DEM Implementation

Computational Efficiency of a DEM Code is Crucial.

In simulating a soft sphere dynamic characteristics are numerically estimated by an iterative time integration of equation of motion for each individual particle.

Developing a general purpose DEM code for granular flow requires a framework for the explicit time integration of equations of motion. This framework requires us to choose a small enough time step for which a disturbance wave propagates a distance not farther than the distance between centers of contacting particles.

Simulation of large scale systems and performing tasks such as contact search, integration etc requires large time complexity and hence needs new algorithms to be optimized.

Program Structure:

A DEM Simulation is implemented according to the given flowchart. Based on this we have several Algorithms for the simulation of Large scale Particles.

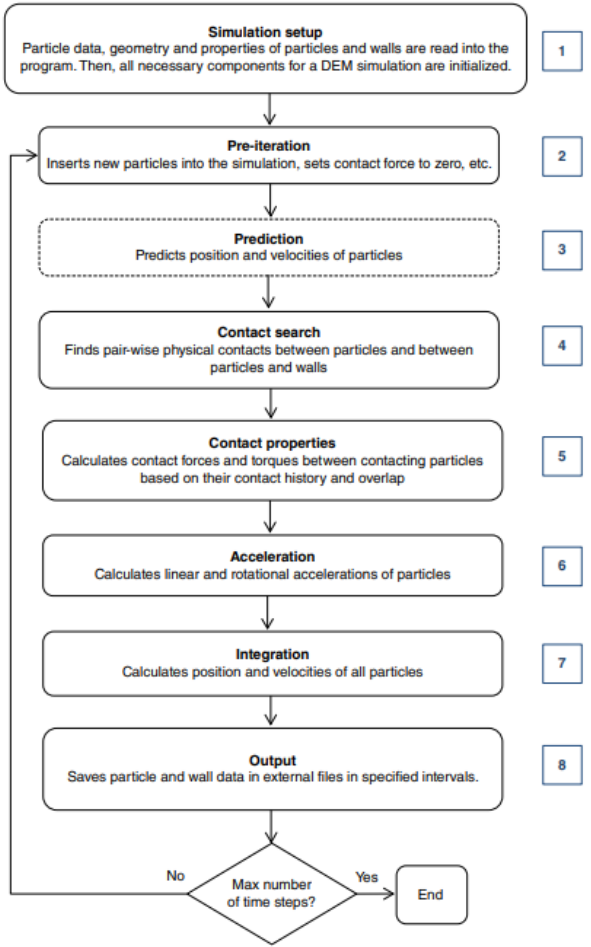

Figure 3.1 Algorithm of the DEM program for the flow of non-cohesive spherical particles without heat or mass transfer and fluid inertial effects

Algorithms for the simulation of Large Scale Particles:

1. Efficient Contact Search Algorithms:

    a. Cell Based : NBS, NBS-Munjiza, Hierarchical Contact Search

    b. Sort Based

    c. Tree Based

1. High Order Integration Methods
2. Selecting small values for spring stiffness
3. Parallelization of Code

Contact Search Algorithms:

DEM Simulation involves a large number of particles in physical or non-physical contact.
Finding all contacts is necessary for successful simulations of granular materials. It is not possible to calculate interaction force and torques in each time step without ample info on pairwise contact interactions.

A contact detection algorithm is a collection of operation and methods that should be done step by step to find all contacting particles and to prevent redundant contact tests between particles which are far from each other.

A Contact detection Algorithm must be robust, CPU & RAM Efficient and easy to implement.

Nested Loop Implementation for Contact Detection has a Time Complexity of O(n^2). However a Contact Detection Algorithm for broad search has a time complexity between O(n) and O(n^2).

Contact Detection Algorithm

A Contact detection Algorithm can be divided into 2 separate phases, broad (neighbour) search and fine(exact) search.

In Broad Search a list of particles in the neighbourhood of the target particle is established.

In Fine Search the exact contact between the particles is tested.

The Contact detection Algorithms are further of 3 types:

1. Cell based
2. Sort based
3. Tree based

Cell based Algorithms:

Definition of the problem:

The aim of the broad search algorithm is to find pairs of spheres with a known radius and position that may have an overlap. Spheres are bounding boxes around each body with identical sizes or a size distribution.

Some algorithms suit mono-sized spheres, but we consider spheres with a size distribution to generalize our implementation.

Each sphere in the system has a unique id number that is used to identify spheres in the code. Obviously, the id number of the first body is 1, that of the second body is 2, and so on.

Two points (Xmin, Ymin, Zmin) and (Xmax, Ymax, Zmax) determine the lower and upper points of the simulation domain. All the entities of the simulation like the particles and wall must remain in this domain.

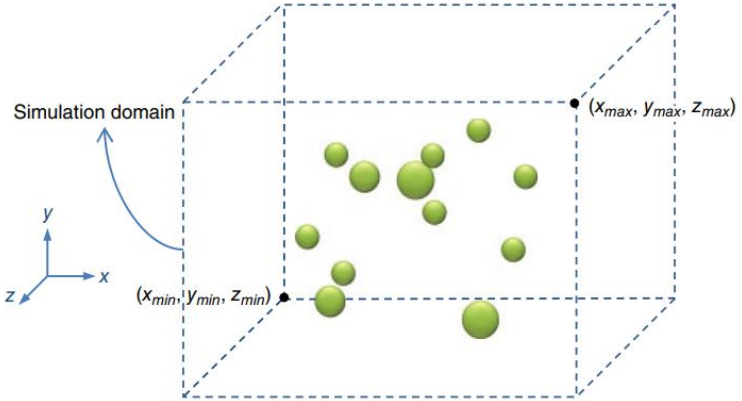

Figure 3.3 Bounding spheres in 3D Cartesian coordinates enclosed by simulation domain

Cell based Algorithms:

In the cell-based algorithms, the simulation domain is discretized into cube cells (square cells in a 2D space) with the same size of dx. Real coordinates of spheres are converted into integer coordinates (ix,iy,iz) by the following equations:

$$ix = \text{int}\left(\frac{x_i - x_{min}}{dx}\right) + 1$$

$$iy = \text{int}\left(\frac{y_i - y_{min}}{dx}\right) + 1$$

$$iz = \text{int}\left(\frac{z_i - z_{min}}{dx}\right) + 1$$

After integerization, spheres are mapped onto the cells according to their integer coordinates.If we choose the size of the cube (dx) as large as the diameter of the largest sphere in the system, we can ensure that each sphere can have contact with spheres in the same cell and adjacent cells.). In this way, we localize the contact detection test of the target sphere to the spheres in these cells. This reduces the number of fine contact searches and increases the accuracy of the contact search algorithm

NBS Contact Search Algorithm:

The name NBS (No Binary Search) has been given to this algorithm because it detects all contacts only once. This algorithm consists of two main steps:
Mapping spheres onto the cells and broad contact search.

In the first step all the elements are mapped onto the cube cells according to the integer coordinates. For this purpose a 3D array called the Head array can be used whose elements represent cells with indices (ix,iy,iz).

Size of Head Array: (nx,ny,nz)

Cell value = -1 if it is empty,
id_sphere if it is not empty

Along with the Head array we also define a Next Array:

Next array is a 1-D array of integers of size N. Head and Next Arrays together create a linked list that can be used to map spheres onto the cube cells.

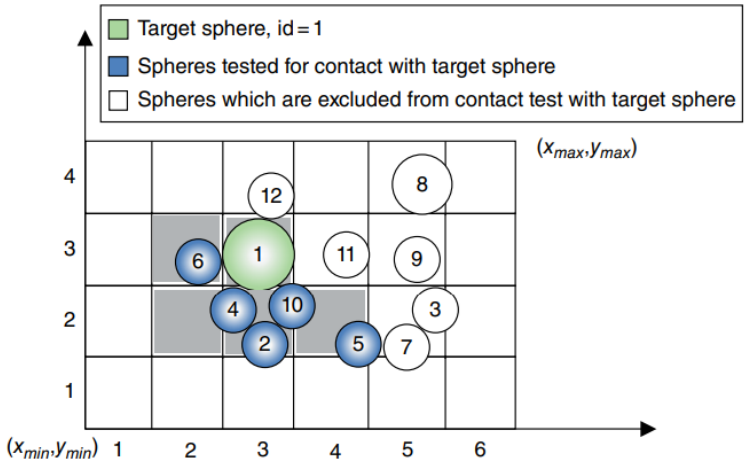

Figure 3.4 Discretized simulation domain and bounding sphere in 2D space

NBS Contact Search Algorithms:

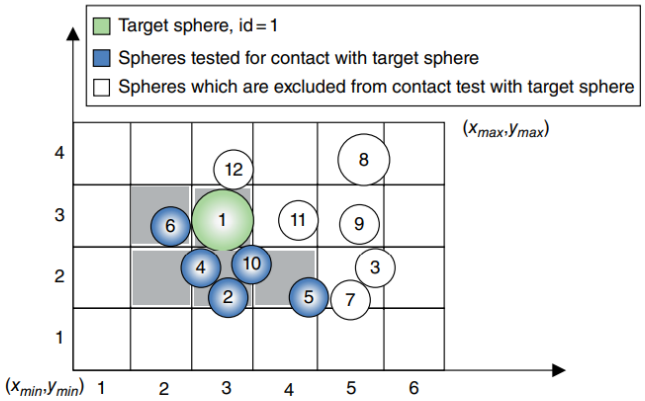

Figure 3.4 Discretized simulation domain and bounding sphere in 2D space

$$\text{Head} = \begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & 10 & 5 & 7 & -1 \\ -1 & 6 & 1 & 11 & 9 & -1 \\ -1 & -1 & 12 & -1 & 8 & -1 \end{bmatrix}$$

Array index	1	2	3	4	5	6	7	8	9	10	11	12
Next =	−1	−1	−1	2	−1	−1	3	−1	−1	4	−1	−1

Figure 3.5 The linked list constructed based on the NBS method for the particulate system shown in Figure 3.4

NBS Mask:

If we consider (0,0) as the coordinate of the target cell in 2D space, the NBS mask includes the following cells.

(-1,0), (-1,-1), (0,-1), (1,-1)

Similarly if we consider (0,0,0) as the coordinates of the target cell in 3D space, the NBS mask includes the following cells:

(0,0,-1), (-1,0,-1), (-1,0,0), (-1,0,1), (0,-1,-1), (0,-1,0), (0,-1,1), (-1,-1,-1), (-1,-1,0), (-1,-1,1), (1,-1,-1), (1,-1,0), (1,-1,1)

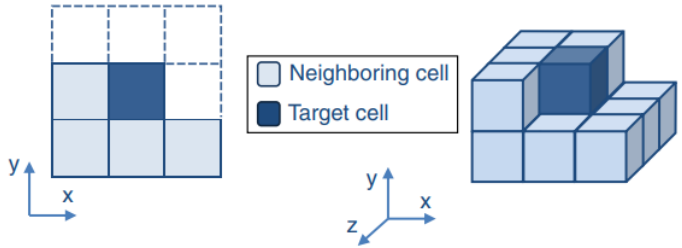

Figure 3.6 The NBS masks in 2D space (a) and 3D space (b)

Algorithm for NBS :

The main algorithm of NBS is divided into 3 different parts:

1. Discretizing the coordinates of spheres (bounding boxes)
2. Loop for creating the Linked List
3. Loop for performing Broad Search and Fine Search in NBS

Time Complexity for the NBS Algorithm:

O(N) where N is the Number of particles.

Space Complexity for the NBS Algorithm:

RAM space consumed is proportional to

(nx x ny x nz)+N which is a cubic function of the simulation domain size.

For a small system with dense packing where the number of particles is comparable to the number of cells, this RAM requirement is not problematic.

However for a large system with loose packing where the number of particles is significantly smaller than the number of cells, the RAM allocation can be very high and much of the RAM space remains useless

NBS-Munjiza Contact Search Algorithm:

The Munjiza Algorithm is similar to the original NBS algorithm except the way particles are mapped onto cells.

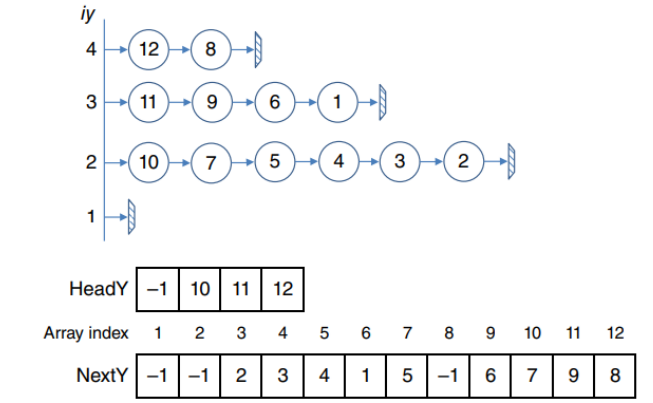

Figure 3.7 Constructed Y-lists of the particulate system of Figure 3.4

Spheres are first mapped onto rows of cells (Y-axis). This operation forms linked lists called Y-lists. It contains ‘ny’ linked lists and each list belongs to a row.

The Y-lists are constructed by 2 One-Dimensional Arrays called ‘ HeadY’ and ‘NextY’ of size ‘ny’ and ‘N’ respectively as shown in the figure alongside.

To this point, spheres are mapped onto rows. To access all spheres in the cell (ix,iy), it is necessary to construct linked list of all spheres in the row iy.

These lists are called X-lists and are constructed according to the integer x-coordinate of sphere in a similar way to what what was described for Y-lists.

NBS Munjiza

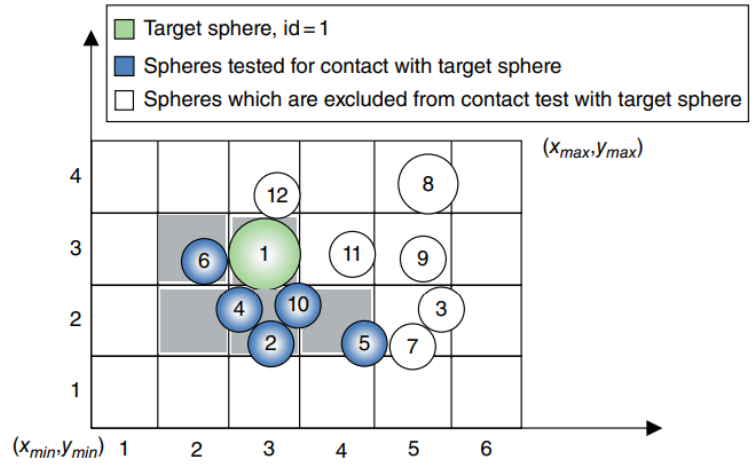

Figure 3.4 Discretized simulation domain and bounding sphere in 2D space

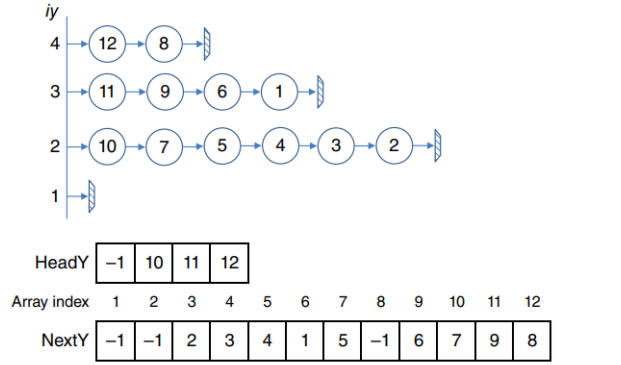

Figure 3.7 Constructed Y-lists of the particulate system of Figure 3.4

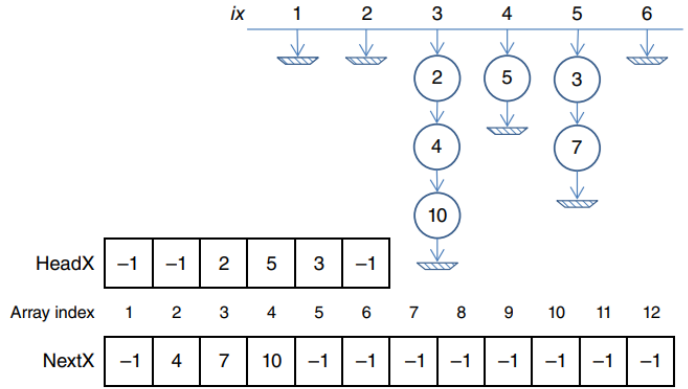

Figure 3.8 X-lists of the second row of the particulate system of Figure 3.4

Hierarchical Contact Search Algorithms:

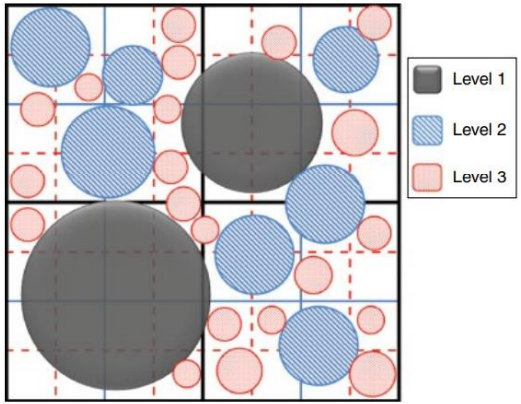

Figure 3.9 A hierarchical contact search with variation of sphere size. Three levels with different cell sizes are considered and spheres are assigned to each level based on their size

1. The basic cell-based algorithms are robust, scalable to large simulations and at the same time easy to implement.
2. The drawback of these algorithms is that they use cells with a fixed size.
3. This is a disadvantage when system contains spheres with a wide size distribution are used. The cell size should be large enough to keep the largest largest sphere inside. However at the same time more small spheres can be found in each cell and the average number of spheres in each row cell increases.
4. Hierarchical Search algorithm can be used in conjunction with NBS or NBS-Munjiza algorithms to generalize them for multisized systems.

Sort and Tree based Contact Search Methods

Sort and Tree based algorithms are a bit advanced Methods which use the Sorting algorithms and the Tree Data Structure. The Time Complexity of these algorithms is usually O(NLogN) while the space complexity is O(N).

Fine Search for Spherical Particles

After the broad search phase, all pair-wise contact candidates (potential contact pairs) are obtained. A contact candidate is a pair of particles that may have an overlap between bounding spheres. The overlap of bounding spheres i and j is simply calculated as:

$$ovrlp = 0.5\left(d_i + d_j\right) - \left|\vec{x}_i - \vec{x}_j\right|$$

Bounding spheres with positive overlap are in contact. For spherical particles, the bounding sphere exactly fits the particle surface. Therefore, no further calculation is required for finding the exact overlap between spherical particles.

High Order Integration Methods

In the framework of the DEM, a large set of coupled ordinary differential equations should be solved over time to obtain the dynamic behavior of the granular flow. The following equations describe translational and rotational motions for a spherical particle (motion with six degrees of freedom in three dimensions):

$$m_i \frac{d\vec{v}_i}{dt} = m_i \frac{d^2\vec{x}_i}{dt^2} = \sum_{j \in CL_i} \vec{f}_{ij}^{\,p-p} + \vec{f}_i^{\,f-p} + \vec{f}_i^{\,ext}$$

$$I_i \frac{d\vec{\omega}_i}{dt} = I_i \frac{d^2\vec{\varphi}_i}{dt^2} = \sum_{j \in CL_i} \left(\vec{M}_{ij}^{\,t} + \vec{M}_{ij}^{\,r} \right)$$

For a system containing N discrete particles in 3D space, 6N coupled ordinary differential equations should be solved. It is evident that a numerical solution is the only choice here.

Table 3.1 Various explicit numerical integration methods used in discrete element simulations

Integration method	Abbreviations	Force evaluations per time step	Extra variables[a]	Accuracy order	
				Position	Velocity
Single-step					
Forward Euler	FE	1	0	1	1
Modified Euler	ME	1	0	2	1
Taylor second order	TY2	1	0	2	1
Taylor third order	TY3	1	1	3	2
Taylor fourth order	TY4	1	2	4	3
Central difference	CD	1	0	2	2
Position Verlet	PV	1	0	2	2
Runge–Kutta fourth order	RK4	4	8	4	4
Multi-step					
Velocity Verlet	VE	1	1	3	2
Adams–Bashforth second order	AB2	1	2	2	2
Adams–Bashforth third order	AB3	1	4	3	3
Adams–Bashforth fourth order	AB4	1	6	4	4
Adams–Bashforth fifth order	AB5	1	8	5	5
Predictor-corrector					
Adams–Moulton third order	AB2AM3	1	5	3	3
Adams–Moulton fourth order	AB3AM4	1	7	4	4
Adams–Moulton fifth order	AB4AM5	1	9	5	5
Gear third order	Gear3	1	4	3	3
Gear fourth order	Gear4	1	5	4	4
Gear fifth order	Gear5	1	6	5	5

[a]Number of vector variables that should be saved in the memory in addition to position, velocity, and acceleration in the current time step.

Evaluation of Integration Methods:

A robust numerical integration method should ideally produce accurate results with the minimum usage of the computational resources.

1. Accuracy:
   High Accuracy is a must for an integration method.

   Time step resolution(alpha) =
   Time of collision/Time step

   For alpha>300 all the integration methods give accurate results.

2. Computational Efficiency:

Complex and accurate integration methods require ample RAM and CPU operations:

I. Computations associated with recalculation of forces acting on a particle. Recalculation of forces includes steps like contact force detection and calculating contact forces between all contacting pairs.

II. Computation associated with performing a predictor step in case of predictor-corrector methods.

III. Computations associated with calculating parameters that appear in the integration expression like bn in TY3.

Parallelization:

1. Data Parallelization
2. Task Parallelization

Data Parallelization involves doing the same task on different sets of data which is shared between processing elements (Processors/Threads). This can be done on Modern Processors which are capable of pipelining on GPUs.

Task Parallelization is doing the same or different data on each processor simultaneously. The processing elements communicate with one another by special function through exchanging data and signals.

It should be noted that parallelization of a real problem (DEM in our case) is not resolved by purely using either of these models.

Many parallel models take advantage of both to achieve maximum speed-up.

Parallelization:

1. Distributed memory Parallelization
1. Shared Memory Parallelization

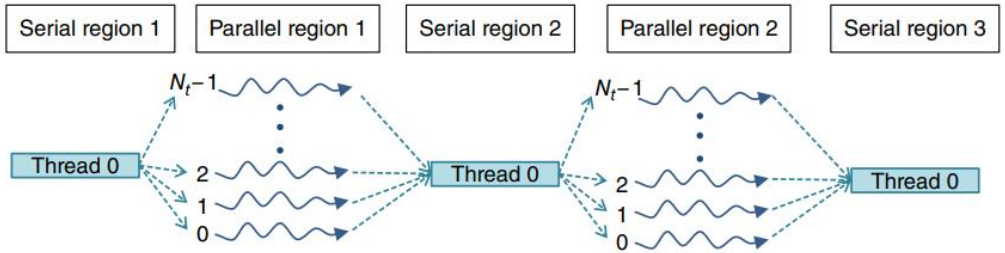

Figure 3.28 The execution paths of serial and parallel regions of the code. Execution proceeds with thread 0 in serial region. Once this thread reaches the parallel region, it creates a team of N_t threads that are executed concurrently and thread 0 becomes the master thread in this team. At the end of parallel region, all private variables are deleted and all threads except the master thread are destroyed. Thread 0 continues the execution of the next serial region. At the end of the parallel region, master thread waits for other threads to finish their execution (implicit synchronization)

Doubts:

1. Spring Stiffness (Section of Chapter 3)
2. Sort based algorithm
3. Tree based algorithm