TU Bergakademie Freiberg

Discrete Element Method - 1. Assignment

Hexagonal Grid and Neighbour Search

Martin A. Haustein

October 15, 2020

1 Introduction

The discrete element method is highly based on a proper initial configuration of particles. Usually, it is possible to set the initial positions via sedimentation, growth algorithms and a predefined packing of particles. Some DEM software packages implement those algorithms by default. So it is possible to create a hexagonal densest packing of particles in a box or differently shaped confinements.

Nevertheless it is of high importance to be able to create such packings via a programming language like *Python* or *C++*. Based on a basic configuration different neighbour search algorithms can be implemented and investigated.

In this first assignment you should create a hexagonal grid of $m \times n$ particles with a radius r. Afterwards, this grid will be filtered and random particles will be removed from the grid. Finally, two different neighbour search algorithms will be implemented and checked for their performance.

2 Creation of the hexagonal grid

Create a hexagonal densest 2D packing of spherical particles (circles) of radius $r = 0.5$. The exact position of the first particle 0 was chosen in this case to be $(x = r; y = 0)$. An example is shown in Fig. 1. The particles were marked with their unique *id* to be able to check the neighbour list afterwards.

3 Filtering the grid

In the next step, the grid should be filtered. As the first step, create a 20×20 grid. Afterwards, remove all particles with a distance $d > 5$ from the point $(10, 10)$. A figure similar to 2 (left) should be created.

Finally the grid obtained in this way should be filted to remove random particles with a propability of p. For $p = 0.4$ an example picture is shown in Fig. 2 (right). Thus, for $p = 0.0$ no particle will be removed, while for $p = 1.0$ all particles are removed.

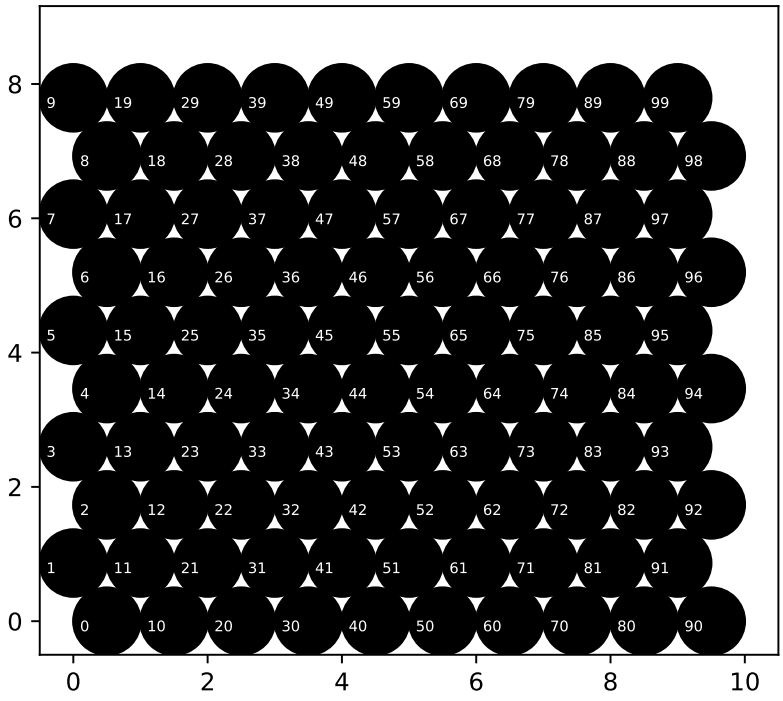


Figure 1: Example of a 10×10 grid of particles with radius of $r = 0.5$.

4 Neighbour search

In DEM simulations the search for neighbouring particles is one of the most time consuming steps. Hence, it is of high importance to take care of this bottle-neck. There are different possibilities to find neighbouring particles. The most simple but also very time consuming algorithm is a brute force approach to check for every particle pair, if their distance $d \leq 2r$. If the distance is smaller or equal to 2 times the particle radius, the particles are overlapping or touching each other.

This situation in shown in Fig. 3 (left). The green particle checks the distance to every other particle in the whole field. If the distance criterion is fulfilled, the particles are assigned as neighbours. Especially for large numbers of particles in dense systems, such an approach is not suitable.

An alternative approach is based on dividing the simulation zone initially into grid cells. Each particle is then assigned to a cell. Finally, it is sufficient to examine the distance

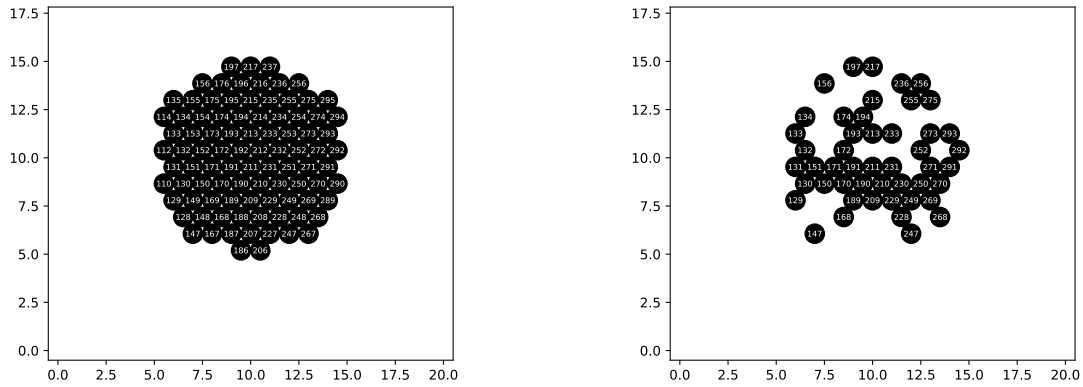


Figure 2: Filtered 20×20 grid (left), and with particles removed with a propability of $p = 0.4$ (right).

criterion for the particles inside the cell of the particle and the surrounding cells. An example of this is shown in Fig. 3 (right).

The particle to be examined (green) is assigned to a cell according to the coordinates of the center of mass of the particle (+). This cell is shown here in red. Then the distance between the green particle and all particles in the red and yellow cells must be checked. This eliminates the need to calculate the distance to the particles that are far away. Especially in larger systems there is an immense time saving.

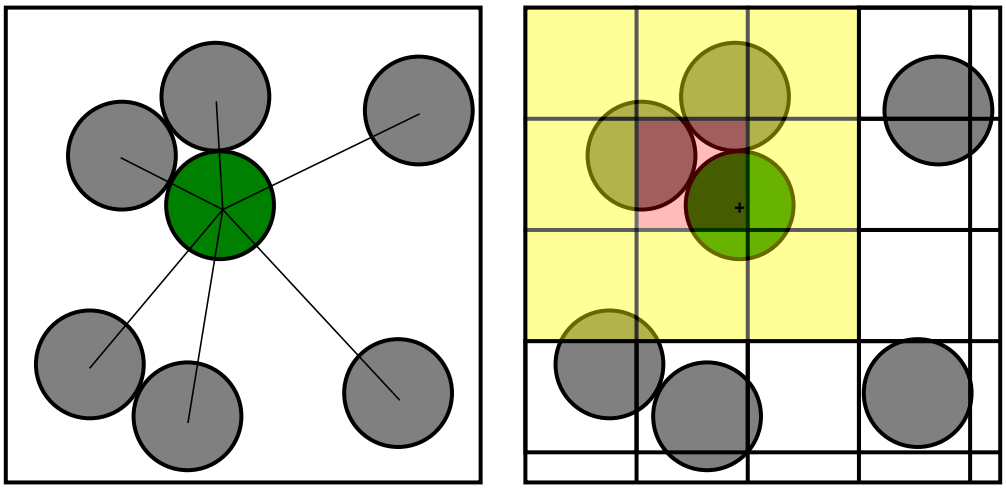
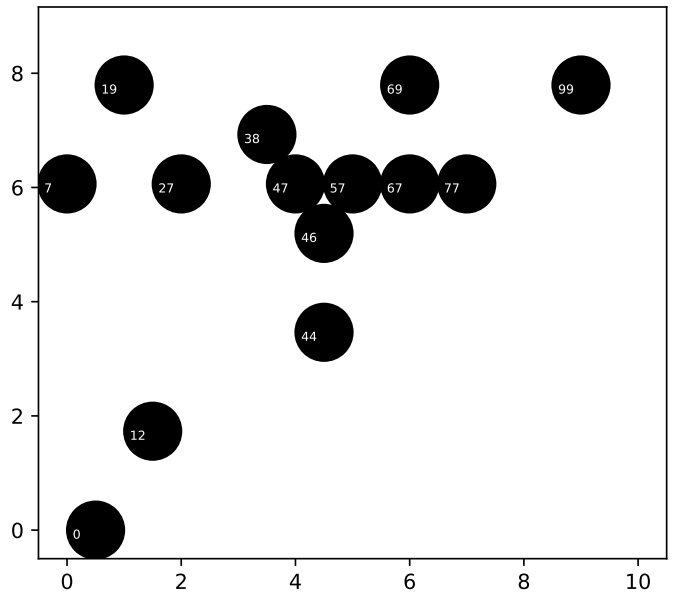


Figure 3: Neighbour search via brute force (left) or on a grid based method (right). The particle to be investigated is colorized in green. The (+) on the right marks the center of mass of the green particle.

Use one of the filtered grids from above to check your implementation of the neighbour search. A simple example is given in Figure/Table 4.



Particle	Brute Force	Grid
0		
7		
12		
19		
27		
38	47	47
44		
46	47, 57	47, 57
47	38, 46, 57	38, 46, 57
57	46, 47, 67	46, 47, 67
67	57, 77	57, 77
69		
77	67	67
99		

Figure 4: Neighbour search results

5 Benchmark of the neighbour search algorithms

To investigate the implementation of the brute force and the grid based neighbour searches, the times for the execution of the code will be studied. For this purpose run the program for grid creation and neighbour search for $n \times n$ grid with $n \in \{2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. Filter each grid with a probability of $p \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. Thus, the program will run for 132 different configurations. It is useful to automatize the process using a simple script or implement the loops directly in the code.

Finally, print the number of particles in the system N and the times needed for the neighbour searches via brute force and the box bases algorithm to a file. Provide a plot $t(N)$ for both algorithms in double logarithmic scale.

6 Tasks

Plase report the results to the questions below until the date given in the description on opal. Please use full sentences and keep the report short but complete. Include all figures and results including a proper figure description.

- Solve the following tasks using C or C++ or Python or Octave or Perl
- Create a 10×10 hexagonal grid similar to Fig. 1. A figure of this raw grid should be included in your report
- Filter a 20×20 grid to include no particle with a distance $d > 5$ from (10,10) similar to Fig. 2. Include graphics of the grid filtered with a propability of particle removal of $p \in \{0.0, 0.3, 0.5, 0.8, 1.0\}$.
- Implement the brute force neighbour search.
- Implement the grid based neighbour search.
- Provide an example like Figure/Table 4 to show the correctness of the implementation.
- Plot for both implementations time t vs number of particles in the system N
- Plot t vs $1 - p$ for $N \in \{2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ for the brute force methode
- Plot t vs $1 - p$ for $N \in \{2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$ for the grid based method
- Explain the scaling of the time for large and very small particle numbers N. In which cases a box-based algorithm should be prefered over the brute force method and v.v.
- Provide the code including the scripts for the figure creation (gnuplot). Do not include compiled code.