 CMES, vol.55, no.1, pp.61-87, 2010

An Object-Oriented MPM Framework for Simulation of Large Deformation and Contact of Numerous Grains

Z. T. Ma[1], X. Zhang[1,2] and P. Huang[1]

Abstract: The Material Point Method (MPM) is more expensive in terms of storage than other methods, as MPM makes use of both mesh and particle data. Therefore, it is critical to develop an efficient MPM framework for engineering applications, such as impact and explosive simulations. This paper presents a new architecture for MPM computer code, developed using object-oriented design, which enables MPM analysis of a mass of grains, large deformation, high strain rates and complex material behavior. It is flexible, extendible, and easily modified for a variety of MPM analysis procedures. An MPM scheme combining contact algorithm with USF, USL and MUSL formulation is presented, and an improved contact detection scheme is proposed to avoid contact occurring earlier than actual time, and several schemes are developed to reduce the memory requirement and computational cost, including the local multi-mesh contact algorithm, dynamic internal state variables for materials, dynamic grid and moving grid technique. Finally, some numerical examples are presented to demonstrate the computational efficiency and memory requirement of the framework.

Keywords: Material point method, Multi-mesh, Contact, Impact, Framework.

1 Introduction

As one of the innovative spatial discretization methods, the Material Point Method (MPM) developed by Sulsky et al. [Sulsky, Chen and Schreyer (1994); Sulsky, Zhou and Schreyer (1995)] is an extension of FLIP [Brackbill and Rupple (1986); Brackbill, Kothe and Rupple (1988)] particle-in-cell method in computational fluid dynamics to the computational solid mechanics. The essential idea is to take advantage of both the Eulerian and Lagrangian methods. It discretizes a material domain by using a collection of material points. The momentum equations are solved on a predefined regular background grid, so that the grid distortion and entanglement

[1] School of Aerospace, Tsinghua University, Beijing 100084, China.
[2] Email: xzhang@tsinghua.edu.cn (corresponding author).

are completely avoided. In the past decade, the MPM has evolved into a robust spatial discretization method capable of handling many challenging engineering problems. It has been applied to large strain problems [Wiechowski (2004); Coetzee, Vermeer and Basson (2005)], calculations with dynamical energy release rate [Tan and Nairn (2002)], fracture mechanics [Joris, Rene and Alan Needleman (2005)], dynamics failure [Chen, Hu, Shen, Xin and Brannon (2002); Chen, Feng, Xin and Shen (2003)], hyper-velocity impact [Zhang, Sze and Ma (2006)] and explosion [Ma, Zhang, Lian and Zhou (2009)], thin membranes [York, Sulsky and Schreyer (1999)], granular materials [Bardenhagen, Brackbill and Sulsky (2000); Cummins and Brackbill (2002); Bardenhagen and Brackbill (1998); Bardenhagen and Brackbill (2000); Bardenhagen and Guilkey (2001)], porous media [Zhang, Wang and Chen (2009)], just to name a few.

Contact phenomena are widely observed in engineering fields. Because a single-valued velocity field is used for updating the positions of material points, the no-slip (or sticking) contact between two different bodies can be handled automatically at no additional cost using the original MPM, and the contact surface need not to be detected. Furthermore, Bardenhagen et al. [Bardenhagen and Brackbill (2000); Bardenhagen and Guilkey (2001)] extended the original MPM to the friction (or slip) contact between deformable solid bodies, which allows Coulomb friction and slip at contact nodes. The contact force between bodies is obtained from the relative nodal velocity at the contact surface. To release the no-slip contact algorithm in MPM, a global multi-mesh mapping scheme was proposed by Hu and Chen [Hu and Chen (2003)]. In the multi-mesh mapping scheme, each material lies in an individual background mesh rather than in the common one. But the algorithm occupies vast memory if there are a lot of granules. Pan et al. [Pan, Xu, Zhang, Zhang, Ma and Zhang (2008)] proposed a three-dimensional global multi-mesh contact algorithm based on [Hu and Chen (2003)], in which the contact force between of bodies is obtained from the normal nodal acceleration continuity requirement at the contact surface.

There are several alternatives for which nodal velocities to use for the updating stress. In 1994, Sulsky et al. [Sulsky, Chen and Schreyer (1994)] presented an approach, referred to as the "Update Stress Last" or USL, in which the nodal velocities is calculated after updating the nodal momenta. The method has serious stability difficulties. Then, two solutions were proposed to solve it. An approach, referred to as the "Modified Update Stress Last" or MUSL was given by Sulsky et al. [Sulsky, Zhou and Schreyer (1995)], in which the particle momenta are extrapolated to the grid twice before updating the nodal velocities. Another fix is to update strains before updating momentum [Bardenhagen (2002)], referred to as "Update Stress First" or USF. In many simulations, both of the above give greatly improved

stability compared to USL. These algorithms were implemented together in one algorithm by Nairn [Nairn (2003)] for crack calculations.

There are also some other improvement and research on MPM. Artificial noise is introduced when the material points move across the cell, due to the discontinuity of the gradient of the interpolation function at the borders of the neighboring cells. Bardenhagen and Kober [Bardenhagen and Kober (2004)] developed the Generalized Interpolation Material Point (GIMP) methods to solve the problem. Then, Ma et al. [Ma, Lu and Komanduri (2006)] presented a refinement scheme for a structured mesh in GIMP. Steffen et al. [Steffen, Wallstedt, Guilkey, Kirby and Berzins (2008)] analyze and demonstrate how the smoothing length within GIMP can affect the error and stability properties, and how various choices of basic functions and boundary treatments affect the spatial convergence properties of MPM. A silent boundary scheme is proposed within the framework of the material point method by Shen and Chen [Shen and Chen (2005)]. An enhancement to the projection operation is developed by Wallstedt and Guilkey [Wallstedt and Guilkey (2007)], which makes use of already available velocity gradient information. The enhancement facilitates exact projection of linear functions and reduces the dependence of projection accuracy on particle location and density for non-linear functions. The thermo-mechanical model is implemented within the framework of MPM by Chen et al. [Chen, Gan and Chen (2008)], so that the different gradient and divergence operators in the governing differential equations could be discretized in a single computational domain and that continuous remeshing is not required with the evolution of failure.

More and more strength models, equations of state and failure models are incorporated into engineering softwares in order to solve multifarious problems. That resulted in many internal state variables of materials need to be saved and relations between materials are more complex than before. Different algorithms and material models need to be incorporated in a single program. However, as the complexity of MPM programs increase, it is obvious that improving the flexibility, expansibility, maintainability and reusability of software is important. Procedural codes contain many complex data structures, which are accessed throughout the program. This global access decreases the flexibility of the system. It is difficult to modify the existing codes and to extend the codes to adapt them for new users, models and solution procedures. The inflexibility is demonstrated in several ways: (1) a high degree of knowledge of the entire program is required to work on even a minor portion of the code; (2) reuse of code is difficult; (3) a small change in the data structures can have a ripple effect throughout the system; (4) the numerous interdependencies between the components of the design are hidden and difficult to determine; (5) the integrity of the data structures is not assured.

Due to these facts, the necessity for a suitable MPM computational environment is evident. The object-oriented programming (OOP) technique has been shown to significantly improve the extendibility and reusability of software. A carefully designed framework or architecture can significantly reduce the effort required for maintaining and extending the software. In the past decades, many structural engineering researchers have pursued objected-oriented design. In 1990, Fenves [Fenves (1990)] described the advantages of OOP for the development of engineering software. One of the first detailed applications of the object-oriented paradigm to finite element method (FEM) was published in 1990 by Forde et al. [Forde, Foschi and Stiemer (1990)]. The authors abstracted out the essential components of the FEM (elements, nodes, materials, boundary conditions, and loads) into a class structure used by most subsequent authors. Since then, complete objected-oriented FEM architectures is presented [Miller and Rucki (1993); Duboispelerin and Zimmermann (1993); Baugh and Rehak (1992); Lu, White, Chen and Dunsmore (1995); Chudoba, Bittnar and Krysl (1995); Archer, Fenves and Thewalt (1999); Yu and Kumar (2001)]. Recently, Luo et al. [Luo, Cai and Zhang (2000)] described an objected-oriented meshless Galerkin method. Atluri et al. [Atluri and Shen (2002); Atluri, Liu and Han (2006)] proposed a general framework for developing the Meshless Local Petrov-Galerkin (MLPG) approach, which provides flexibility in choosing the local weak forms, the trial functions, and the independent test functions for solving systems of partial differential equations. Zhang and Subbarayan [Zhang and Subbarayan (2006)] proposed a design-analysis integrated CAD framework termed jNURBS, developed using the Java language. It enabled meshless analysis of physical behavior and optimal design.

Efficiency of C++ as an object-oriented language is higher than that of languages based on virtual machine or common language runtime (such as Java and C#) and equate to that of traditional languages (such as FORTRAN and C). This paper presented a new object-oriented MPM computational framework, named MPM3Dpp, designed for large deformation, high strain rates and complex material behavior, developed using C++ language. MPM3Dpp evolved from MPM3D [Peng, Zhang, Ma and Wang (2008)] supporting OpenMP parallelization, developed using FORTRAN, and is redesigned with object-oriented approach. Considering that implementation efficiency and memory consumed are important for computational program, we have taken advantage of object-oriented mechanisms and memory allocation of C++ to optimize them. Besides, some examples were implemented to demonstrate high efficiency and low memory required of the new framework.

2 MPM formulation

2.1 Initial Discretization

The MPM procedure begins by discretizing the material domain with a set of material points or particles. A weak formulation [Sulsky, Chen and Schreyer (1994); Sulsky, Zhou and Schreyer (1995)] of the MPM algorithm of solid mechanics is given and the method is framed in terms of the finite element method. The momentum equation

$$\rho \frac{d\mathbf{v}}{dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{b} \tag{1}$$

is solved in a Lagrangian frame on a background grid. In Eq.(1), $\mathbf{b}$ is the body force, ρ is the mass density, $\mathbf{v}$ is the velocity and $\boldsymbol{\sigma}$ is the stress tensor.

In MPM, the continuum bodies are discretized with N_p material particles. Each material particle carries the information of position, velocity, mass, density, stress, strain and all other internal state variables necessary for the constitutive model. At each time step, the mass and velocities of the material particles are mapped onto the background computational mesh. The mapped nodal mass m_i and momentum $\mathbf{p}_i$ are obtained through the following equations respectively,

$$m_i = \sum_{p=1}^{n_p} m_p N_{ip} \tag{2}$$

$$\mathbf{p}_i = \sum_{p=1}^{n_p} m_p \mathbf{v}_p N_{ip} \tag{3}$$

where, m_p is the particle mass, $\mathbf{v}_p$ the velocity of the particle, and N_{ip} is the value of shape function associated with node i evaluated at particle p.

2.2 Time integration with contact algorithm

Constraints on the grain motion are necessary only when grains are approaching. If the momenta of two bodies are projected on to the same node, the contact may occur and the contact between bodies r and s is detected by comparing the nodal velocities $\mathbf{v}_i^r$ and $\mathbf{v}_i^s$ [Pan, Xu, Zhang, Zhang, Ma and Zhang (2008)],

$$(\mathbf{v}_i^r - \mathbf{v}_i^s) \cdot \mathbf{n}_i^r > 0 \tag{4}$$

where $\mathbf{n}_i^r$ is the unit outward normal of body r at node i along the boundary.

Bardenhagen and Guilkey [Bardenhagen and Guilkey (2001)] proposed a method to include the normal traction t_n^α, which is computed at a contact grid node by

interpolating individual material point contributions using the surface normals (and the relation $t_n^\alpha = \mathbf{n}^\alpha \cdot \boldsymbol{\sigma}^\alpha \cdot \mathbf{n}^\alpha$), in the contact logic to more appropriately determine the free separation criterion.

Both contact detection methods of the above will make contact occur earlier than the actual time (if the space between two bodies is less than two times cell size, contact may take place). An improved contact detection method is proposed here to avoid the earlier contact (see Fig. 1). Assume two bodies may contact at node i and they are approaching, then distance between them D_i^{rs} can be calculated as

$$D_i^{rs} = D_i^r + D_i^s \tag{5}$$

$$D_i^g = -\max(\mathbf{X}_{ip}^g \cdot \mathbf{n}_i^g + l_p^g)$$

$$\mathbf{X}_{ip}^g = \mathbf{X}_p^g - \mathbf{X}_i$$

where $\mathbf{X}_i$ is the position of node i, l_p^g and $\mathbf{X}_p^g$ are the length and position of some particle of body g at the current time step, respectively. If D_i^{rs} is less than specified distance D, it means that the contact occurs.

As the normal momentum must be conserved, nodal momenta of contacting bodies need to be updated [Pan, Xu, Zhang, Zhang, Ma and Zhang (2008)],

$$\bar{\mathbf{p}}_i^r = \mathbf{p}_i^r - \frac{m_i^s \mathbf{p}_i^r - m_i^r \mathbf{p}_i^s}{m_i^r + m_i^s} \cdot \mathbf{n}_i^r \mathbf{n}_i^r \tag{6}$$

$$\bar{\mathbf{p}}_i^s = \mathbf{p}_i^s + \frac{m_i^s \mathbf{p}_i^r - m_i^r \mathbf{p}_i^s}{m_i^r + m_i^s} \cdot \mathbf{n}_i^r \mathbf{n}_i^r \tag{7}$$

Grid accelerations can be calculated as

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{f}_i = \mathbf{f}_i^{\text{int}} + \mathbf{f}_i^{\text{ext}} \tag{8}$$

where the internal and external forces at grid nodes can then be calculated as

$$\mathbf{f}_i^{\text{int}} = -\sum_p^{N_p} \frac{m_p}{\rho_p} \boldsymbol{\sigma}_p \cdot \nabla N_{ip} \tag{9}$$

$$\mathbf{f}_i^{\text{ext}} = \sum_p^{N_p} N_{ip} \mathbf{b}_p \tag{10}$$

Once bodies r and s contact, their acceleration along the normal is equal, so that their nodal force must be updated [Pan, Xu, Zhang, Zhang, Ma and Zhang (2008)],

$$\bar{\mathbf{f}}_i^r = \mathbf{f}_i^r - f_i^{\text{nor}} \mathbf{n}_i^r - f_i^{\text{fric}} \mathbf{s}_i^r \tag{11}$$

$$\bar{\mathbf{f}}_i^s = \mathbf{f}_i^s + f_i^{\text{nor}}\mathbf{n}_i^r + f_i^{\text{fric}}\mathbf{s}_i^r \tag{12}$$

$$f_i^{\text{nor}} = \frac{m_i^s \mathbf{f}_i^{r,\text{int}} - m_i^r \mathbf{f}_i^{s,\text{int}}}{m_i^r + m_i^s} \cdot \mathbf{n}_i^r \tag{13}$$

$$f_i^{\text{fric}} = \min\left(\mu f_i^{\text{nor}},\, f_i^{\tan}\right) \tag{14}$$

$$f_i^{\tan} = \frac{\left(m_i^s \mathbf{p}_i^r - m_i^r \mathbf{p}_i^s + \left(m_i^s \mathbf{f}_i^{r,\text{int}} - m_i^r \mathbf{f}_i^{s,\text{int}}\right)\Delta t\right) \cdot \mathbf{s}_i^r}{\left(m_i^r + m_i^s\right)\Delta t} \tag{15}$$

where $\mathbf{s}_i^r$ is the unit tangential at node i along the boundary, μ is the coefficient of friction, Δt is the time step.

An explicit time integrator is used to solve Eq. (7) for the nodal accelerations, with the time step satisfying the stability condition. The critical time step is the ratio of the smallest cell size to the sum of wave speed and particle velocity.

Update particle position and velocity [Sulsky, Zhou and Schreyer (1995)], respectively,

$$\bar{\mathbf{x}}_p = \mathbf{x}_p + \sum_{i=1}^{n_i} \frac{\mathbf{p}_i}{m_i} N_{ip} \Delta t \tag{16}$$

$$\bar{\mathbf{v}}_p = \mathbf{v}_p + \sum_{i=1}^{n_i} \frac{\mathbf{f}_i}{m_i} N_{ip} \Delta t \tag{17}$$

2.3 The calculations of stress and strain

Strain and vorticity increment of a particle are obtained through the following equations respectively,

$$\Delta\varepsilon_{p\alpha\beta} = \frac{1}{2}\sum_{i=1}^{n_i}\left(G_{ip\beta} v_{i\alpha} + G_{ip\alpha} v_{i\beta}\right)\Delta t \tag{18}$$

$$\Delta\Omega_{p\alpha\beta} = \frac{1}{2}\sum_{i=1}^{n_i}\left(G_{ip\beta} v_{i\alpha} - G_{ip\alpha} v_{i\beta}\right)\Delta t \tag{19}$$

where G_{ip} is defined as

$$\mathbf{G}_{ip} = \nabla N_{ip} \tag{20}$$

Input the strain and vorticity increment into a material constitutive law and update the particle stresses. Any constitutive law may be used.

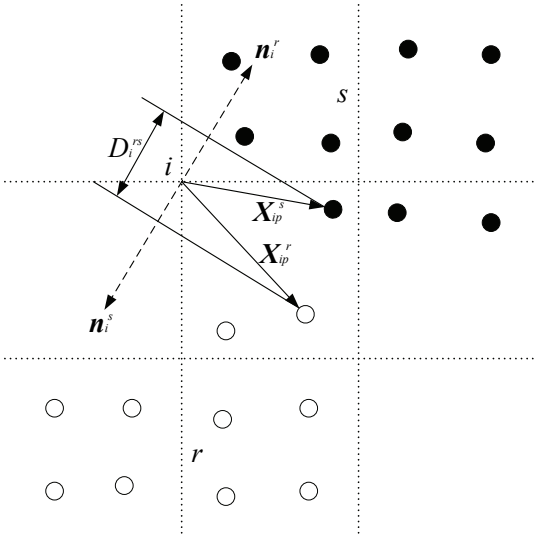

Figure 1: Improved contact detection scheme

2.4 Flow chart of MPM3Dpp

MPM3Dpp flow is drawn in Fig. 2 to include contact algorithm for three operations for updating particle stresses and strains denoted as USF, MUSL, USL.

3 Overall architecture

The key concept for the design of the system architecture is the separation of tasks into distinct classes of objects. A general overview of main classes of the object design is given in Fig. 3. UML (Unified modeling language) static model are used to show the architecture. Class names are shown within boxes, and the reference associations between the classes are drawn as solid line with solid filled diamond connecting the boxes.

The top-level class CMPM3DPP is developed to control the overall computational process and creates the class CSolution, CWriteResult and CDomain to assist itself in completing the task. This class parses of an input file at the beginning of the computational process, and then starts solving. CSolution is an expert of the entire analysis procedure. It performs an analysis on a domain by obtaining the properties of the CDomain object and outputs the result via CWriteResult object. Solution time is recorded by the Clock class. The CDomain class is in charge of creation and initialization of background grid (CGrid class), materials (CMatGeneric class), bodies (CBody class). The other classes will be demonstrated as follows.

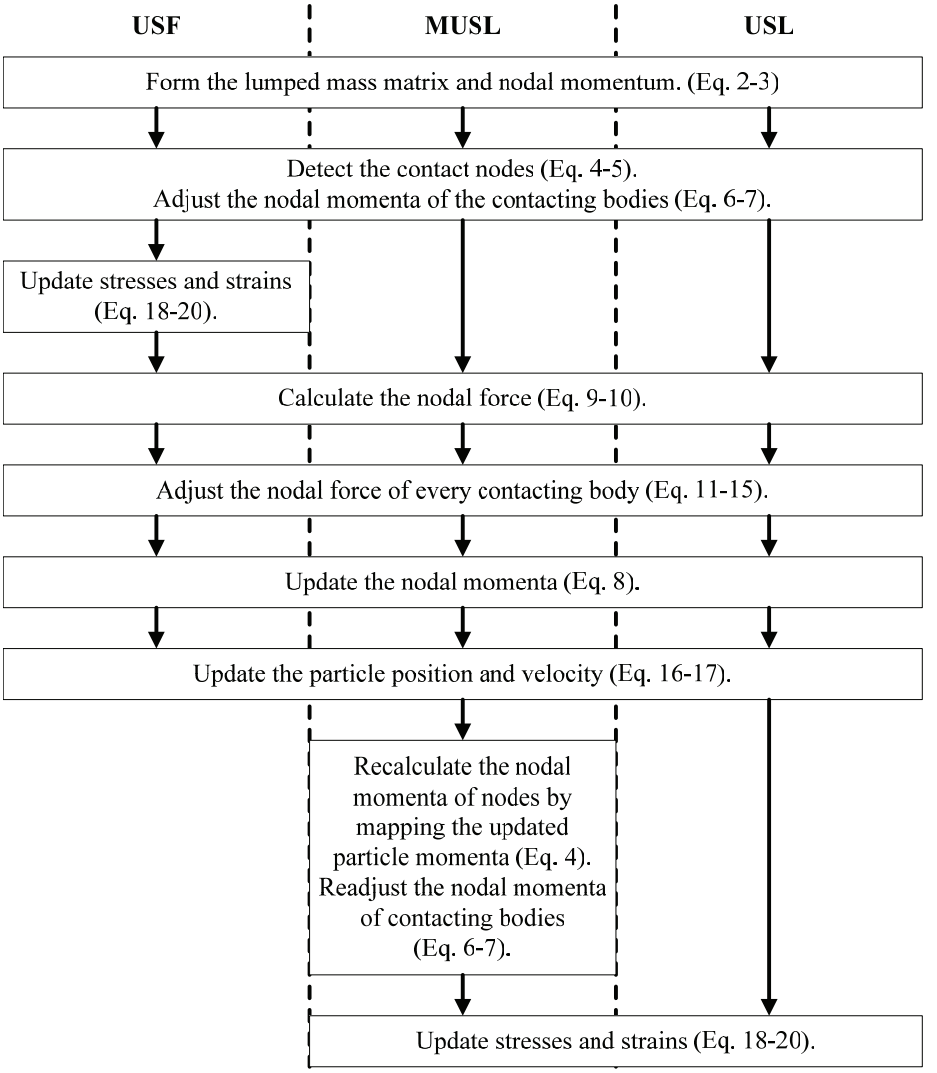

Figure 2: Sketch of MPM3Dpp algorithm

3.1 Particle structure

MPM particle is described by CParticle class (created by the CBody class, see Fig. 3), which contains coordinate, velocity, volume, artificial bulk viscosity, mean stress, failure, internal energy, mass, sound speed and extra properties. For saving the memory to the largest extent, only needed internal state variables of materials

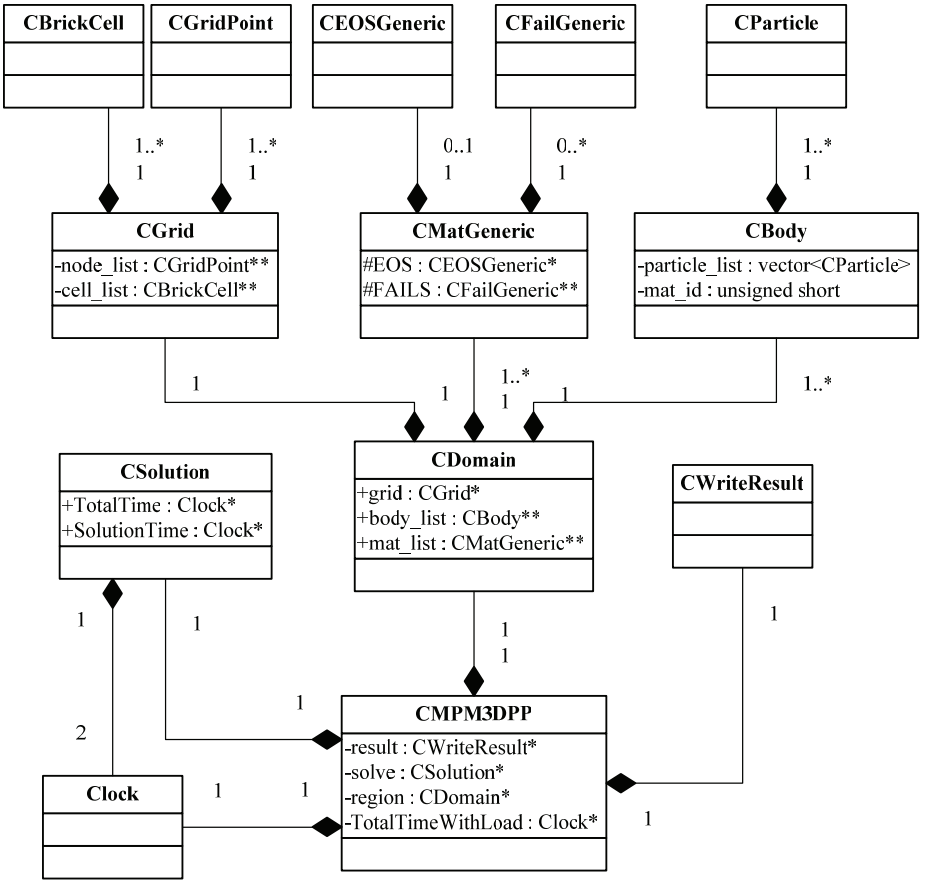

Figure 3: General structure of MPM3Dpp code.

and variables related to different schemes are stored in a dynamic allocated array named ExtraProp (extra properties). New variables can be appended to ExtraProp in CBody, CDomain and all of material classes.

A namespace named MPM is used to define program version, algorithm types, particle properties, global settings, etc. Optional types of extra properties ExtraParticleProperty are defined as ENUM data type in the MPM namespace. Subscript operator overloading is implemented for convenient and fast storage and retrieval of ExtraProp, namely,

```
inline double & operator[] (MPM::ExtraParticleProperty seq) {
       return ExtraProp[ExtraParticlePropPos[seq]
       }
```

where ExtraParticlePropPos is a pointer pointing to an array storing positions of extra properties defined in CBody, seq is some extra property type.

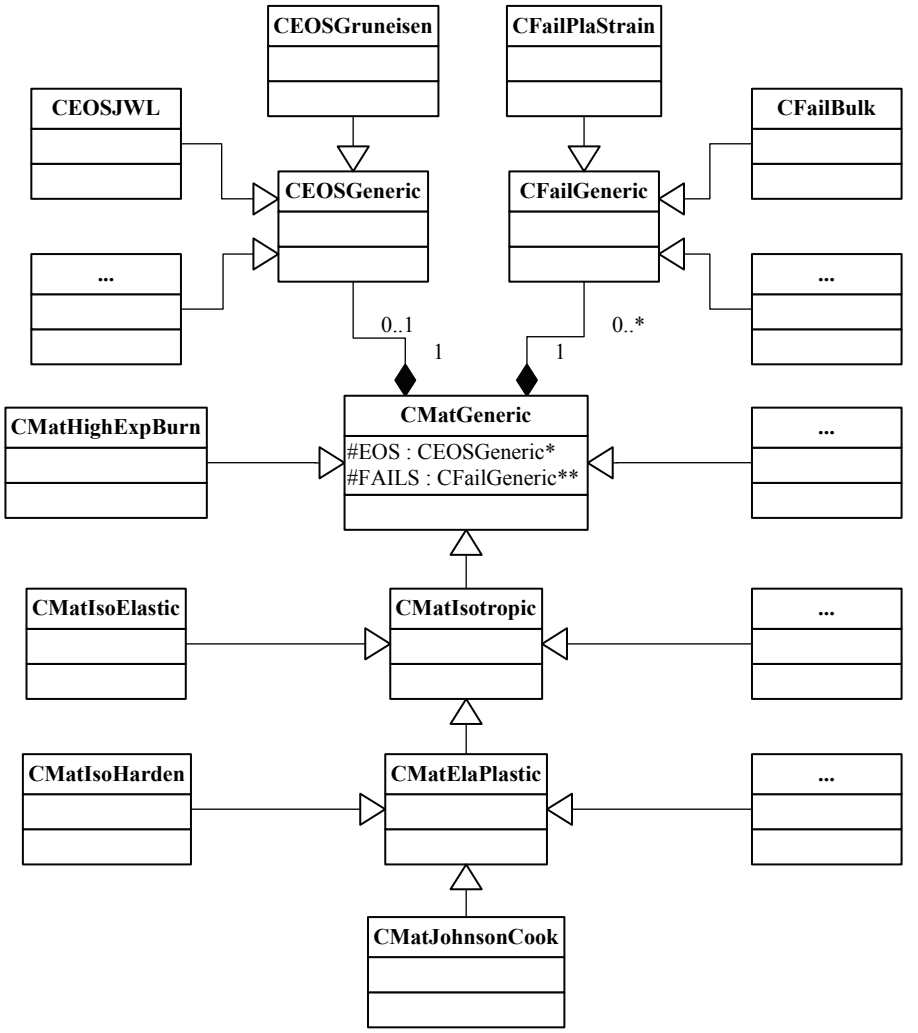

Figure 4: The reference and inheritance associations of material classes.

3.2 Material structure

The reference and inheritance associations between the material classes are shown in Fig. 4. The class CMatGeneric (strength model), CEOSGeneric (equation of

state, EOS) and CFailGeneric (failure model) are abstract classes, requiring their implementation by the specialized derived classes, which compose material definitions. The derived classes are allowed to overload these general implementations to reflect the specific needs. A material can have one EOS at most and more than one failure model. A variable named material_id is designated as serial number of materials in each CBody object. Listing 1 is an outline of the partial definition of the CMatGeneric class. Internal state variables of some materials will be appended to Extra properties of particles via ApplyParticleProperties function.

3.3 Local multi-mesh structure

In MPM the grid servers as a scratch pad for the solution of conservation of momentum, from which particle states are updated. A contact algorithm was presented by Bardenhagen et al. to simulate the interactions of the grains of granular material [Bardenhagen, Brackbill and Sulsky (2000)]. To release the contact algorithm in MPM, a global multi-mesh mapping scheme was proposed by Hu and Chen [Hu and Chen (2003)]. In the multi-mesh mapping scheme, each grain lies in an individual background mesh rather than in the common one, as demonstrated in Fig. 5. For a two-dimensional case, these individual meshes are comprised of the same types of cells as the common background mesh, which are uniformly distributed in the computational domain. Much memory is needed and wasted in the method, especially for cases with many grains. So, local multi-mesh is presented here, i.e. multi-mesh is only created at overlapped nodes, as illustrated in Fig. 6.

Listing 1: Partial definition of the CMatGeneric class.

```
protected:

CEOSGeneric *EOS;

CFailGeneric ** FAILS;

void ApplyParticleProperties(CBody *body);

public:

virtual void UpdateStress(double(&de)[6],
        double (&vort)[6], CParticle
        *p, double &vold, double &dt);

virtual double SoundSpeed(double rho, CParticle *p);
```

For both contact and non-contact cases, connectivities between nodes are defined in CBrickCell class (Fig. 3, cell_list); CGridNodeForContact and CGridNodeForNoContact, inheriting CGridPoint (contains coordinate variable, boundary con-

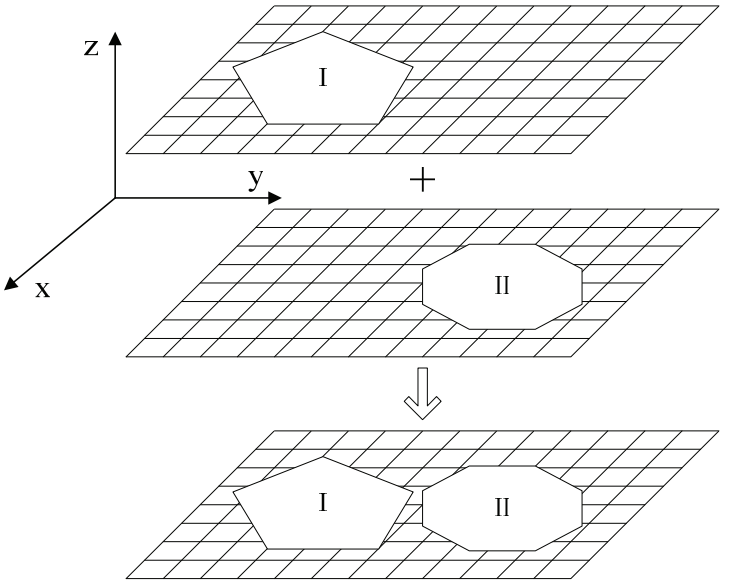

Figure 5: The global multi-mesh mapping scheme [Hu and Chen (2003)].

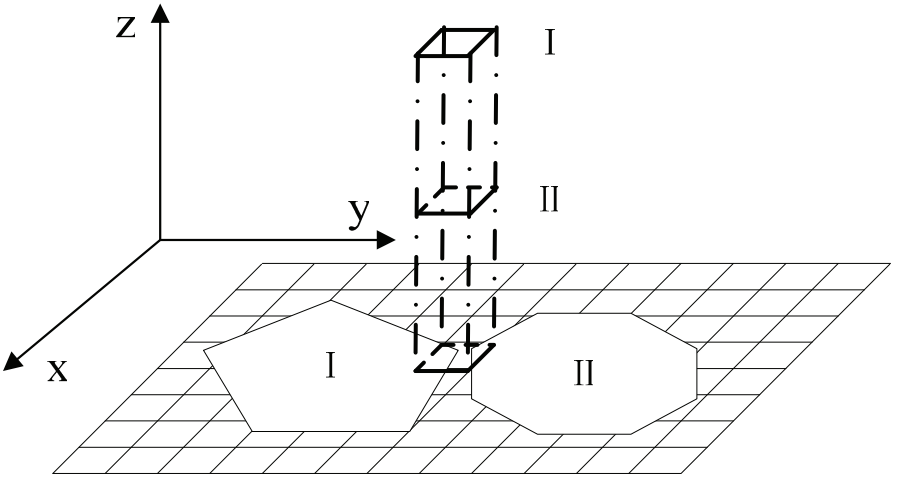

Figure 6: The local multi-mesh mapping scheme.

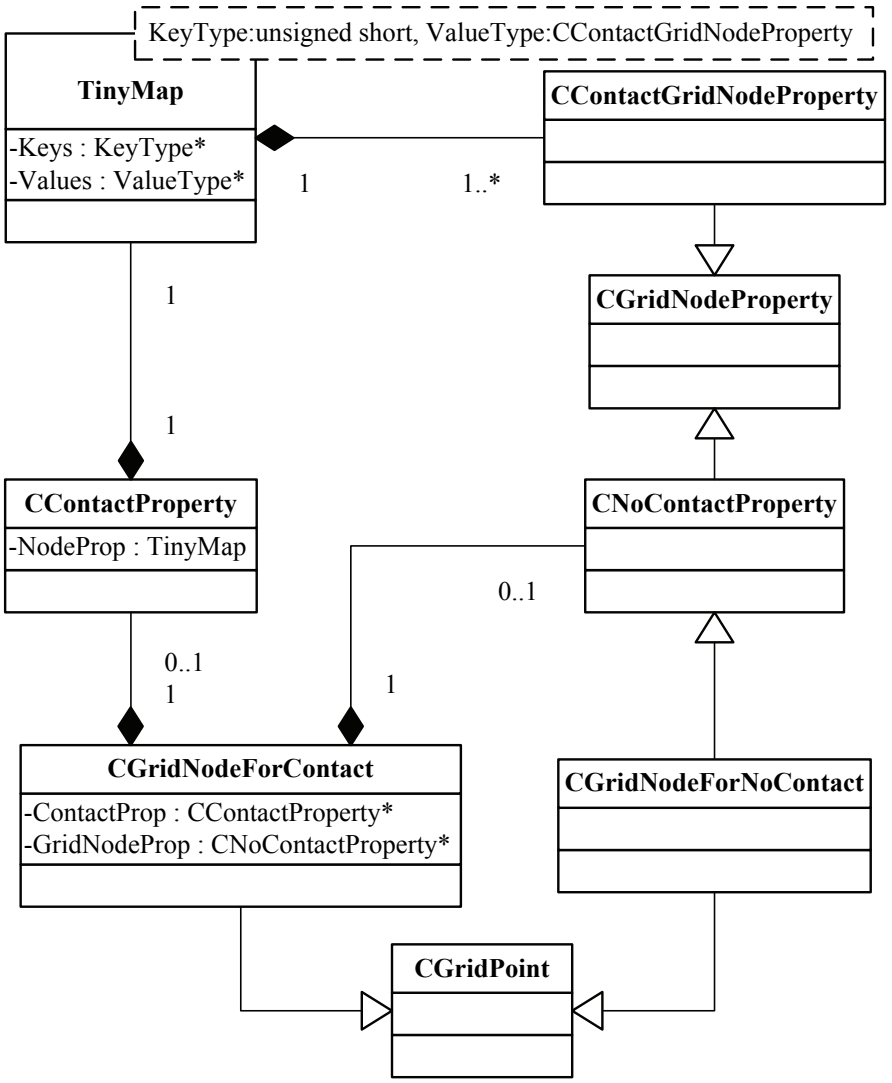

Figure 7: The reference and inheritance associations of GridNode classes.

dition, abstract functions and basic functions called by derived classes) abstract class, are created according to contact status of the node, respectively. The derived classes are allowed to overload general implementations to reflect the specific needs. This allows using the derived class instances on a very abstract level, hiding the particular implementation details, by using only the general services declared by the base classes. Listing 2 is an outline of the partial definition of the basic

grid node class (CGridPoint). The reference and inheritance associations between the classes are drawn as solid line with a solid filled diamond and a hollow arrow connecting the boxes in Fig. 7, respectively.

For non-contact case, the CGridNodeForNoContact class inherits both CGridPoint and CNoContactProperty (contains mass, momentum, internal/external force, etc.) classes.

For contact case, the CGridNodeForContact class inherit CGridPoint class and contains instances of one of CNoContactProperty and CContactProperty (contains normal vector, whether contact or not, number of grains, properties of every grain, etc.) classes. The state of contact or non-contact of a node can switch each other according to number of grains in a node. If there is only one grain in a node, it will be stored in a CNoContactProperty instance, otherwise, they will be stored in TinyMap, which is a container template class like map class in C++ STL. TinyMap is used for the fast storage and retrieval of a little of data from a collection in which the each element is a pair that has both a data value and a sort key. It has higher efficiency and needs less memory than the map class. Though there are a lot of grains in simulations, MPM3Dpp can solve them effectively and costs tiny memory with the method.

3.4 Dynamic grid

Usually, particles will hold part of computation region, so that the residual region will be useless and waste much memory. In order to solve the problem, a dynamic grid scheme is developed. At the beginning of a new computation, pointer arrays of nodes and cells will be created and initialized to NULL, while memory of nodes won't be allocated immediately, until there are particles around a node or in a cell. At the beginning of every time step, instances of nodes and cells, around or in which there are no particles, can be destructed according to specified step interval, otherwise they will be reseted. If a node isn't instantiated, it won't participate in computation. The method will decrease memory allocated and improve efficiency.

Listing 2: Partial definition of the CGridPoint class.

```
protected:

double Xg[3];

void ApplyBoundaryConditions(CGridNode *gd,
                             unsigned char fixed);

void InitiateMomentum(CGridNode *gd);

void IntegrateMomentum(CGridNode *gd,
```

```
                    unsigned char fixed, double DTx);
public:
virtual bool Clear (bool check=false);
virtual CGridNode * GetGridNodeByCom (unsigned short comid);
virtual void ApplyBoundaryConditions (unsigned char fixed);
virtual void InitiateMomentum ();
virtual void IntegrateMomentum(unsigned char fixed, double DTx);
```

3.5 Moving grid

With the moving of particles, original grid may not cover the computational region, so that some particles will be omitted when computing. To solve the problem, a moving grid is presented here. When the situation occurs, the grid will be moved and enlarged. The step of the method is listed as follows,

1. Loop over all particles, if there are particle approach boundaries (except for fixed boundary and symmetrical boundary), record directions (i.e. x, y or z) and position of grid need be changed.
2. Enlarge grid along the recorded directions.
3. Adjust cell size according to the new grid size.

3.6 Optimized computational flow

MPM3Dpp computational flow is optimized to improve efficiency in this section, including several alternatives for which nodal velocities to use for the updated process. The optimized MPM with the local multi-mesh presented in this paper as follows,

1. Initialize pointers of all grid nodes and cells to NULL.
2. Initialize the properties of existent grid nodes or destruct instances of unused nodes and cells at every specified step interval (default value is 10 in MPM3Dpp).
3. If there are particles in a cell, create it and its nodes. Form the lumped mass matrix and nodal momentum; record nodes and cells around or in which there are particles for contact case.

4. This task is only for contact case. Loop over the mesh nodes, if a node exists and contains more than one grain (i.e. size of TinyMap is greater than 1), create and initialize normal vector arrays for the grains and record the serial number of the node to an array named ContactNodeList. Loop over the mesh cells, if there is more than one grain on a cell, mark it in an array named CellMayContact (i.e. contact may occur on it). Loop over the particles in cells of potential contact to compute normal vectors of grains. Loop over the mesh nodes of potential contact, if bodies contact at a node, adjust the nodal momenta of the contacting bodies.
5. This task is only for USF method. Calculate the rate of the deformation gradient for each particle, compute the increment of strain using an appropriate strain measure and solve constitutive equations to update the stress.
6. Calculate the nodal internal force of existent nodes.
7. This task is only for contact case. Adjust the nodal force of every contacting body.
8. Update the nodal momenta of existent nodes.
9. Mapping kinematics variables to particles to update their position and velocity.
10. This task is only for MUSL method. Extrapolate the new particle velocities to the grid to get a revised set of nodal momenta. Readjust the nodal momenta for contact case.
11. This task is for MUSL and USL method. Update the stresses of particles based on the updated grid nodal velocity at time $t^{n-1/2}$ (i.e. step v).
12. Define a new mesh, if necessary, and return to step ii to begin a new time step.

4 Example problems

In the section, we apply MPM3Dpp to a variety of problems to demonstrate the performance of the framework. The increased flexibility of the object-oriented software architecture impacts the execution speed and memory required of the program. To investigate the extent of the effect, the program was compared to MPM3D in section 4.1. As MPM3D can deal with only two grains, the other example just performed by MPM3Dpp.

All simulations are performed on a computer with Intel CPU Q6600 2.4GHz and 4G memory installed, running under Microsoft windows XP operating system.

4.1 Penetration

In order to compare the implementation efficiency and memory consumed, 3-dimension analyses have been performed on penetration of an ogive-nose 4340 steel projectile with a striking velocity 575m/s against a 6061-T651 aluminum target with an obliquity of 30^o (see Fig. 8) [Piekutowski, Forrestal, Poormon and Warren (1996)]. The projectile is modeled as an elastic material. And the Mie-Gruneisen equation of state and the John-Cook strength model were applied for target in this study. The image after impact is shown in Fig. 9. And the residual velocity (441m/s) is agreed well with experimental data (455m/s).

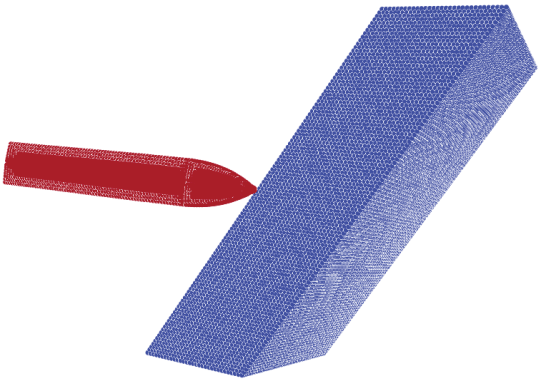

Figure 8: MPM models for simulation of oblique penetration.

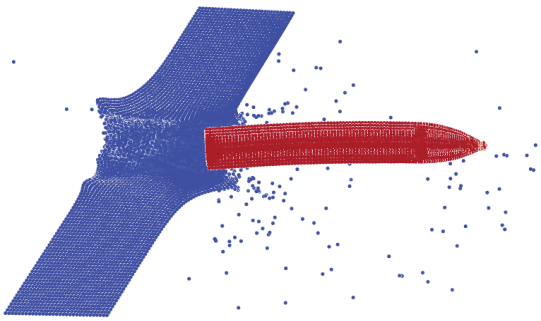

Figure 9: Configuration after penetration

In order to make results comparable, the use of the same time step and solution algorithm (USF) has been explicitly enforced, and moving grid isn't enabled. Fig. 10

shows time history of percentage of used cells during the simulation. And maximal percentage of used cells in the example is 13.13%. The results have been compared with those obtained by MPM3D to demonstrate the optimization associated with object oriented nature of the code (see Table 1). The average time of every step and maximal memory consumed are reported as well the ratio between MPM3D and MPM3Dpp. The dramatic memory reduction and performance improvement, compared to MPM3D, is caused by optimized flow and dynamically allocated memory of mesh and particles, respectively.

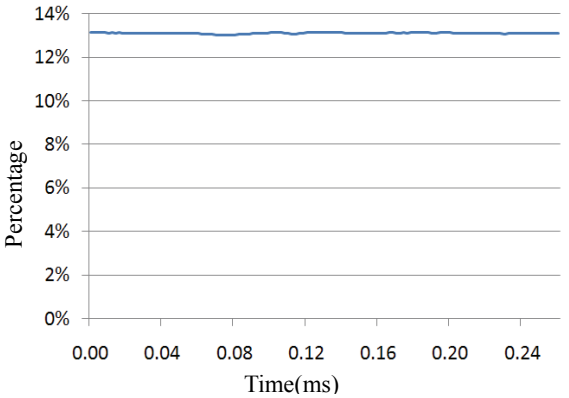

Figure 10: Time history of percentage of used cells.

Table 1: Computational performance and memory consumed in penetration simulation

Contact	Particles/ Cells	Average time (s)			Maximal memory(M)		
		MPM3D	MPM3Dpp	Ratio	MPM3D	MPM3Dpp	Ratio
Disable	200864/ 170688	0.5455	0.4064	1.342	73.18	54.25	1.349
Enable		0.8146	0.4313	1.888	109.36	55.02	1.988

4.2 Spheres under pressure

In order to demonstrate the capability of simulating contact of numerous grains, 1000 aluminum spheres ($10\times10\times10$, distribution of spheres in 3-dimension) are held in a container covered with a slab under distributed load (see Fig. 11). There are 810 contact surfaces in this example, and solution algorithm is MUSL.

The results have been compared with non-contact case to further demonstrate the high efficiency and low memory requirement of the code (see Table 2). The average

time of every step and maximal memory consumed are reported between contact case and non-contact case. Additional CPU time and memory caused by contact are not much, though numerous grains contact. Fig. 12 shows stress wave propagation, reflection and dissipation through spheres simulated with contact algorithm.

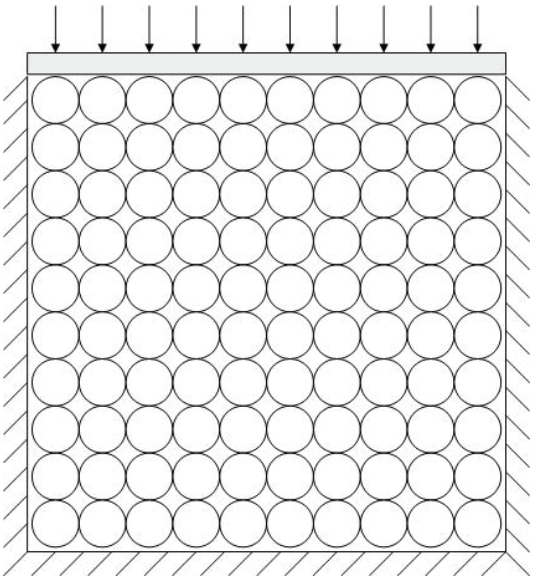

Figure 11: Planar sketch of spheres under pressure

Table 2: Computational performance and memory consumed in the simulation of spheres under pressure

Contact	Particles	Cells	Average time(s)	Maximal memory(M)
Disable	652000	137500	1.4263	288.60
Enable			2.2187	305.19

4.3 Shakes of grains

In order to prove the ability to simulate long term loading of MPM3Dpp, 3 big grains and 207 small grains, under periodic force and gravity, are held in a steady container. The example is 2-dimension, solution algorithm is MUSL, and contact algorithm is enabled. Each small and big grain is modeled by 205 and 1821 particles, while the container by 4832 particles. Near 1,700,000 steps were used in

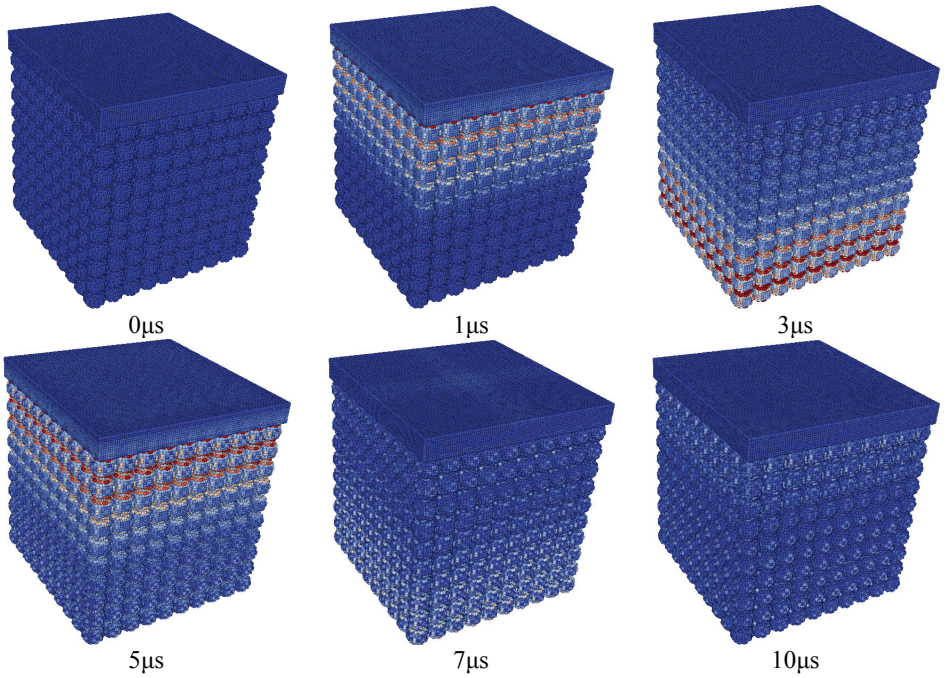

Figure 12: 1000 spheres under pressure simulated with contact algorithm.

the simulation. As shown in Fig. 13, the hundreds of grains moved under periodic force and gravity and the big ones were filtrated.

4.4 Collision of two elastic rings

The example is used to verify the proposed improved contact detection method. Two elastic rings collide at a relative speed of 60m/s, and then they bounce off. The inner radius, outer radius and thickness of every ring are 30mm, 40mm and 10mm, respectively, which is modeled by 29760 particles. The simulation results without and with the improved contact detection method are shown in Fig. 14 and Fig. 15, respectively. Comparing Fig. 14 and Fig. 15, it can be seen that the method avoids the gap between two rings when collision.

5 Conclusions

An object-oriented design for a MPM program has been presented. The program is flexible, extendible, and easily modified for a variety of MPM analysis procedures. The difficulties inherent in procedural based MPM programs are eliminated by de-

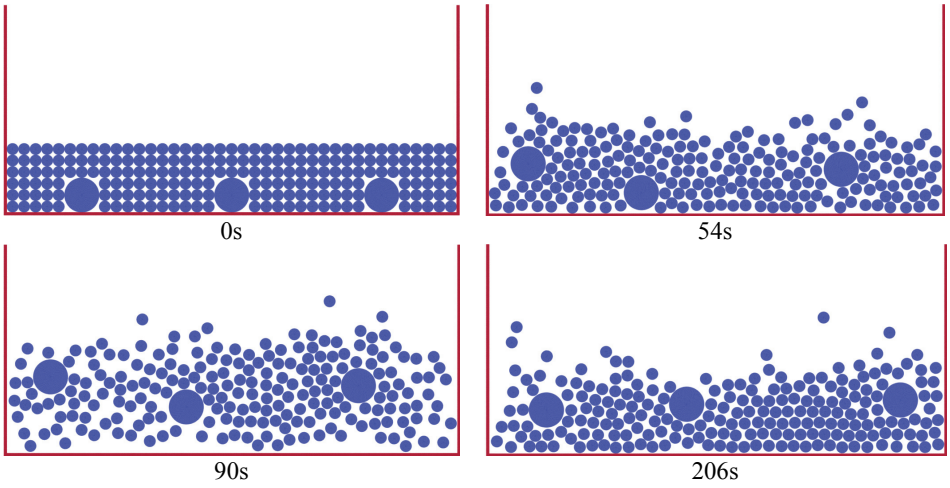

Figure 13: Simulation of shakes of grains

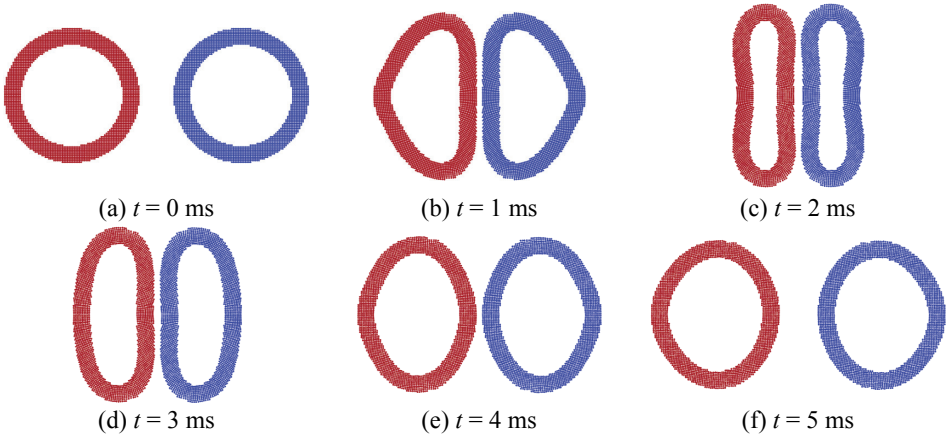

Figure 14: Collision simulation of elastic rings without improved contact detection method

sign. To modify or extend the program, the required knowledge of the components is restricted to the public interfaces of the classes. The reuse of code is promoted by the use of inheritance.

An improved contact detection method is proposed to avoid the contact occur earlier than the actual time. High implementation efficiency and low memory con-

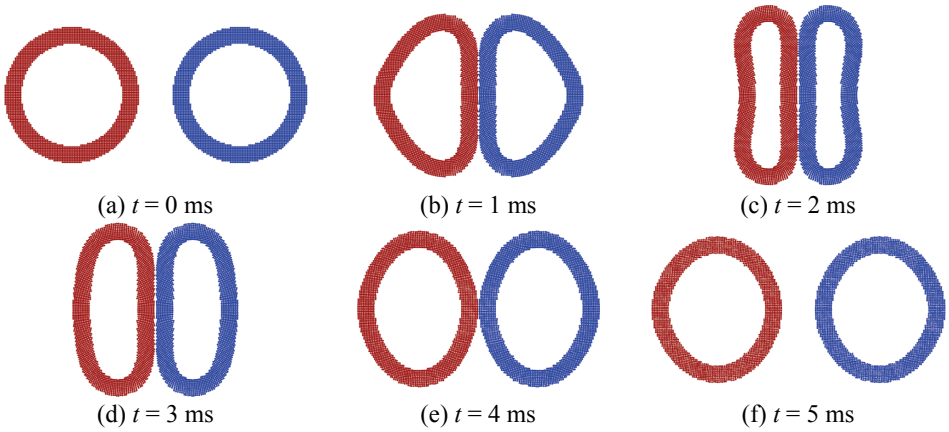

Figure 15: Collision simulation of elastic rings with improved contact detection method

sumed of MPM3Dpp are also achieved. Different operations for updating particle stresses and strains and contact algorithm are combining in MPM3Dpp. Local multi-mesh is presented, which is only created at overlapped nodes and needs less memory than global multi-mesh. Dynamic internal state variables and dynamic grid are presented instead of static ones to reduce memory consumed. Moving grid is proposed to prevent particles from flying out of computational region.

The example problems demonstrate the high computational efficiency and lower memory requirement of the framework whatever contact and non-contact cases. The advantage of the improved contact detection method is also illustrated by the collision of two elastic rings case. Successful implementation using C++ language verifies the designed program structure and provides a robust computational tool for MPM modeling.

Acknowledgement: This work is supported by the National Natural Science Foundation of China (10872107) and National Basic Research Program of China (2010CB832701).

References

Archer, G. C.; Fenves, G.; Thewalt, C. (1999): A new object-oriented finite element analysis program architecture. *Computers and Structures*, vol. 70, pp. 63-75.

Atluri, S. N.; Liu, H. T.; Han, Z. D. (2006): Meshless local Petrov-Galerkin (MLPG) mixed finite difference method for solid mechanics. *CMES: Computer Modeling in Engineering & Sciences*, vol. 15, no. 1, pp. 1-16.

Atluri, S. N.; Shen, S. P. (2002): The meshless local Petrov-Galerkin (MLPG) method. A simple & less-costly alternative to the finite element and boundary element methods. *CMES: Computer Modeling in Engineering and Sciences*, vol. 3, no. 1, pp. 11-51.

Bardenhagen, S. G. (2002): Energy conservation error in the material point method. *J. Comp. Phys.*, vol. 180, pp. 383–403.

Bardenhagen, S. G.; Brackbill, J. U. (1998): Dynamic stress bridging in granular material. *J. Appl. Phys.*, vol. 83, pp. 5732-5740.

Bardenhagen, S. G.; Brackbill, J. U. (2000): Numerical study of stress distribution in sheared granular material in two dimensions. *Phys. Rev. E*, vol. 62, pp. 3882-3890.

Bardenhagen, S. G.; Brackbill, J. U.; Sulsky, D. (2000): The material-point method for granular materials. *Comput. Methods Appl. Mech. Engrg.*, vol. 187, pp. 529-541.

Bardenhagen, S. G.; Guilkey, J.E. (2001): An improved contact algorithm for the material point method and application to stress propagation in granular material. *CMES: Computer Modeling in Engineering & Sciences*, vol. 2, no. 4, pp. 509–522.

Bardenhagen, S. G.; Kober, E. M. (2004): The generalized interpolation material point method. *CMES: Computer Modeling in Engineering & Sciences*, vol. 5, no. 6, pp. 477-495.

Baugh, J. W.; Rehak, D. R. (1992): Data abstraction in engineering software development. *Journal of Computing in Civil Engineering*, vol. 6, pp. 282-301.

Brackbill, J. U.; Kothe, D. B.; Rupple, H. M. (1988): FLIP: A low-dissipation, part method for fluid flow. *Comput. Phys. Commun.*, vol. 48, pp. 25-38.

Brackbill, J. U.; Rupple, H. M. (1986): FLIP: A method for adaptively zoned particle-in-cell calculations in two dimensions. *J. Comput. Phys.*, vol. 65, pp. 314-343.

Chen, Z.; Feng, R.; Xin, X.; Shen, L. (2003): A computational model for impact failure with shear-induced dilatancy. *Int. J. Numer. Meth. Engng.*, vol. 56, pp. 1979-1997.

Chen, Z.; Gan, Y.; Chen, J. K. (2008): A coupled thermo-mechanical model for simulating the material failure evolution due to localized heating. *CMES: Computer Modeling in Engineering & Sciences*, vol. 26, no. 2, pp. 123-137.

Chen, Z.; Hu, W.; Shen, L.; Xin, X.; Brannon, R. (2002): An evaluation of the MPM for simulating dynamic failure with damage diffusion. *Eng. Fract. Mech.*, vol. 69, pp. 1873-1890.

Chudoba, R.; Bittnar, Z.; Krysl, P. (1995): Explicit finite element computation: an object-oriented approach. Computing in Civil and Building Engineering, Pahl, pp. 139-145.

Coetzee, C. J.; Vermeer, P. A.; Basson, A. H. (2005): The modelling of anchors using the material point method. *Int. J. Numer. Anal. Meth. Geomech.*, vol. 29, pp. 879-895.

Cummins, S. J.; Brackbill, J. U. (2002): An implicit particle-in-cell method for granular materials. *J. Comput. Phys.*, vol. 180, pp. 506-548.

Duboispelerin, Y.; Zimmermann, T. (1993): Object-oriented finite element programming 3, an efficient implementation in C++. *Computer Methods in Applied Mechanics and Engineering*, vol. 108, pp. 165-183.

Fenves, G. L. (1990): Object-oriented programming for engineering software development. *Engineering with computers*, vol. 6, pp. 1-15.

Forde, B. W. R.; Foschi, R. O.; Stiemer, S. F. (1990): Object-oriented finite element analysis. *Computers and structures*, vol. 34, pp. 355-374.

Hu, W.; Chen, Z. (2003): A multi-mesh MPM for simulating the meshing process of spur gears. *Computers and Structures*, vol. 81, pp. 1991-2002.

Joris, J. C. Remmers; Rene, de Borst; Alan, Needleman (2005): Simulation of fracture in heterogeneous materials with the cohesive segments method. Proceedings of VIII International Conference on Computational Plasticity, Barcelona.

Lu, J.; White, D. W.; Chen, W. F.; Dunsmore, H. E. (1995): A matrix class library in C++ for structural engineering computing. *Computers and Structures*, vol. 55, pp. 95-111.

Luo, S. M.; Cai, Y. C.; Zhang, X. G. (2000): Object-oriented element free Galerkin method. *Chinese journal of mechanical engineering*, vol. 36, pp. 23-26.

Ma, J.; Lu, H. B.; Komanduri, R. (2006): Structured mesh refinement in generalized interpolation material point (GIMP) method for simulation of dynamic problems. *CMES: Computer Modeling in Engineering & Sciences*, vol. 12, no. 3, pp. 213-227.

Ma, S.; Zhang, X.; Lian, Y. P.; Zhou X. (2009): Simulation of high explosive explosion using adaptive material point method. *CMES: Computer Modeling in Engineering & Sciences*, vol. 39, no. 2, pp. 101-123.

Miller, G. R.; Rucki, M. D. (1993): A program architecture for interactive nonlin-

ear dynamic analysis of structures. Proc. 5 Int. Conf. Comput. Civ. Build Engng. V ICCCBE, New York ASCE, pp. 529-536.

Nairn, J. A. (2003): Material point method calculations with explicit cracks. *CMES: Computer Modeling in Engineering & Sciences*, vol. 4, no. 6, pp. 649-663.

Pan, X. F.; Xu, A. G.; Zhang, G. C.; Zhang, P.; Zhu, J. S.; Ma, S.; Zhang, X. (2008): Three-dimensional multi-mesh material point method for solving collision problems. *Commun. Theor. Phys.*, vol. 49, pp. 1129-1138.

Peng, H.; Zhang, X.; Ma, S.; Wang, H. K. (2008): Shared memory OpenMP parallelization of explicit MPM and its application to hypervelocity impact. *CMES: Computer Modeling in Engineering & Sciences*, vol. 38, no. 2, pp. 119-148.

Piekutowski, A. J.; Forrestal, M. J.; Poormon, K. L.; Warren, T. L. (1996): Perforation of aluminum plates with ogive-nose steel rods at normal and oblique impacts. *Int. J. Impact Engng.*, vol. 18, pp. 877-887.

Shen, L. M.; Chen, Z. (2005): A Silent Boundary Scheme with the Material Point Method for Dynamic Analyses. *CMES: Computer Modeling in Engineering & Sciences*, vol. 7, no. 3, pp. 305-320.

Steffen, M.; Wallstedt, P. C.; Guilkey, J. E.; Kirby, R.M.; Berzins, M. (2008): Examination and analysis of implementation choices within the material point method (MPM). *CMES: Computer Modeling in Engineering & Sciences*, vol. 31, no. 2, pp. 107-127.

Sulsky, D.; Chen, Z.; Schreyer, H. L. (1994): A particle method for history-dependent materials. *Comput. Methods Appl. Mech. Engrg.*, vol. 118, pp. 179-196.

Sulsky, D.; Zhou, S.; Schreyer, H. (1995): Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, vol. 87, pp. 236-252.

Tan, H. L.; Nairn, J. A. (2002): Hierarchical, adaptive, material point method for dynamic energy release rate calculations. *Comput. Methods Appl. Mech. Engrg.*, vol. 191, pp. 2095-2109.

Wallstedt, P. C.; Guilkey, J. E. (2007): Improved velocity projection for the material point method. *CMES: Computer Modeling in Engineering & Sciences*, vol. 19, no. 3, pp. 223-232.

Wiechowski, Z. (2004): The material point method in large strain engineering problems. *Comput. Methods Appl. Mech. Engrg.*, vol. 193, pp. 4417-4438.

York, A. R.; Sulsky, D.; Schreyer, H. L. (1999): The material point method for simulation of thin membranes. *Int. J. Numer. Meth. Engng.*, vol. 44, pp. 429-1456.

Yu, L. C.; Kumar, A. V. (2001): An object-oriented modular framework for implementing the finite element method. *Computers and Structures*, vol. 79, pp.

919-928.

Zhang, H. W.; Wang, K.P.; Chen, Z. (2009): Material point method for dynamic analysis of saturated porous media under external contact/impact of solid bodies. *Comput. Methods Appl. Mech. Engrg.*, vol. 198, pp. 1456-1472.

Zhang, X. F.; Subbarayan, G. (2006): jNURBS: An object-oriented, symbolic framework for integrated, meshless analysis and optimal design. *Advances in engineering software*, vol. 37, pp. 287-311.

Zhang, X.; Sze, K. Y.; Ma, S. (2006): An explicit material point finite element method for hyper-velocity impact. *Int. J. Numer. Meth. Engng.*, vol. 66, pp. 689-706.