newtonian particle dynamics and finite element method have been introduced, and the rules to bridge these two methods also have been discussed for solid material that assumes the forces and displacements are continuous at the interface of these two methods. The fundamental theories of friction mechanisms are built upon the surface roughness, adhesion and deformation at the contact between two surfaces. At last, the simulation model of sliding process is presented with the hybrid method, and its visualization and result analysis has been given. At the same time, this thesis also includes the procedures of establishing the simulation of the hybrid methods with C++ programming like the program framework, structure and the major pieces of the program.

ACKNOWLEDGEMENTS

I am sincerely thankful to everyone who helped, encouraged and supported me in the past three years. I would like to express appreciation and thanks to my advisor Prof. Linbing Wang for his kindness, patience and knowledge. It is beneficial to have his guidance not only on research activities but also about living a meaningful life. I would like to thank Prof. Montasir Abbas for serving as my committee member and offers insightful comments. And also, the same thanks go to the committee member Prof. Surot Thangjitham who provides me valuable suggestions and an impressive and beneficial course instruction of fracture mechanics. The graduate coordinator in the department, Mrs. Leigh Anne R. Byrd, always gives me timely help on administrative affairs, so I would like to thank her here as well. Moreover, I would like to thank my fellows of the research group who are really nice to encourage me and discuss with me on the thesis. I would like to specially mention the thanks to my friend Wenjuan Sun for her kindhearted help. At this special moment, I would like to thank my beloved family who always unconditionally support me move forward.

Table of Contents

List of figures

List of tables

1 Introduction

For the pavement structure, there are different types of surface layer constructed with different objectives such as reducing noise, improving drainage, resisting thermal cracks, and minimizing the rutting. And, with all of these surface layers, the friction characteristics of pavement surface could be the common concern to researchers and engineers, because the tire-pavement interaction will cause a complicated stress-state near pavement surface with a great potential of leading to pavement distresses and damages. So far, the mechanisms of pavement friction are still not very clear, especially for its complicity at small scales. This study tries to demonstrate a creative method that can help to understand pavement friction across different scales.

At the beginning of this chapter, the background of the project is introduced about the importance of pavement friction to driver safety, social economics and environmental impact. Sequentially, the overall description of tire-pavement friction problem is given with the assumptions for simulation and modeling. At last, the research outline and objectives of the hybrid multiscale method modeling is presented.

1.1 Background

The friction force between pavement and tires acts as driving and resisting force for vehicles which is crucial for sustaining the stable wheel course and providing the controllable brake, especially in the situation like corning, bad weather, and other emergencies. In the meantime, the friction force is always accompanying of wearing phenomena for both tires and pavements. Pavement surfaces would deteriorate, and aggregates would be abraded and polished under daily traffic loadings, leading to great losses in pavement surface texture. In return, these losses in pavement surface texture would result in a decrease of friction between tires and pavements, leading to catastrophic traffic accidents. When the pavement condition becomes poor and no longer sustains efficient friction force, it brings vehicles into risk of accidents due to weak controlling. Peter (2008) indicated that crash rate is higher in those roadway sections where the pavement surface macrotexture recedes to a low level or the roughness keeps extremely high.

According to the statistics from transport department of UK, while the skidding friction coefficient reduces below 0.45, the accident risk will increase 20 times than it keeps at a value above 0.6, and the accident risk will increase 300 time higher if it goes down to 0.30 (Transport Department of UK, 1994). From the highway safety improvement program conducted by Federal Highway Administration, 70% of wet traffic crashes can be prevented or minimized by improving the friction of pavement (Federal Highway Administration, 2010). Since, the pavement friction is proved to be related to traffic accidents, the pavement friction will be associated to the social economic when traffic accidents bring enormous negative impact on individuals and society including injuries, death, damage of infrastructure, and the indirectly influence such like pain, family suffering, value of losing life. Besides the driving safety, pavement friction owns influence on the gas consumption of cars. Cars driving on pavements with the medium or appropriate friction level generally will consume little gas; and conversely it will consume more gas if the friction is extreme high or low, or with fierce changes in the tire-pavement friction along the roadway. (Hong, 2005; Federal Highway Administration, 2005) explains that a decrease of pavement roughness will cause a decrease of rolling resistance so that promises reduction of fuel consumption. Moreover, pavement friction produces influences on surrounding environment. The examples about these adverse environment impacts include CO_2 and PM (particle matter) emission into the air around the roadway, the rubber particles wearing from tires permeate into soil by rain rushing, and the vibration in friction is the major source of acoustic upset near roadway. Researchers find out the high concentration of inhalable particles around roadways during winter and spring (Amemiya, 1984; Swietlicki, 2004). Also, (Mats, 2008) indicates that particles mechanically generated from wearing are the primary reason for high PM concentration in busy streets and roads environment. The PM is the general index to describe or quantify the fine particles suspended in a gas or liquid, and it supposes to be important for health effects (Brunekreef, 2005). The emission of CO_2 is directly related to the consumption of fuel, which can be effected by rolling resistance of pavement surface. It claims that a reduction of 3.3% fuel consumption can bring in the benefit of saving 48 million liters of fuel and 45.000 tons greenhouse gases per year in Denmark (Bjarne, 2012). About the noise problem due to pavement friction, from the filed experiments, there is no substantial conflict between the requirements for high friction and low noise, or between the low rolling friction and low noise. It provides evidence that it is not necessary to improve noise emission by sacrificing wet friction or rolling resistance and it is possible to find a way to consider both noise reduction and reasonable resistance at the same time (Anon, 2012).

In summary, pavement friction is related to roadway safety, and the accidents will bring personal, economics and society loss. The degradation of pavement by friction will produce effects on environment like CO_2 and PM emission. And also, environment noise can be another concern in pavement friction research.

1.2 Literature review of pavement friction

The tire-pavement friction depends on various factors rather than the inherent property of pavement alone. Only when all the necessary conditions are all considered, it's possible to give an accurate prediction of the tire-pavement friction interaction. Critical factors that affect the pavement friction can be discretized into four categories, shown as Table 1 (Wallman, 2001).

Table 1 Four categories of influencing factors for the tire-pavement friction interaction

Pavement Surface Characteristics	Vehicle Operating Parameters	Tire Properties	Environment
• Micro-texture • Macro-texture • Unevenness • Material Properties • Temperature	• Slip speed • Driving maneuver	• Footprint • Tread design and condition • Rubber composition • Inflation pressure • Load • Temperature	• Climate • Contaminants

The macro-texture can provide the large scale asperities and drainage routes when in a rainy or water situation. Micro-texture of pavement is made of micro-asperities that exist on aggregates surface, and depends on petrographic properties of aggregate which determines the skid resistance performance under various traffic volumes (Webb, 1970; Gramling, 1974). For the aggregate mineralogy, different mineral components bring aggregate different hardness that is essential to skid resistance (Mullen, 1974). Webb (1970) proved that the harder minerals will be more beneficial with a longer polishing. The common used aggregate in pavement concrete such as limestone and dolomite are the types with uniform texture and low hardness that can be easily polished. (Bunett, 1968) shows that the limestone and dolomite aggregate usually owns a friction coefficient below 0.32 because of the quartz grains within the matrix of aggregates. Moreover, the elemental magnesium content, specific gravity, and total acid insoluble residue are important factors to pavement surface resistance since they are indicators of impurity of limestone or dolomite aggregate (West, 2000). The seasonal environment changing can cause the variations on texture and hysteresis loss in tires, leading to great influence on pavement friction (Giles, 1959a). Usually, the friction coefficient will be like a sinusoidal curve versus time, while it is lower in later summer through fall and higher in winter through spring (Giles, 1959a; Jayawickrama, 1998). The reason of seasonal changing curve can be explained as there is more chemical weathering of deicing salts and mechanical weathering of tire chain-lock during winter which leads to more abrasive phenomena on micro-asperities and the voids between macro-asperities filling up with polished fine aggregates (Giles, 1959a; Jayawickrama, 1998). Seasonal temperature changing also has a significant influence on friction. The mechanical properties of tires are sensitive to temperature. As the temperature increases, the tire friction will reduce (Giles, 1959b). In summary, pavement texture, mineralogy and seasonal effects are the three major categories that influence the pavement friction. The other influencing factors include binder content, tire treads, vehicle speed and so on.

Besides characterizing the factors in pavement friction, there are also various classical theories and laboratory tests available for analyzing the friction mechanisms between tire and pavement. In classical theories, the friction force between pavement and tire is calculated as proportional part to the normal force on the interface, and consisting of longitudinal and lateral frictional forces. Friction forces are usually calibrated by using the indices like skidding resistance coefficient and friction number, which can be obtained by both laboratory and field tests. These measurement indices are designed based on the tests of either friction coefficient or surface texture properties, which are significantly different from each other due to test methods, data acquisition and data processing. Hall (2009) presented a comprehensive summarization on pavement friction measurement methods, especially in ASTM and AASHTO. For the friction coefficient, there are four different types of experiments with the corresponding indices (Henry, 2000), including the locked-wheel test calculating the friction number or sliding number, the Mu-Meter measuring the side-force, the Side-Force Coefficient Road Inventory Machine, measuring devices calculating the parameter side-force coefficient, and the fixed-slip devices or variable-slip devices measuring the parameter percent slip. For the surface texture properties, there are three major types of methods to quantify texture including sand patch method, outflow meter, and circular texture meter.

However, these laboratory and field experiments focus on the statistic regression analysis of a couple of parameters based on the observations of phenomena so that lack quantitative explanations about the friction mechanisms. The shortcomings will appear: 1) the various test methods lack of a unified criterion of evaluating the friction condition, especially when both environment conditions and daily traffic significantly influence the test results; 2) these methods are oversimplified and inaccurate, and usually statistical model fail to predict future conditions.

1.3 Description of problem

The friction process can be summarized as the complex physical and chemical interactions at the contact interface of pavement and tires, which includes the mechanical interlock of asperities, the deformation at the real contact area, the microscopic fracture of the aggregate, the molecular adhesion and so on. Since there are some disadvantages of classical research and practice which relies on the empirical relationships observed from the field and laboratory tests, it is important to improve the understandings and quantitative modeling on the pavement friction with creative methods and technologies. According to modern tribology science, traditional friction theory hypothesizes are no longer applicable in various situations under small scales, and they cannot reflect the truth of how the friction is formed at small scales. Friction forces cannot be simply calculated as a proportional part of normal load while it depends on the types of contacts and the various conditions at the interfaces, so further studies should be conducted using microscopy techniques to develop the precise percepts (Hong, 2005). At the microscale, the friction presents more distinguished and complicated features than it looks like at macroscale. The nano tribology methods have strong advantages of studying the characteristics of the surfaces and contact with the help of top technological nano equipment (John, 1994). The essential work to achieve the better modeling includes clarifying mechanisms of the atomic and micro origin of friction, and building quantitative relationships between the micro and macro features of contacting surfaces. Within in small scales, the friction can be discretized into three categories: adhesion friction, deformation friction and elastic rolling friction, and each one has different calculating principles (Gohar, 2008). Multiscale methods have the advantage of incorporated the features at small scales such like nanostructure and property of surfaces, the real contact area mechanics, and the molecular dynamics modeling into the simulation, and also properly consider the model as a relative macro behavior.

In summary, the multiscale methods provide the possibilities that studying the friction problems with combinations of different theories across different scale levels such like continuum mechanics, adhesion theory, micro mechanics and fracture mechanics. Within this thesis, it majorly focuses on developing and proving the multiscale method and its application on friction problem. Moreover, there are basic assumptions have been made to made to simplify the problem including the elastic material, no gravity, and simple geometry surface shape. And, the friction behavior is simply defined as two dimensional sliding with stable velocity.

1.3 Research outline and objective

The primary objective of this study is to develop and prove the multiscale method of investigating friction mechanisms. The hybrid method of Newtonian particle dynamics and finite element method is implemented as the multiscale modeling tool. The modeling and simulation can help us to better understand how the friction at the nanoscale, and to establish the relationship between the pavement texture and friction properties.

The project is implemented in the following steps: 1) review of multiscale methods and the hybrid methods of discrete element and finite element method, and present the programing scheme in C++; 2) propose the fundamental theories that will applied for the simulation, such like the mathematical description of surface roughness, adhesion and deformation at the contact surface; 3) then perform simulations with visualization and parameter analysis.

2 Multiscale modeling with hybrid methods

2.1 Introduction

Multiscale modeling methods take advantage of studying complicated problems at different scales to reveal the fundamental mechanisms of phenomena. The scales associated to multiscale modelling often can be both spatial and temporal across different levels. Classical theories at macroscale may not be applicable or accurate for microscale situations, and the theories at single microscale may cover too little information for macroscale, however the multiscale modeling methods can effectively combine these two types of theories ranging from microscale to macroscale.

There are various multiscale modeling techniques due to the diversities of fundamental methods associated to different scales. For example, in chemistry and bio-science, the quantum mechanics (QM) and molecular dynamics (MD) are often integrated together to investigate the atomic-scale problems. And, for the material science and computational mechanics, it often requires to bridge atomistic and continuum scales. Moreover, even under the same methodology category, it also has different schemes of fulfilling the multiscale modeling, where the scheme of multiscale method means the overall and systemic design to put the modeling into effect. In the QM/MD multiscale method, the schemes will vary with specific choices of computation configurations, inter-atomic potentials, time integration algorithms and so on. For the atomistic-continuum multiscale method, Curtin (2003) presented a comprehensive review on various fulfillment approaches and schemes such like quasicontinuum method (Tadmor, 1996; Shenoy, 1998), the mixture of finite elements, molecular dynamics and semi-empirical tight-binding (Rudd, 2000), the finite element to atomistic method (Kohlhoff, 1991), the fully non-local quasicontinuum method of (Knap, 2001), and the Coupled Atomistic and Discrete Dislocation method (Shilkrot, 2002).

This thesis proposes a scheme of continuum multiscale method across nanoscale to macroscale with the application in studying friction mechanisms. And, it integrates the finite element method and Newtonian particle dynamics method through a lay of interface particles so that it divides the computational domain into three regions, finite element region, Newtonian particle dynamics region and mixture of both named as interface region. The interface region can transfer forces and displacements from the particle dynamics region to the finite element region. The reason of using such a hybrid method is that the particle dynamics is more compatible and flexible to the friction and wear behaviors at the interface, while the finite element method is much less computation demanding, with better promises to extend the system spatial and temporal scale.

2.2 Newtonian dynamics of particles

The discrete analysis method is the numerical approach of describing the physical and chemical behavior under Newtonian mechanics frame which assumes the system is driven and govern by the Newton’s Law. Newtonian dynamics is the classical discipline that describes the motion of objectives over time, and its numerical simulation methods include classical molecular dynamics, discrete element method, granular dynamics and dissipative particle dynamics. The following section presents a brief introduction to fundamental knowledge about Newtonian dynamics, including setting up the motion equations of particles system and the numerical solution method to motion equations.

2.2.1 Particles system

To present a particles-based discrete analysis, it first needs to discretize the materials into the assembly of particles system which consists of a large number of particles with different shapes, and various physical properties can be assigned to particles. To describe the motion evolution of particles system, it needs to identify both environmental influence and internal change of the particles system. The environmental influence here means the external forces and energies that exerts on the system. The internal change indicates the change of system state due to interaction among particles. Among the system, these particles interplay each other via various pairwise forces which can be categorized into two types, contact forces and long range forces. The contact force is produced via direct mechanical contact while the long range forces are computed via potential functions or the force field, without direct contact between particles. The determination of interaction force type in a particles system is dependent on what kind of simulation problem it is. In this paper, both two kinds of interaction types have been employed. Since the particles are subjected to numerous pairwise trajectories forces, according to Newton's laws these particles will move at certain accelerations and velocities, forming different trajectories.

2.2.2 Principles of Newtonian dynamics

1) Position, velocity, and acceleration

The particles system consists of N particles of mass $m_1, m_2, m_3, \dots, m_N$. All the particles $i\ (i = 1, \dots., N)$ are simplified as the spheres with different radius and material properties, but the masses are concentered at the point of center. Setting up an orthonormal basis coordinate system server as the reference frame for particles system, the position of particles $i\ (i = 1, \dots., N)$ at time t can be expressed as the position vector $\boldsymbol{r}_i$ at the time t relative to the origin O of coordinates system,

$$\boldsymbol{r}_i = x_i \boldsymbol{e_1} + y_i \boldsymbol{e_2} + z_i \boldsymbol{e_3}$$

The velocity of the particles denoted $\boldsymbol{v}_i$ is the rate of change of position and the acceleration of the particles denoted $\boldsymbol{a}_i$ is the rate of change of velocity is defined as

$$\boldsymbol{v}_i = \dot{\boldsymbol{r}}_i$$

$$\boldsymbol{a}_i = \dot{\boldsymbol{v}}_i = \ddot{\boldsymbol{r}}_i$$

By giving $\boldsymbol{r}_i$ of particle i, the trajectory of particle can be obtained and the velocity and acceleration can be derived by definition. The rules of decomposing these vectors are ignored here but are technically employed in simulation programs, because decomposing of vectors are more applicable to be programmed.

2) Equation of motion

For each individual particle in two-dimension, it has both translational and rotational motions subjected to the Newton's laws at the same time. These two types of motions are respectively produced by force and torque acting on the particles.

As for a single particle i, the total force on it can be divided into two contributions, external force from environment and internal force due to the contacts of particles in the system. The external forces are usually given by initial conditions, and it won't be discussed here. The total internal forces $\boldsymbol{F}_i$ on particle i, can be further written as,

$$\boldsymbol{F}_i = \sum_{i \neq j} \boldsymbol{F}_{ij}$$

where $\boldsymbol{F}_{ij}$ is the interact force between particle i and particle j. According to Newton's third law,

$$\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}$$

The torque generally acting on a particle can be described with the cross product of position vector and force as follows.

$$\boldsymbol{T}_i = \boldsymbol{r}_i \times \sum_{i \neq j} \boldsymbol{F}_{ij}$$

The transitional motion is produced by contact forces and gravitational forces. the rotational motion only considers the contact forces, thus the Newton's law of motions can be given as (Tsuji, 1992),

$$\ddot{\boldsymbol{r}}_i = \boldsymbol{F}_i / m_i + \boldsymbol{g}$$

$$\dot{\boldsymbol{\omega}}_i = \boldsymbol{T}_i / I_i$$

where $\boldsymbol{r}_i$ is the position vector of the mass center of particle, m_i is the particle mass, $\boldsymbol{g}$ is the gravity acceleration vector, $\boldsymbol{T}_i$ is the summation of torque caused by the contact forces, and $\boldsymbol{\omega}$ is the angular velocity. I_i is the moment of inertia of the circular disk respect to z direction given by

$$I_{iz} = \frac{\mathbf{1}}{\mathbf{2}} m_i R_i^2$$

Applying the Newton's second law for each particle in the system it will get $N \times 2$ equations of motion in total.

2.2.3 Pairwise interactions between particles

The interactions between particles are the driving force of the system and particles' trajectory. These interactions are pairwise between every two particles and described by various theoretical models or laboratory experiments data. The theoretical models can be further divided as contact force models and long range force models. The contact force models are usually related to contact mechanics which is the classic theory first introduced by Hertz who tried to resolve the problem of two elastic bodies contact. And the long range force models are mostly associated with the kind of force effects within a range of distance, such as gravitational, electrostatic, magnetic and intermolecular forces. In this paper, the macroscopic Van der Waals' force has been considered, which the equations are presented by (Hamaker, 1937) that obtain the Van der Waals' force between circular particles using integration method.

1) Contact models of particles

When two solid bodies are brought into contact with each other, there are pressure and adhesion acting in normal and tangential directions on the contact surfaces. Contact mechanics is the theory about describing the stress and deformation at the vicinity of contacting solid bodies.

a) Contact kinematics of spherical particles

The contact kinematics describes the contact direction, deformation, linear velocity and angular velocity due to contact forces and torques. Assuming two spherical particles i and j with radii R_i and R_j, and their centers locate at $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ with traveling velocity $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ and rotational velocity $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_j$ respectively before they contact with each other . The two-dimensional illustration of contact scheme is shown in the following picture.

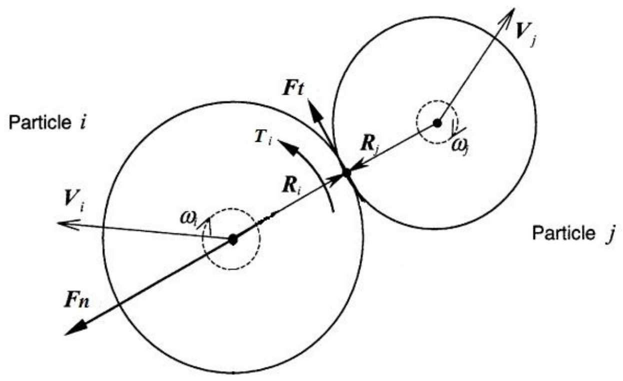


Figure 1 Two-dimensional illustration of two contacting spheres

The pairwise interaction force on particle i that produced from particle j can be decomposed into the normal and tangential part as,

$$\boldsymbol{F}_{ij} = \boldsymbol{f}^n + \boldsymbol{f}^t$$

Besides the translation motion, the rotational motion of particles is produced by the torque of tangential contact forces, for a spherical particle of radius R_i, the torques $\boldsymbol{T}_{ij}$ can be given as,

$$\boldsymbol{T}_{ij} = R_i\, \boldsymbol{n}_{ij} \times \boldsymbol{f}^t$$

Some models also take account of a torque component by rolling friction forces (Zhou, 1999) which is not considered here. In above equation, $\boldsymbol{n}_{ij}$ is the normal contact direction is the unit vector from the center of particle i to particle j,

$$\boldsymbol{n}_{ij} = (\boldsymbol{r}_j - \boldsymbol{r}_i)/\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|$$

The velocity of particle i relative to particle j can be given as,

$$\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j$$

The normal component of relative velocity will be,

$$\boldsymbol{v}_{ij}^n = (\boldsymbol{v}_{ij} \cdot \boldsymbol{n}_{ij})\boldsymbol{n}_{ij}$$

With the influence of angular velocity of both two particles, the relative velocity at the contact point in tangential direction can be givens as,

$$\boldsymbol{v}_{ij}^{s} = \boldsymbol{v}_{ij} - (\boldsymbol{v}_{ij} \cdot \boldsymbol{n}_{ij})\boldsymbol{n}_{ij} + (R_i\boldsymbol{\omega}_i + R_j\boldsymbol{\omega}_j) \times \boldsymbol{n}_{ij}$$

Thus it can use the direction of tangential velocity along the tangential contact direction,

$$\boldsymbol{t}_{ij} = \frac{\boldsymbol{v}_{ij}^{s}}{|\boldsymbol{v}_{ij}^{s}|}$$

The deformation of the contact can be decomposed into normal and tangential deformation, the normal deformation is the overlap of two particles so that it can be directly calculated by

$$\delta_n = (R_i + R_j) - (\boldsymbol{r}_j - \boldsymbol{r}_i) \cdot \boldsymbol{n}_{ij}$$

The δ_n is called mutual compression of particle i and j. In above equations, when the sum of two particle's radii is larger than the distance between their centers ($\delta_n > 0$), it says the two particle i and j are in contact. This rule has been employed to detect contact condition in the programming algorithm.

The tangential deformation is the accumulated tangential displacement and can be calculated by integrating the tangential velocity with time (H. Kruggel-Emden, 2008),

$$\delta_t = \int_{t_0}^{t} \boldsymbol{v}_{ij}^{s} dt$$

Where t_0 is the time of first contact, $t - t_0$ is the duration of contacting. Here use an incremental approach to calculate the total tangential deformation δ_t,

$$\delta_{t+1} = \delta_t + \Delta\delta$$

$$\Delta\delta = \boldsymbol{v}_{ij}^{s}\Delta t$$

Where the *t+1* and *t* means the following and previous time step, Δt is the single time step gap between these two steps.

b) Normal contact forces

The normal contact model is composed of two components which are elastic force represented with spring and energy dissipation represented with a damping. Specifically within the normal contact, both linear and non-linear spring is available to represent elastic force part, and a damping term represents the dissipative contribution to normal forces. Here will present some typical and popular normal contact model.

Linear spring and damping model

The linear spring and damping model is first given by Cundall and Strack (Cundall, 1979) and now has been widely applied. In the model it consists of a dissipative and a linear conservative part, which involves the viscous and elastic forces respectively. The total normal force can be given by the sum of these two parts (Kruggel-Emden, 2007; Tsuji, 1992)

$$\boldsymbol{f}^n = (-k^n \delta_n - \gamma^n \dot{\delta}_n)\boldsymbol{n}_{ij}$$

where k^n is the spring stiffness constant and γ^n is the viscous damping coefficient. And, it assumes that the interaction forces are central directed along the line joining two particles, specifically along the $\boldsymbol{n}_{ij}$ direction from the center of particle i to particle j.

For pairwise collisions, the forces with linear spring and damping model will cause a decrease of the relative normal velocity of the particles by a factor which called coefficient of restitution. The coefficient of restitution is an important characteristic of material properties and can be obtained with experiments (Pöschel, 2005). The relationships between coefficient of restitution and spring stiffness and damping coefficient have been given by (Kruggel-Emden, 2007).

Nonlinear spring and damping model

Based on Hertz theory, there are also several nonlinear contact models have been developed, of which also will take into account of conservative and dissipative components. The notable contributions to nonlinear contact model of normal force have been made by Lee and Herrmann (1993), Kuwabara and Kono (1987). This paper will employ the model set up by Tsuji (Tsuji,

1992) which modify classical Hertz model by changing the exponent of conservative and dissipative term as,

$$f^n = (-k^n \delta_n^{3/2} - \gamma^n \dot{\delta_n}) \boldsymbol{n}_{ij}$$

Within above equations, the methods of deriving the spring and damping coefficient are correspondingly provided by various researchers (Hinrichsen, 2004; Nathan Bell, 2005; Tsuji, 1992 and Cundall, 1979; Zhang, 2005). Tsuji (Tsuji, 1992) also proposed the analytical procedures to determine the γ^n which finally can be expressed with coefficient of restitution. Also, Kruggel-Emden (Kruggel-Emden, 2007) pointed out that the γ^n can be related to bulk viscosities of material involved in the collision. In the situation of lacking bulk viscosities information, the γ^n has been treated as the adjustable parameter in the model (Brilliantov, 1996).

In this paper, to avoid a complicated calculation of contacts so that to improve total computation time consume, it employs following equations to determine the value of k^n and γ^n (Bell, 2005; Tsuji, 1992; Cundall, 1979, Stevens, 2005),

$$k^n = \frac{4}{3} E^* \sqrt{R^*}$$

$$\gamma^n = 2\sqrt{m^* k^n}$$

In above equations, E^* and R^* are the effective Young's modulus and radius of two spheres, and ν_i is the Poisson ratio, and m^* is the effective mass,

$$\frac{1}{E^*} = \frac{1-\nu_i^2}{E_i} + \frac{1-\nu_j^2}{E_j}$$

$$\frac{1}{R^*} = \frac{1}{R_i} + \frac{1}{R_j}$$

$$\frac{1}{m^*} = \frac{1}{m_i} + \frac{1}{m_j}$$

Hysteretic, adhesive model

The hysteretic models are the advanced models that usually take into account of plasticity and adhesive effect in the contact. And they employ only the spring with different stiffness during different periods of loading and unloading for the plasticity. Thus, the hysteretic models also can be divided into linear and non-linear ones which depend on what kind of spring it uses. While in the case of dissipation occurs in the large deformation of particles, it can also add a component of dissipative forces related to viscous and velocity of particles. The important contributions to this discussion can be found in (Walton, 1986; Thornton, 1997, 1998; Tomas, 2003; Sadd, 1993; Vu-Quoc, 1999).

It is believed that the adhesive forces are also important contribution to contact mechanisms, especially for the materials behavior with heavy adhesions. Thus, there are also various theories to describe the adhesive model, such as JKR and DMT theory. Brilliantov (Brilliantov, 2007) have discussed about elastic adhesive contact force and offered a series of references about adhesive contact model.

In this paper, due to a consideration of computation simplicity, it doesn't consider hysteretic and adhesive effect at the contact of particles. The adhesive force components are directly represented by long range force model which specifically means the macroscopic Van der Waals forces.

c) Tangential contact forces

The sliding, rolling and torsion of two particles all will produce the forces and torques at the contact which will result in tangential contact forces (Luding, 2008). There are also various models that similarly define the tangential forces at the contact.

Linear spring and damping model

To build up the tangential contact model, the one simple way is to treat it the same as normal contact model but with different ways to determine stiffness and damping coefficient. For a linear spring and damping model, the tangential force can be expressed as (Tsuji, 1992; Cundall, 1979, Mindlin, 1949; and Di Renzo, 2005).

$$f^t = (-k^t \delta_t - \gamma^t \dot{\delta_t}) \boldsymbol{t}_{ij}$$

Within above equations, it employs the similar strategy to obtain the spring and damping coefficient so that avoid a complicated calculation so that to improve computation time consume, the k^t and γ^t are calculated by the following equations (Kruggel-Emden, 2007; Tsuji, 1992; Cundall, 1979),

$$k^t = 8\sqrt{R^*} G^* \delta_n^{1/2}$$

$$\gamma^t = 2\sqrt{m^* k^t}$$

$$G^* = (\frac{2-\nu_i}{G_i} + \frac{2-\nu_j}{G_j})^{-1}$$

where G_i is the shear modulus as $G_i = E_i/2(1+\nu_i)$.

Coulomb's law

Moreover, tangential forces are attributed to surface friction between particles so that Coulomb's law should be applicable of friction behaviors. After the initial contact, there will be a nonzero tangential velocity between these two particles, resulting in the increases of frictional forces. However, for intuitive experiences, there will be limited increases of tangential forces, which usually is given by Coulomb cutoff law in the following expression.

$$F_{max}^t = \mu F^n$$

There are also some advanced and complicated tangential force models, considering the nonlinear, adhesive and plasticity nature. Also, there are various methods have been given to achieve the stiffness and damping coefficient in normal and tangential direction. Consider the implementation difficulty and the computation efficiency, this paper only implement the non-linear model for normal and linear model for tangential forces using spring and both normal and tangential forces contain an damping components.

2) Long range forces

Van der Waals force is a well-known long range force, responsible for viscosity, diffusion, adhesion and surface tension problems. Under the interesting revealing of the adhesive force between small particles, the Van der Waals force equations between spherical particles have been derived (Hamaker, 1937) by calculating the energy of interaction between volumetric bodies as shown in Figure 2.

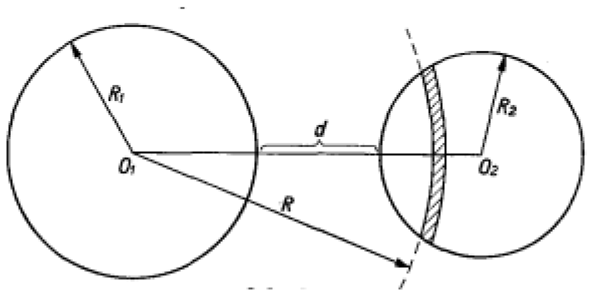


Figure 2 Marcoscopic LJ forces between spheres (Hamaker, 1937)

The potential energy between two spherical bodies is given as,

$$E = -A\frac{1}{12}\left\{\frac{y}{x^2+xy+x}+\frac{y}{x^2+xy+x+y}+2\ln\frac{x^2+xy+x}{x^2+xy+x+y}\right\}$$

Where A is called Hamaker constant related to material properties, and its unit is erg. With the discussion from Hamaker's paper, he concluded that A generally may be assumed to vary between 10^{-14} and 10^{-11} erg as extreme limits and to value between 10^{-13} and 10^{-12} erg in most cases. Moreover, He proposed that the 0.7×10^{-12} erg as an average value.

The x and y in the equation are related to distance and diameter of two particles,

$$x = \frac{d}{2R_1}$$

$$y = \frac{R_2}{R_1}$$

Then the forces can be derived by the differentiation of energy with respect to d,

$$F = \frac{\partial E}{\partial d} = -\frac{A(2x+y+1)}{24R_1}\left\{\frac{-y}{(x^2+xy+x)^2} + \frac{-y}{(x^2+xy+x+y)^2} + \frac{2}{x^2+xy+x} - \frac{2}{x^2+xy+x+y}\right\}$$

2.2.4 Numerical integration algorithms

Newton's equations of motion for conservative physical systems are the ordinary differential equations. Due to the large number of particles and the complicated interaction force, it is usually difficult to find analytic solutions to motion equations. Thus, there are many numerical integration algorithms have been developed with acceptable stability and efficiency. The common algorithms includes velocity Verlet algorithm, leapfrog method, Runge-Kutta method and so on. For a consideration of computation efficiency and simplicity, this paper implements the Euler’s and velocity Verlet algorithm. These two algorithms will provide more options for different simulation applications concern on precision or calculation speed.

(1) Euler method

The advantage of Euler method is to reduce the complexity of computing strategies and computer consumption in single calculation step. And, the disadvantage is that the error item at single step is the second order of time step, thus it requires the time step to be small enough, resulting in the increase of total steps. Consider a function represents position vector x of time t which satisfies,

$$\dot{r}(t) = v(t)$$

$$\dot{v}(t) = a(t)$$

Given the initial value of r_0 and v_0 at time $t = 0$, the above differential equation determines a unique solution of $r(t)$. It’s easy to assume the solution of r and v is a smooth function with well-behaved derivatives, and gives its approximation by Taylor expansion as follows.

$$r(t+dt) = r(t) + v(t)dt + O(dt^2)$$

$$v(t+dt) = v(t) + a(t)dt + O(dt^2)$$

Let the dt be the fixed time step Δt between two solutions, from the equations given by Euler method, it indicates that the local error of both position and velocity is the local truncation error $\mathrm{O}(\Delta t^2)$. And the global error will be sum of all the local errors after n steps which will be of order

$$\sum_{k=1}^{n} \mathrm{O}(\Delta t^2) \sim T\ \mathrm{O}(\Delta t)$$

Where T is the total time. As the equation shows, the Euler method is the first order algorithm for the global error. This implies that the attempt of increasing one more decimal accuracy requires 10 times the number of integration steps. Moreover, the convergence speed of Euler method is relatively slow.

(2) Verlet method

Another commonly used algorithm used in integrating Newton's equations of motion is the Verlet method, which is associated to the work of Verlet (Verlet, 1976) and Störmer. Consider the differential equation of second order of the type,

$$\ddot{r}(t) = a(t)$$

And given the initial conditions of $r(0) = r_0$ and $\dot{r}(0) = v_0$. The Verlet method can present the approximate numerical solution $r(t)$ by considering the sum of Taylor expansion corresponding to forward and reverse time steps,

$$r(t + \Delta t) = 2r(t) - r(t - \Delta t) + a(t)(\Delta t)^2$$

Where Δt is the time step.

Above is the basic Verlet algorithm which doesn't include the velocity explicitly. And note that the basic Verlet algorithm relies on two previous time steps, t and $t - \Delta t$ and gives the solution of forward time $t + \Delta t$. The Verlet algorithm uses positions and accelerations at time t and the positions from time $t - \Delta t$ to calculate new positions at time $t + \Delta t$. This characterization brings the disadvantage effect in computation which requires storing all the information of two previous steps and as well as the calculation result of current step. To overcome such a disadvantage, it can be modified by adding the term of velocity, and it is called velocity Verlet algorithm. By adding the description with velocity, it will be convenient to express the kinetic energy properties of system.

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$

$$v(t+\Delta t) = v(t) + \frac{a(t) + a(t+\Delta t)}{2}\Delta t$$

The local error in position will be $O(\Delta t^4)$ while local error in velocity will be $O(\Delta t^2)$. And, the global error of both position and error are $O(\Delta t^2)$. Usually, the Verlet algorithm is called forth order algorithm which promise a better convergence and precision than Euler method.

2.3 Finite element method

Finite element analysis is one such notable numerical method that gives stable and reliable approximation solutions to differential equations. The methodology of finite element analysis is to discretize the domain of problem into subdomains that connected to each other with specified rules. Because it can choose different shapes of elements and the types of connecting, it becomes available to solve the problems with complicated shapes. Moreover, it allows choosing different approximation function in each element so that it makes the problem in continuum space with infinite freedoms degrade to limited and discrete freedoms. The accuracy will increase as the numbers of elements increase and using better interpolation functions. In this thesis, a basic and simple C++ programming of finite element method has been developed.

2.3.1 Basic equations of continuum mechanics

To solve the problems of stress, strain and deformation in solid mechanics, it needs to establish the governing equations with the kinematic relations, equilibrium equations and constitutive relations. And, the boundary conditions are necessary to solve the initial value problem in finite element method. Therefore, the general form of governing equations is composed of four parts which are equilibrium equations, kinematic relations, constitutive relations and boundary conditions.

Assuming the stress $\boldsymbol{\sigma}$, displacement $\boldsymbol{u}$ and strain at any point $P(x, y, z)$ can be given by matrix,

$$\boldsymbol{\sigma} = [\sigma_x\ \sigma_y\ \sigma_z\ \tau_{xy}\ \tau_{yz}\ \tau_{zx}]^T$$

$$\boldsymbol{u} = [u\ v\ w]^T$$

$$\boldsymbol{\varepsilon} = [\varepsilon_x\ \varepsilon_y\ \varepsilon_z\ \gamma_{xy}\ \gamma_{yz}\ \gamma_{zx}]^T$$

Then these four parts of governing equations can be explained as following:

1) Equilibrium equations

The equilibrium equations are derived from the principals that the force and momentum in the solid body are balanced. But, it is also very common to introduce the first law of thermal dynamics to establish the equilibrium equations. Here, the equilibrium equations relates the internal stress state to the external forces such like point force, surface force or body force like below,

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \overline{f_x} = 0$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \overline{f_y} = 0$$

$$\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \overline{f_z} = 0$$

By transferring above equations to matrix formulation,

$$\boldsymbol{A\sigma} + \overline{\boldsymbol{f}} = \boldsymbol{0}$$

Where $\boldsymbol{A}$ is the matrix of differential operator,

$$\boldsymbol{A} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

The tensor form of equilibrium equations are givens as,

$$\sigma_{ij,j} + \overline{f_i} = 0$$

2) Kinematic relations

The kinematic relations describe the geometrical changing of solid so that it relates the strain to the displacement. Considering the condition of small deformation, the strain can be given by the derivation of displacement in the space,

$$\varepsilon_x = \frac{\partial u}{\partial x}, \varepsilon_y = \frac{\partial v}{\partial y}, \varepsilon_z = \frac{\partial w}{\partial z},$$

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \gamma_{yx}, \gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} = \gamma_{zy}, \gamma_{zx} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} = \gamma_{xz}$$

Thus, the matrix form of relations between displacement and strain can be expressed as,

$$\boldsymbol{\varepsilon} = \boldsymbol{Lu}$$

Where $\boldsymbol{L}$ is the matrix of different operator,

$$\boldsymbol{L} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} = \boldsymbol{A}^T$$

For the tensor form of kinematic relations, it will be more concise,

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

3) Constitutive relations

Both equilibrium equations and kinematic relations are independent of material properties, and it needs to employ physical laws to define the behavior of material, which describes the relations between strain and stress. For this part, it only introduces the simplest one – the elasticity mechanics. For the homogenous and linear elastic material, relation between stress and strain can be given as,

$$\boldsymbol{\sigma} = \boldsymbol{D\varepsilon}$$

Where D is elasticity constant matrix,

$$D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & \frac{\nu}{1-v} & & & \\ \frac{\nu}{1-\nu} & 1 & \frac{\nu}{1-v} & & 0 & \\ \frac{\nu}{1-\nu} & \frac{\nu}{1-\nu} & 1 & & & \\ & & & \frac{1-2v}{2(1-v)} & 0 & 0 \\ & 0 & & 0 & \frac{1-2v}{2(1-v)} & 0 \\ & & & 0 & 0 & \frac{1-2v}{2(1-v)} \end{bmatrix}$$

With the tensor, the equations can be expressed as,

$$\sigma_{ij} = D_{ijkl}\varepsilon_{kl}$$

4) Boundary conditions

The boundary conditions are the forces and displacement constrains that solid body and this will be the initial value of the governing equations to ensure it will have for solutions.

$$\boldsymbol{n\sigma} = \overline{\boldsymbol{T}}$$

$$\boldsymbol{u} = \overline{\boldsymbol{u}}$$

$\boldsymbol{n}$ is the normal direction of the surface, $\overline{\boldsymbol{T}}$ and $\overline{\boldsymbol{u}}$ are the extern constrains at the surface

2.3.2 Numerical solutions of partial differential equations

The numerical analysis of governing equations is to find the approximate solutions to partial different equations with boundary values, and estimate the error to make sure the solution is stable and acceptable. Since the partial different equations appear different properties in different disciplines and problems, and subjected to various constrain conditions, the choices of numerical techniques of finding an approximate solution will be different from case to case. Generally, the numerical analysis techniques of FEM is based on the calculus of variations, and it needs to establish the integral weak form that is equivalent to governing equations, then solve the approximation solution to the weak form of governing equations by energy minimization principles or weighted residual methods. Here, it only introduces the basic and fundamental theories of deriving the numerical solution to problems in continuum mechanics, specifically for the elasticity theory.

The partial differential equations given above are strong form of governing equations which requires strong continuity on the dependent field variables. Usually, it is difficult to obtain the exact solution to strong form of governing equations, thus it adapts the equivalent form to reduce the requirement on the differentiable functions in the field. The weak form is often an integral form.

To obtain the weak form of governing equations, it introduces the principals of virtual work to reform the equilibrium and kinematic functions. The principle of virtual work can be explained as “the sum of works of the internal and external forces done by virtual displacements is zero”. The deformed body should be in balance of forces and the virtual displacements are infinitesimal change. The principle of virtual work includes virtual displacement method and virtual force method which can be employed to reform the equilibrium and kinematic equations to weak form.

1) Virtual displacement method

Given the equilibrium equations and the boundary conditions of forces,

$$\sigma_{ij,j} + \overline{f_i} = 0$$

$$\sigma_{ij} n_j - \overline{T_i} = 0$$

Then the equivalent integral form can be given by,

$$\int_V \delta u_i (\sigma_{ij,j} + \overline{f_i}) dV - \int_{S_u} \delta u_i (\sigma_{ij} n_i + \overline{T_i}) dS = 0$$

And the first part can be further extended and notices that the tensor $\sigma_{ij,j}$ is symmetric and δu_i is the variation of real deformation and it can derive the strain as

$$\delta \varepsilon_{ij} = \frac{1}{2}(\delta u_{i,j} + \delta u_{j,i})$$

So that from the principle of integration by parts,

$$\int_V \delta u_i \sigma_{ij,j} dV = -\int_V \frac{1}{2}\left(\delta u_{i,j} + \delta u_{j,i}\right)\sigma_{ij} dV + \int_{S_u} \delta u_i \sigma_{ij} n_i dS$$

Thus, the final result of weak form of equilibrium and force boundary equations can be given as,

$$\int_V (-\delta \varepsilon_{ij} \sigma_{ij} + \delta u_i \overline{f_i}) dV + \int_{S_u} \delta u_i \overline{T_i} dS = 0$$

2) Virtual force method

Considering the kinematic equations and boundary condition of displacement,

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i})$$

$$u_i = \overline{u_i}$$

Then the equivalent integral form can be given by,

$$\int_V \delta\sigma_{ij}[\varepsilon_{ij} - \frac{1}{2}(u_{i,j} + u_{j,i})]dV + \int_{S_u} \delta T_i(u_i - \overline{u_i})dS = 0$$

It notices that the $\delta\sigma_{ij}$ is the variation of real stress which satisfies the equilibrium $\delta\sigma_{ij,j} = 0$ so that at the boundary surface $\delta T_i = \delta\sigma_{ij}n_j = 0$ and by employing the rule of integration by parts, the final weak form of kinematic and displacement boundary condition equations can be given as,

$$\int_V \delta\sigma_{ij}\varepsilon_{ij}dV - \int_{S_u} \delta T_i\overline{u_i}dS = 0$$

The above weak form of equilibrium and kinematic equations are dependent of material properties so that they apply for linear or non-linear elastic problem, elastic-plastic problems or other material behavior types. Moreover, since the equilibrium and kinematic equations are given under the constrain conditions of small deformations, the weak form should obey the assumptions as well.

2.3.3 Triangular element

The linear triangular element has the simplest formulation among all 2D solid elements and owns good adaptation to complex geometry. Moreover, the topological property of triangular configuration makes it easier to develop automated meshing algorithms than other shape of 2D solid elements.

1) Element displacement mode and interpolation function

Considering a triangular element of uniform thickness t, the nodes of the element are numbered i, j, m in counter-clockwise and each node has two components of displacement u and v as shown in Figure 3 and following equation,

$$\boldsymbol{a}_i = \begin{Bmatrix} u_i \\ v_i \end{Bmatrix} \qquad (i, j, m)$$

So that the element has six node displacement components and six degrees of freedom,

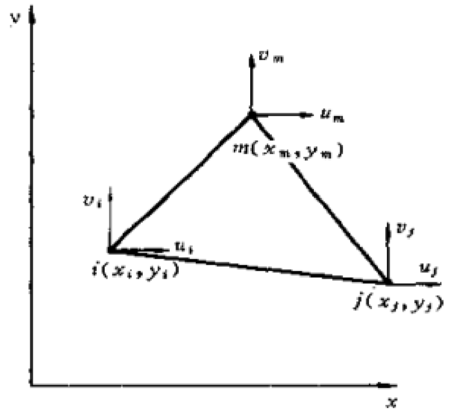


Figure 3 Triangular element and nodes displacement components

The displacement mode or the displacement function of the element is introduced to interpolate the field value of displacement inside the element and at the edge of element. The displacement mode usually uses polynomial functions which ensure simple arithmetic computation and good approximation to smooth curve by increasing their orders. For constructing the displacement function of triangular element, it employs the first-order polynomial function,

$$u = \beta_1 + \beta_2 x + \beta_3 y$$

$$v = \beta_4 + \beta_5 x + \beta_6 y$$

$(u. v)$ is the displacement at point (x, y) and $\beta_1 \sim \beta_6$ are the coefficients to be determined. And, the displacement at nodes with the equation will be,

$$u_i = \beta_1 + \beta_2 x_i + \beta_3 y_i$$

$$u_j = \beta_1 + \beta_2 x_j + \beta_3 y_j$$

$$u_m = \beta_1 + \beta_2 x_m + \beta_3 y_m$$

Then, it gets the solution to $\beta_1 \sim \beta_3$

$$\beta_1 = \frac{1}{D} \begin{vmatrix} u_i & x_i & x_i \\ u_j & x_j & y_j \\ u_m & x_m & y_m \end{vmatrix} = \frac{1}{2A} (a_i u_i + a_j u_j + a_m u_m)$$

$$\beta_2 = \frac{1}{D}\begin{vmatrix} 1 & u_i & x_i \\ 1 & u_j & y_j \\ 1 & u_m & y_m \end{vmatrix} = \frac{1}{2A}(b_i u_i + b_j u_j + b_m u_m)$$

$$\beta_3 = \frac{1}{D}\begin{vmatrix} 1 & x_i & u_i \\ 1 & x_j & u_j \\ 1 & x_m & u_m \end{vmatrix} = \frac{1}{2A}(c_i u_i + c_j u_j + c_m u_m)$$

Where,

$$D = 2A$$

$$a_i = x_j y_m - x_m y_j$$

$$b_i = y_j - y_m$$

$$c_i = -x_j + x_m$$

A is the area of triangular element, in the equations (i, j, m) is in the iterator of replacing each other $\rightarrow j, j \rightarrow m, m \rightarrow i$.

For the same calculation, it can get $\beta_4 \sim \beta_6$,

$$\beta_4 = \frac{1}{2A}(a_i v_i + a_j v_j + a_m v_m)$$

$$\beta_5 = \frac{1}{2A}(b_i v_i + b_j v_j + b_m v_m)$$

$$\beta_6 = \frac{1}{2A}(c_i v + c_j v_j + c_m v_m)$$

Thus, the displacement function can be rewritten as,

$$u = N_i u_i + N_j u_j + N_m u_m$$

$$v = N_i v_i + N_j v_j + N_m v_m$$

Where N_i (i, j, m) is called interpolation function or shape function,

$$N_i = \frac{1}{2A}(a_i + b_i x + c_i y) \qquad\qquad (i, j, m)$$

And it can gives the matrix of shape function $\boldsymbol{N}$,

$$\boldsymbol{N} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix}$$

Then the displacement filed can be given by shape function and displacement of nodes,

$$\boldsymbol{u} = \boldsymbol{N}\begin{Bmatrix} \boldsymbol{a_i} \\ \boldsymbol{a_j} \\ \boldsymbol{a_m} \end{Bmatrix} = \boldsymbol{N}\boldsymbol{a^e}$$

2) Strain matrix and stress matrix

Once the displacement is given by displacement function, it can derivate the strain and stress using the kinematical relations and constitutive relations. The strain is the differential of displacement in the space,

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = L\boldsymbol{u} = L\boldsymbol{N}\boldsymbol{a}^e = L[\boldsymbol{N_i} \quad \boldsymbol{N_j} \quad \boldsymbol{N_m}]\boldsymbol{a^e}$$
$$= [\boldsymbol{B_i} \quad \boldsymbol{B_j} \quad \boldsymbol{B_m}]\boldsymbol{a}^e = \boldsymbol{B}\boldsymbol{a^e}$$

Where L is the differential operator,

$$\boldsymbol{L} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

Then, the strain matrix can be calculated,

$$\boldsymbol{B} = [\boldsymbol{B_i} \quad \boldsymbol{B_j} \quad \boldsymbol{B_m}] = \frac{1}{2A}\begin{bmatrix} b_i & 0 & b_j & 0 & b_m & 0 \\ 0 & c_i & 0 & c_j & 0 & c_m \\ c_i & b_i & c_j & b_j & c_m & b_m \end{bmatrix}$$

The stress of element can be calculated by the constitutive relations D,

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \boldsymbol{D\varepsilon} = \boldsymbol{DBa^e} = \boldsymbol{Sa^e}$$

The stress matrix is $\boldsymbol{S}$,

$$\boldsymbol{S} = \boldsymbol{DB} = \boldsymbol{D}[\boldsymbol{B_i} \quad \boldsymbol{B_j} \quad \boldsymbol{B_m}] = [\boldsymbol{S_i} \quad \boldsymbol{S_j} \quad \boldsymbol{S_m}]$$

$$\boldsymbol{S}_i = \frac{E_0}{2(1-\mu_0^2)A}\begin{bmatrix} b_i & \mu_o c_i \\ \mu_0 b_i & c_i \\ \frac{1-\mu_o}{2}c_i & \frac{1-\mu_0}{2}b_i \end{bmatrix}$$

For plane stress problem,

$$E_0 = E, \mu_0 = \mu$$

For plane strain problem,

$$E_0 = \frac{E}{1-\mu^2}, \mu_0 = \frac{\mu}{1-\mu}$$

3) Stiffness matrix of element

By using the principal of virtual work or variation method, the stiffness matrix can be derivated as,

$$\boldsymbol{K^e} = \int_{\Omega^e} \boldsymbol{B^T DB} t dx dy$$

Since linear triangular element is constant strain element, then the $\boldsymbol{K^e}$ can be further given by,

$$\boldsymbol{K^e} = \boldsymbol{B^T DB} tA = \begin{bmatrix} \boldsymbol{K_{ii}} & \boldsymbol{K_{ij}} & \boldsymbol{K_{im}} \\ \boldsymbol{K_{ji}} & \boldsymbol{K_{jj}} & \boldsymbol{K_{jm}} \\ \boldsymbol{K_{mi}} & \boldsymbol{K_{mj}} & \boldsymbol{K_{mm}} \end{bmatrix}$$

Where the sub-matrix can be expressed with equations

$$\boldsymbol{K}_{rs} = \frac{E_0 t}{4(1-\mu_0^2)A}\begin{bmatrix} \boldsymbol{K_1} & \boldsymbol{K_3} \\ \boldsymbol{K_2} & \boldsymbol{K_4} \end{bmatrix} \qquad (r, s = i, j, m)$$

$$\boldsymbol{K}_1 = b_r b_s + \frac{1-\mu_0}{2} c_r c_s$$

$$\boldsymbol{K}_2 = \mu_0 c_r b_s + \frac{1-\mu_0}{2} b_r c_s$$

$$\boldsymbol{K}_3 = \mu_0 b_r c_s + \frac{1-\mu_0}{2} c_r b_s$$

$$\boldsymbol{K}_4 = c_r c_s + \frac{1-\mu_0}{2} b_r b_s$$

2.3.4 Solution to linear algebra equations

With finite element analysis, it will generate a linear algebra equations system by assembling the element properties matrices. The finite element problem finally comes as the way of solving the linear algebra equations. Usually, there are two categories of obtaining solution of linear algebra equations, the direct methods and iterative methods. The direct methods typically are based on Gauss elimination method which owns high accuracy but will consume much computation for high order matrix. The iterative methods use the approximation strategy corresponding sequence converges which provides the balance between accuracy and efficiency. There are many implementation methods of solving linear algebra equations respects to the matrix properties. For the linear algebra equations in FEM, it owns advantageous properties like symmetry, band diagonal, positive definite and diagonally dominant. In this paper, it adopts the successive over relaxation method. It supposes to be one effective method for large sparse matrix with considerable convergence speed. It is a variant of Gauss-Seidel method which first devised by Young (Young, 1954).

For a linear algebra equations like,

$$Ax = b$$

Its successive over relaxation method solution can be given as,

$$x_i^{k+1} = (1-\omega)x_i^k + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{k+1} - \sum_{j>i} a_{ij}x_j^k\right), i = 1,2,\cdots n.$$

The ω is relaxation factor which determinates the convergence speed. $\omega = 1$ is the situation same as Gauss-Seidel method. It has been proved that the solution will definitely convergent while A is symmetry and positive definitely and $0 < \omega < 2$.

2.4 Hybrid method of FEM and Newtonian particles dynamics method

2.4.1 Bridging rules

The hybrid method of multiscale modeling in this paper is to present the integration of finite element method and Newtonian particle. The important step is to bridge these two methods with a layer of interface region as shown in Figure 4, where the particles and element nodes are overlapped and directly tied together as the master-slave pairs.

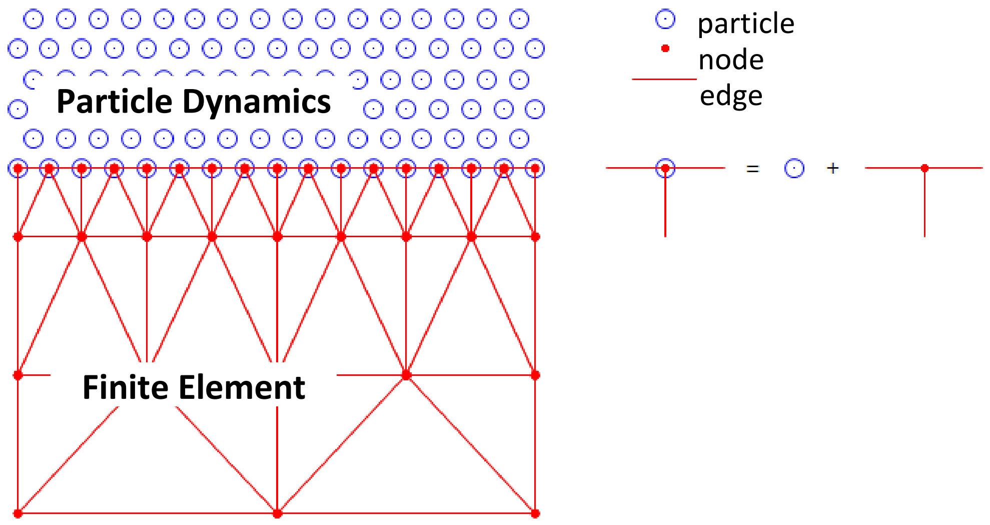


Figure 4 The particle dynamics/finite element coupling rules

The interface region is the touching area between particles and element nodes, where the particles have been directly tied to element nodes. This interface is acting the role of transferring the effect between particle dynamics and finite element method. Moreover, these particles on the interface have full complement of neighbor particles. Their forces F and velocities in particle system can be given at the end of each step. And, displacements Δu can be calculated within the particles system at the end of each step. In this way, since the dynamical particles on the interface coincide with the nodes in FEM mesh, it can subsequently transfer the forces and displacements of interface particles to element nodes and pass the computation process to FEM domain. Thus, the concurrent bridge between two domains can be established by the effect of boundary conditions. This rule directly assigns the force and displacement status of interface particles to finite element method as boundary conditions.

However, these forces and displacement of interface particles need be adjusted before it is applied to FEM region, because they are actually constrained by FEM region like collision against a wall. When calculating the particle dynamics system, it doesn't take account of any influence from FEM region, and particles are just treated at free status at the interface, thus this adjustment is necessary to eliminate such a defect. The adjust rule is simplified as there is an elastic wall against the particles so that it will be the process of Hertzian collision between particles and wall, as the following figure shows.

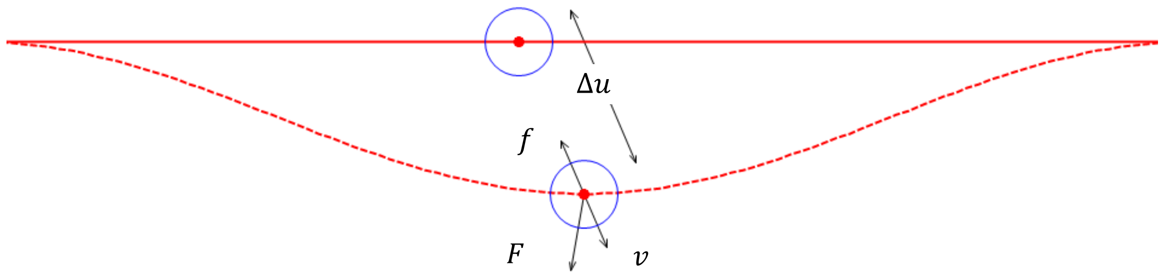


Figure 5 Collision between particle and wall

Where, f is the force from elastic wall and also is the demanded part of adjustment, F is the force calculated from particles dynamics system, v is the velocity of particle and Δu is the actual displacement of particle.

When assume the collision happens in very extremely short time and velocity won't change, it will obtain the adjustment as,

$$\Delta u = v\,\Delta t$$

$$f^n = (-k^n \Delta u^{3/2} - \gamma^n \Delta \dot{u} \quad) \boldsymbol{n}_{ij}$$

Where $\boldsymbol{n}_{ij}$ is the direction of v.

This bridge rule may be only suitable for continuum solid materials, in which the force and displacement can be smoothly passed within the body and between neighbor particles. However, for molecules in molecular dynamics or gas material in granular dynamics, the behavior of particles may vary with high frequency and changing like chaos, it may be better to perform a local generalization before transfer the information of particles into finite element region. This means you can calculate a cluster of particles in a local range to obtain its average value, and then pass these average values into finite element method.

2.4.2 Programming Scheme

The scheme of coupling the finite element method and Newtonian particle dynamics can be explained with following flow chart.

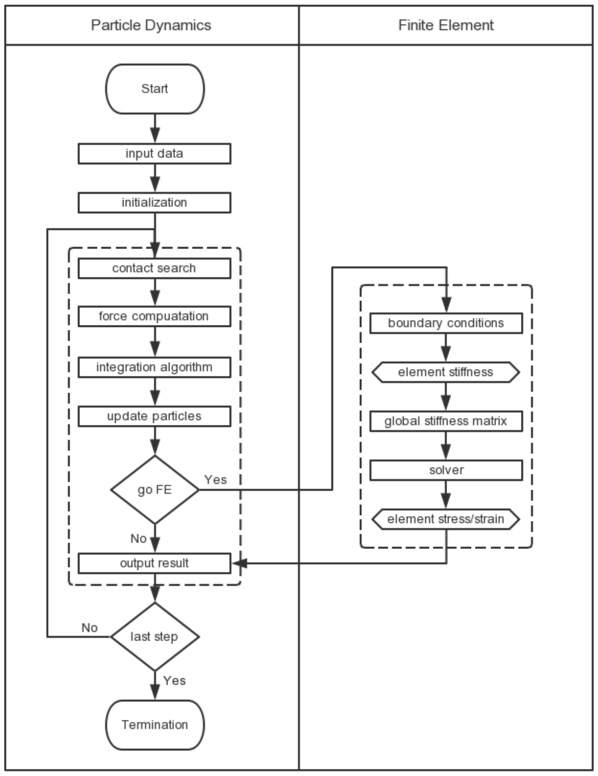


Figure 6 The program scheme of hybrid method

1) Input data: to input initial particle positions and velocities, FEM mesh information and an individual file that contains global and environmental setting of model like total steps, step time, Young’s modulus of material and so on;
2) Initialization: to set up global parameters of simulation, create interface region variables and do stationary computation like the elasticity constant matrix;
3) Contact search: to calculate the distance between particles and build up the neighbor list of contact force and long range force;
4) Force computation: to calculate pair-wise interaction force;
5) Integration algorithm: Euler method or Velocity Verlet method to calculate the changing variables of velocity and position vectors;
6) Update particles: update the variables related to new time steps;
7) Go FE: control structure that determinate if pass the computation process to FE part. Since there are hundreds thousands of steps in particle dynamics and the particles position changes little after each single step, it is not necessary to compute FE every single step corresponding to PD update. It can go to FE within specified steps like every 100 or 1000 steps, this will save considerable computations. However, it should note that it is the total displacement of all the specified steps not the single step. But the force is only related to the last step before going into FE, because the elasticity material is not sensitive to force loading history as long as it hasn’t reached yielding;
8) Boundary conditions: fixed nodes at the edge and also the force and displacement of interface particles;
9) Element stiffness: loop element by element to calculate its stiffness;
10) Global stiffness matrix: to assemble all the element stiffness; this step will also set up all governing equations;
11) Solver: Successive over relaxation method to solve the displacement-based governing equations;
12) Element stress/strain: loop element by element to calculate the stress and strain with the displacement form solver;
13) Output result: print out the results for post-processing.

2.4.3 Programming structure

The major structure and functions of program are related to the simulation scheme. The program will be organized with a main function and contains four different categories of functions which are particle dynamics, finite element method, input/output and general mathematical functions. Moreover, there will be some accessories files to help manage the program including the head files to separate the variations declaim and the make file to easily compile the program.

(1) main function

The main function is the console of the program which determines the computing process.

(2) subfunctions related to particle dynamics

These subfunctions include the input/output, inter-particles forces calculation and integrator of Newton’s equations.

(a) inter-particles forces calculation functions

(b) input/output functions

(c) integrator – Euler method

(3) subfunctions related to finite element method

These subfunctions include the input/output functions and the FEM calculation which integrates all the steps described in the programming scheme such like, global stiffness matrix, boundary conditions and solver of linear equations.

(a) FEM calculation

This function will include the procedures of fulfilling FEM like defining the nodes and elements, calculating the element stiffness, assembling the global stiffness matrix and the solver for the linear equations system.

(b) Input/output functions of finite element method

(4) general mathematical functions

For the whole program, there will have some repeatedly used mathematical functions such as matrix and array operations.

The major piece of the program that illustrates the aforementioned functions has been given in the appendix.

3 Fundament theories of friction mechanisms

In the early age, the friction is described macroscopically with Amontons' laws, which friction is the proportional to normal load, and it is independent of the apparent contact area. Nowadays, researchers find out the friction is more complicated than it looks like. At the small scale level, the friction is influenced by numerous factors such as surface properties, mechanical properties of objects, temperature distribution, localized chemistry properties and so on. Generally, friction force is the integration of several different fragment generated by different laws of mechanics, and mainly from the adhesion force and plowing force in small scale. Thus, the current efforts of understanding and quantifying the friction are to characterize the contact surface properties, to analyze the contact characteristics of contact surfaces, and to simulate the principles of friction. And, a possible approach is to adopt nanotribology and nanochemistry techniques to establish the link between geometrical structure and physical property.

At small scale, friction is produced due to interactions between the two solid surfaces, such as adhesion, plowing, and deformation. Among which, the adhesion and deformation are the major contribution factors to the dry friction in the contact. The adhesion contribution to the friction relates to the interatomic forces that attraction occurs between those atoms are very close and tight. It will grow with the increase of the pressure and the area of contact area, or it will drop while contaminant films are in the contact. The deformation component of friction is due to the physical interaction of two surfaces at the places of asperity.

3.1 Surface asperities and roughness

The texture of real surface is extremely complicated and presents the deviations of peaks and valleys in a three-dimensional space. And, surface topography, asperities or roughness may significantly affect the contact properties, such like the contact area, deformation, adhesion, energy dissipation, and so on. To study the influence of pavement surface in pavement-tire friction, it needs to characterize the pavement roughness and texture of the single aggregate. The interlocks between asperities is the main source of forming mechanical friction force at the macro scale, and surface texture of aggregates can significantly influence the adhesive friction force between tires and aggregates. The general mathematical representation of surface features has analogue solutions and discrete interval solution two major methods.

The pavement texture is classified into four major types according to wavelengths: microtexture, macrotexture, megatexture and roughness, as shown in Figure 8 (Bitelli, 2012).

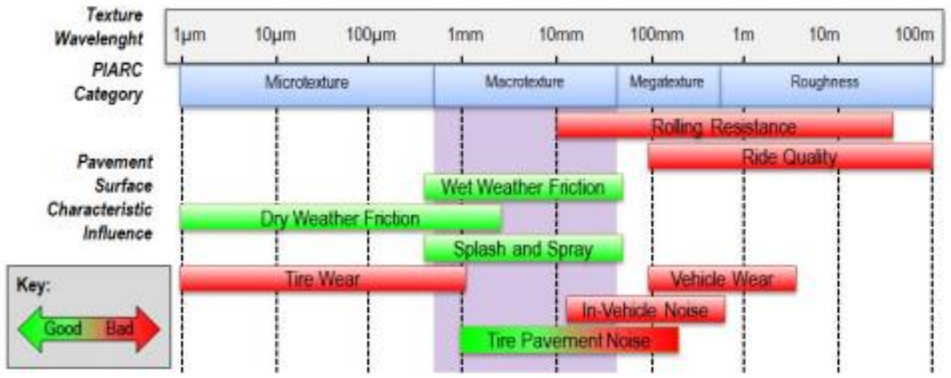


Figure 7 Four classifications of pavement texture (Bitelli, 2012)

Traditional measurements of pavement surface texture are usually in 2D, and now more and more 3D geometry modeling and non-contact tests are available to provide sufficient accuracy to evaluate the surface texture of pavements. Besides the notable results that characterize geometrical structure of pavement surface, there are some indices of quantifying texture features. The currently parameters are mostly the 2D-dependent, and simply geometry calculation based on the 3D surface should be further developed.

The parameters for the description of surface texture of pavements based on 2D profile have following typical ones, (1) average roughness: the average value of (absolute) deviations with reference to mean profile line; (2) peak to valley height: the maximum vertical distance between the highest peak value and the lowest profile valley; (3) leveling depth: the depth resulting from the distance between an average line and a straight line tangential to the profile peak; (4) mean depth: the distance between an average line and a parallel line tangential to the most accentuated cavity, which is the lowest point.

The problem of 2D parameter is that it is hard to describe the integrity shape and there are loss of geometry information due to problem simplification. It suggests developing the 3D parameters that can quantify the curve surface features such like the geodesic torsions, normal curvatures and the angle between surfaces, and so on.

3.2 Deformation

When the surfaces are pressed against each other, there will produce some touch area due to the asperities. The two surfaces are interplayed through the contact area, and the surfaces topographies will change as the contact occurs and evolves subjected to forces. Thus, the deformation appears for both two surfaces. The deformation of the contact can be explained from the atomic changing of surfaces material, when the external forces break the balance between attractive and repulsive forces of atoms so that part of atoms occur relative motions from each other. At the contact, it usually owns both elastic deformations and plastic deformations of surfaces. There are two types of deformation friction. One is the produced force to conquer the shearing displacement with force along the tangent angle. The other one is more important, with the component caused by the ploughing action, which needs to displace a wall of the softer material between these two surfaces. For the ploughing deformation, it relates to the damaging of the softer materials. The analytical solutions of elastic deformations vary as the different assumptions of contact footprint, such as the concentrated contact of hemisphere, the line contact of long cylinders, and the elliptical footprint contacts.

3.3 Adhesion

To study the adhesion in friction process, Bowden and Tabor first introduce the model to calculate the adhesion influences (Bowden, 1954). At the beginning of the loading, it first causes some soften to the material at the tips of the asperity, and produces some plastic yield, and the contact area is only occurring at the area of these asperity tips. And when the loading is keeping applying to the surfaces, the total contact area will increase consequently. The plastic yield at the asperity tips will grow little by little, and at the same time, some new contact area will appear and more part of surface makes the contact. When the loading is completed, the area contact area due to the elastic deformation and plastic yield will result in the strong adhesive bonds at the atomic scale. The adhesive bonds may be the summation of various components including the Van Der Waasls, ionic, and metallic bonds.

When the applied load is W, and assuming that the real contact area is A, and p_m is the mean contact pressure, then the applied load W can be described as,

$$W = A\, p_{\mathrm{m}}$$

And, since the area of the asperities is very close to the fully plastic, it can be assumed that the mean contact pressure roughly equals to the hardness of the softer material, H, so,

$$W \approx A\, H$$

Then, (Bhushsan, 1999) gives out the conclusion that the shears strength at the contacting junction due to the adhesion is very close to the bulk maximum shear stress of the softer material, k. Once the stress of the adhesion area is bigger than k, the contact area will be separated under the force. Thus, the adhesion friction can be give as the value of the real contact area times the maximum stress strength k that the maximum can afford,

$$Fa = A\, k$$

Hence, it gives the frequently cited parameter, coefficient of adhesive friction, μ_a.

$$\mu_{\mathrm{a}} = \frac{Fa}{W} \approx \frac{k}{H}$$

However, this theory is found some bias from the observation in practice.

3.4 Contact area and friction coefficient

Contact area and friction coefficient are two common index used to evaluate the friction behavior. When two surfaces are placed in contact, some regions of the two surfaces are close to each other, and the other regions are still apart. The real contact is sum of the area of junctions where the atom-to-atom forces in these regions is remarkable to be responsible for the friction force. The interaction of two surfaces mainly happens on the real contact area.

To identify the real contact area is extremely complicated, as the real contact area is typically small and being charged in an evolution. Thus, it usually introduces the contact mechanics to estimate the real area contact. To accomplish such an analysis of real contact area, it needs to determine the input parameters such as the geometry of contact area, deformation of contact surfaces, and the forces on the contact, and then to run a finite element analysis based on the models for estimating the real area of contact. The frequently used model is the Greewood and Williamson model, which can predict the average area of contact for plastic or elastic deformation contacts.

4 Simulation modeling and analysis

The friction mechanism of pavement surface can help us understand the degradation of pavement and promise us the improvement of pavement design and construction in future. The asperities of aggregate can be removed and abraded under the deformation or frictional force. And due to the macromolecular forces existed between the tire and aggregate stone, the adhesion between tire and aggregate will also cause shear damage of asperities. The existed models for friction mechanisms relies on plenty of simplification assumptions and few of them can be directly applied in real projects with well acceptable prediction results, so that it requires more accurate and well-developed models for friction process (Meng, 1995). Most of current researches on pavement wearing have been set up to study the statistics assessment and long-term degradation evaluation with little concerns on the mechanical mechanisms (Alexandros,1998; Ibrahim, 2007; Brillet, 1985; Li, 2011). This thesis tries to study pavement friction mechanisms by focusing on developing the multiscale method. Numerical simulation can assist in describing the mechanical process of pavement surface degradation. For the existing numerical analysis methods, it usually employs classical ones like finite element, discrete element and boundary element method. The articles (Podra, 1999; MacGinley, 2001; Yen, 2004) present the FEM (Finite Element Method) modeling of wearing. The BEM (Boundary Element Method) also has been employed in the wearing simulation by various researches (Serre, 2001; Sfantos, 2006; Rodrǵuez, 2010). The method with discrete method can be found in articles (Nicolas, 2004; Dubujet, 1995; Elrod, 1995). Other available friction simulation method may be DEM (Li, 1999; Heinz, 2009). The FEM is widely accepted in friction simulation because of the availability of commercial FEM software, but FEM has its problem for such complicated problem like friction. The feature of friction calls for a power of simulating the particles detaching the main body which is relatively complex and difficult to fulfill through FEM. Due to the limitation on available commercial software, demand of skillful computer language programming, and weakness of building up complicated geometry models, the BEM is still not very popular in addressing the engineering simulation, and also the algorithm for nonlinear problem will be more complicated due to the integration at the singular point. DEM is the most close to the nature of fracture and friction process among these three methods, thus this thesis introduces the similar method to study the microscale process of pavement wearing, the particle dynamics method which is also based on the particle system and Newton’s law.

Within this thesis, it will present the simulation of sliding process of friction. Moreover, the hybrid method of particle dynamics and finite element method will be applied. As it mentioned above, the benefit of such a hybrid method is that the particle dynamics has the advantage of simulating the discontinuity behavior such like crack, large deformation and wear, and also the finite element method is good at simulating the continuum material with low computation consumption.

4.1 Model description

The model is set up as the sliding process of friction so that it will contain two rectangle bodies, namely are the lower and upper body. Each body contains the particles of dynamics system and elements of finite element method. Moreover, the lower body is treated as the substrate layer which is still in the whole process while the upper body will move with a steady velocity that acts as the sliding block. Within the model, the particles of lower and upper body will occur interaction and produce the influence its own finite element region via the interface particles.

4.1.1 Geometry

The geometry and dimension of model is shown as Figure 13.

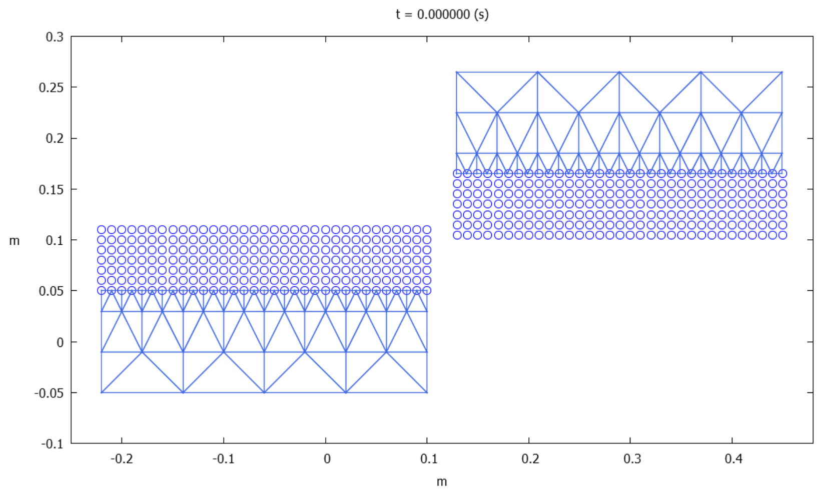


Figure 8 The geometry and dimension of model

Both of the lower and upper body is 0.32 m in length and 0.15 m in height of total. Then, each particle region is 0.32 m long and 0.05 m high. Each finite element method region is 0.32 m long and 0.1 m high.

4.1.2 Model setting

Besides the geometry and dimension set of the model, it also needs to define various parameters for the material. The information all involved in the model has been concluded in Table 2.

Table 2 Parameters of model

	Parameters	Value	Unit
PD	Shape	spherical	-
	Number of particles	462	-
	Diameter	5X10^-3	m
	Density	1000	kg/m^3
	Coefficient of friction	0.5	-
	Young's modulus	1X10^7	Pa
	Poisson's ration	0.333	-
	Time step	1X10^-5	sec
	Number of steps	1.5X10^5	-
FEM	Young's modulus	1X10^7	Pa
	Poisson's ration	0.333	-
	Total elements	168	-
	Element type	plane stress	-
	Element thickness	0.001	m
	Invoking steps	1000	-

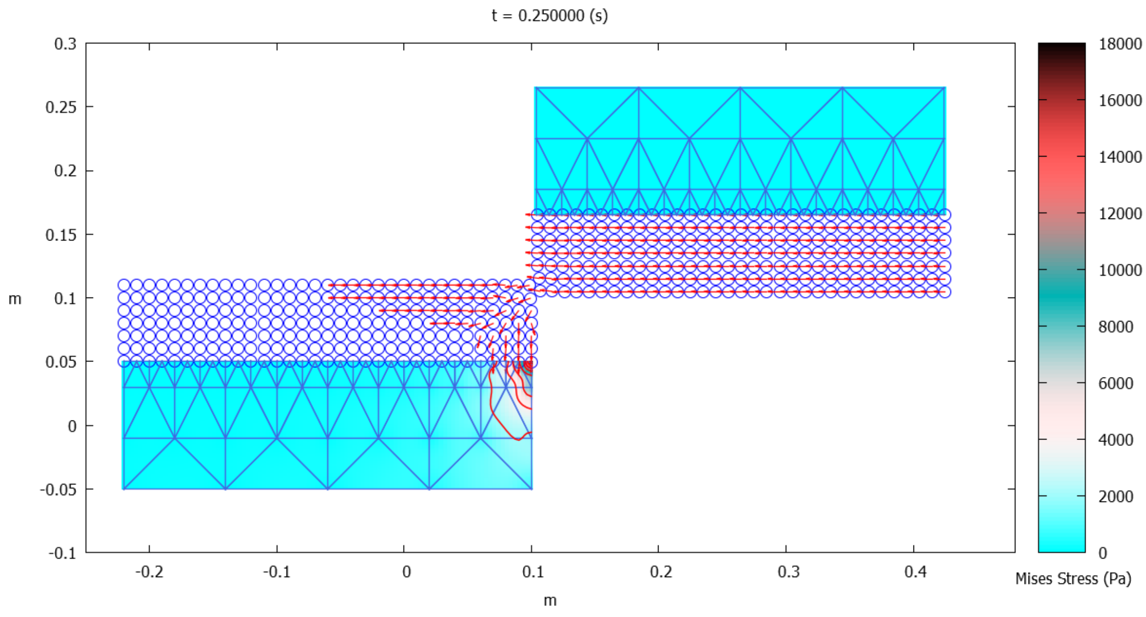
Within particle dynamics, the particle diameter is 0.005 m and the total number of particles are 462. The material properties are set as elasticity with 10 Mpa of Young's modulus, 0.333 of Poisson's ration and density is defined as 2000 Kg/m^3. The time is 1×10^{-5} second per step and 1×10^{5} steps in total. The material properties in FEM model are the same as those in the particle dynamics model. There are 168 plane stress triangle elements in total, with the thickness of 0.001 m. It will invoke the finite element calculation every 1000 steps.

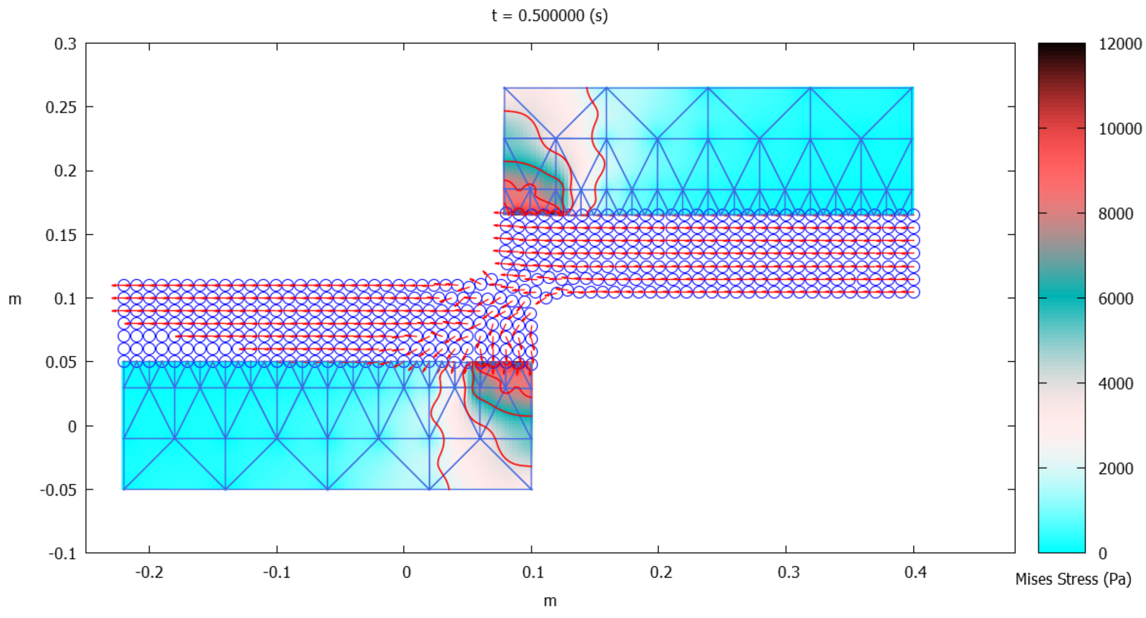
4.2 Results and Analysis

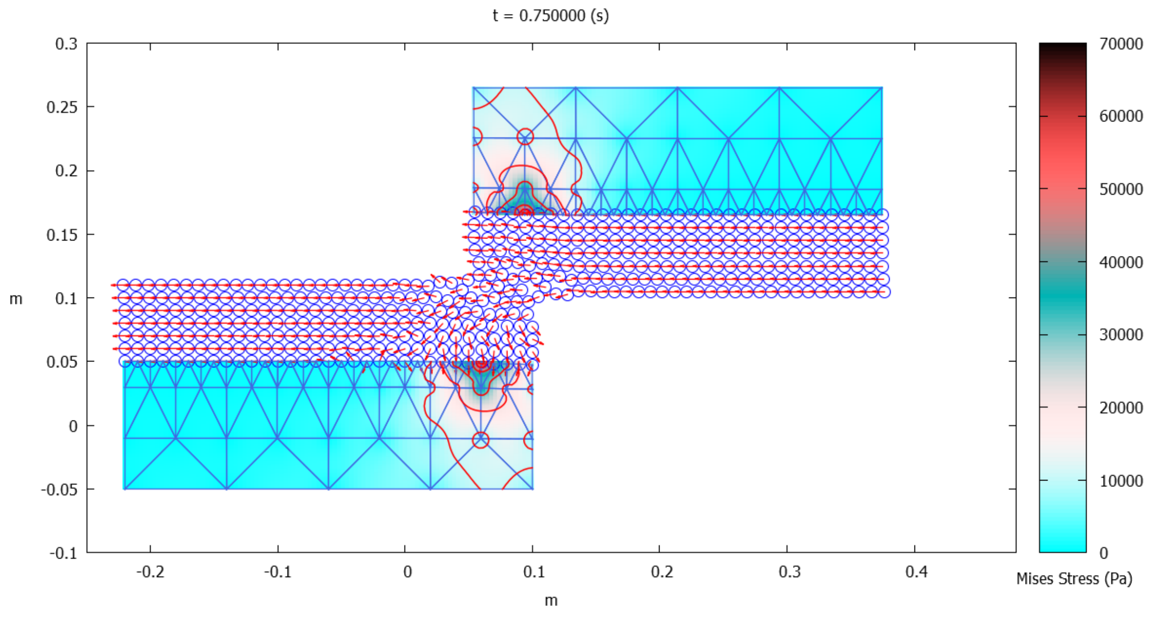
The visualization of simulation is fulfilled using GNUPLOT, a script type graph plot tool in LINUX system. This tool is much simpler than profession post-preprocessing packages or software such like OPENGL, VTK and PARAVIEW. The scripts of GNUPLOT is given as below, the contour functions have been applied to draw the contour of Mise stress, and vector functions are used to plot the velocity field.

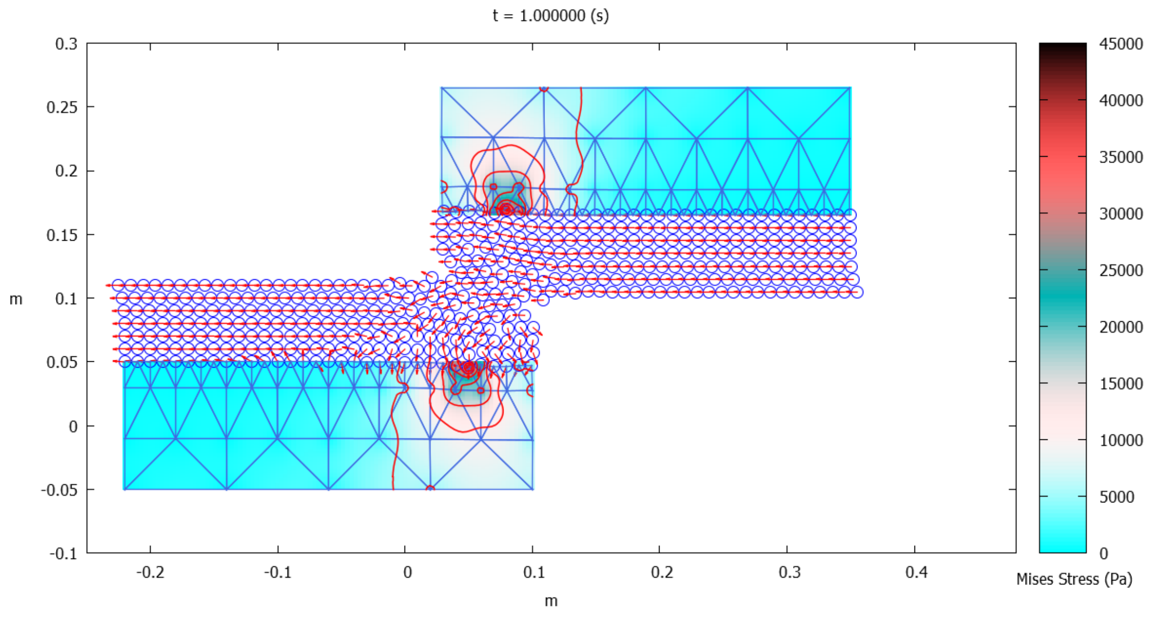
```
system('rm -rf ./animation')
system('mkdir -p animation')

do for [i=0:150000:5000] {
reset
filename1 = sprintf('./output-particle/particle%d.dat',i)
filename2 = sprintf('./output-fem/FEM1gnuplot%d.dat',i)
filename3 = sprintf('./output-fem/FEM2gnuplot%d.dat',i)
filename4 = sprintf('./output-fem/FEM1misesstress%d.dat',i)
filename5 = sprintf('./output-fem/FEM2misesstress%d.dat',i)

reset
set xrange [-0.25:0.48]
set yrange [-0.1:0.3]
set isosample 250, 250
set table 'test.dat'
set dgrid3d 100,100,2
splot filename4 u 1:2:3 w lines
unset table

reset
set xrange [-0.25:0.48]
set yrange [-0.1:0.3]
set isosample 250, 250
set table 'test2.dat'
set dgrid3d 100,100,2
splot filename5 u 1:2:3 w lines
unset table

reset
set xrange [-0.25:0.48]
set yrange [-0.1:0.3]
set contour base
set cntrparam level auto 8
unset surface
set table 'cont.dat'
set dgrid3d 100,100,2
splot filename4 u 1:2:3 w lines
unset table

reset
set xrange [-0.25:0.48]
set yrange [-0.1:0.3]
set contour base
set cntrparam level auto 8
unset surface
set table 'cont2.dat'
set dgrid3d 100,100,2
splot filename5 u 1:2:3 w lines
unset table

reset
set term pngcairo size 1280,640
outfile = sprintf('./animation/animation%d.png',i)
set output outfile
set xrange [-0.25:0.48]
set yrange [-0.1:0.3]
#set autoscale xy
time = sprintf('t = %f (s)',i*1e-5)
set title time
set xlabel "m"
set ylabel "m" rotate  by -360
set label "Mises Stress (Pa)" at 0.48,-0.12
set size ratio -1
set style line 1 lc rgb "red" lt 1 lw 1 pt 7 pi -1 ps 1.5
set style arrow 1 head lt 1 lw 1.2
unset key
set palette rgbformulae 17,10,10
p  'test.dat' w image, 'test2.dat' w image,\
   filename1 u 2:3 w points pt 6 ps 2.0 lc rgb "blue" notitle,\
   '' u 2:3:(0.009*$5)/sqrt($5*$5+$6*$6):(0.009*$6)/sqrt($5*$5+$6*$6) w vec
arrowstyle 1 notitle,\
   filename2 w lp lt 7 ps 0.1 lc rgb "royalblue" lw 1.2 notitle,\
   filename3 w lp lt 7 ps 0.1 lc rgb "royalblue" lw 1.2 notitle,\
   'cont.dat' w l lt 1 lw 1.5, 'cont2.dat' w l lt 1 lw 1.5, \
}
```

Since there are 15000 steps in total and it’s difficult to plot them all, here only gives the frame shot at ever 25000 steps as the following figures,


t = 0.250000 (s)
m
m
Mises Stress (Pa)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.2
-0.1
0
0.1
0.2
0.3
0.4
18000
16000
14000
12000
10000
8000
6000
4000
2000
0


t = 0.500000 (s)
m
m
Mises Stress (Pa)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.2
-0.1
0
0.1
0.2
0.3
0.4
12000
10000
8000
6000
4000
2000
0


t = 0.750000 (s)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
m
-0.2
-0.1
0
0.1
0.2
0.3
0.4
m
70000
60000
50000
40000
30000
20000
10000
0
Mises Stress (Pa)


t = 1.000000 (s)
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
m
-0.2
-0.1
0
0.1
0.2
0.3
0.4
m
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Mises Stress (Pa)

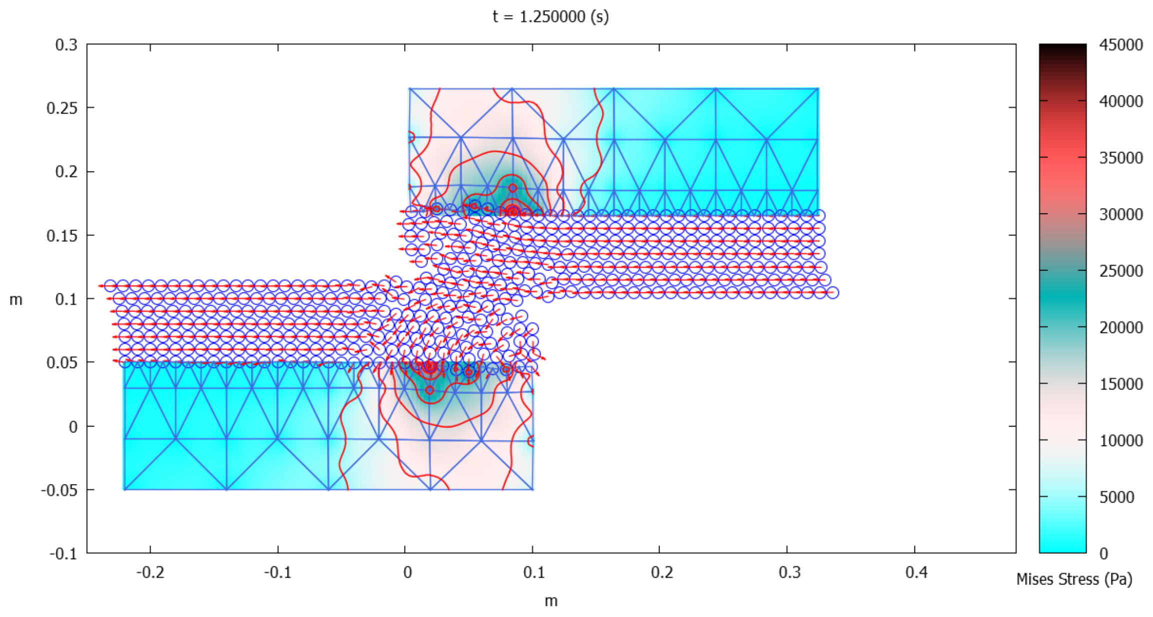


Figure 9 The frame shots of sliding process

4.2.1 Von Mises stress in finite element region

The von Mises stress of each elements have been calculated, and the average value of tall elements in upper and lower body are compared at specified time intervals.

Table 3 Von Mises stress of lower body

	steps	25000	50000	75000	100000	125000	150000
Lower Body	1	6.31628	22.9383	92.3961	72.1213	171.163	144.832
	2	5.95941	21.4933	73.4057	69.3622	158.43	136.118
	3	6.38973	22.9744	69.4152	75.2292	167.972	144.207
	4	7.30073	25.718	72.8418	85.6158	185.619	160.182
	5	8.007	27.9362	74.8829	92.4038	198.419	172.832
	6	10.2266	34.7176	88.634	117.514	239.469	219.427
	7	11.0436	37.3326	91.6518	129.144	252.689	235.175
	8	13.4071	44.813	105.218	155.01	288.79	264.968
	9	14.7955	49.3187	110.712	170.976	308.904	288.011
	10	18.8657	61.9668	134.119	219.431	370.957	355.277
	11	20.7998	68.3672	140.729	242.67	395.385	384.293
	12	27.1363	88.1877	176.042	312.913	498.171	445.316
	13	30.8659	103.242	192.503	356.305	623.79	544.876
	14	42.202	129.856	264.12	480.03	735.182	652.746
	15	46.4372	134.758	285.392	512.318	761.949	672.269
	16	58.26	166.113	351.467	676.205	896.116	710.952

	17	64.9339	182.336	414.596	690.319	1141.17	1053.85
	18	83.5221	230.944	522.061	989.621	1427.64	1233.34
	19	93.2495	256.233	572.689	1064.99	1598.75	4155.89
	20	124.535	332.413	897.189	1452.86	4344.03	8532.18
	21	143.263	378.227	1251.24	2136.24	9736.35	16483.1
	22	193.091	495.202	1771.04	3270.41	15083	32755.7
	23	218.107	544.951	2822.18	5528.31	25259	51326.2
	24	288.957	734.421	4613.16	6778.96	30427.2	21775.9
	25	331.258	887.942	9300.15	10109.8	41301	21633.8
	26	437.289	1329.78	21984.5	13321.7	24710.6	33901.6
	27	534.497	1919.43	32592.3	30810.1	19616.9	31949
	28	759.494	3297.4	37021.6	42264.1	26417.3	32961.8
	29	973.304	11793.6	65281.4	23103.9	19843.7	27000.3
	30	1437.2	11004.6	39754.3	5708.15	17024.7	18973.8
	31	3528.84	10162.5	31352.6	5588.66	21499.5	17289.9
	32	7845.06	11429	19369.6	3768.29	6873.69	5469.04
	33	16429.5	7249.54	7016.5	4493.81	9394.8	8492.64
	34	4.69576	16.4415	57.4366	53.3905	121.174	105.913
	35	5.21089	17.7582	51.4117	58.8773	127.945	114.793
	36	7.66028	25.6928	68.743	85.9792	180.127	163.363
	37	10.6028	34.5894	86.6872	119.796	233.605	226.885
	38	14.1688	46.1618	107.185	159.81	291.004	276.38
	39	20.6584	66.0826	148.628	235.658	406.067	399.582
	40	31.3598	99.6529	207.758	350.877	585.788	532.284
	41	45.4778	135.081	302.514	494.514	761.318	712.714
	42	61.9357	176.254	375.08	647.055	1145.86	867.615
	43	91.6728	257.716	590.804	973.389	2282.94	3813.47
	44	141.976	377.09	1128.76	1723.22	8204.71	12796.4
	45	205.637	529.908	2891.91	4452.08	17729.6	25086.4
	46	314.803	888.024	8322.3	9161.21	26254.2	20665
	47	503.76	2044.38	19139.2	21660	18423.5	27900.1
	48	1077.25	7452.45	35060.1	20778.6	17231.7	23105.5
	49	2377.43	8374.86	18772.9	6327.19	10416.1	10114.2
	50	7346.26	8778.77	8194.67	3999	4695.37	4861.21
	51	4.53157	14.3151	40.2024	52.0531	105.038	109.369
	52	7.95792	25.1295	69.8987	92.0988	181.829	194.819
	53	15.7836	49.4988	132.838	183.628	354.702	393.605
	54	33.7822	105.256	282.675	394.742	749.601	862.609
	55	67.5111	208.869	601.216	838.264	1764.24	2054.04
	56	140.154	431.047	1647.79	2275.09	6454.54	8785.42
	57	306.98	1032.86	6221.76	7531.35	14121.2	16297.5

	58	792.304	3405.81	8378.06	8950.99	10379.4	14165.9
	59	1888.57	5155.77	8611.94	5234.96	4159.62	5639.05
	60	7.29913	23.9028	71.9313	92.2037	180.701	193.743
	61	19.3988	61.7573	173.278	235.79	462.157	508.273
	62	81.5986	258.378	815.945	1135.13	2577.28	3313.85
	63	381.763	1373.26	4074.44	4902.81	8505.55	11080.3
	64	817.829	3050.44	6629.9	5988.81	7294.37	9968.11
Average		1163.694	2427.469	7494.132	5754.401	8820.516	10705.04

Table 4 Von Mises stress of upper body

	steps	25000	50000	75000	100000	125000	150000
Upper Body	65	0	7217.18	7010.15	4551.4	9389.55	8481.06
	66	0	11419.7	19379.6	3795.76	6877.12	5465.11
	67	0	10176.3	31347.1	5611.6	21510.5	17304.6
	68	0	11012.5	39754.7	5716.59	17039.5	18980.7
	69	0	11799.2	65290.3	23088.8	19822.8	26977.4
	70	0	3297.86	37025.1	42260.5	26409.6	32965.6
	71	0	1913.7	32605.9	30816.6	19618.9	31951.6
	72	0	1324.4	22000	13325.8	24705.2	33915.1
	73	0	883.175	9305.11	10110.7	41305.3	21644.5
	74	0	730.073	4615.68	6780.63	30435.5	21769.8
	75	0	541.72	2820.62	5529.7	25257.9	51328.7
	76	0	492.319	1769.59	3271.06	15083.3	32760.5
	77	0	376.149	1248.99	2136.2	9736.96	16485.3
	78	0	330.6	894.629	1452.54	4344.83	8533.61
	79	0	254.955	570.406	1064.82	1598.58	4156.37
	80	0	229.815	519.916	989.186	1427.61	1233.6
	81	0	181.532	412.701	689.928	1140.48	1053.73
	82	0	165.411	349.726	675.829	895.302	710.409
	83	0	134.276	283.872	512.008	761.203	671.687
	84	0	129.439	262.69	479.742	734.414	652.075
	85	0	103.04	191.19	356.104	622.801	544.163
	86	0	88.0335	174.86	312.742	497.226	444.672
	87	0	68.3511	139.657	242.565	394.384	383.538
	88	0	61.981	133.116	219.351	369.989	354.509
	89	0	49.4489	109.771	170.964	307.86	287.174
	90	0	44.9717	104.329	155.025	287.763	264.135
	91	0	37.5713	90.8497	129.219	251.633	234.306
	92	0	34.9839	87.8762	117.62	238.423	218.525
	93	0	28.3064	74.1999	92.5945	197.314	171.853
	94	0	26.1258	72.2014	85.8438	184.516	159.177

	95	0	23.4579	68.8574	75.5316	166.843	143.136
	96	0	21.9808	72.9163	69.6843	157.356	135.096
	97	0	23.5173	91.9456	72.5298	170	143.716
	98	0	8749.22	8197.85	4006.57	4684.68	4847.14
	99	0	8369.93	18778.3	6327.51	10414.8	10108.5
	100	0	7454.7	35071.6	20777.1	17229.1	23098.8
	101	0	2040.97	19147.4	21665.9	18426.1	27905.9
	102	0	883.519	8327.58	9164.79	26254.9	20668.4
	103	0	526.835	2891.89	4454.09	17730.4	25086.9
	104	0	374.957	1126.88	1723.14	8206.09	12798
	105	0	256.399	589.034	973.041	2283.19	3814.34
	106	0	175.408	373.633	646.696	1145.68	867.715
	107	0	134.499	301.474	494.233	761.05	712.536
	108	0	99.3136	206.801	350.665	585.282	531.929
	109	0	65.8958	147.924	235.523	405.614	399.264
	110	0	46.1317	106.501	159.741	290.399	275.911
	111	0	34.6019	86.1499	119.765	233.086	226.481
	112	0	25.8382	68.2183	86.0323	179.462	162.801
	113	0	17.9357	51.0361	58.9759	127.374	114.289
	114	0	16.7191	57.0817	53.5744	120.486	105.285
	115	0	5140.22	8612.87	5220.92	4148.02	5621.47
	116	0	3398.61	8381	8945.67	10372.5	14157.2
	117	0	1028.94	6224.18	7536.24	14122.2	16300.2
	118	0	429.206	1647.01	2275.68	6455.82	8787.23
	119	0	207.932	600.455	837.987	1764.47	2054.46
	120	0	104.778	282.207	394.564	749.651	862.713
	121	0	49.241	132.567	183.506	354.731	393.681
	122	0	25.0062	69.667	92.0182	181.762	194.809
	123	0	14.2318	40.0185	51.989	104.965	109.361
	124	0	3043.16	6634.06	5984.73	7292.79	9966.13
	125	0	1368.63	4074.23	4903.41	8506.4	11082
	126	0	257.319	814.824	1134.82	2577.44	3314.29
	127	0	61.5767	172.79	235.706	461.886	508.08
	128	0	23.9564	71.5892	92.2299	180.322	193.432
Average		0	1681.996	6440.084	4283.531	7004.552	8528.042

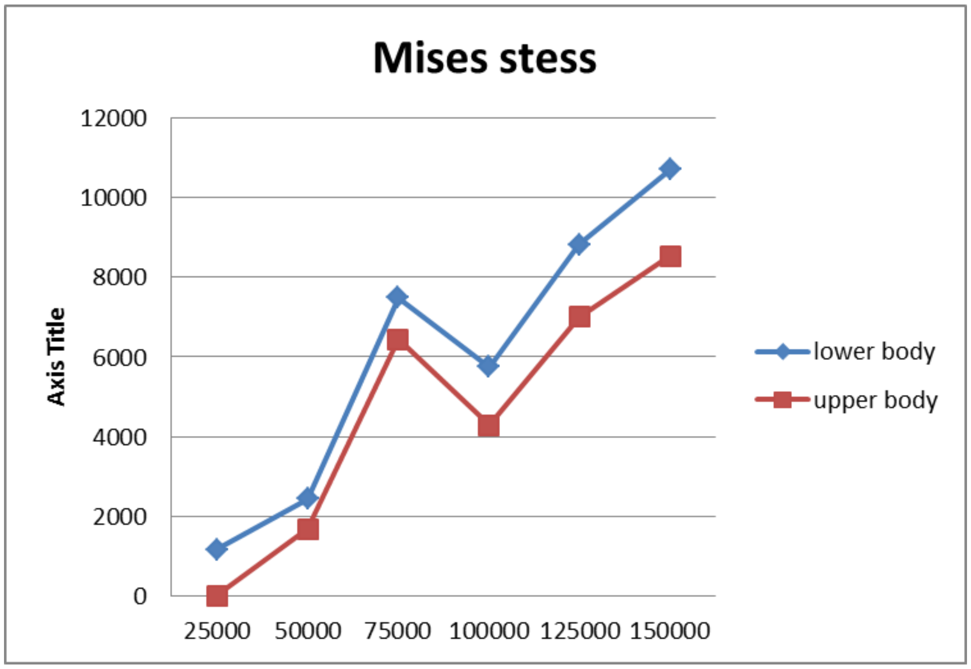


Figure 10 Von Mises stress of lower and upper finite element body

From figure 10, it finds that the average von Mises stress of lower body is always greater than upper body, and both of these two bodies have a drop in von Mises stress synchronously. This may be related the phenomena called stick-slip, i.e., the contact between surfaces may occur stick and slip alternatively.

4.2.2 Velocity and forces of particles

Table 5 and Figure 14 show the average value of velocities and forces of all particles.

Table 5 Average velocities and forces of all particles

steps	*V(m/s)*	*f(N)*
25000	0.500975	0.194851
50000	0.502658	0.170806
75000	0.507214	0.247361
100000	0.506817	0.600103
125000	0.509608	0.769408
150000	0.511583	0.521502

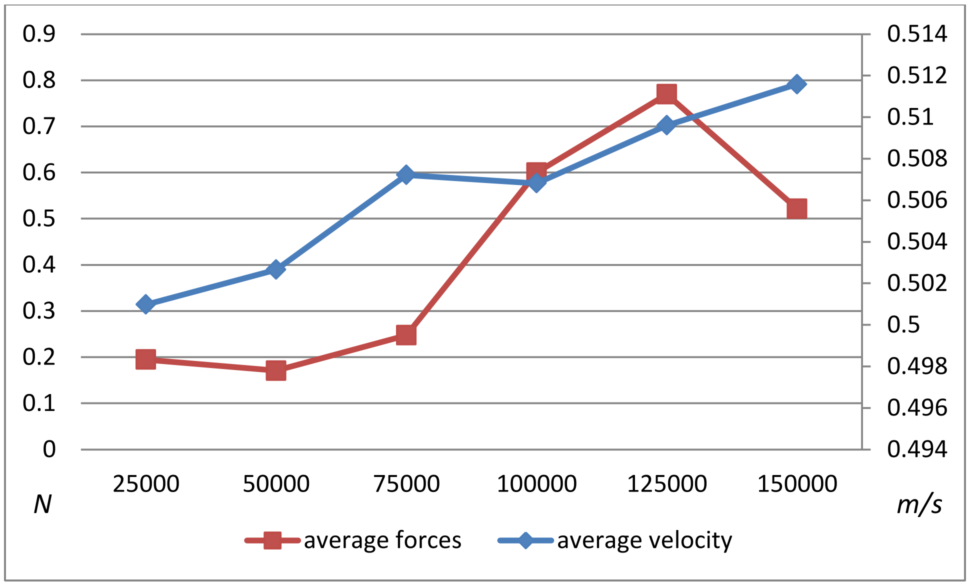


Figure 11 Average velocities and forces of all particles

From the table and figure, the average value of all velocities of particles is increasing as the time elapses. And, the average force shows that at the beginning phase of sliding with a very slight decreasing, and then the sliding becomes more drastic and involves more particles, the average force increases significantly. After that, the average force goes down to a distinguished range, this may be because the particles have been kind of re-arranged and the inner arrangement of particles becomes looser.

5 Summaries and future work

This thesis presents a multiscale modeling program that accomplishes the hybrid methods that combines Newtonian particle dynamics and finite element, and apply it to friction mechanism simulation basically. This multiscale modeling method proves the possibility of develop advanced modeling tool with the consideration of nanoscale surface morphology and nanoscale mechanics and even computation chemistry to study friction or other complicated interface problems.

5.1 Summaries

This hybrid multiscale model combines the finite element method and particle dynamics through the interface particles, with the force and deformation information passing from particles to elements. In particles dynamics, the Hertzian contact has been introduced and Euler method is employed to determine particle motion. Moreover, the Lennard-Jones (LJ) potential function and adhesive force component has been defined between particles so that the modes actually achieves the Hertz and LJ hybrid pair style between interaction particles. In finite element region, the solver of equations is successive over relaxation method which is suitable for the matrix of finite element method. As this thesis mainly focuses on the programming, there is not a lot of quantitative analysis for the post-preprocessing. This may be enhanced in future research.

5.2 Future work

It would rather to see this work as the pro-type program that can truly and precisely describe the friction process on the tire-pavement interface. The C++ codes in this thesis need to be improved in the following aspects.

1) This is 2D model, but it is designed to be applicable for both 2D and 3D problems. The particles dynamics is already compatible to 3D, but it needs add 3D element type to extend finite element model.

2) The preprocessing part of this program only can handle the rectangle and circle geometry, limiting the geometry boundary of this model.
3) The contact forces model between particles are only limited to Hertzian contact.
4) Euler method is not the best choice for accuracy of integrating particles' motion, and the other method may be considered to improve calculation accuracy.
5) The visualization actually is the very important part for the simulation. This thesis uses GNUPLOT, which may not good enough. Other software such as PARAVIEW can be introduced to better visualize outputs from the multiscale model.

The challenge can exist everywhere in this project. Though the multiscale methods and atomic mechanism analysis have the huge advantages in studying the topics, they also inevitably have some their own inadequacies, which can be instanced by the recognized difficulty that the tribological phenomena are the highly non-equilibrium processes, and includes frequent detachment and re-attachment within the microscopic contacts between the surfaces in relative motion. However, it's also necessary for us to precede the research gradually rather than abandon it by fearing its difficulties.

References

Alexandros G. Kokkalis, Olympia K. Panagouli, Fractal Evaluation of Pavement Skid Resistance Variations. II: Surface Wear, Chaos, Solitons & Fractals, Volume 9, Issue 11, 1 November 1998, Pages 1891-1899.

Amemiya S., Y. Tsurita, T. Masuda, etc. Investigation of environmental problems caused by studded tires of automobiles using PIXE, Nucl Instrum Methods Phys Res, B3 , pp. 516–521, 1984.

Anon, Ulf Sandberg and Jerzy A. Ejsmont, Noise emission, friction and rolling resistance of car tires - Summary of an experimental study, 2012.

Bell, Nathan, Yizhou Yu, and Peter J. Mucha. "Particle-based simulation of granular materials. " Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation. ACM, 2005.

Bhushsan, Principles and Applications of Tribology, John Wiley and Sons Press (1999)

Bitelli G., A. Simone, F. Girardi, and C. Lantieri. Laser Scanning on Road Pavements: A New Approach for Characterizing Surface Texture. Sensors. Vol. 12, 2012, pp. 9110-9128.

Brilliantov, Nikolai V., et al. "Collision dynamics of granular particles with adhesion." Physical Review E 76.5 (2007): 051302.

Bowden, Tabor, The Friction and Lubrication of Solids, Clarendon Press (1954).

Brillet F., Skid-resistance properties of highway pavements: Assessment of fourteen years of national skid-resistance surveying (In French) : Bull Liaison Lab Ponts Chaussees N134, Nov–Dec 1984, P5–20, International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts, Volume 22, Issue 6, December 1985, Page 188.

Brilliantov, Nikolai V., et al. "Model for collisions in granular gases." Physical review E 53.5 (1996): 5382.

Brunekreef B., B. Forsberg, Epidemiological evidence of effects of coarse airborne particles on health, Eur Respir J, 26 (2), pp. 309–318, 2005.

Burnett, W.C., Gibson, J.L. and Kearney, E.J., 1968. Skid resistance of bituminous surfaces. Highway Research Record (236): 49-60.

Bjarne Schmidt, Jeppe C. Dyre, CO2 Emission Reduction by Exploitation of Rolling Resistance Modelling of Pavements, Procedia - Social and Behavioral Sciences, Vo.48, pp 311-320, 2012.

Curtin, Willian A., and Ronald E. Miller. "Atomistic/continuum coupling in computational materials science." Modelling and simulation in materials science and engineering 11.3 (2003): R33.

Cundall P. A., Strack O. D. L., A discrete numerical model for granular assemblies. Geotechnique, Vol. 29 No.1, 1979, pp. 47-65.

Di Renzo, Alberto, and Francesco Paolo Di Maio. "An improved integral non-linear model for the contact of particles in distinct element simulations."Chemical engineering science 60.5 (2005): 1303-1312.

Dubujet, P., Ghaoudi, A., Chaze, M., and Sidoroff, F, ‘‘Particulate and Granular Simulation of Third Body Behavior,’’22nd Leeds-Lyon Symposium on Tribology, Elsevier Tribology Series, 31, pp. 355–365. 1995.

Elrod, H. G., 1995, ‘‘Numerical Experiments With Flows of Elongated Granules-Part I,’’22nd Leeds-Lyon Symposium on Tribology, Elsevier Tribol-ogy series,31, pp. 347–354.

Federal Highway Administration, Surface Texture for Asphalt and Concrete Pavements, Technical Advisory 5040.36, Washington, DC, 2005.

Federal Highway Administration, Highway Safety Improvement Program, Pavement Friction, Washington, DC, 2010.

Giles, C.G. and Sabey, B.E., 1959a. A note on the problem of seasonal variation in skidding resistance, First International Skid Prevention Conference. Virginia Council of Highway Investigation and Research, Charlottesville, VA, pp. 563-568.

Giles, C.G. and Sabey, B.E., 1959b. Recent investigations on the role of rubber hysteresis in skidding resistance measurements. In: V.C.o.H.I.a. Research (Editor), First International Skid Prevention Conference. Virginia Council of Highway Investigation and Research, Charlottesville, VA, pp. 219-225.

Gohar R., H. Rahnejat, Fundamentals of Tribology, Imperial College Press, London, 2008.

Gramling, W.L. and Hopkins, J.G., III, 1974. Skid resistance studies: aggregateskid resistance rel ationship as applied to Pennsylvania aggregates. 654, Pennsylvania Department of Transportatio n, Harrisburg, PA.

Hall J.W.. K.L. Smith. L. Titus-Glover, 2009. Guide for Pavement Friction, National Cooperative Highway Research Program Project 01-43, pp. 37-44.

Hamaker, H. C. "The London—van der Waals attraction between spherical particles." physica 4.10 (1937): 1058-1072.

Heinz Kloss, Rolf Wasche, Analytical approach for wear prediction of metallic and ceramic materials in tribological applications, Wear, vol266, 2009.

Henry, J.J., "Evaluation of Pavement Friction Characteristics," NCHRP Synthesis 291, National Cooperative Highway Research Program (NCHRP), Washington, D.C. (2000).

Hertz, Heinrich. " Über die Berührung fester elastischer Körper." J. für die reine u. angew. Math. 92 (1882).

Hinrichsen and D. Wolf (eds.), The Physics of Granular Media, Wiley-VCH (Berlin, 2004), p. 189-209.

Hong Liang, David Craven, Tribology in Chemical-Mechanical Planarization, CRC Press, Boca Raton, FL, 2005.

Ibrahim M. Asi, Evaluating skid resistance of different asphalt concrete mixes, Building and Environment, Volume 42, Issue 1, January 2007, Pages 325-329.

Jayawickrama, P.W. and Thomas, B., 1998. Correction of field skid measurements for seasonal variations in Texas. Transportation Research Record (1639): 147-154.

John Dinardo, Nanoscale Characterization of Surfaces and Interfaces, VCH Press, New York, 1994.

Knap, J., and M. Ortiz. "An analysis of the quasicontinuum method." Journal of the Mechanics and Physics of Solids 49.9 (2001): 1899-1923.

Kohlhoff, S., P. Gumbsch, and H. F. Fischmeister. "Crack propagation in bcc crystals studied with a combined finite-element and atomistic model."Philosophical Magazine A 64.4 (1991): 851-878.

Kruggel-Emden, H., et al. "Review and extension of normal force models for the discrete element method." Powder Technology 171.3 (2007): 157-173.

Kuwabara, K. Kono, Restitution coefficient in collision between two spheres, Japanese Journal of Applied Physics 26 (1987) 1230 –1233.

Lee, H.J. Herrmann, Angle of repose and angle of marginal stability mole cular- dynamics of granul ar particle s, Journa l o f Physics. A, Mathematical and General 26 (2) (1993) 373– 383.

Leon, K.K. McGhee, and I.L. Al-Qadi. 2003. “Pavement Surface Macro-texture Measurement and Application,” Transportation Research Board (TRB), Washington, D.C.

Li Beixing, Guoju Ke, Mingkai Zhou, Influence of manufactured sand characteristics on strength and abrasion resistance of pavement cement concrete, Construction and Building Materials, Volume 25, Issue 10, October 2011, Pages 3849-3853.

Li D.Y., Khaled Elalem, M.J. Anderson, S. Chiovelli, A microscale dynamical model for wear simulation, Wear, 225–229. pp.380–386. 1999.

Luding, Stefan. "Introduction to discrete element methods: Basic of contact force models and how to perform the micro-macro transition to continuum theory." European Journal of Environmental and Civil Engineering 12.7-8 (2008): 785-826.

MacGinley, T., Monaghan, J., Modelling the orthogonal machining process using coated cemented carbide cutting tools. J. Mater. Proc. Technol. vol.118, 293–300. 2001.

Mats Gustafsson, Göran Blomqvist, Anders Gudmundsson, Properties and toxicological effects of particles from the interaction between tyres, road pavement and winter traction material, Science of The Total Environment, vol.393(2–3), pp 226-240, 2008.

Mats Gustafsson, Göran Blomqvist, Anders Gudmundsson, Andreas Dahl, Per Jonsson, Erik Swietlicki, Factors influencing PM10 emissions from road pavement wear, Atmospheric Environment, vol.43(31), pp 4699-4702, 2009.

Meng H.C., Ludema K.C. Wear models and predictive equations: their form and content. Wear, vol.181-183, pp. 443-457, 1995.

Mindlin, R.D., 1949. Compliance of elastic bodies in contact. Journal of Applied Mechanics 16, 259–268.

Mullen, W.G., Dahir, S.H.M. and El Madani, N.F., 1974. Laboratory evaluation of aggregates, aggregate blends, and bituminous mixes for polish resistance. Transportation Research Record(523): 56-64.

Nicolas Fillot , Ivan Iordanoff, Yves Berthier, A granular dynamic model for the degradation of material, Journal Of Tribology-Transactions Of The Asme. pp.606-614. 2004.

Peter Cairne, Paul Bennett, ARRB Group, Relationship between Road Surface Characteristic and Crashes on Victorian rural roads, 23rd ARRB Conference – Research Partnering with Practitioners, Adelaide Australia, 2008.

Podra, P., Andersson, S., Simulating sliding wear with finite element method. Tribol. Int. vol.32, 71–81. 1999.

Pöschel, Thorsten, and Thomas Schwager. Computational granular dynamics: models and algorithms. Springer, 2005.

Rodríguez-Tembleque L., R. Abascal, M.H. Aliabadi, A boundary element formulation for wear modeling on 3D contact and rolling-contact problems, International Journal of Solids and Structures. vol.47. pp.2600–2612. 2010.

Rudd, Robert E., and Jeremy Q. Broughton. "Concurrent coupling of length scales in solid state systems." physica status solidi (b) 217.1 (2000): 251-291.

Sadd, Q.M. Tai, A contact law effects on wave-propagation in particulate materials using distinct element modeling, International journal of Non-Linear Mechanics 28 (2) (1993) 251 –265.

Serre, I., Bonnet, M., Pradeilles-Duval, R.-M., Modelling an abrasive wear experiment by the boundary element method. CR Acad.Sci. Paris 329 (Série II b), 803–808. 2001.

Sfantos G.K., M.H. Aliabadi, Application of BEM and optimization technique to wear problems, International Journal of Solids and Structures, vol.43 pp.3626–3642, 2006.

Shenoy, V. B., et al. "Quasicontinuum models of interfacial structure and deformation." Physical Review Letters 80.4 (1998): 742.

Shilkrot, L.E., R.E. Miller, and W.A. Curtin. "Coupled atomistic and discrete dislocation plasticity." Physical review letters 89.2 (2002): 025501.

Stevens, A. B., and C. M. Hrenya. "Comparison of soft-sphere models to measurements of collision properties during normal impacts." Powder Technology 154.2 (2005): 99-109.

Swietlicki E, T. Nilsson, A. Kristensson, Traffic-related source contributions to PM_{10} near a highway, J Aerosol Sci, 35, pp. S795–S796, 2004.

Tadmor, Ellad B., Michael Ortiz, and Rob Phillips. "Quasicontinuum analysis of defects in solids." Philosophical Magazine A 73.6 (1996): 1529-1563.

Thornton, Coefficient of restitution for collinear collisions of elastic perfectly plastic spheres, Journal of Applied Mechanics—Transactions of the ASME 64 (2) (1997) 383 –386.

Thornton, Z.M. Ning, A theoretical model for the stick/bounce behavior of adhesive, elastic-plastic spheres, Powder Technology 99 (2) (1998).

Transport Department of UK, Design Manual for Roads and Bridges - Skidding Resistance, London, 1994.

Tomas, Mechanics of nanoparticle adhesion --a continuum approach, in: K.L. Mittal (Ed.), Particles on Surfaces 8: Detection Adhesion andRemoval, VSP, 2003, pp. 1 –47154–162.

Tsuji, T. Tanaka, T. Ishida, Lagrangian numerical simulation of plug flow of cohesionless particles in a horizontal pipe, Powder Technology 71 (1992) 239 – 250.

Verlet, Loup. "Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules." Physical review 159.1 (1967): 98.

Vu-Quoc, X. Zhang, An elastoplastic force-displacement model in the normal direction: displacement-driven version, Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences 455 (1999) 4013 – 4044.

Wallman, C.G. and H. Astrom, "Friction Measurement Methods and the Correlation Between Road Friction and Traffic Safety," Swedish National Road and Transport Research Institute, VTI Meddelande 911A, Linkoping, Sweden (2001).

Walton, R.L. Braun, Viscosity, granular temperature and stress calculations for shearing assemblies of inelastic, frictional disks, Journal of Rheology 30 (1986) 949 –980.

Webb, J.W., 1970. The wearing characteristics of mineral aggregates in highway pavements. VH RC-70-R7, Virginia Highway Research Council, Washington, D.C.

West, T.R. and Cho, K.H., 2000. Development of a Procedure to Identify Aggregates for Bituminous Surfaces in Indiana. FHWA/IN/JHRP-2000/28, Joint Transportation Research Program, West Lafayette, IN.

Wriggers, Peter, and G. Zavarise. Computational contact mechanics. John Wiley & Sons, Ltd, 2002.

Yen, Y.-C., So ̇hner, J., Lilly, B., Altan, T., Estimation of tool wear in orthogonal cutting using the finite element analysis. J. Mater. Proc. Technol. 146, 82–91, 2004.

Young, David. "Iterative methods for solving partial difference equations of elliptic type." Transactions of the American Mathematical Society (1954): 92-111.

Zhang, H. P., and H. A. Makse. "Jamming transition in emulsions and granular materials." Physical Review E 72.1 (2005): 011301.

Appendix: Source code

Main cpp file:

```
/**************************************************************************
* Program: GD/DEM/MD-FEM hybrid method simulation                        *
* Author: Xinfei Wang @ Blacksburg, VA, USA.                             *
* Purpose: for multiscale simulation                                     *
* Use: cross platform                                                    *
* Restriction:  the input data file has some format                      *
* Version: June/01/2014.                                                 *
* Features:                                                              *
**************************************************************************/
#include "PD_declartion.h"
#include "FEM_declartion.h"
#include "./preprocessing/preprocessing_model.h"
#include <iomanip>

int Frame_rate=1000;//the frequency of capture FEM calculation
map<int, int> fix_type;

int main()
{
//program running time calculation
clock_t program_run_time;

//pre-processing
preprocessing_model();

/*******************************INPUTDATA(FEM)****************************
Input data: nodes, elements, material properties, element thickness, plane
type(plane strain or plane stress).
**************************************************************************/
//setting of model
const char plane_type[15]="planestress";
double element_thickness=0.001;                                  //unit m
//material
MATERIAL concrete1 = {10e6, 0.33333};                           //unit Pa
MATERIAL concrete2 = {10e6, 0.33333};

//>>>>>>>>>>>>>>>>>>Initialization of FEM BLOCK<<<<<<<<<<<<<<<<<<<<<<<<<//
FEM_input_node_element("input_fem1.dat", node1, element1, Trans_nodes1,
Fix_nodes1);
Fix_nodes1.pop_back();

//boundary condtions of fixed nodes
for(vector<int>::iterator it=Fix_nodes1.begin();
it!=Fix_nodes1.end();++it)
    {   BC_DISPLACEMENT displ1={*it,0,0};
        fix_bc1.push_back(displ1);    }

//boundary conditions of trans nodes, these will vary with particle motion
for(vector<int>::iterator it=Trans_nodes1.begin();
it!=Trans_nodes1.end();++it)
    {   //initial displacement on trans nodes
```

```
        BC_DISPLACEMENT temp;
        temp.ID=*it;
        temp.x=0;
        temp.y=0;
        trans_nodes_displacement1.push_back(temp);
        //initial force loading on trans nodes
        BC_FORCELOAD temp2;
        temp2.ID=*it;
        temp2.x=0;
        temp2.y=0;
        trans_nodes_force1.push_back(temp2);}

//integrate these boundary condtions into displacement and force variables
BC_displacement1.clear();
BC_displacement1.insert(BC_displacement1.end(),trans_nodes_displacement1.b
egin(),trans_nodes_displacement1.end());
BC_displacement1.insert(BC_displacement1.end(),fix_bc1.begin(),fix_bc1.end
());

BC_forceload1.clear();
BC_forceload1.insert(BC_forceload1.end(),trans_nodes_force1.begin(),
trans_nodes_force1.end());

//>>>>>>>>>>>>>>>>>>>>>Initialization of FEM BLOCK<<<<<<<<<<<<<<<<<<<<<<<<//
FEM_input_node_element("input_fem2.dat", node2, element2, Trans_nodes2,
Fix_nodes2);
Fix_nodes2.pop_back();

for(vector<int>::iterator it=Fix_nodes2.begin();
it!=Fix_nodes2.end();++it)
    {   BC_DISPLACEMENT displ1={*it,0,0};
        fix_bc2.push_back(displ1);    }

for(vector<int>::iterator it=Trans_nodes2.begin();
it!=Trans_nodes2.end();++it)
    {   BC_DISPLACEMENT temp;
        temp.ID=*it;
        temp.x=0;
        temp.y=0;
        trans_nodes_displacement2.push_back(temp);

        BC_FORCELOAD temp2;
        temp2.ID=*it;
        temp2.x=0;
        temp2.y=0;
        trans_nodes_force2.push_back(temp2);}

BC_displacement2.clear();
BC_displacement2.insert(BC_displacement2.end(),trans_nodes_displacement2.b
egin(),trans_nodes_displacement2.end());
BC_displacement2.insert(BC_displacement2.end(),fix_bc2.begin(),fix_bc2.end
());

BC_forceload2.clear();
BC_forceload2.insert(BC_forceload2.end(),trans_nodes_force2.begin(),
trans_nodes_force2.end());
```

```
/*********************************INPUTDATA(PD)***************************
Input data: global parameters, particle information
***********************************************************************/
//input global setting and particles information
PD_input_data("input_PDglobal.dat", "input_particle.dat");
particle.pop_back();

/**************************************SUMMARY***************************
display the summary of simulation on console
***********************************************************************/
cout<<"Total nodes:"<<node2.size()+node1.size()<<endl;
cout<<"Total elements:"<<element2.size()+element1.size()<<endl;
cout<<"Total fixed nodes: "<<Fix_nodes2.size()+Fix_nodes1.size()<<endl;
cout<<"Total transition nodes:
"<<Trans_nodes2.size()+Trans_nodes1.size()<<endl;
cout<<"Element type: "<<plane_type<<endl;
cout<<"Element thickness: "<<element_thickness<< "(m)"<<endl;
cout<<endl;
cout<<"Total particles:"<<particle.size()<<endl;
cout<<"Total steps: "<<nstep<<endl;
cout<<"Time step: "<<steptime<<" (s)"<<endl;
cout<<"Output particle steps: "<<nprint<<endl;
cout<<"Invoke FEM calculation steps: "<<Frame_rate<<endl;

/************************Transitionregion*******************************
map the FEM nodes to PD particle & examine if the number of transition
nodes and labeled particles match
***********************************************************************/
//map the particle ID to Transition region 1
vector<int>::iterator iter1=Trans_nodes1.begin();
int counter1 = 0;
for(unsigned int i=0; i<particle.size(); ++i)
    {if(particle[i].type==0)
        {Trans_map1[particle[i].ID]=counter1;
        ++iter1;
        ++counter1;}
    }
if(iter1!=Trans_nodes1.end())
    {cerr<<"Error: the number of trans nodes and type 0 particles doesn't
match"<<endl;
    exit(0);}
//map the particle ID to Transition region 2
vector<int>::iterator iter2=Trans_nodes2.begin();
int counter2 = 0;
for(unsigned int i=0; i<particle.size(); ++i)
    {if(particle[i].type==1)
        {Trans_map2[particle[i].ID]=counter2;
        ++iter2;
        ++counter2;}
    }
if(iter2!=Trans_nodes2.end())
    {cerr<<"Error: the number of trans nodes and type 0 particles doesn't
match"<<endl;
    exit(0);}

/******************************Initialization(PD)***********************
initial velocity, dynamics of particles
```

```
*************************************************************************/
//the type of particles that is fixed or has no acceleration
//fix_type.insert(pair<int, int> (4,4));
//fix_type.insert(pair<int, int> (3,3));

//the type of particles that has initial velocity
map<int, double *> initial_velocity_type;
double va[2] = {-1,0};
initial_velocity_type.insert(pair<int,double *>(1,va));
initial_velocity_type.insert(pair<int,double *>(3,va));

//update the mass of particles, initial velocity and set force to 0
for(unsigned i=0;i<particle.size();i++)
   {//specify the type that has initial velocity
    int type1=particle[i].type;
    DYNAMICS dytemp;
    dytemp.ID=i;
    map<int, double *>::iterator iter;
    iter = initial_velocity_type.find(type1);
    if(iter!=initial_velocity_type.end())
        {double *p = iter->second;
        dytemp.vx=p[0];dytemp.vy=p[1];  }
    else
        {dytemp.vx=0;dytemp.vy=0;   }
    dytemp.w=0;
    dytemp.ax=0;dytemp.ay=0;
    dytemp.wa=0;
    dynamics.push_back(dytemp);
    FORCE fotemp;
    fotemp.x=0;fotemp.y=0;
    force.push_back(fotemp);
    //particle mass
    double r=particle[i].radius;
    double rho=mtype[type1].Rho;
    particle[i].mass=PI*r*r*rho;
    }

double velfem1[2]={0,0};
double velfem2[2]={-1,0};
/*******************************STEP LOOP*********************************
iterating by step, and goes to FEM every "Frame_rate" steps, output
particle result every "nprint" steps
*************************************************************************/
for (int istep=0;istep<=nstep;istep++)
{
    for(map<pair<int, int>, struct contact_index>::iterator iter2 =
contactmap.begin(); iter2!=contactmap.end(); )
        {if(iter2->second.status)
            {iter2->second.status = false;
            ++iter2;}
        else
            {contactmap.erase(iter2++);}
        }

    //contact search and contact force calculate
    for (unsigned int i=0;i<particle.size();i++)
        {for (unsigned int j=i+1;j<particle.size();j++)
```

```
            {double r1, r2;
            double distance;
            r1=particle[i].radius;
            r2=particle[j].radius;
            distance=sqrt((particle[i].x-particle[j].x)*(particle[i].x-
particle[j].x)+\
                          (particle[i].y-particle[j].y)*(particle[i].y-
particle[j].y));      //2D simulation

            //LJ potential force
            if (distance<=2.5&&distance>1)
                {force_LJ(i,j,distance);}

            //Hertzian contact force
            double eps_d =-1e-7;
            if(distance-r1-r2<eps_d)
                {force_Hertzian(i, j, distance, contactmap);}
            }
        }

    //integration of dynamics and update particle information
    integrator(istep);

    //go to FEM calculation and output results
    if(istep % Frame_rate == 0)
        {//add the displacement boundary conditions
        if(istep==0)
            {for(unsigned int i=0;i<trans_nodes_displacement1.size();++i)
                {trans_nodes_displacement1[i].x-=velfem1[0]*steptime;
                 trans_nodes_displacement1[i].y-=velfem1[1]*steptime;}}
        else
            {for(unsigned int i=0;i<trans_nodes_displacement1.size();++i)
                {trans_nodes_displacement1[i].x-
=velfem1[0]*Frame_rate*steptime;
                 trans_nodes_displacement1[i].y-
=velfem1[1]*Frame_rate*steptime;}}

        BC_displacement1.clear();

      BC_displacement1.insert(BC_displacement1.end(),trans_nodes_displacem
      ent1.begin(),trans_nodes_displacement1.end());
        // concatenate the transition nodes and fixed boundary nodes BC

      BC_displacement1.insert(BC_displacement1.end(),fix_bc1.begin(),fix_b
      c1.end());
        BC_forceload1.clear();

      BC_forceload1.insert(BC_forceload1.end(),trans_nodes_force1.begin(),
      trans_nodes_force1.end());

      FEM_calculation(1, concrete1, plane_type, node1, element1,
      BC_displacement1, BC_forceload1, element_thickness, istep, velfem1);
        //clean up the transition nodes displacement to 0 for next loop
        for(unsigned int i=0;i<trans_nodes_displacement1.size();++i)
            {trans_nodes_displacement1[i].x=0;
            trans_nodes_displacement1[i].y=0;}
        if(istep==0)
            {for(unsigned int i=0;i<trans_nodes_displacement2.size();++i)
                {trans_nodes_displacement2[i].x-=velfem2[0]*steptime;
                 trans_nodes_displacement2[i].y-=velfem2[1]*steptime;}}
        else if(istep>0&&istep<=30000)
            {{for(unsigned int i=0;i<trans_nodes_displacement2.size();++i)
            {trans_nodes_displacement2[i].x=0;
             trans_nodes_displacement2[i].y=0;}}
            }
        else
            {for(unsigned int i=0;i<trans_nodes_displacement2.size();++i)
                {trans_nodes_displacement2[i].x-
=velfem2[0]*Frame_rate*steptime;
                 trans_nodes_displacement2[i].y-
=velfem2[1]*Frame_rate*steptime;}}


        BC_displacement2.clear();

      BC_displacement2.insert(BC_displacement2.end(),trans_nodes_displacem
      ent2.begin(),trans_nodes_displacement2.end());
        // concatenate the transition nodes and fixed boundary nodes BC

      BC_displacement2.insert(BC_displacement2.end(),fix_bc2.begin(),fix_b
      c2.end());
        BC_forceload2.clear();

      BC_forceload2.insert(BC_forceload2.end(),trans_nodes_force2.begin(),
      trans_nodes_force2.end());
      FEM_calculation(2, concrete2, plane_type, node2, element2,
      BC_displacement2, BC_forceload2, element_thickness, istep, velfem2);
        //clean up the transition nodes displacement to 0 for next loop
        for(unsigned int i=0;i<trans_nodes_displacement2.size();++i)
            {trans_nodes_displacement2[i].x=0;
            trans_nodes_displacement2[i].y=0;}
        }

    /***************************OUTPUTRESULT**************************/
    if(istep%nprint==0)
        {PD_output_data(istep);      }

    /***************************RESET NEXTSTEP**************************/
    for(unsigned int i=0;i<force.size();++i)
        {   force[i].x=0;
            force[i].y=0;
        }

    for(unsigned int i=0;i<dynamics.size();i++)
        {
        dynamics[i].wa=0;       }

    //display the process percentage on console
    printf("%.2lf%% \r", istep*100.0/nstep);
}
/*********************************ProgramTermination********************
```

```
*************************************************************************/
program_run_time = clock() - program_run_time;
printf("it took %.4lf seconds\n",
(double)program_run_time/CLOCKS_PER_SEC);

//combine the shell commands to organize the output files
// modifications are needed if under windows OS
system("rm -rf output-particle");
system("mkdir output-particle");
system("mv particle*.dat ./output-particle");

system("rm -rf output-fem");
system("mkdir output-fem");
system("mv FEM*.dat ./output-fem");

return (0);
}

/**
 *
 *
 * below are sub functions *
 *
 *
 * */


void force_LJ(int i, int j, double distance)
{   //basic information of particle
    double m1=particle[i].mass;
//mass
    double m2=particle[j].mass;
    double r1[3]={particle[i].x, particle[i].y, 0};      //position vector
    double r2[3]={particle[j].x, particle[j].y, 0};

    //contact forces
    double epsilon=1.0;
    double sigma=1.0;
    double k=24*epsilon/sigma*(2*pow(sigma/distance,14)-
pow(sigma/distance,8));
    double f[2]={k*(r1[0]-r2[0]), k*(r1[1]-r2[1])};

    //update force
    force[i].x+=f[0];
    force[i].y+=f[1];
    force[j].x+=-f[0];
    force[j].y+=-f[1];

    //update dynamics
    dynamics[i].ax+=f[0]/m1;
    dynamics[i].ay+=f[1]/m1;
    dynamics[j].ax+=-f[0]/m2;
    dynamics[j].ay+=-f[1]/m2;

}

void force_Hertzian(int i, int j, double distance, map<pair<int,int>,
contact_index> &contactmap)
{   //basic information of particle,mass, position, velocity, radius
    double m1=particle[i].mass;
//mass
    double m2=particle[j].mass;
    double r1[2]={particle[i].x, particle[i].y};
//position vector
    double r2[2]={particle[j].x, particle[j].y};
    double v1[2]={dynamics[i].vx, dynamics[i].vy};
//velcotiy
    double v2[2]={dynamics[j].vx, dynamics[j].vy};
    double radius1=particle[i].radius;
//radius
    double radius2=particle[j].radius;

    //particle material information, elastic modulus and Poisson's ratio
    double type1=particle[i].type;
//material type
    double type2=particle[j].type;
    double E1=mtype[type1].E;
//Young's modulus
    double E2=mtype[type2].E;
    double Miu1=mtype[type1].Miu;
//Poisson's ratio
    double Miu2=mtype[type2].Miu;

    //particle angular motion information
    double w1=dynamics[i].w;
//angular velocity, it is in z direction
    double w2=dynamics[j].w;

    double I1=0.5*m1*radius1*radius1;
//en.wikipedia.org/wiki/list_of_moments_of_inertia
    double I2=0.5*m2*radius2*radius2;

    //contact kinematics, deformation and relative velocity
    double delta_n=radius1+radius2-distance;
//normal deformation(overlap)
    double n_ij[2]={(r2[0]-r1[0])/distance,(r2[1]-r1[1])/distance};
//unit vector at normal direction
    double v_ij[2]={v1[0]-v2[0], v1[1]-v2[1]};
//relative velocity of i respect to j
    double vn_scale=v_ij[0]*n_ij[0]+v_ij[1]*n_ij[1];
//v_ij*n_ij
    double vn[2]={vn_scale*n_ij[0],vn_scale*n_ij[1]};
//normal component of v_ij
    double vw_1[2]={-radius1*w1*n_ij[1],radius1*w1*n_ij[0]};
//angular velocity at contacting point due to omega
    double vw_2[2]={-radius2*w2*n_ij[1],radius2*w2*n_ij[0]};
    double vt[2]={v_ij[0]-vn[0]+vw_1[0]+vw_2[0],v_ij[1]-
vn[1]+vw_1[1]+vw_2[1]};      //tangential velocity
    double vt_scale = sqrt(vt[0]*vt[0]+vt[1]*vt[1]);
    double t_ij_scale = sqrt(vt[0]*vt[0]+vt[1]*vt[1]);
    double t_ij[2] = {vt[0]/t_ij_scale,vt[1]/t_ij_scale};
    if(t_ij_scale==0)
        {t_ij[0]=0;t_ij[1]=0;}
    //contact force -> normal force
    double r_eff=(radius1*radius2)/(radius1+radius2);
//effective radius
    double E_eff=1/((1-Miu1*Miu1)/E1+(1-Miu2*Miu2)/E2);
//effective Young's modulus
    double m_eff=(m1*m2)/(m1+m2);
//effective mass
    double kn=4/3*E_eff*sqrt(r_eff);
//normal stiffness
    double rn=2*sqrt(m_eff*kn);
//normal damping coefficient
    double FN=-kn*delta_n*sqrt(delta_n)-rn*vn_scale;
    double fn[2]={FN*n_ij[0],FN*n_ij[1]};

    //contact force -> tangential force
    double delta_t=vt_scale*steptime;
//tangential deformation produced by current step;
//>>>>>>>> map contact pair, read and store tangential deformation history
    //create current contact pair
    pair<int, int> contact_pair(i,j);
    //to make sure the i < j
    if(contact_pair.first > contact_pair.second)
        {int middle = contact_pair.first;
        contact_pair.first = contact_pair.second;
        contact_pair.second = middle;}
    //find the current pair if in the old and historical contact mapping
    map<pair<int, int>, struct contact_index>::iterator iter;
    iter = contactmap.find(contact_pair);
    if(iter!=contactmap.end())      //check if new contact pair if found
        {//add the history value to current delta_t
        delta_t += iter->second.delta_t_history;
        //pass new struct to contact pair, true is marked so that it will
exist as false before next loop while false will be deleted
        contact_index s;
        s.status = true;
        s.delta_t_history = delta_t;
        iter->second = s;    }
    else
        {//if current pair doesn't exist in history map, it will be
inserted as new contact pair
        contact_index s;
        s.status = true;
        s.delta_t_history = delta_t;
        contactmap.insert(pair<pair<int, int>, struct contact_index>
(contact_pair, s));    }
//<<<<<<<<<<<<<<
    double G1=0.5*E1/(1+Miu1);
//shear modulus
    double G2=0.5*E2/(1+Miu2);
    double G_eff=1/((2-Miu1)/G1+(2-Miu2)/G2);
//effective shear modulus
    double kt=8*sqrt(r_eff)*G_eff*sqrt(delta_n);
//tangential stiffness
    double rt=2*sqrt(m_eff*kt);
//tangential damping coefficient
    double f_miu=0.5;
//friction coefficient set the maximum tangential force, problem, manually
```

```
setting value
    double FT=-kt*delta_t-rt*vt_scale;
    if (FT<f_miu*FN)
//since both FN and FT are negative, the judge condition is <
        {FT=f_miu*FN;}

    double ft[2]={FT*t_ij[0],FT*t_ij[1]};

    //contact force -> torque
    double T_direction=n_ij[0]*t_ij[1]-n_ij[1]*t_ij[0];
    double T1=T_direction*radius1*FT;
    double T2=T_direction*radius2*FT;

    //update force
    force[i].x+=fn[0]+ft[0];
    force[i].y+=fn[1]+ft[1];
    force[j].x+=-fn[0]-ft[0];
    force[j].y+=-fn[1]-ft[1];

    //update dynamics
//  dynamics[i].ax+=(fn[0]+ft[0])/m1;
//  dynamics[i].ay+=(fn[1]+ft[1])/m1;
//  dynamics[j].ax+=(-fn[0]-ft[0])/m2;
//  dynamics[j].ay+=(-fn[1]-ft[1])/m2;
    dynamics[i].wa+=T1/I1;
    dynamics[j].wa+=T2/I2;


}

void integrator(int step)
{   //update the force and accelerations
    for(unsigned int i=0;i<dynamics.size();i++)
        {//update gravity
        double m=particle[i].mass;
        force[i].x+=G.x*m;      //the index of dynamics and particle are
supposed to be on the same page
        force[i].y+=G.y*m;
        //reudction the forces of transition nodes
        map<int,int>::iterator iter1;
        iter1 = Trans_map1.find(i);      //i is the particle ID
        map<int,int>::iterator iter2;
        iter2 = Trans_map2.find(i);
        dynamics[i].ax=force[i].x/m;
        dynamics[i].ay=force[i].y/m;

        //update fixed particles or those has no acceleration
        map<int,int>::iterator iter;
        iter = fix_type.find(particle[i].type);
        if(iter!=fix_type.end())
            {dynamics[i].ax=0; dynamics[i].ay=0;}
        }

    //update position and velocity of particles
    //Euler Method: v(n+1)=v(n)+a*delta(t); x(n+1)=x(n)+v(n)*delta(t).
    for(unsigned int i=0;i<particle.size();i++)
        {//save displacement of transition nodes
        map<int,int>::iterator iter1;
        iter1 = Trans_map1.find(particle[i].ID);
        if(iter1!=Trans_map1.end())
            {int id = iter1->second;

trans_nodes_displacement1[id].x+=dynamics[i].vx*steptime+0.5*dynamics[i].a
x*steptime*steptime;

trans_nodes_displacement1[id].y+=dynamics[i].vy*steptime+0.5*dynamics[i].a
y*steptime*steptime;
            trans_nodes_force1[id].x = force[i].x;
            trans_nodes_force1[id].y = force[i].y;
            }

        map<int,int>::iterator iter2;
        iter2 = Trans_map2.find(particle[i].ID);
        if(iter2!=Trans_map2.end())
            {int id = iter2->second;

trans_nodes_displacement2[id].x+=dynamics[i].vx*steptime+0.5*dynamics[i].a
x*steptime*steptime; //problem

trans_nodes_displacement2[id].y+=dynamics[i].vy*steptime+0.5*dynamics[i].a
y*steptime*steptime;
            trans_nodes_force2[id].x = force[i].x;
            trans_nodes_force2[id].y = force[i].y;
            }

        //update particle position and dynamics

particle[i].x+=dynamics[i].vx*steptime+0.5*dynamics[i].ax*steptime*steptim
e;

particle[i].y+=dynamics[i].vy*steptime+0.5*dynamics[i].ay*steptime*steptim
e;
        dynamics[i].vx+=dynamics[i].ax*steptime;
        dynamics[i].vy+=dynamics[i].ay*steptime;
        dynamics[i].w+=dynamics[i].wa*steptime;
        }      //angular velocity is in z direction
}

void PD_input_data(const char *fname, const char *fname2)
{
    ifstream file(fname);
    MTYPE mtemp;
    if(!file)
        {cerr<<"error(1):[input_data]couldn't open file: "<<fname<<endl;
        exit(0);}
    while(file.peek()=='#')
        {string keyword;
        file>>keyword;
        if(keyword=="#gravity:")
            {file>>G.x;
            file>>G.y;
            file.ignore(100,'\n');}
        else if(keyword=="#nstep:")
            {file>>nstep;
            file.ignore(100,'\n');}
        else if(keyword=="#steptime:")
            {file>>steptime;
            file.ignore(100,'\n');}
        else if(keyword=="#nprint:")
            {file>>nprint;
            file.ignore(100,'\n');}
        else if(keyword=="#mtype:")
            {file>>mtemp.ID;
            file>>mtemp.Rho;
            file>>mtemp.E;
            file>>mtemp.Miu;
            file.ignore(100,'\n');
            mtype.push_back(mtemp);}
        else
            {cerr<<"error(2):[input_data]unknown keyword of global
setting: "<<keyword<<endl;
            exit(0);}
        }
    file.close();

    ifstream file2(fname2);
    PARTICLE ptemp;
    if(!file2)
        {cerr<<"error(3):[input_data]couldn't open file of particles:
"<<fname2<<endl;
        exit (0);}
    while (!file2.eof())
        {file2>>ptemp.ID;
        file2>>ptemp.type;
        file2>>ptemp.x;
        file2>>ptemp.y;
        file2>>ptemp.radius;
        particle.push_back(ptemp);
        file2.ignore(100,'\n');}
    file2.close();

}

void PD_output_data(int istep)
{     //create a string to store variable filename
      char fname1[30];
      sprintf(fname1,"particle%d.dat",istep);
      // write data into files
      runtime=istep*steptime;
      ofstream o_particle(fname1);
      //write data, particle information
      o_particle<<"#Time: "<<runtime<<endl;
      o_particle<<"#Step: "<<istep<<endl;
      o_particle<<"#ID x y r vx vy w fx fy"<<endl;
      for(unsigned int i=0;i<particle.size();i++)
         {o_particle<<particle[i].ID<<" "<<particle[i].x<<"
"<<particle[i].y<<" "<<particle[i].radius<<" " \
                   <<dynamics[i].vx<<" "<<dynamics[i].vy<<"
"<<dynamics[i].w<<" " \
                   <<force[i].x<<" "<<force[i].y<<endl;}
      o_particle.close();
```

```
}

void FEM_input_node_element(const char *fname, vector<struct NODE> &node,
vector<struct ELEMENT> &element, vector<int> &Trans_nodes, vector<int>
&Fix_nodes)
//Input: text file
//Output: variables of 1)node 2)element 3)Trans_nodes and 4)Fix_nodes
{
    int indicator=0;       //indicator to node or element data //problem,
unnecessary to initialize, but if not, appear warning
    NODE temp1;
    ELEMENT temp2;
    int fixnode;
    int transnode;
    ifstream file(fname);
    if(!file)
        {cerr<<"Error 001: can not open file. \n";
        exit(0);}
    while (!file.eof())
        {while(file.peek()=='#')
            {string type;
            file>>type;
            if(type=="#node")
                indicator=1;
            else if(type=="#element")
                indicator=2;
            else if(type=="#transition_nodes")
                indicator=3;
            else if(type=="#fixed_boundary_nodes")
                indicator=4;
            else
                {cerr<<"Error 002: unknown type of data "<<type<<endl;
                exit(0);}}
        if(indicator==1)
            {file>>temp1.ID;
            file>>temp1.x;
            file>>temp1.y;
            file>>temp1.z;
            file.ignore(100,'\n');
            node.push_back(temp1);}
        else if(indicator==2)
            {file>>temp2.ID;
            file>>temp2.N1;
            file>>temp2.N2;
            file>>temp2.N3;
            file.ignore(100,'\n');
            element.push_back(temp2);}
        else if(indicator==3)
            {file>>transnode;
            Trans_nodes.push_back(transnode);
            file.ignore(100,'\n');}
        else if(indicator==4)
            {file>>fixnode;
            Fix_nodes.push_back(fixnode);
            file.ignore(100,'\n');}
        else
            {cerr<<"Error 003: data should be either node or
```

```
element"<<endl;
            exit(0);}
        }
    file.close();
}

void FEM_output_data(int FEM_ID, vector<struct NODE> &node, vector<struct
ELEMENT> &element, vector<vector<double> > &STRESS, vector<double> &U, int
istep, double *velocity_fem)
{   //output stress and displacement
//    //create a string to store variable filename
//    char fname[30];
//    char fname2[30];
//    sprintf(fname,"FEM%ddisplacement%d.dat",FEM_ID,istep);
//    sprintf(fname2,"FEM%dstress%d.dat",FEM_ID,istep);
//    runtime=istep*steptime;
//
//  ofstream o_displacement(fname);
//  //write data, node displacement
//  o_displacement<<"#Time: "<<runtime<<endl;
//  o_displacement<<"#Step: "<<istep<<endl;
//  o_displacement<<"#ID x y ux uy"<<endl;      //for 2D problem, no z
displacement
//  for(unsigned int i=0;i<node.size();i++)
//      {o_displacement<<node[i].ID<<" "<<node[i].x<<" "<<node[i].y<<"
"<<U[2*i]<<" "<<U[2*i+1]<<endl;}
//  o_displacement<<flush;
//  o_displacement.close();
//
//  //write data, stress informaiton
//  ofstream o_stress(fname2);
//  o_stress<<"#Time: "<<runtime<<endl;
//  o_stress<<"#Step: "<<istep<<endl;
//  o_stress<<"#element_ID N1 N2 N3 sigma_x sigma_y tao_xy"<<endl;
//  for(unsigned int i=0;i<element.size();i++)
//      {o_stress<<element[i].ID<<" "<<element[i].N1<<"
"<<element[i].N2<<" "<<element[i].N3<<" "\
//                  <<STRESS[i][0]<<" "<<STRESS[i][1]<<"
"<<STRESS[i][2]<<endl;}
//  o_stress<<flush;
//  o_stress.close();


    char fname3[30];
    sprintf(fname3,"FEM%dgnuplot%d.dat",FEM_ID,istep);
    ofstream o_gnuplot(fname3);     //file for gnuplot of element lines
    for(unsigned int i=0;i<element.size();i++)
        {o_gnuplot<<node[element[i].N1].x<<"
"<<node[element[i].N1].y<<endl;
        o_gnuplot<<node[element[i].N2].x<<"
"<<node[element[i].N2].y<<endl;
        o_gnuplot<<node[element[i].N3].x<<"
"<<node[element[i].N3].y<<endl;
        o_gnuplot<<node[element[i].N1].x<<"
"<<node[element[i].N1].y<<endl;
        o_gnuplot<<endl;}
    o_gnuplot<<flush;
    o_gnuplot.close();


    char fname5[30];
    sprintf(fname5,"FEM%dmisesstress%d.dat",FEM_ID,istep);
    ofstream o_plot_stress(fname5);
    vector<vector<double> > nodestress(node.size(),vector<double>(2));
    for(unsigned int i=0; i<element.size();++i)
        {//Mises stress
        double stress_plot =
sqrt(STRESS[i][0]*STRESS[i][0]+STRESS[i][1]*STRESS[i][1]-
STRESS[i][0]*STRESS[i][1]+3*STRESS[i][2]*STRESS[i][2]);
        nodestress[element[i].N1][0]+=stress_plot;
        nodestress[element[i].N1][1]+=1;
        nodestress[element[i].N2][0]+=stress_plot;
        nodestress[element[i].N2][1]+=1;
        nodestress[element[i].N3][0]+=stress_plot;
        nodestress[element[i].N3][1]+=1;     }

    for(unsigned int i=0;i<node.size();i++)
        {o_plot_stress<<node[i].x<<" "<<node[i].y<<"
"<<nodestress[i][0]/nodestress[i][1]<<endl;}
    o_plot_stress<<flush;
    o_plot_stress.close();
}

void FEM_calculation(int FEM_ID, struct MATERIAL &material, const char
*plane_type, vector<struct NODE> &node, vector<struct ELEMENT> &element,
        vector<struct BC_DISPLACEMENT> &BC_displacement, vector<struct
BC_FORCELOAD> BC_forceload, double element_thickness, int istep, double
*velocity_fem)
{
    /*****************GLOBAL STIFFNESS MATRIX***************************
      current method is to assemble the completed global stiffness matrix,
      which is low effi-ciency on computation.
    *******************************************************************/
    int NN=2*node.size();       //degree of freedom
    int NE=element.size();       //number of elements
    vector<vector<double> >KK(NN,vector<double>(NN));
    double E0;
    double Miu0;
    double Kcoe1,Kcoe2;
    plane_general(material, plane_type, E0, Miu0, Kcoe1, Kcoe2);
    vector<vector<double> >Be;       //shape function parameters
    for(int i=0;i<NE;i++)
        {
        double Ke[6][6];
        elementstiffness(i,element,node, Kcoe1, Kcoe2, Miu0,
element_thickness, Ke, Be);
        int p1=element[i].N1;
        int p2=element[i].N2;
        int p3=element[i].N3;
        int DOF[6]={2*p1,2*p1+1,2*p2,2*p2+1,2*p3,2*p3+1};
        for(int n1=0;n1<6;n1++)
            {for(int n2=0;n2<6;n2++)
                KK[DOF[n1]][DOF[n2]]+=Ke[n1][n2];}
        }
    /***********************BOUNDARY CONDITIONS*************************
      modify the global stiffness matrix for global load vector &
      displacement constrains method is to multiply a big number on
      diagonal element according to equation 2.2.57 in Reference[1]
     *******************************************************************/
    //load vector, P
    vector<double>P(NN,0);
    if(BC_forceload.size()!=0)
        {for(unsigned int i=0;i<BC_forceload.size();i++)
            {int id=BC_forceload[i].ID;
            P[2*id]+=BC_forceload[i].x;
            P[2*id+1]+=BC_forceload[i].y;   }
        }
    //displacement constrains
    for(unsigned int i=0;i<BC_displacement.size();i++)
        {int id=BC_displacement[i].ID;
              P[2*id]=BC_displacement[i].x*KK[2*id][2*id]*BIGNUMBER;
    //BIGNUMBER is constant
        KK[2*id][2*id]=KK[2*id][2*id]*BIGNUMBER;
        P[2*id+1]=BC_displacement[i].y*KK[2*id+1][2*id+1]*BIGNUMBER;
        KK[2*id+1][2*id+1]=KK[2*id+1][2*id+1]*BIGNUMBER;}

    /****************************SOLVER*******************************
    To solve the linear equations system: K*u=p, K is global stiffness
    matrix, u is the displacement vector, p is the global load vector.
    Successive Over Relaxation method, page 223 in Reference[1]
    ********************************************************************/
    vector<double>U(NN);
    SOR(KK,P,U,1e5);      //result "u" is in "U" vector

    /**********************STRESS and STRAIN ANALYSIS******************
    element stress, principal stress, principal stress direction
    ********************************************************************/
    // elastic constant matrix of material
    vector<vector<double> >D(3, vector<double >(3));
    D[0][0]=E0/(1-Miu0*Miu0);
    D[0][1]=Miu0*E0/(1-Miu0*Miu0);
    D[0][2]=0;
    D[1][0]=Miu0*E0/(1-Miu0*Miu0);
    D[1][1]=E0/(1-Miu0*Miu0);
    D[1][2]=0;
    D[2][0]=0;
    D[2][1]=0;
    D[2][2]=0.5*(1-Miu0)*E0/(1-Miu0*Miu0);
    // equation 2.2.18, stress=D*B*ae
    vector<vector<double> >STRESS(NE,vector<double >(3));
    vector<vector<double> >B(3, vector<double >(6));
    for(int i=0;i<NE;i++)

{B[0][0]=0.5*Be[i][1]/Be[i][0];B[0][2]=0.5*Be[i][2]/Be[i][0];B[0][4]=0.5*B
e[i][3]/Be[i][0];

B[1][1]=0.5*Be[i][4]/Be[i][0];B[1][3]=0.5*Be[i][5]/Be[i][0];B[1][5]=0.5*Be
[i][6]/Be[i][0];

B[2][1]=0.5*Be[i][1]/Be[i][0];B[2][3]=0.5*Be[i][2]/Be[i][0];B[2][5]=0.5*Be
[i][3]/Be[i][0];

B[2][0]=0.5*Be[i][4]/Be[i][0];B[2][2]=0.5*Be[i][5]/Be[i][0];B[2][4]=0.5*Be
[i][6]/Be[i][0];
        vector<vector<double> > C;      //C=D*B
        Matrix_multiply(D,B,C);
        int p1=element[i].N1;      //number of 1, 2, 3 node of element
        int p2=element[i].N2;
        int p3=element[i].N3;
        vector<vector<double> >ae;
        vector<double>ui(1); vector<double>vi(1);
        vector<double>uj(1); vector<double>vj(1);
        vector<double>um(1); vector<double>vm(1);
        ui[0]=U[2*p1];vi[0]=U[2*p1+1];
        uj[0]=U[2*p2];vj[0]=U[2*p2+1];
        um[0]=U[2*p3];vm[0]=U[2*p3+1];
        ae.push_back(ui); ae.push_back(vi);
        ae.push_back(uj); ae.push_back(vj);
        ae.push_back(um); ae.push_back(vm);
        vector<vector<double> > stress;
        Matrix_multiply(C,ae,stress);      //stress=C*ae=D*B*ae
        for(int n=0;n<3;n++)
            STRESS[i][n]=stress[n][0];
        }
    /**********************UPDATE NODES POSITION*************************
    add the displacement to the nodes, so it will generate the new
    position of NODES
    *********************************************************************/
    for(unsigned int i=0;i<node.size();i++)
        {node[i].x=node[i].x+U[2*i]+velocity_fem[0]*Frame_rate*steptime;

    node[i].y=node[i].y+U[2*i+1]+velocity_fem[1]*Frame_rate*steptime;}

    /*************************OUTPUT RESULT******************************
    node displacement, stresses
    *********************************************************************/
    FEM_output_data(FEM_ID, node, element, STRESS, U, istep,velocity_fem);

}

void plane_general(struct MATERIAL &material, const char *plane_type,
double &E0, double &Miu0, double &Kcoe1, double &Kcoe2)
//Input: 1)material; 2)global setting of plane type, plane stress or plane
strain
//Output: E0, Miu0, Kcoe1 and Kcoe2. equation 2.2.21, 2.2.22, 2.2.34 and
2.2.35 in Reference[1]
{   if(strcmp(plane_type,"planestress")==0)
        {E0=material.E;
        Miu0=material.miu;  }
    else if(strcmp(plane_type,"planestrain")==0)
        {E0=material.E/(1-material.miu*material.miu);
        Miu0=material.miu/(1-material.miu);}
    else
        {cerr<<"Error 004: please specify the type of plane problem,
stress or strain "<<endl;
        exit(0);    }
    Kcoe1=E0/(4*(1-Miu0*Miu0));
    Kcoe2=0.5*(1-Miu0);
}

void elementstiffness(int ID, vector<struct ELEMENT> &element,
vector<struct NODE> &node, double Kcoe1, double Kcoe2, double Miu0, double
element_thickness, double (*Ke)[6], vector<vector<double> > &Be)
//Input: element ID, begin with 0, assuming ID is the same as index
//Output: Ke[6][6], equations 2.2.33~2.2.35 are on page 44 of Reference[1]
//Notes: triangular element, first-order polynomial shape function.
{   int p1=element[ID].N1;      //number of 1, 2, 3 node of element
    int p2=element[ID].N2;
    int p3=element[ID].N3;
    double x1=node[p1].x
    double y1=node[p1].y;
    double x2=node[p2].x;
    double y2=node[p2].y;
    double x3=node[p3].x;
    double y3=node[p3].y;
    double A=(x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2))/2;   //area of element
    if(A<0)
        {cerr<<"Error 005: the value of element area is negative. Please
ensure nodes numbered counter-clockwise. "<<endl;
        exit(0);}
    double b1=y2-y3;      //the coefficient of shape function
    double b2=y3-y1;
    double b3=y1-y2;
    double c1=x3-x2;
    double c2=x1-x3;
    double c3=x2-x1;
    //store strain matrix into Be
    vector<double> temp(7);
    temp[0]=A; temp[1]=b1; temp[2]=b2; temp[3]=b3; temp[4]=c1; temp[5]=c2;
temp[6]=c3;
    Be.push_back(temp);

    double B[4]={0,b1,b2,b3};      //the "0" is useless
    double C[4]={0,c1,c2,c3};
    double cons1=Kcoe1/A*element_thickness;
    double cons2=Kcoe2;
    for(int i=1;i<=3;i++)
        {for(int j=1;j<=3;j++)
            {Ke[2*i-2][2*j-2]=cons1*(B[i]*B[j]+cons2*C[i]*C[j]);
            Ke[2*i-2][2*j-1]=cons1*(Miu0*C[i]*B[j]+cons2*B[i]*C[j]);
            Ke[2*i-1][2*j-2]=cons1*(Miu0*B[i]*C[j]+cons2*C[i]*B[j]);
            Ke[2*i-1][2*j-1]=cons1*(C[i]*C[j]+cons2*B[i]*B[j]);}
        }
}

void SOR(vector<vector<double> > &A,vector<double> &b,vector<double>
&c,int n)
//Input: A 2-D matrix, b 1-D vector, n maximum iterations
//Output: the solution is stored in vector "b"
//Notes: Successive Over Relaxation method for solution to A*X=b; equation
6.9.10
//        omega=1.2;
{   //declaration of variables
    int order=A.size();
    vector<double>x(order,0);
    double e0,ek;
    //process of iterations
    for(int k=1;;k++)
        {for(int i=0;i<order;i++)
            {double sum1=0;
            for(int j=0;j<i;j++)
                {sum1+=A[i][j]*x[j];}
            double sum2=0;
            for(int j=i;j<order;j++)
                {sum2+=A[i][j]*x[j];}
            x[i]=x[i]+1.2*(b[i]-sum1-sum2)/A[i][i];       //omega=1.2
            //examine error, ek is the maximum error at kth iteration
            if(i==0)
                e0=fabs((b[i]-sum1-sum2)/A[i][i]);
            ek=fabs((b[i]-sum1-sum2)/A[i][i]);
            if(ek>=e0)
                e0=ek;
            }
        //judge if finish the computation by reaching acceptable error or
exceed maximum iterations.
        if(e0<=1e-9)       //1e-9 is the minimum error allowed.
            break;
        else
            {if(k>n)       //n is the maximum iterations allowed.
                {cerr<<"Error 006: exceed the maximum iterations of
solving equations."<<endl;
                break;}
            }
        }
    //assign value to "c" vector
    for(int i=0;i<order;i++)
        {c[i]=x[i];}
}

void Matrix_scalar(vector<vector<double> > &A, double k)
//Input: 2D matrix A, coefficient k
//Output: A=k*A
{   for(unsigned int i=0;i<A.size();i++)
        {for(unsigned int j=0;j<A[0].size();j++)
            A[i][j]=A[i][j]*k;
        }
}

void Matrix_multiply(vector<vector<double> > &A,vector<vector<double> >
&B, vector<vector<double> > &C)
//matrix multiply: C=A*B
//Notes: It needs to declare C in advance.
//       It includes the error examination of the matrix size.
//       Even A or B is 1D matrix, it should be declared in the form as
the function parameters.
{   unsigned int ia=A.size();
    unsigned int ja=A[0].size();
    unsigned int ib=B.size();
    unsigned int jb=B[0].size();
    if(ja!=ib)
        {cerr<<"Error: size mismatch of two matrices.";
        exit(0);}
    vector<double> temp(jb);
    for(unsigned int i1=0;i1<ia;i1++)
        {for(unsigned int j2=0;j2<jb;j2++)
            {double sum=0;
            for(unsigned int k=0;k<ja;k++)
                sum+=A[i1][k]*B[k][j2];
            temp[j2]=sum;}
        C.push_back(temp);
        }
}
```

Preprocessing cpp:

```
/* Statement:
Application: to create intput data for hybrid method model of FME and PD
Outcome:
1)a file contains nodes and element data as following format,
#node
ID x y
.
.
.
#element
ID node1 node2 node3
.
.
.
#transition_nodes
node1
node2
node3
.
.
#fixed_boundary_nodes
node1
node2
node3
.
.
(NO counts from 0; node1, node2, node3 form count-clockwise combination)
2)a file contains particle inforamtion as following format,
ID type x y r
.
.
.
(ID counts from 0, type = 0 is reserved as transition area particles which
are nodes in FEM as well)
3) a map to store the transition area like map<int i, int j>, i is the
NODES id in FEM, j is the PARTICLE ID in PD
4) a vector store the nodes that used as fixed boundary nodes
Notes:
1) generate triangular elements to cover rectangle region
2) generate rectangle or line shape particles assembling
3) the top edge of FME rectangle is set as transition area
4) the bottom edge of FEM rectangle is set as fixed boundary
5) transition aeaa nodes is set as type 0 particles in PD, and it always
```

```
is the first part of PD particles information
6) the diameter of particles as the nodes in FEM is set as the length
between two neighboring nodes
7) in PD_rectangle function, particle_ID and vector particle are not used
as virtual parameter, they are directly operated in global scope
8) in function FEM mesh, map Transition_area and vector bc_fix_nodes is
handled as above statement
*/
#include "preprocessing_model.h"
#include <cmath>
#include <iomanip>

//containers of particle and transition nodes
struct PARTICLE_pre {int ID; int type; double x, y; double radius;};
vector<struct PARTICLE_pre> particle_pre;

//function
void FEM_MESH_2D(int FEM_ID, double *region, int degree_x, vector<struct
PARTICLE_pre> &par, int type);
void PD_rectangle(vector<struct PARTICLE_pre> &par, double p_region[],
double d, int type);


int preprocessing_model()
{
    /**********For FEM***********/
    // region contains [xmin, xmax, y1, y2]
    double region[4] = {-0.22, 0.1, -0.05, 0.05};
    FEM_MESH_2D(1, region, 32, particle_pre,0);

    // region contains [xmin, xmax, y1, y2]
    double region2[4] = {0.13, 0.45, 0.265,0.165 };
    FEM_MESH_2D(2, region2, 32, particle_pre,1);

    /***********For  PD***********/
    double p_region[4];
    p_region[0] = -0.22; p_region[1] = 0.1;
    p_region[2]=0.06; p_region[3]=0.11;
    PD_rectangle(particle_pre, p_region, 0.01, 2);
    p_region[0] = 0.13; p_region[1] = 0.45;
    p_region[2]=0.105; p_region[3]=0.155;
    PD_rectangle(particle_pre, p_region, 0.01, 3);

    /***********write data to file, particle information***********/
    ofstream o_particle("input_particle.dat");
    //ID type x y r
    for(unsigned int i=0;i<particle_pre.size();i++)
        {o_particle<<particle_pre[i].ID<<" "<<particle_pre[i].type<<"
"<<particle_pre[i].x<<" "<<particle_pre[i].y<<"
"<<particle_pre[i].radius<<endl;}
    o_particle<<flush;
    o_particle.close();

    //write the global setting file, for PD only
    ofstream out_fileg("input_PDglobal.dat");
    out_fileg<<"#gravity: 0 0"<<endl;
    out_fileg<<"#steptime: 0.000001"<<endl;
    out_fileg<<"#nstep: 450000"<<endl;
    out_fileg<<"#nprint: 1000"<<endl;
    out_fileg<<"#mtype: 0 3000 2000000000 0.333"<<endl;
    out_fileg<<"#mtype: 1 2000 500000000 0.333"<<endl;
    out_fileg<<"#mtype: 2 3000 2000000000 0.333"<<endl;
    out_fileg<<"#mtype: 3 2000 500000000 0.333"<<endl;
    out_fileg<<flush;
    out_fileg.close();

    return (0);
}

void FEM_MESH_2D(int FEM_ID, double *region, int degree_x, vector<struct
PARTICLE_pre> &par, int type)
//argument: FEM_ID, the number of FEM block, region is the rectangle area,
degree_x is the number of divisions of x coordinates, par is the particle
container, type is the type assigned to interface nodes
//input: region = {xmin, xmax, y1, y2}, y2 is the refined edge; degree_x,

{
int particle_ID = par.size()+1-1;

//initialization -> region
double xmin = region[0];
double xmax = region[1];
double y1 = region[2];
double y2 = region[3];

//initialization -> degree
int nx = degree_x;
double dx = (xmax-xmin)/nx;
int ny;
if(y2>y1)
    {ny = int((y2-y1)/(4*dx))+1;
    if((y2-y1)>(ny*4*dx+2*dx))
        {ny+=1;     }
    }
else
    {ny = int((y1-y2)/(4*dx))+1;
    if((y1-y2)>(ny*4*dx+2*dx))
        {ny+=1;     }
    }

//error inspection
if(xmax<=xmin)
    {cerr<<"error[FEM_mesh_2D]: the FEM region should be set as [xmin,
xmax, y1, y2]"<<endl;}
if(nx%8!=0)
    {cerr<<"error, incorrect setting of degree x"<<endl;}

//x and y coordinates
double *x = new double[nx+1];
for (int i=0; i<=nx; ++i)
    {x[i]=xmin+i*dx;    }
double *y = new double[ny+1];
for (int i=0; i<=ny; ++i)
    {if(i==0)
```

```
        {y[i]=y2;     }
    else if(i==1)
        {if(y2>y1)
            y[i]=y2-2*dx;
        else
            y[i]=y2+2*dx;
            }
    else
        {if(i==ny)
            {y[i]=y1;}
         else
            {if(y2>y1)
                y[i]=y2-2*dx-4*dx*(i-1);
            else
                y[i]=y2+2*dx+4*dx*(i-1);
            }
        }
    }

//output nodes information: ID x y, and ID counts from 0, and find
transition area and boundary nodes
char fname[30];
sprintf(fname,"input_fem%d.dat",FEM_ID);
//output nodes information: ID x y, and find transition area and boundary
nodes
ofstream out_file(fname);
out_file<<"#node"<<endl;
int IDnode=0;
vector<int> bc_fix_nodes;
vector<int> trans_nodes;
for(int i=0; i<=ny; ++i)
    {for(int j=0; j<=nx;)
        {out_file<<IDnode<<" "<<x[j]<<" "<<y[i]<<" "<<0<<endl;
        if(i==0)
            {PARTICLE_pre temp;
            temp.ID = particle_ID;
            temp.type = type;
            temp.x = x[j];
            temp.y = y[i];
            temp.radius = dx/2;
            particle_pre.push_back(temp);
            particle_ID+=1;
            trans_nodes.push_back(IDnode);  }
        if(i==ny)
            {bc_fix_nodes.push_back(IDnode);}
        if(i<=2)
            {j+=pow(2,i);   }
        else
            {j+=8;          }
        IDnode+=1;
        }
    }

//create the element
int IDelement=0;
out_file<<"#element"<<endl;
for(int i=0; i<=ny-1; ++i)
```

```
     {
     int elenum;       //number of elements at y = i
     if(i<=2)
         elenum = nx/(pow(2,i+1));
     else
         elenum = nx/8;
     for(int j=1; j<=elenum; ++j)
         {
         if(y2>y1)
             {int p1, p2, p3, p4, p5;
             if(i<=2)
                 {p1 = 2*(1-pow(0.5,i))*nx + i + (j-1)*2;
                 p2 = p1 + 1;
                 p3 = p2 + 1;
                 p4 = 2*(1-pow(0.5,i+1))*nx + (i+1) + (j-1);
                 p5 = p4 + 1;

                 out_file<<IDelement<<" "<<p1<<" "<<p4<<" "<<p2<<endl;
                 IDelement+=1;

                 out_file<<IDelement<<" "<<p4<<" "<<p5<<" "<<p2<<endl;
                 IDelement+=1;

                 out_file<<IDelement<<" "<<p5<<" "<<p3<<" "<<p2<<endl;
                 IDelement+=1;             }
             else
                 {p1 = int((7/4 + 1/8*(j-3)))*nx + i + j-1;
                 p2 = p1 + 1;
                 p3 = int((7/4 + 1/8*(j-2)))*nx + i + j-1;
                 p4 = p3 + 1;

                 out_file<<IDelement<<" "<<p1<<" "<<p3<<" "<<p4<<endl;
                 IDelement+=1;

                 out_file<<IDelement<<" "<<p4<<" "<<p2<<" "<<p1<<endl;
                 IDelement+=1;
                 }}
         else
             {int p1, p2, p3, p4, p5;
             if(i<=2)
                 {p1 = 2*(1-pow(0.5,i))*nx + i + (j-1)*2;
                 p2 = p1 + 1;
                 p3 = p2 + 1;
                 p4 = 2*(1-pow(0.5,i+1))*nx + (i+1) + (j-1);
                 p5 = p4 + 1;

                 out_file<<IDelement<<" "<<p1<<" "<<p2<<" "<<p4<<endl;
                 IDelement+=1;

                 out_file<<IDelement<<" "<<p4<<" "<<p2<<" "<<p5<<endl;
                 IDelement+=1;

                 out_file<<IDelement<<" "<<p5<<" "<<p2<<" "<<p3<<endl;
                 IDelement+=1;             }
             else
                 {p1 = int((7/4 + 1/8*(j-3)))*nx + i + j-1;
                 p2 = p1 + 1;
```

```
                p3 = int((7/4 + 1/8*(j-2)))*nx + i + j-1;
                p4 = p3 + 1;

                out_file<<IDelement<<" "<<p1<<" "<<p4<<" "<<p3<<endl;
                IDelement+=1;

                out_file<<IDelement<<" "<<p4<<" "<<p1<<" "<<p2<<endl;
                IDelement+=1;
                }
            }
        }
    }

//output transition and fixed boundary nodes
out_file<<"#transition_nodes"<<endl;
for(unsigned int i=0;i<trans_nodes.size();++i)
    {out_file<<trans_nodes[i]<<endl;    }
out_file<<"#fixed_boundary_nodes"<<endl;
for(unsigned int i=0;i<bc_fix_nodes.size();++i)
    {out_file<<bc_fix_nodes[i]<<endl;    }

//delete memory of array x and y
delete [] x;
delete [] y;

//flush the buffer and close output file
out_file<<flush;
out_file.close();

}

void PD_rectangle(vector<struct PARTICLE_pre> &par, double p_region[],
double d, int type)
{
int particle_ID = par.size()+1-1;

//initialization -> region
double xmin = p_region[0];
double xmax = p_region[1];
double ymin = p_region[2];
double ymax = p_region[3];

double p0[2]={xmin,ymin};
for(double y=p0[1];y<=ymax+d/2;y+=d)
    {for(double x=p0[0];x<=xmax+d/2;x+=d)
        {PARTICLE_pre temp;
        temp.ID=particle_ID;
        temp.type=type;
        temp.x = x;
        temp.y = y;
        temp.radius = d/2;
        par.push_back(temp);
        particle_ID+=1;}
    }

}
```

Header file 1: FEM

```
#define PI 3.14159
#define BIGNUMBER 1e10
//the number used to modify global stiffness matrix to introduce boundary
conditions page 53, equation 2.2.57, "有限单元法基本原理和数值方法" （by 王勖成）
//variables in iteration
struct NODE {int ID;double x,y,z;};
struct ELEMENT {int ID;int N1,N2,N3;};
struct MATERIAL {double E;double miu;};
struct BC_DISPLACEMENT {int ID;double x;double y;};
struct BC_FORCELOAD {int ID;double x;double y;};
//functions--IO
void FEM_input_node_element(const char *fname, vector<struct NODE> &node,
vector<struct ELEMENT> &element, vector<int> &Trans_nodes, vector<int>
&Fix_nodes);
void FEM_output_data(int FEM_ID, vector<struct NODE> &node, vector<struct
ELEMENT> &element, vector<vector<double> > &STRESS, vector<double> &U, int
istep, double *velocity_fem);
//functions--modulue
void elementstiffness(int ID,vector<struct ELEMENT> &element,
vector<struct NODE> &node, double Kcoe1, double Kcoe2, double Miu0, double
element_thickness,double (*Ke)[6],vector<vector<double> > &Be);
void plane_general(struct MATERIAL &material, const char *plane_type,
double &E0, double &Miu0, double &Kcoe1, double &Kcoe2);
void SOR(vector<vector<double> > &A,vector<double> &b,vector<double> &c,
int n);
//functions--matrix operation
void Matrix_scalar(vector<vector<double> > &A, double k);
void Matrix_multiply(vector<vector<double> > &A,vector<vector<double> >
&B,vector<vector<double> > &C);
//functions--main
void FEM_calculation(int FEM_ID, struct MATERIAL &material, const char
*plane_type, vector<struct NODE> &node, \
        vector<struct ELEMENT> &element, vector<struct BC_DISPLACEMENT>
&BC_displacement, vector<struct BC_FORCELOAD> BC_forceload, double
element_thickness, int istep, double *velocity_fem);
//global variables: nodes, elements, transition and fixed boundary nodes
vector<struct NODE> node1;
vector<struct ELEMENT> element1;
vector<int> Trans_nodes1;
vector<int> Fix_nodes1;
//boundary condition: add BC on fixed nodes and transtion nodes
vector<struct BC_DISPLACEMENT> fix_bc1;
vector<struct BC_DISPLACEMENT> trans_nodes_displacement1;
vector<struct BC_FORCELOAD> trans_nodes_force1;
vector<struct BC_DISPLACEMENT> BC_displacement1;
vector<struct BC_FORCELOAD> BC_forceload1;
//map the particle to trans BC
map<int, int> Trans_map1;
//global variables: nodes, elements, transition and fixed boundary nodes
vector<struct NODE> node2;
vector<struct ELEMENT> element2;
vector<int> Trans_nodes2;
vector<int> Fix_nodes2;
//boundary condition: add BC on fixed nodes and transtion nodes
vector<struct BC_DISPLACEMENT> fix_bc2;
```

```
vector<struct BC_DISPLACEMENT> trans_nodes_displacement2;
vector<struct BC_FORCELOAD> trans_nodes_force2;
vector<struct BC_DISPLACEMENT> BC_displacement2;
vector<struct BC_FORCELOAD> BC_forceload2;
//map the particle to trans BC
map<int, int> Trans_map2;
```

Header file 2: PD

```
#ifndef _HybridM_h
#define _HybridM_h

#include <fstream>
#include <vector>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <map>
#include <utility>
using namespace std;

//constructors
struct PARTICLE {int ID; int type;double x,y; double radius; double
mass;};
struct MTYPE {int ID; double Rho; double E, Miu;}; //material: density, E
and Poisson ration
struct DYNAMICS {int ID;double vx, vy;double w;double ax, ay;double wa;};
struct FORCE {int ID; double x, y;}; //total force acting on particle
struct contact_index {bool status; double delta_t_history;};

//simulation environment setting variables
double runtime;
double steptime;
int nstep, nprint;
struct gravity {double x, y;} G;

//containers
vector<struct PARTICLE> particle;
vector<struct MTYPE> mtype;
vector<struct DYNAMICS> dynamics;
vector<struct FORCE> force;
map<pair<int,int>, struct contact_index > contactmap;

//functions
void integrator(int step);
void PD_input_data(const char *fname, const char *fname2);
void PD_output_data(int istep);
void force_Hertzian(int i, int j, double distance, map<pair<int,int>,
contact_index> &contactmap);
void force_LJ(int i, int j, double distance);

#endif
```

Header file 3: preprocessing

```
#ifndef _Pre_h
#define _Pre_h

#include <iostream>
#include <fstream>
#include <cstdio>
#include <vector>
#include <map>
#include <utility>
//#include <array>
using namespace std;

//preprocessing functions
int preprocessing_model();

#endif
```