ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2005, Vol. 34, No. 1

DEMMAT CODE FOR NUMERICAL SIMULATION OF MULTI-PARTICLE SYSTEMS DYNAMICS

Robertas Balevičius, Rimantas Kačianauskas

Laboratory of Numerical Modelling, Vilnius Gediminas Technical University
Sauletekio al. 11, Vilnius, Lithuania

Algis Džiugys

Laboratory of Combustion Processes, Lithuanian Energy Institute
Breslaujos g. 3,Kaunas, Lithuania

Algis Maknickas

Computer centre, Vilnius Gediminas Technical University
Sauletekio al. 11, Vilnius, Lithuania

Kęstutis Vislavičius

Dept. of Strength of Materials, Vilnius Gediminas Technical University
Sauletekio al. 11, Vilnius, Lithuania

Abstract. The paper presents three versions of software codes for the simulation of granular material dynamics based on the discrete element method. The codes DEMMAT_F90 and DEMMAT_PAS are used for the implementation of a purely procedural approach by the programming languages FORTRAN 90 and OBJECT PASCAL, while the code DEMMAT_CPP represents a purely object-oriented programming approach based on applying C++.

1. Indroduction

The discrete element method (DEM), initiated by Cundall and Strack [0], is one of the most powerful tools for simulation of granular material as an assembly of particles. The DEM is based on the Lagrangian approach, used to track the position, velocity, orientation and other parameters of particles during the simulation.

The application of DEM is usually associated with computational capabilities which are currently limited by a huge number of particles, small time step and finding the contacts between the particles. However, the capacities of computers are growing dramatically, making DEM accessible for use on personal computers. As a result, personal computers have become more popular in solving granular material problems by DEM.

The efficiency of DEM implementation in software depends on numerical methods, programming concepts and languages as well as on the compilers used. The choice of the programming concept and the programming language has the crucial influence on the efficiency of software.

Two programming concepts based on procedural and object-oriented programming (OOP) approaches may be implemented now through different programming languages. By using a procedural approach, the problem is decomposed into several composite algorithms which are implemented as their subroutines or functions, while all the data related to particles is kept in separate arrays. To extend functionality, new subroutines or functions are required in a code, which can make it large and complicated. At the same time, any changes of the code require some additional time in order to avoid coding errors. For these reasons, the procedural approach, according to Peters [2], is not sufficiently flexible for the creation of software for modelling particles as an integral object. The discussion of a procedural DEM code written in VISUAL BASIC and based on the previous FORTRAN code [3] is presented by Asmar et al. [4].

Instead of using a set of functions showing a global state, object orientation uses a set of interacting individual objects with their information hiding, encapsulation, inheritance and polymorphism. All data related to particles can be kept inside the object, and all operations with these data are described by methods defined within the classes of the objects. A

comprehensive study concerning the object-oriented approach to DEM programming is given by Peters and Džiugys [0], where programming modules are written in C++ and implemented in the TOSCA software package. According to the authors, OOP facilitates code maintenance and introduction of changes in a program as well as improving its transparency and extensibility.

However, the codes based on the procedural approach have one indubitable advantage in comparison with the object-oriented programs. They use less CPU time. The review of the performance of both approaches shows that early attempts to apply the OOP technique were disappointing. Forslund [0], using particle-in-cell methods in plasma physics simulation, found that the procedural FORTRAN code version ran about twice as fast as the comparable similar OOP C++ version. In [0], the typical benchmarks showed C++ lagging behind FORTRAN 90's performance by 20% to a factor of ten. Haney [0] and Robison [0] also commented on some aspects of applications of C++ language in considering computational speed.

The goal of the present investigation is to compare the computational performances of procedural and object-oriented approaches to discrete element method software. For this purpose, three versions of DEMMAT code have been created. Programming languages C++, FORTRAN 90 and OBJECT PASCAL were used.

2. DEM Technique

2.1. Governing Relations for Dynamics of Granular Material

The granular material is considered as a space filled with discrete spherical particles referred to as discrete elements with a geometric representation of their surfaces and a description of physical state. The motion of the particles is calculated from the applied forces and moments according to the Newton's second law. The boundary conditions are determined by walls, which are treated as particles with infinite radius and mass. The external action is induced with kinematics boundary conditions.

Two spherical particles in contact, i and j, are defined by the positions $\mathbf{x}_i$ and $\mathbf{x}_j$ of their centres of gravity O_i and O_j and by translation and rotation velocities $\mathbf{v}_i$ and $\mathbf{v}_j$, $\mathbf{w}_i$ and $\mathbf{w}_j$, respectively (Figure 1).

The details for explicit evaluation of the parameters relating to the depth of the overlap h_{ij}, the normal unit vectors $\mathbf{n}_{ij}$ and $\mathbf{n}_{ji}$ pointed in the direction of the contact surface through the centre of the overlap area towards the particles i and j, tangential direction unit vector $\mathbf{t}_{ij}$, the relations between normal tangential relative velocities $\mathbf{v}_{n,ij}$, $\mathbf{v}_{t,ij}$, etc. may be found in [0, 0].

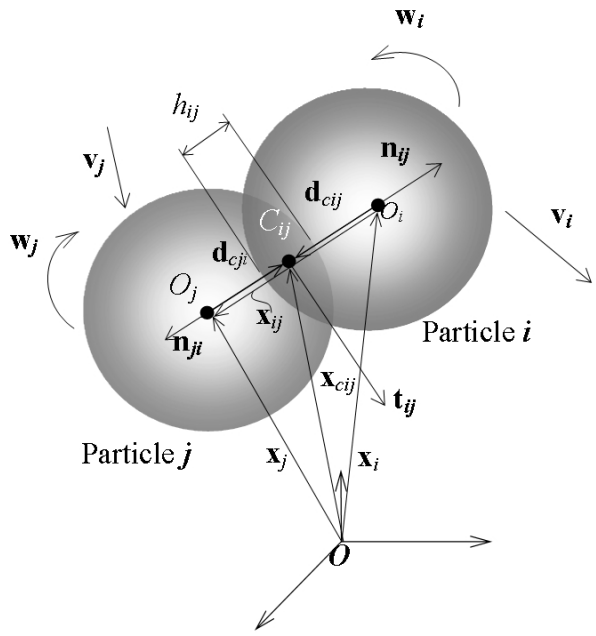

Figure 1. Particles i and j: geometry of contact

The composition of the media is time-dependent because individual particles change their position due to free rigid body motion or because of contacting with the neighbouring particles or walls. Thus, Newton's second law is applied to each particle i to update its translational and rotational displacements according to:

$$m_i \frac{d^2\mathbf{x}_i}{dt^2} = \sum_{j=1, j\neq i}^{N} \mathbf{F}_{ij} + m_i\mathbf{g}\,, \tag{1}$$

$$I_i \frac{d^2\theta_i}{dt^2} = \sum_{j=1, j\neq i}^{N} \mathbf{d}_{cij} \times \mathbf{F}_{ij}\,, \tag{2}$$

where $\mathbf{x}_i$, $\boldsymbol{\theta}_i$ are vectors of the position of the centre of gravity and orientation of the particle, m_i is the mass of the particle i ($i = 1, N$), I_i is the inertia moment of the particle,

$\mathbf{g}$ is vector of gravity acceleration, $\mathbf{d}_{cij}$ is the vector pointed from particle centre to contact point C_{ij}, $\mathbf{F}_{ij}$ is the vector presenting the sum of contact forces and torques.

The numerical solutions of differential equations (1)–(2) for each particle i at the time $t + \Delta t$ (where Δt is the time step) are performed by using the *5th-order Gear predictor-corrector* [0] scheme. Therefore, the positions $\mathbf{x}_i^p$, velocities $\mathbf{v}_i^p$, accelerations $\mathbf{a}_i^p$ and higher order derivatives $\mathbf{b}_{ni}^p$ (where $n = 3, 4, 5$) of the particles denoted hereafter by superscript p may be predicted by the following expressions:

$$\mathbf{x}_i^p(t+\Delta t) = \mathbf{x}_i(t) + \Delta t\mathbf{v}_i(t) + \frac{1}{2}\Delta t^2\mathbf{a}_i(t) + \mathbf{b}_{3i}(t) + \mathbf{b}_{4i}(t) + \mathbf{b}_{5i}(t)\,, \tag{3}$$

$$\mathbf{v}_i^p(t+\Delta t)=\mathbf{v}_i(t)+\Delta t\mathbf{a}_i(t)+\frac{1}{\Delta t}\left(3\mathbf{b}_{3i}(t)+4\mathbf{b}_{4i}(t)+5\mathbf{b}_{5i}(t)\right), \quad (4)$$

$$\mathbf{a}_i^p(t+\Delta t)=\mathbf{a}_i(t)+\frac{2}{\Delta t^2}\left(3\mathbf{b}_{3i}(t)+6\mathbf{b}_{4i}(t)+10\mathbf{b}_{5i}(t)\right), \quad (5)$$

$$\mathbf{b}_{3i}^p(t+\Delta t)=\mathbf{b}_{3i}(t)+4\mathbf{b}_{4i}(t)+10\mathbf{b}_{5i}(t), \quad (6)$$

$$\mathbf{b}_{4i}^p(t+\Delta t)=\mathbf{b}_{4i}(t)+5\mathbf{b}_{5i}(t), \quad (7)$$

$$\mathbf{b}_{5i}^p(t+\Delta t)=\mathbf{b}_{5i}(t). \quad (8)$$

When the prediction is made, the accelerations denoted hereafter by superscript c are calculated according to the forces $\mathbf{F}_i\left(\mathbf{x}_i^p,\mathbf{v}_i^p\right)$ obtained by evaluating new positions and velocities of particles:

$$\mathbf{a}_i^c(t+\Delta t)=\frac{\mathbf{F}_i\left(\mathbf{x}_i^p,\mathbf{v}_i^p\right)}{m_i}. \quad (9)$$

Then, the changes between the accelerations of the correction and prediction states are calculated:

$$\Delta\mathbf{a}_i(t+\Delta t)=\mathbf{a}_i^c(t+\Delta t)-\mathbf{a}_i^p(t+\Delta t), \quad (10)$$

Finally, the positions, velocities and higher-order time derivatives are corrected taking into consideration the changes of accelerations obtained:

$$\begin{pmatrix}\mathbf{x}_i^c(t+\Delta t)\\ \mathbf{v}_i^c(t+\Delta t)\\ \mathbf{a}_i^c(t+\Delta t)\\ \mathbf{b}_{3i}^c(t+\Delta t)\\ \mathbf{b}_{4i}^c(t+\Delta t)\\ \mathbf{b}_{5i}^c(t+\Delta t)\end{pmatrix}=\begin{pmatrix}\mathbf{x}_i^p(t+\Delta t)\\ \mathbf{v}_i^p(t+\Delta t)\\ \mathbf{a}_i^p(t+\Delta t)\\ \mathbf{b}_{3i}^p(t+\Delta t)\\ \mathbf{b}_{4i}^p(t+\Delta t)\\ \mathbf{b}_{5i}^p(t+\Delta t)\end{pmatrix}+\begin{pmatrix}0.5c_0\Delta t^2\\ 0.5c_1\Delta t\\ c_2\\ 0.5c_3\Delta t^2\\ 0.5c_4\Delta t^2\\ 0.5c_5\Delta t^2\end{pmatrix}\Delta\mathbf{a}_i(t+\Delta t). \quad (11)$$

The values of the constants c_i depend on the desired accuracy, and for the second order differential equation are $c_0 = 3/16$, $c_1 = 251/360$, $c_2 = 1.0$, $c_3 = 11/18$, $c_4 = 1/6$ and $c_5 = 1/60$ [0], respectively.

The validation tests presented in [0] illustrate the performance of the above scheme producing low artificial damping.

The contact forces $\mathbf{F}_{ij}$ found between two particles i and j act at the contact point C_{ij} (Figure 1) and can be expressed in terms of the normal and the tangential components $\mathbf{F}_{n,ij}$ and $\mathbf{F}_{t,ij}$ according to [0, 0, 0].

2.2. Contact Searching and Referencing Technique

In order to evaluate forces $\mathbf{F}_i$ as a sum of forces acting on the particles and including all forces due to viso-elastic contacts between the particles and their neighbours, the fast contact searching approach procedure should be applied. Thus, in a system containing N particles, the general contact detection problem is of the size $O(N^2)$, when the neighbouring particles search algorithm is applied sequentially. In order to reduce the number of all particle pair combinations, one of the simplest *zoning* algorithms [0] for finding the neighbours was used for the contact detection. It is described below.

Three-dimensional domain of the granular media is divided into cubic cells (Figure 2) of the size equal to the diameter of the largest particle. The neighbour-searching algorithm comprises referencing of individual particles to the cells and constructing of the *neighbour list of particles*.

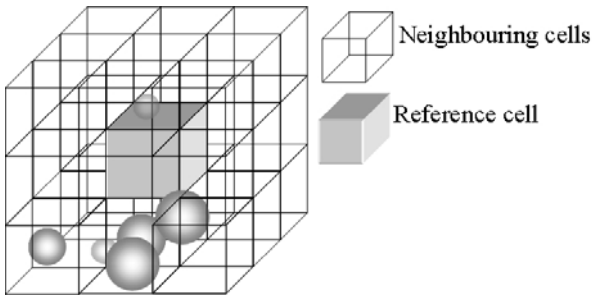

Figure 2. The fragment of schematic diagram of 3D domain divided into cubic cells

All particles, whose centres of gravity are within the cells (Figure 2), are referenced by using the `ceiling` of the components of vector $\mathbf{x}_i$, the size of the cell and the geometric data of the domain of granular media. Thus, the referenced particles indices, cells indices and the number of particles per cell are stored in the memory. When referencing is made, *the neighbour list of particles* is constructed by assembling particles indices from the neighbouring cells which are around the reference cell. In a 3D case, the reference cell includes 27 neighbouring cells (Figure 2). The neighbouring cells also include the boundary zones, i.e. zones which are beside the walls. The *boundary neighbour list of particles* was constructed in the manner described above.

Finally, the procedure of contact detection containing sequential checking of contacts of the type particle-particles and particle-walls is performed by using particle indices obtained from the *neighbour list of particles* and the *boundary neighbour list of particles.*

2.3. Algorithm Order

The application of DEM to the code DEMMAT consists of the following steps and items:

Pre-processor

1. Set-up of initial conditions of the particles and the walls

Processor

1. Assigning time t = t + Δt
2. Predicting the dynamical state: predicting positions, velocities, accelerations and higher-order time derivatives according to (3)–(8)
3. Searching for and referencing to the contacts particle–particle:
   Loop for each particle *i*
   - For $i = 1, N - 1$

   Loop for each particle *j*
   - For $j = j, N_p$
4. Contact initialization from the neighbour list of particles by indices *j* and summation of forces and moments for particles *i* and *j*.
5. Searching for and referencing to the contacts particle–wall:
   Loop for each wall *i*
   - For $i = 1, N_{walls}$

   Loop for each particle *j*
   - For $j = 1, N$
6. Initialization of contact from *the boundary neighbour list of particles* by indices *j,* summation of forces for walls *i* and summation of forces and moments for particles *j*.
7. Global summation of forces and moments obtained in steps 4 and 6.
8. Application of the Newton's second law and determination of translational and angular accelerations for particles according to equation (9).
9. Correction of the dynamical state: the dynamical state from step 2 is corrected according to (11)
10. If current time is within the entire time period ($t \le t_{end}$) – go to step 1, otherwise, go to the next step.

Post-processor

1. Visualization.

Note: N_p is the number of neighbours for particle *i*, N_{walls} is the number of walls.

The only *processor* stage of this algorithm will be considered below.

3. Program Implementation

First, the software DEMMAT_CPP written within the OOP approach was developed. The application of this approach to simulate the motion of granular media was disappointing with respect of CPU time of simulations. Obviously, it is not worth worrying too much about getting a faster software code when the problem can be numerically solved in minutes, but when you have to wait for weeks for the results, then it is better to spend some time and find a way to make a faster software code. Therefore, other codes such as DEMMAT_F90 (FORTRAN 90) and DEMMAT_PAS (OBJECT PASCAL) relying on the purely procedural approach have been written in order to examine the potential losses of computational efficiency, which are avoided by programming according to the object-oriented approach. The code DEMMAT_PAS was used additionally to determine the differences in the efficiency between the procedural approaches.

3.1. The Procedural Concept for the Codes DEMMAT_F90 and DEMMAT_PAS

The procedural concept and the algorithm described above have been implemented in the original program called DEMMAT_F90 which was written using FORTRAN 90 and compiled on the *Microsoft Visual Fortran 6.1* compiler. The code comprises `subroutines`, `functions` and `module` implementations and uses `intrinsic` and supporting functions from IMSL library for vector algebra. Some peculiarities of programming are described below.

All subroutines of the code DEMMAT_F90 have been written as `external` procedures. The `interface` blocks were used to define the procedures argument details. The vectors presenting the main dynamical parameters of particles, such as their positions, velocities, accelerations and high-order derivatives for translational and rotational motions according to equations (3-11) as well as the vectors of the particle forces $\mathbf{F}_i$ and torques calculated by expressions (1-2) are described as two-dimensional `real(8)` precision type arrays with `allocatable` attributes having the local entities. Rows of these arrays present the indices of the particles, while the columns present the Cartesian components of vectors. The dynamical parameters are accessible by a number of different program subroutines by specifying `intent` attributes only. For example, prediction and correction of the dynamical states of particles were calculated by calling the subroutines named `gear_predictor` and `gear_corrector`. The list of `actual` arguments of these subroutines is the same because the list of `dummy` arguments is declared by using `intent(inout)` attributes.

The procedural approach usually shows preference to the initial declaration of invariable bounds of arrays. The memory and length of arrays that are related to the contact detection algorithm always fluctuate during the iteration process and can be precisely determined only at runtime. This problem is often solved by using the linked list method based on the implementation of pointers and target variables. The present analysis has shown that this flexible alternative to arrays allocation in DEM simulations gives the increase of CPU time because of the dynamical extension or reduction of the capacity of memories for arrays must be swapped almost in every iteration. Therefore, the specification of arrays bounds approximately prescribed as initial invariable values with the additional control of their overestimation was implemented in the DEMMAT_F90 code.

The code DEMMAT_PAS corresponds to the code DEMMAT_F90 in all aspects: it has the same parameters, arrays, procedures (subroutines, functions) and the same type of variables is used in it. The main difference between the investigated codes is as follows: in DEMMAT_F90 the dynamical parameters of particles are accessible to a number of different program units by specifying `intent` attributes only, while in DEMMAT_PAS these parameters are declared as global. The code DEMMAT_PAS was compiled on the *Borland Delphi 6* compiler.

3.2. The Object-Oriented Concept for the Code DEMMAT_CPP

The original program DEMMAT_CPP based on the purely object-oriented programming concept has been written using *C++* and compiled by *Microsoft Visual C++* 2003 compiler. The code implements the numerical algorithm described above and used in the program DEMMAT_F90.

All spherical particles were defined as separate objects of the same C++ `class` of `spherical_particle`, while walls were also defined as particles referring, however, to the objects of C++ `class` of `plane`. We used a pure approach of the OOP methodology implying that all parameters of each particle, such as position, velocity, etc. were kept inside the object of the particle. This approach is an extreme case of OOP methodology, which has a certain advantage for code writing and revising, but is more expensive in terms of CPU time, whereas referencing to any parameter of any particle is doubled by one reference to the particle object and another reference to its own parameter inside the particle object.

A list of neighbours of any particle was defined as a list of `pointers` to objects of the `neighbour` class, which are kept inside the index of a neighbouring particle. The search for the neighbours of a particle, contact detection and other parts of the algorithm were similar to those used to code the program DEMMAT_F90.

4. The Computational Experiment

The efficiency of the presented concept for numerical simulation of the dynamics of granular materials by DEM is investigated taking into consideration the results of the numerical test described below.

The test simulates free compacting of granular material in a cubic box, which is assumed to be the computation domain. The side of the box is 2.0 m long. An assembly of a particular number of particles presents granular material. The values of the particle radii R_i varying between 0.003 m and 0.005 m are defined randomly with uniform distribution. Initially, the particles were distributed in the space over the bottom as an orthogonal and uniform grid. Initially the particles are placed into the centres of the cells to ensure that they are not in contact at the beginning of testing (Figure 3). The initial velocities are also defined randomly with uniform distribution, while their magnitudes range from 0 to 1 m/s. The compacting of particles in a box is driven by particles moving under the force of gravity defined by gravity vector **g** ($g_x = g_y = 0$, $g_z = 10$ m/s^2). The time integration of equation (1-2) was carried on by the constant time step $\Delta t = 10^{-5}$ s. The data on the granular material are given in Table 1.

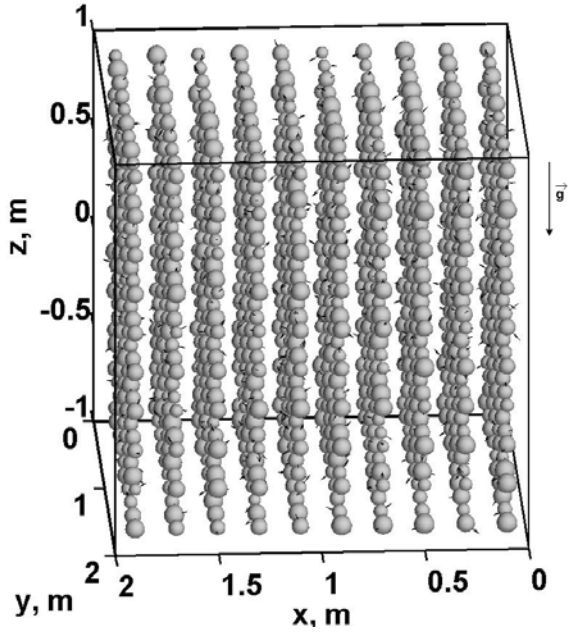

Figure 3. Initial positions and velocity vectors of 1000 particles

Table 1. Major data on the particles

Quantity	Symbol	Value
Particle radii, m	R	0.003 – 0.005
Material density, kg/m^3	ρ	1000
Poisson's ratio	ν	0.30
Elastic modulus, Pa	E	$0.3 \cdot 10^6$
Shear modulus, Pa	G	$0.11 \cdot 10^6$
Normal damping coefficient, 1/s	γ_n	60.0
Shear damping coefficient, 1/s	γ_t	60.0
Friction coefficient	μ	0.6

It is important to know that the efficiency of contact searching in order to reduce the CPU time depends on the establishing of the ideal cell size of 3D domain. In general, the ideal cell size is determined by a balance between two conflicting requirements: a) the cells should be small in order to reduce contact check depending on the number of their particles; b) the cells should be reasonably coarse, otherwise, the number of particles associated with their transitioning between cells becomes too large. It has been found that, in order to achieve the lowest computing time, the ideal ratio between the radii of the largest particles and the

cell size is equal to two. Figure 4 shows the CPU time spent by using various numbers of particles with different cell sizes when 10000 cycles are completed.

As shown in Figure 4 if a 3D domain is divided into $2R_{max}$ cells, the CPU time linearly relates with the number of particles, while if this domain is divided into $4R_{max}$ cells, the CPU time increases non-linearly depending on the number of particles. When the sequential contact detection algorithm is used (no cells are applied), the CPU time grows dramatically as in the problem $O(N^2)$. The graphs presented in Figure 4 also prove that for the number of particles not exceeding 200, the sequential contact searching and zoning algorithms yield approximately the same CPU time. However, if the 3D domain contains 1000 particles, the contact searching and referencing algorithms gain in CPU time by about 12 times in comparison with the domain not divided by cells.

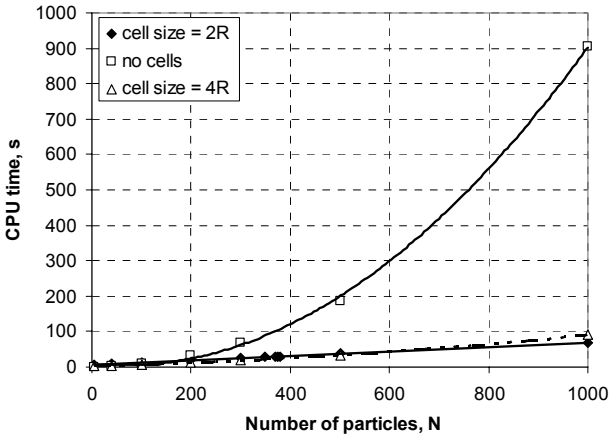

Figure 4. The relationship between the number of particles and CPU time for cells of various sizes

The limit of the CPU time ratio versus the number of particles obtained by using the sequential contact searching and zoning algorithms, taking into account $4R_{max}$ cell size, may be written as $\lim_{N\to\infty} \frac{0.001N^2 - 0.131N + 9.32}{5\cdot 10^{-5}N^2 + 0.041N + 1.71}$. The above solution shows that cell size is an important factor while $N \to \infty$. When using $4R_{max}$ cells the contact searching and referencing algorithm gains in CPU time by about 20 times in comparison with the domain not divided by cells. When $N \to \infty$ and $2R_{max}$ cells are used the CPU time can be saved by more than 200 times.

In the present test $2R_{max}$ cells are used.

The test consists of two stages. The first stage ends when the granular material state gained in free compacting in a cubic box (Figure 5) is at rest. This state is assumed to be the initial state for the second test characterized by the compression of the granular material caused by a wall moving.

At this stage, the main part of the codes such as neighbour searching and referencing works properly on the ultimate output because all of the particles are in contact with each other, the latter depending only on the changing numbers of particle neighbours and the number of particles contacting with the walls during the compressive motion of the right wall (Figure 6)**.** In this case, the right wall, which is at constant position at $x = 0$ m as shown in Figure 3, starts to move in x-direction with constant velocity $v_x = -0.5$ m/s (Figure 6). As a result, the granular material is compressed. The duration of the compression simulation is limited by the time interval of three seconds.

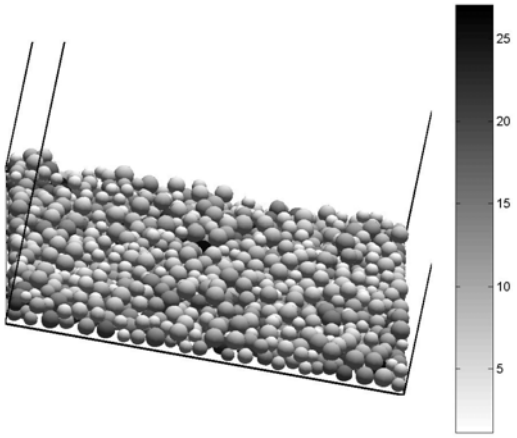

Figure 5. The initial state and forces (N) of particles being compressed by a moving wall

The analysis of the results obtained at the second test stage shows that the CPU time per time step used by the predictor and corrector of time integration is stable throughout the simulation, whereas the operation of the predictor and corrector depends on the total number of particles, being independent of the number of particle neighbours.

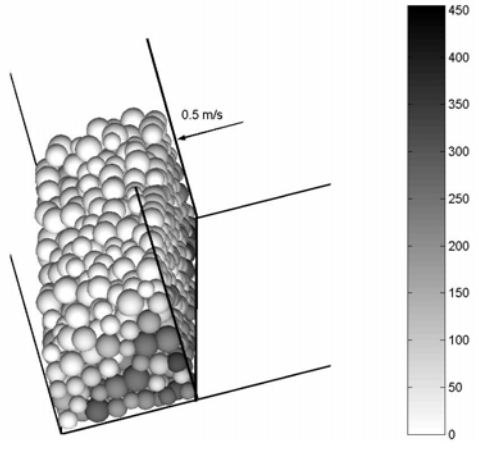

Figure 6. The state and particle forces (N) of granular material compressed by a moving wall for three seconds

As mentioned above, the main result of the software implementation efficiency is the ratio of CPU time taken by the software execution. The graphs in Figure 7 show the CPU time per step used by the software DEMMAT_F90, DEMMAT_CPP and DEMMAT _PAS. As can be seen the application of a conven-

tional purely object-oriented programming concept (software DEMMAT_CPP) costs by about 5-5.5 times more than the CPU time used by the procedural concept (DEMMAT_F90, DEMMAT_PAS). A comparison of the procedural approaches based on the codes DEMMAT_F90 and DEMMAT_PAS shows that the code DEMMAT_F90 runs by about 2 times faster than DEMMAT_PAS.

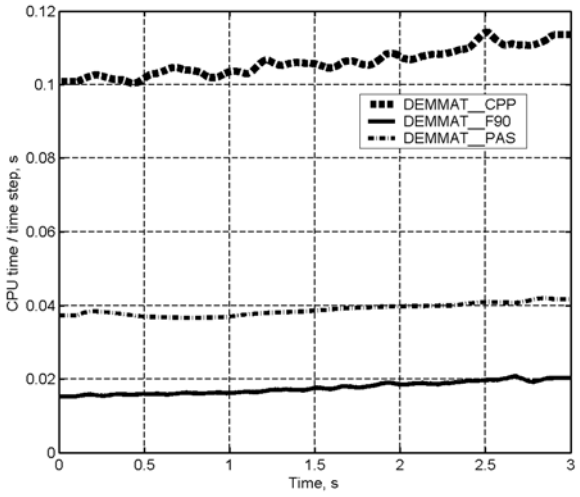

Figure 7. CPU time comparison between the described codes

The additional analysis of the influence of the neighbour searching and referencing algorithms on the CPU time, as well as the calculation of the particles forces, prediction and correction of time integration of the dynamical state were also performed. The results obtained by using the code DEMMAT_F90 show that the CPU time per step required by predicting and correcting the dynamical state when the time is stable throughout the simulation is equal approximately to 0.00025 s for predictor and 0.00018 s for corrector, while for contact searching and referencing algorithm including the calculation of particle forces the time taken is approximately 0.016 s. The execution time for the code DEMMAT_CPP is as follows: the predictor and corrector have taken 0.0321 s and 0.0167 s, respectively, while contact searching and referencing algorithms, including the calculation of forces for each of the particles, have taken 0.0542 s. Taking into consideration all the details, DEMMAT_F90 procedures involving the use of predictor and corrector are approximately by 130 and 90 times faster than the corresponding procedures applying the code DEMMAT _CPP. The procedures for contacts evaluation in the code DEMMAT_F90 are by 5-4 faster than the respective procedures in DEMMAT_CPP. This is the most important result obtained in the present investigation dealing with highly time-consuming DEM simulations.

The test has been run on a personal computer of 2.4 GHz Athlon CPU with 1.0 GB RAM, operating under OS Windows XP. All three software codes described above yielded the same results of the dynamics of granular material.

5. Conclusions

1. Three versions of software codes for the simulation of granular material dynamics based on the discrete element method have been developed and tested. The use of the codes DEMMAT_F90 and DEMMAT_PAS was based on a purely procedural approach implemented by the programming languages FORTRAN 90 and OBJECT PASCAL, while the code DEMMAT_CPP relied on a purely object-oriented programming approach implemented by the programming language C++. Other conceptual parts including the neighbour searching and referencing algorithms, the model of contact forces and time integration method were implemented in the same way in all three codes.
2. The numerical results obtained by simulating identical representative cases of the dynamic behaviour of granular material clearly illustrate better performance of the procedural approach. The code developed in FORTRAN 90 by using the procedural approach runs by about 5-5.5 times faster than the code developed in C++ within the object-oriented approach in the manner described above.
3. The comparison of OBJECT PASCAL and FORTRAN 90 software shows that the FORTRAN 90 code is by about 1.5-2 times faster than the OBJECT PASCAL code. It is believed that some special programming tricks could be found to optimize the software and to make OBJECT PASCAL as fast as FORTRAN.
4. It is determined that the CPU time largely depends on the ratio of the particle to the ideal cell size of the 3D domain. The minimum CPU time can be recorded if the 3D domain is divided into cells of $2R_{max}$ size.

References

[1] **P.A. Cundall, O.D.L. Strack.** A discrete numerical model for granular assemblies. *Geotechnique*, *Vol*.29, *No*.1, 1979, 47–65.

[2] **B. Peters.** Efficient software development and use for engineering applications with TOSCA (Tools of Object-oriented Software for Continuum Mechanics Applications). *CFD 96 proceedings, Fourth Annual Conference of the CFD Society of Canada, Ottawa, Ontario, Canada, June* 2-4, 1996, 18-27.

[3] **P.A. Langston, U. Tuzun, D.M. Heys.** Discrete element simulation of granular flow in 2D and 3D hoppers: dependence of discharge rate and wall stress on particle interactions. *Chem. Engineering Science*, *Vol*.50, 1995, 967-981.

[4] **B.N. Asmar, P.A. Langston, A.J. Matchett, J.K. Walters.** Validation tests on a distinct element model of vibrating cohesive particle systems. *Computers and Chemical Engineering, Vol*. 26, 2002, 785-802.

[5] **B. Peters, A. Džiugys.** Numerical simulation of the motion of granular material using object-oriented techniques. Comput. *Methods Appl. Engrg., Vol*.191 2002, 1983-2007.

[6] **D.W. Forslund, C. Wingate, P. Ford, J. Jackson, S.C. Pope.** Experiences in Writing a Distributed Particle Simulation Code in C++. *Poc*. 1990 *USENIX* C++ *Conf*., 1990, 28-40.

[7] **T.L. Veldhuizen, M.E. Jernigan.** Will C++ be faster than Fortran? *Proceedings of the* 1*st International Scientific Computing in Object-Oriented Parallel Environments* (ISCOPE'97), 1997, 40-45.

[8] **S.W. Haney.** Is C++ fast enough for scientific computing? *Computer Physics, Vol*.8, 1994, 690-694.

[9] **A.D. Robison.** C++ gets faster for scientific computing. *Computers in Physics*, Vol. 10, *Sep/Oct*, 1996, 458-462.

[10] **R. Balevičius, A. Džiugys, R. Kačianauskas.** Discrete element method and its application to the analysis of penetration into granular media. *Journal of Civil Engineering and Management, Vol.*10, 2004, *No*.1, 3-14.

[11] **R. Balevičius, A. Džiugys, R. Kačianauskas.** Simulation of penetration in granular media. *CD ROOM proc. of* 15*th International Conference on Computer Methods in Mechanics CMM*-2003/1*st Central European Association for Computational Mechanics Conference on Computational Mechanics, June* 3-6, 2003, *Gliwice/Wisla, Poland*.

[12] **M.P. Allen, D.J. Tildesley.** Computer simulation of liquids. *Oxford: Clarendon Press*, 1991.

[13] **G.H. Kohring.** Dynamical simulations of granular flows on multi-processor computers. *Computational methods in applied sciences* '96, *John Wiley & Sons Ltd*., 1996, 190-196.

DOI: 10.5755/j01.itc.34.1.11959