REPORT

UNDERGRADUATE PROJECT - II

Department of Chemical Engineering

SEMESTER - VI

DEM Simulation & Package Generation Algorithm

Advisor- Dr. Anurag Tripathi
Assistant Professor
Department of Chemical Engineering IIT Kanpur

Mentor- Mr. Nikhil Meena
M.Tech
Department of Chemical Engineering IIT Kanpur

- # Musen

It is a GPU accelerated open-source DEM simulation software.

Installation

Installed DEM on machine 205 by following the instructions given in the readme of the musen GitHub repository. But, I need to downgrade the version of GCC c++ compiler from 10 to 9.

Demo Simulation

To learn the basics of DEM and musen software, I tried to run a simple simulation in musen in which there is a plate and a spherical particle is allowed to free-fall under gravity and elastically/non-elastically collide with the wall and plotted velocity vs time and velocity vs distance graphs in musen.

The aim was also to understand the various features of the musen along with to understand the files in which musen saves the simulation output.

Understanding the code of Musen

In the initial weeks of the project, we tried to understand the flow of the musen software. How it takes input parameters and takes an overview of the code of the implementation of different models for contact forces, contact detection, package generation, dynamic generation etc.

We tried to understand the code in two ways. First by using a debugger and second by finding the main file of the code and then following the flow of the file. But, in both cases, the code diverges into various parts, and it becomes difficult to understand the proper working of the code.

We also have generated Doxygen documentation of the musen code. It has given us various insights into different classes, methods, and inheritance.

- # Learning GPU Computing

Completed three weeks out of 7 weeks of GPU computing course by Nvidia and also solved the assignments.

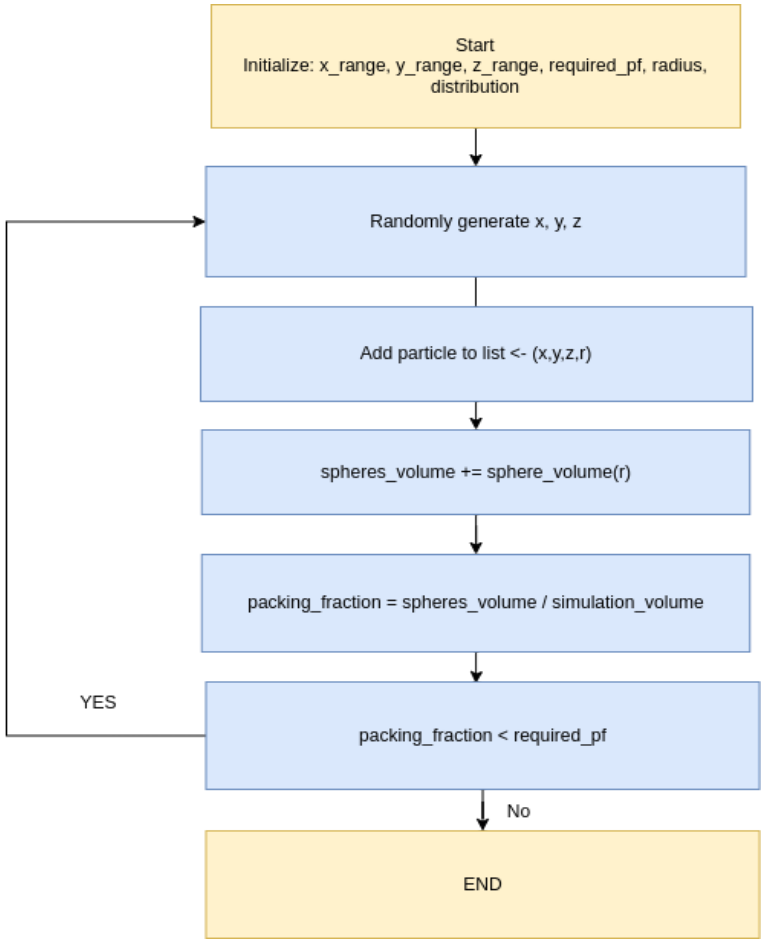
- # Learning DEM Algorithms

Completed chapters 2nd and 3rd of the book Coupled CFD-DEM Modeling. Chapter 2nd is about DEM formulation, and chapter 3rd is about various efficient DEM algorithms for contact detection, wall implementation, and parallel computing in DEM.

- # Package Generation:

In this part, I have tried to develop a robust and time-efficient algorithm to fill a cuboidal simulation volume of any dimension with particles of a given radius without using any physics engine. The algorithm is based on randomly inserting many particles by using a uniform distribution along all three dimensions and then deleting particles using some heuristics.

Insertion



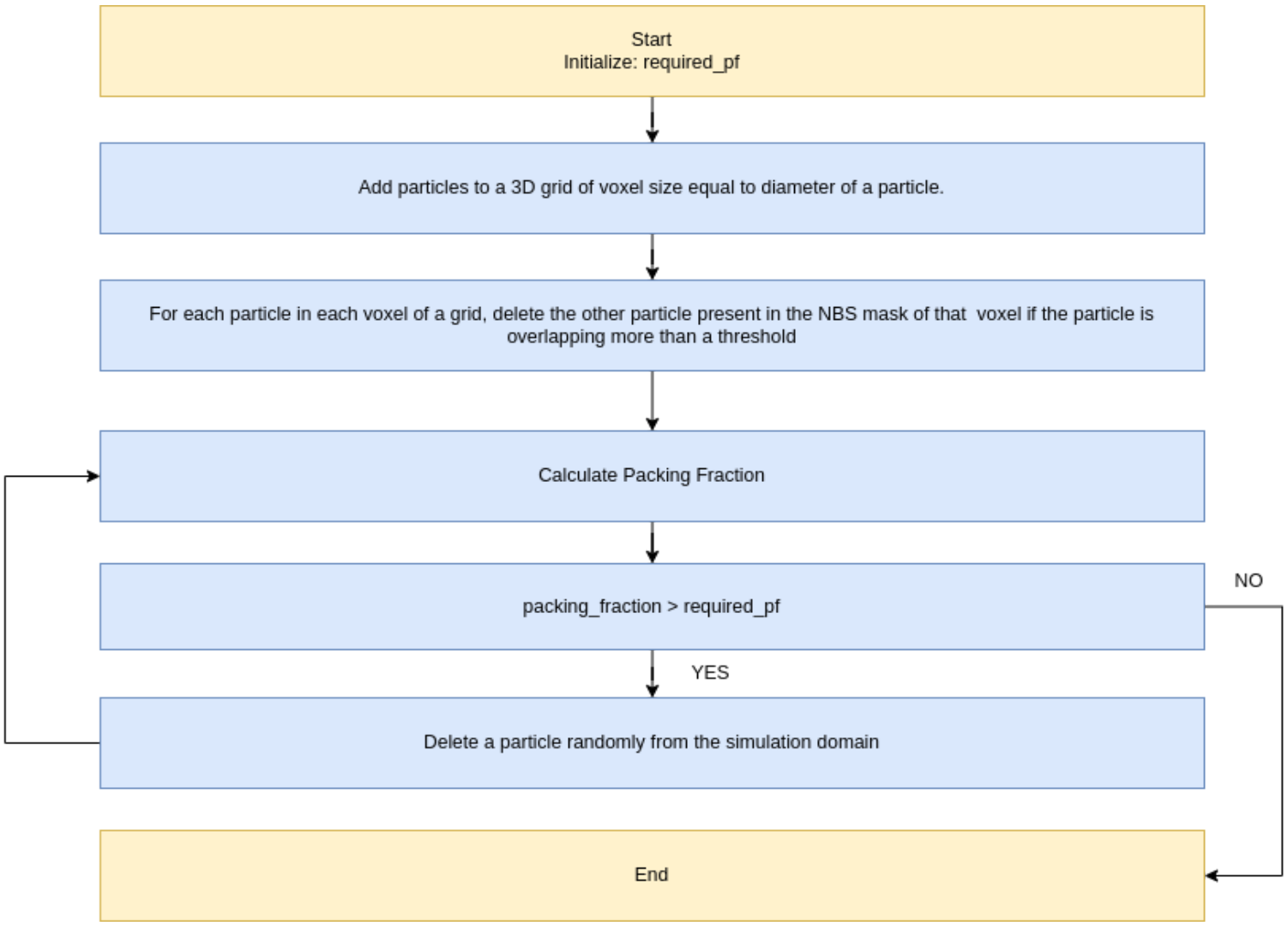
It is the first part of the package generation algorithm. This algorithm uniformly fills the simulation domain by generating random particles in the simulation domain.

Deletion:

Deletion is the second part of the package generation algorithm. It is the most difficult part of the package generation algorithm. The question is how to delete particles efficiently to get the required packing fraction? If you delete particles most optimally, you will get a packing fraction as high as HCC, 0.72. But, when the particles are polydisperse, one can even achieve higher packing fractions.
I have proposed two different algorithms for the deletion part. The first algorithm is faster than the second algorithm, but the second algorithm usually gives a higher packing fraction.

Algorithm-1:

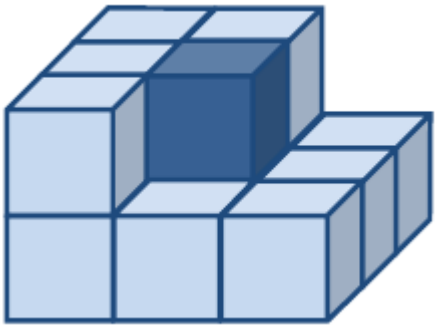
In this algorithm, first, we make a grid in which each dimension of a voxel is of the size of the diameter of the particle. Then, we find the contacts of each particle using *the NBS Contact Search Algorithm.* We delete all the particles overlapping with the current particle more than a threshold value. All such particles can only be present in the NBS mask of the current particle. After deleting all overlapping particles we move to next particle and repeat the same process.



The main computational part of this algorithm is the NBS Contact Search algorithm to find all overlapping particles with a particular particle. Since the **time complexity** of the NBS Contact search algorithm is of the order of the number of particles.

So, the **time complexity** of this algorithm is of the order of the number of particles.

The **space complexity** of this algorithm is also of the order of the number of particles.



NBS MASK IN 3D

Algorithm-2:

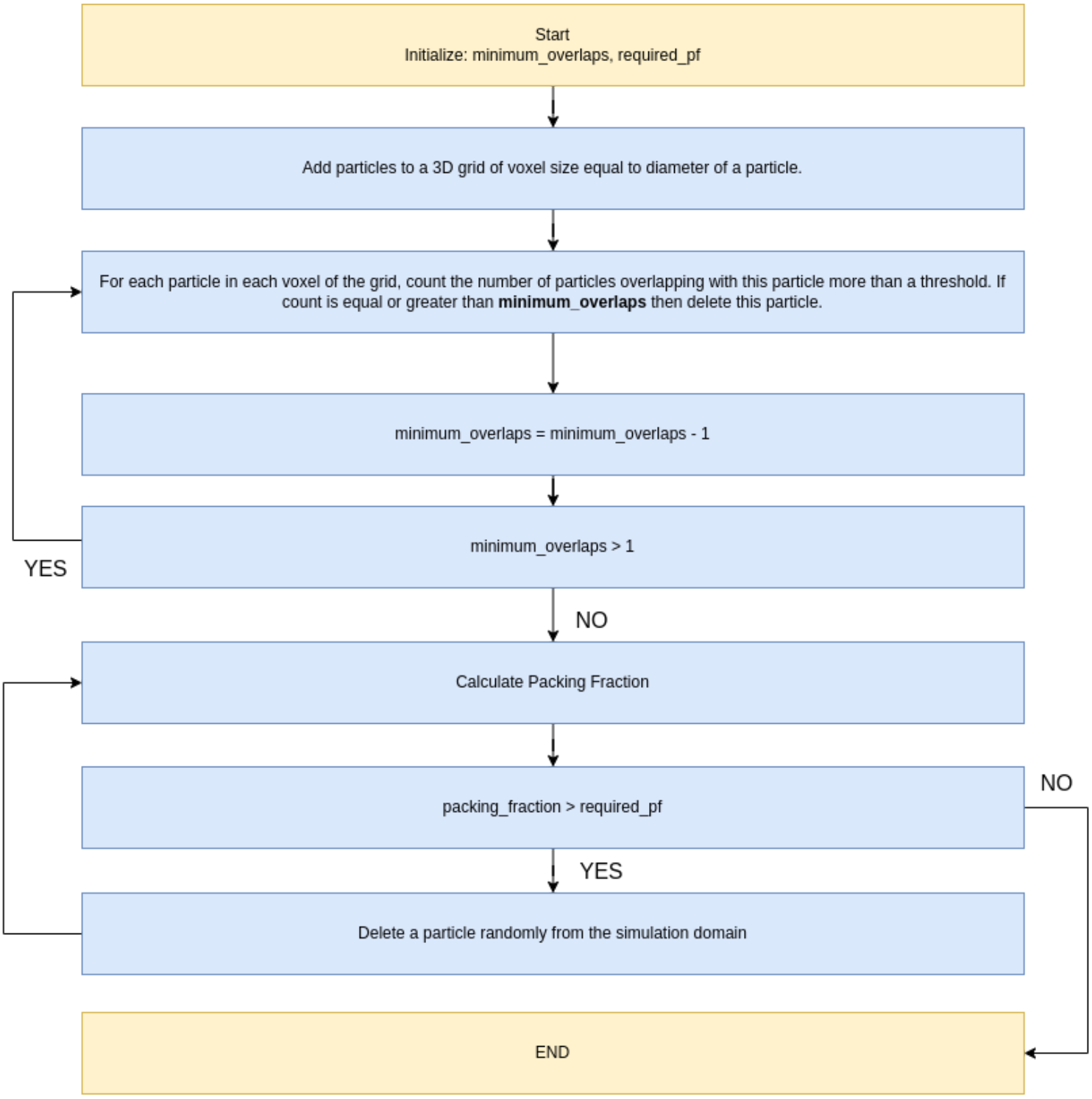
This algorithm depends on a parameter minimum_overlaps. Its value decides that at any given iteration of the algorithm, any particle present in the simulation domain at max can overlap with how many other particles.

In this algorithm, for each particle in each voxel, we find the total number of particles that are overlapping with the current particle. We delete all such particles which have total number overlaps greater than or equal to the minimum_overlaps parameter. Then in the next iteration of this algorithm, we decrease the value of this parameter by one and do the above step again. We keep on doing this step until the value of minimum_overlaps is 1. After this, we check if the required packing fraction is more than the current packing fraction. If so, we randomly delete particles until we achieve the required packing fraction. Otherwise, we do nothing.

In this algorithm, since we need to find all overlapping particles of a given particle so instead of using the NBS mask in the NBS Contact Search Algorithm, we use a cubical mask which is subdivided into 27 voxels, and the center voxel of this mask overlaps with the voxel in which given particle is present.

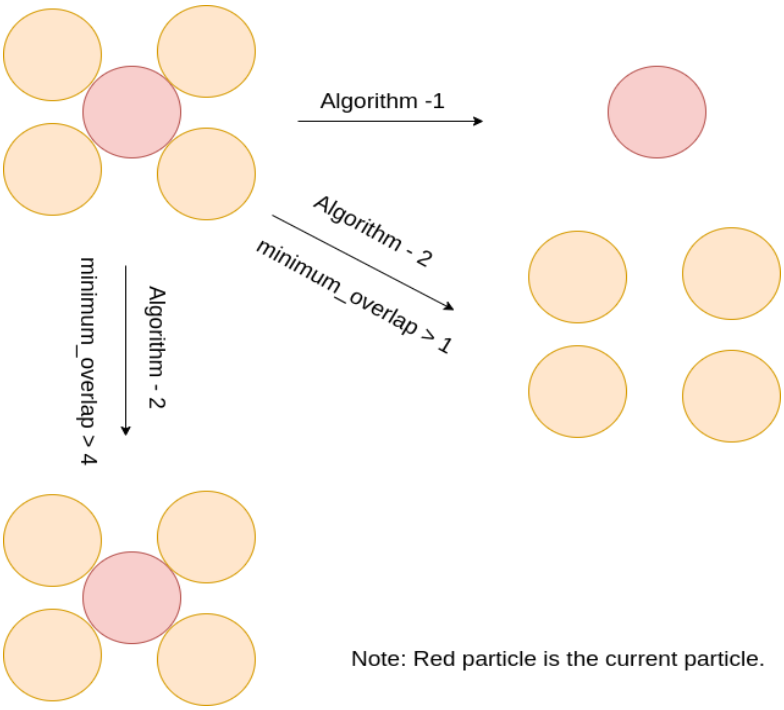
The **time complexity** of this algorithm is the order of the product of the minimum_overlaps parameter and the number of particles.

The **space complexity** of this algorithm is the order of the number of particles.


Start
Initialize: minimum_overlaps, required_pf
Add particles to a 3D grid of voxel size equal to diameter of a particle.
For each particle in each voxel of the grid, count the number of particles overlapping with this particle more than a threshold. If count is equal or greater than **minimum_overlaps** then delete this particle.
minimum_overlaps = minimum_overlaps - 1
minimum_overlaps > 1
YES
NO
Calculate Packing Fraction
packing_fraction > required_pf
NO
YES
Delete a particle randomly from the simulation domain
END

Analysis

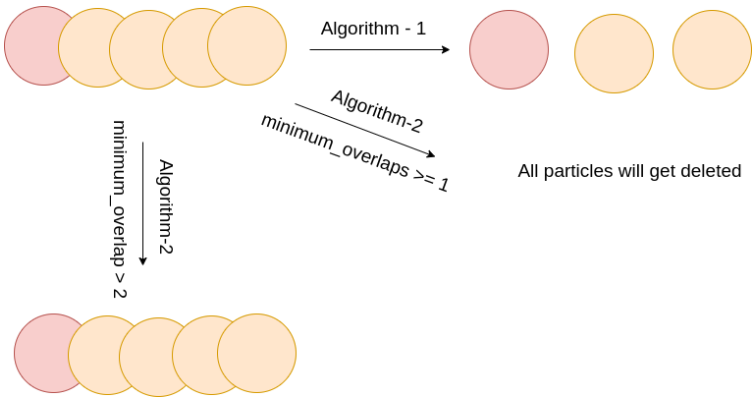
Example - 1



As we can see, if we have used algorithm one in this case, we must have left with only one particle at the end, but if we have used the second one, we have left with four particles.

Note: Red particle is the current particle.

Example - 2



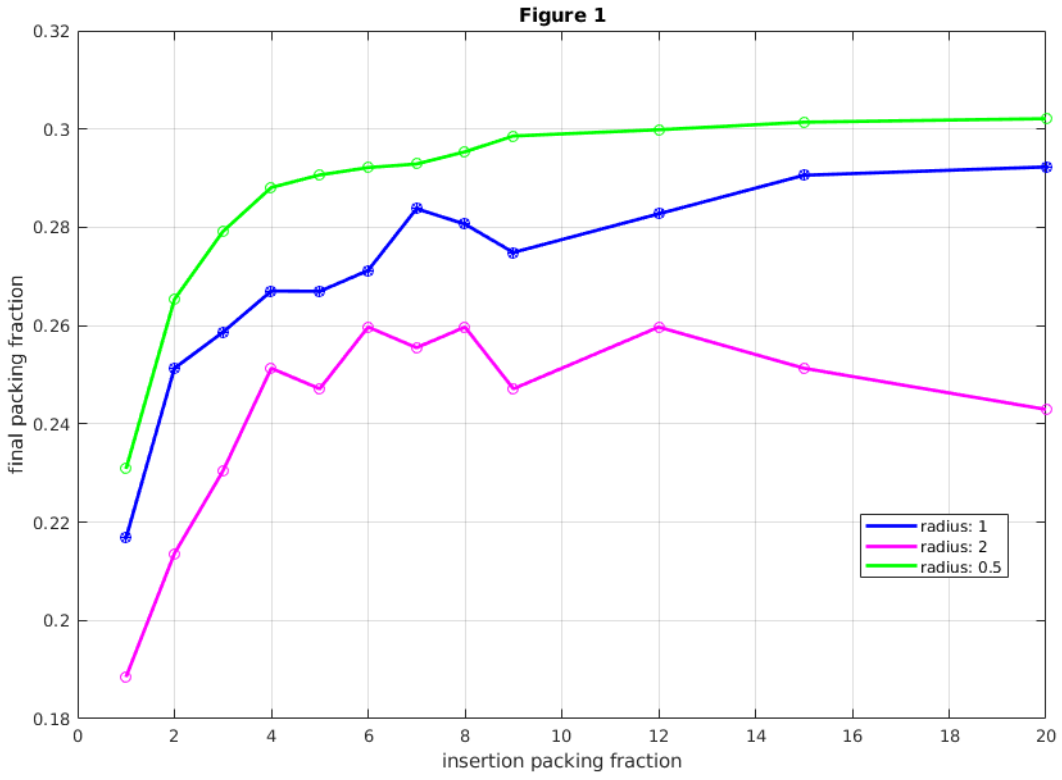
If we have chosen the minimum_overlaps value to be 1, then we have left with no particle. If we choose a higher value of the minimum_overlaps parameter, we get the same configuration at the end in both algorithms.

Also, intuitively thinking algorithm 2 makes more sense because first we should delete those particles which are overlapping with many other particles and then do the same thing on the remaining particles until we have non-overlapping particles in the simulation domain.

Observations

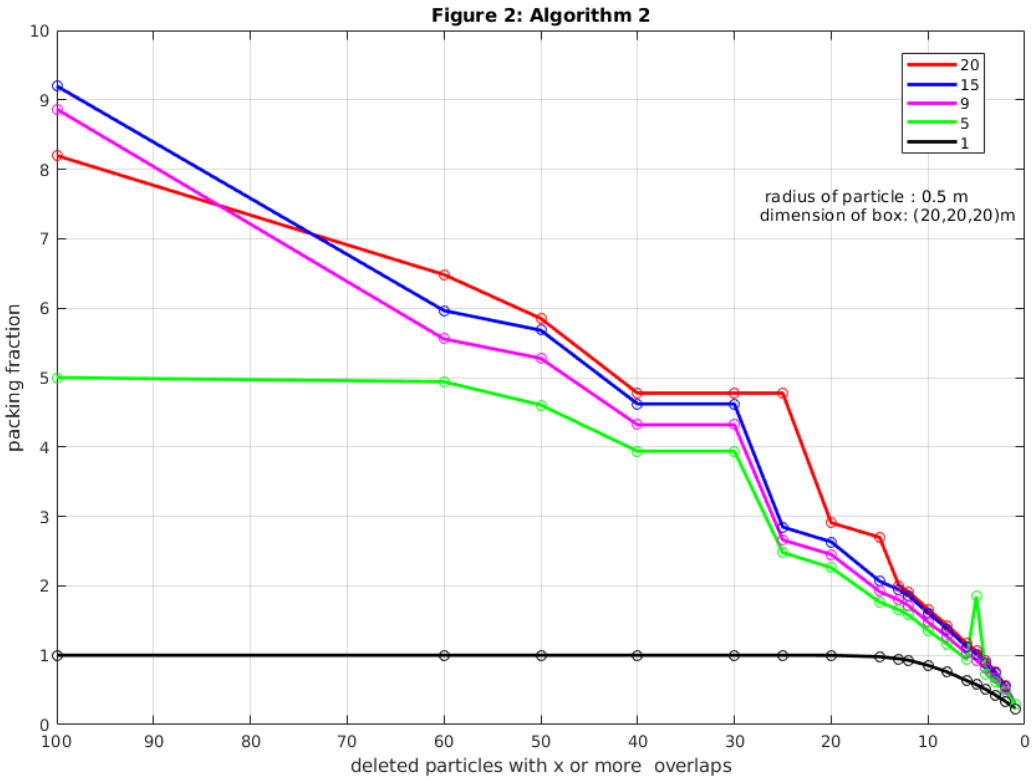
- Simulation domain size (20,20,20) m.
- Algorithm 2 is used for deletion.
- Insertion packing fraction refers to the packing fraction of particles inserted by the insertion algorithm.

1) The final packing fraction first increases with the insertion packing fraction and then saturates because initially, as we insert more and more particles by insertion algorithm, the deletion algorithm gets more choices to find any one of the optimal ways to get a high final packing fraction, but as we increase insertion packing fraction before a limit, a huge number of particles start overlapping more than 90%, and all such particles get removed in the first iteration of deletion algorithm.

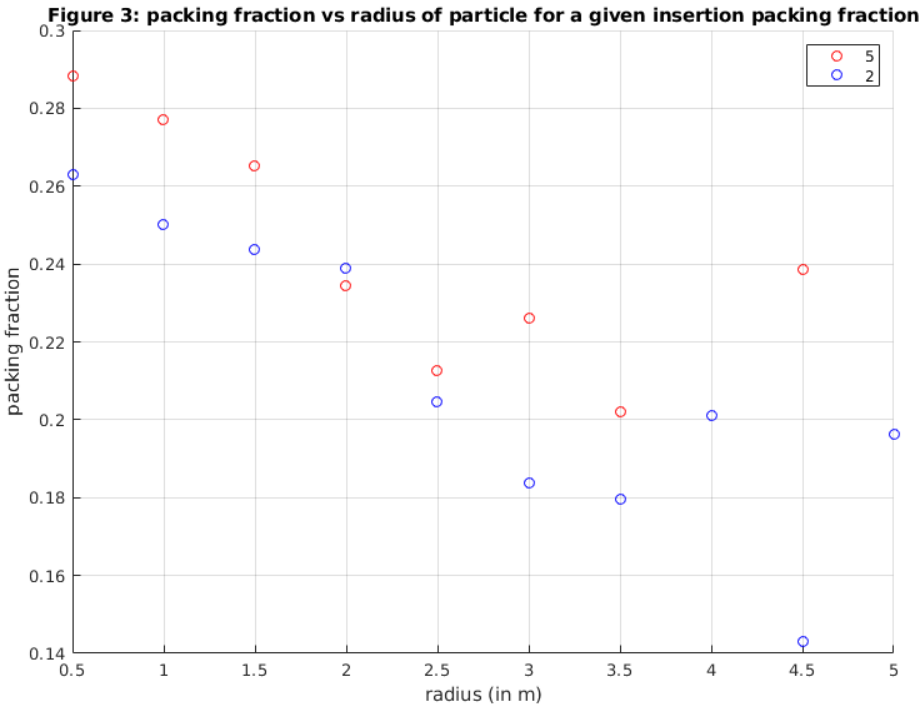


2) This figure shows the variation of packing fraction when the particles which are overlapping with x or more than x number of other particles are deleted.

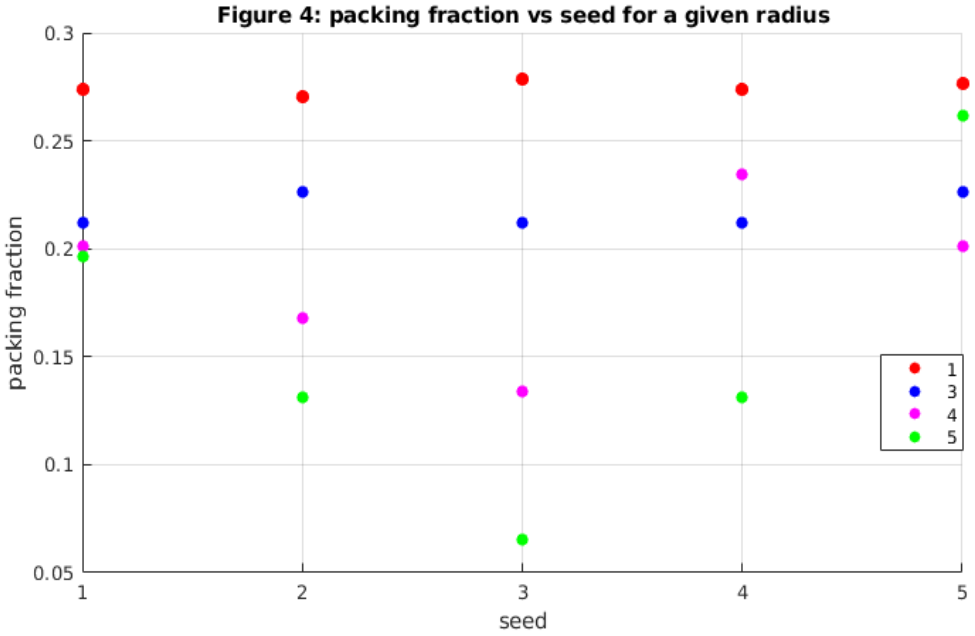
Note: The legend in this figure is the insertion packing fraction.



3) As the radius of the particle increases in a fixed simulation domain, the packing fraction decreases for a fixed insertion packing fraction. As the radius of particles increases, fewer particles are inserted in the simulation domain, and then the choice of order of deletion of particles to get a high packing fraction becomes much more difficult.



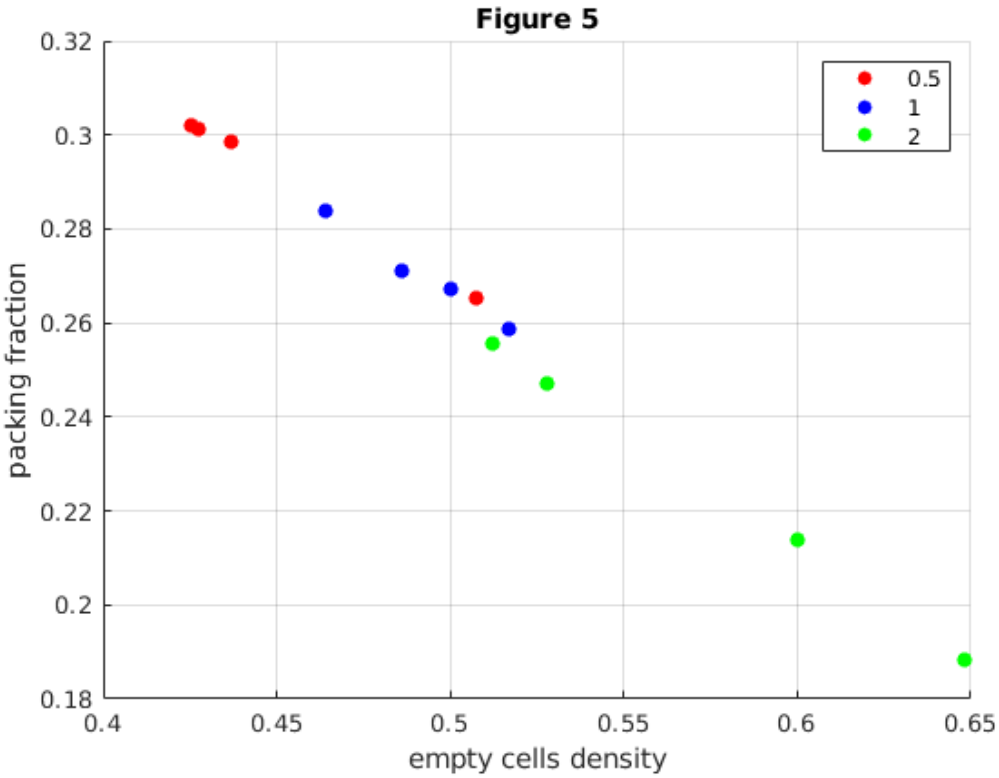
4) We can see that if the radius of particles is comparatively smaller than the simulation domain, we get nearly the same packing fraction in every seed on the randomized algorithm. But, if the diameter is nearly equal to half of the simulation domain length, then in every seed, there is quite a large variation in the final packing fraction achieved. The reason for the same is that when the radius increases, then the number of particles decreases. So, the order of deletion of particles matters much more here because the wrong order of deletion can result in the configuration in which very few non-overlapping particles are left inside the simulation domain.



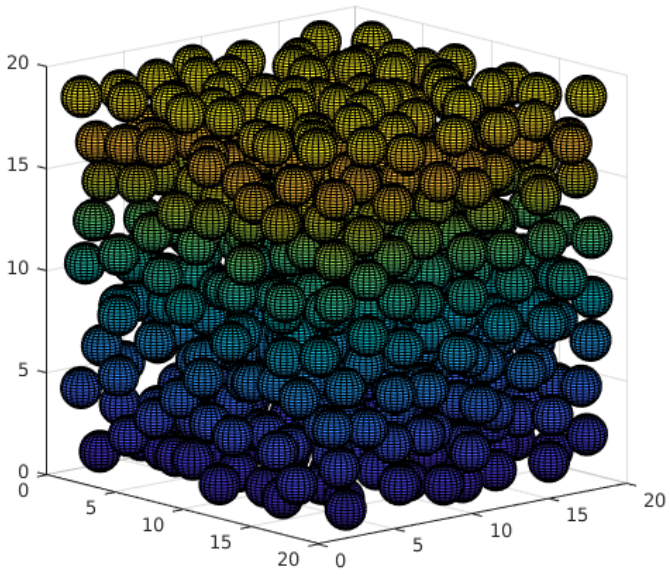
5) Packing fraction linearly decreases as the empty cell density (empty cell density refers to the number of empty cells / the total number of cells in the grid) increases in the simulation domain irrespective of the radius of particles.

Another inference that can be made from the figure is that, more or less, only one particle is present in each voxel other than the voxels which are completely empty after the algorithm terminates. This is one of the reasons why the packing fraction is so less. One may think to fill one particle in each voxel rather than using this algorithm and can achieve a 0.52 packing fraction. But, this algorithm will fail in the case of polydisperse particles but our algorithm also works when the particles are polydisperse.

One may think of refilling the empty voxels, but it is not trivial as we can not just insert the particle at the center of each empty voxel.

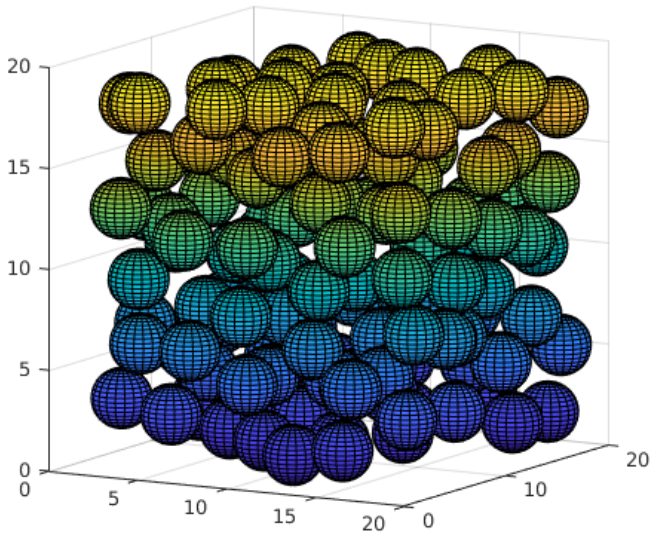


Result



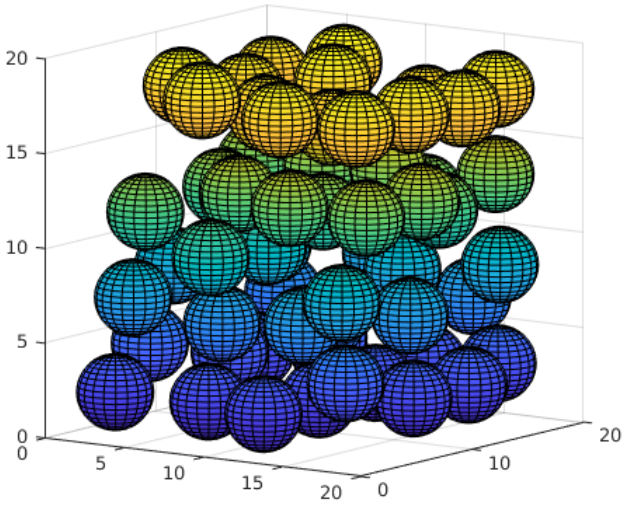
radius - 1 m

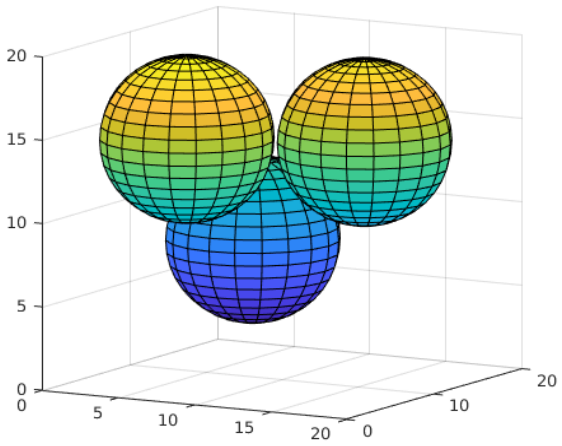
packing fraction - 0.27671



radius - 1.5 m

packing fraction - 0.258003





radius - 2 m

packing fraction - 0.238761

radius - 5 m

packing fraction - 0.19625

In all four cases above, the simulation domain is of the size (20, 20, 20) m.

The **final packing fraction** that can be achieved by this algorithm is in the range of **0.27 to 0.30**.

Future Work

- To increase the final packing fraction achieved by this algorithm. To achieve higher packing fractions, we need to try many different patterns of deleting particles and then try to come up with some reasoning that can explain the behavior, and then using all this; we can make a more sophisticated overlapping particle deletion algorithm which can achieve packing fraction of around 0.5 - 0.6.
- Testing of this algorithm on polydisperse particles.
- Testing of this algorithm in which particles are monodisperse, but the size of particles is much smaller than the dimension of the simulation domain(~ order of 0.001).
- Implement dropping and particle expansion algorithms and check the range of packing fraction achieved by both algorithms.
- A much more rigorous analysis of the current deletion algorithm should be done to achieve higher packing fractions.