MesoMake

Russell, Truman M.

November 10, 2023

1 Introduction

The purpose of the mesoMake program is to drop a number of circles or spheres into a square or cube domain, and iterativly solving to remove overlap between objects. A program like this is useful for setting up meso-scale finite element simulations. This program has been specificaly set up for use with CTH, an Eularian hydrocode produced by Sandia National Laboratory. One application might be the simulation of a projectile impacting sand. While sand grains might not be perfectly spherical, it might be acceptale to simulte them as spheres.

The basic principal of the mesoMake program is the determine the distance between two spheres, determine if they are overlapping. If they are overlapping, the particles are moved such that the edges are touching. This is a simple problem for a few particles, but complexities arise when resolving a large number of particles.

MesoMake is not the first of its kind; but it does have a number of convienient features. MesoMake is written in python, so it is easy for users to modify the program and add their own features. MesoMake allows for several species of particle, which can have different radii and volume fractions. A single species of particle can even have varying radii according to any kind of sampling distribution. The user can use the Sobol sampling method to create an initial sample that has been sampled to reduce the overlap from the start. This is very helpful in creating evenly spaced particles.

Versions of mesoMake have been setup to allow for zoning. Zoning allows the user to subdivide the domain into several subdomains. This can darasticaly reduce the time to solving time, by not requiring the solver to compute distances between particles not in the same or neighboring subdomains. Zoning is most effective for thousands of particles, and can reduce computational time by up to 75%.

2 Removing overlap

The whole purpose of the mesoMake program is to remove overlap from a set of circles or spheres. The heuristic for the system is total the amount of radial overlap. Once the heuristic becomes zero, the problem is solved. For a set of two circles this is a simple problem.

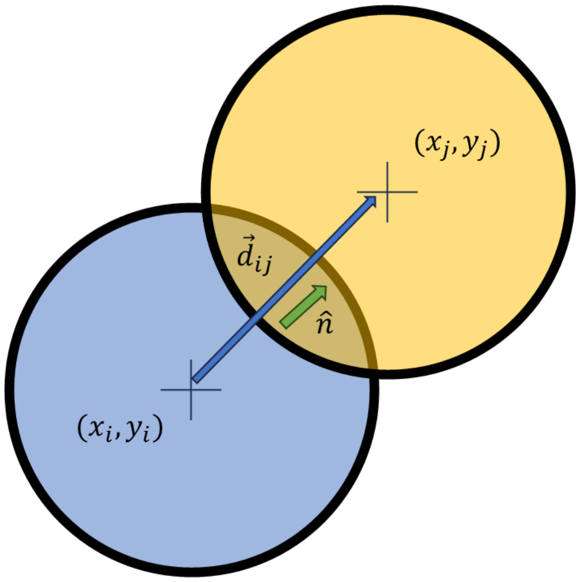

Figure 1: Two Overlapping Circles. Vector Deffinitions

In Figure 1, two circles are overlapping by some ammount. There are an infinite number of solutions to this problem, but the following method will be used to solve the problem in the most reasonable method. The vector $\vec{d_{ij}}$ represents the center-to-center vector from circle i to circle j. This vector has length d and direction $\hat{n}$. The vector d is calculated as:

$$\vec{d_{ij}} = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \tag{1}$$

The magnitude of the vector is the euclidean norm of the vector, and the direction is the vector divided by its magnitude. If there is no overlap, the center-to-center distance should be at least $R = r_i + r_j$. The heuristic can fe formulated such that: $err = R - d$. It can be seen that $\frac{\partial err}{\partial d} = -1$ and $\frac{\partial err}{\partial \hat{n}} = 0$. According to Newton's method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

It can be seen that the direction of the vector has no effect on the error value, so it should not change (even though the zero gradient causes an error in the algorithm). On the other hand, we find that $d_{n+1} = d_n + err$. This results in the solution $d = R$, which means that the overlap has been removed. But there are stil multiple solutions. However if the location of circle i is fixed, there is finaly a unique solution:

$$\vec{x}_j^{n+1} = \vec{x}_j^n + err * \hat{n} \tag{3}$$

But if circle i is not fixed, each circle can be moved such that the center of mass of the particles is the same from step n to step $n+1$.

$$\vec{x}_j^{n+1} = \vec{x}_j^n + \frac{r_i^2}{r_i^2 + r_j^2} err * \hat{n} \tag{4}$$

$$\vec{x}_i^{n+1} = \vec{x}_i^n - \frac{r_j^2}{r_i^2 + r_j^2} err * \hat{n} \tag{5}$$

This will remove overlap, while mantaining the center of mass of the system. To generalize this to three-dimensions, each r^2 will become an r^3, and the vectors work out the same way. The vector math is easy to generalize for circular particles, since the heuristic is independant of the rotation of the particle. For a system of polygons, the vector math may become complex and include cross products.
This method assumes that the two particles have the same density–which is fairly trivial at this point. If considering particles of varying densities is important, the densities can be multiplied in. In systems with many points of varying sizes, mantaining the center of area is critical for solver stability.

For a set of two particles, all the overlap can be imediatly removed in one step, but for a large number of particles, overlap needs to be removed iteratively

2.1 Systems of Many Particles

A system of many particles can be solved the same way, by going through each particle one-by-one, and determining what particles it overlaps with, and moving accordingly.

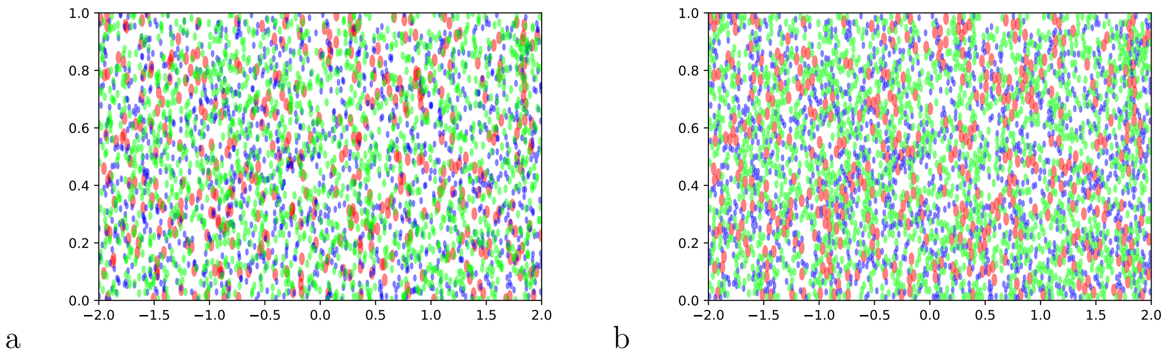

Figure 2: Example system of points of varying radius. Before (a) and after 10 steps (b).

3 Zoning

4 Sobol Sampling

5 MesoMake Program

The mesoMake program is written in python and uses an object-oriented approach to solving these complex problems. The setup is heirarchical, wherein the primary object the main method interacts with is the domain.

The domain has several properties and methods, and does most of the problem solving, as it contains the system of points and is responsible for resolving the overlaps between points. In zoned problems, the domain creats the zones and is responsible for the eularian update that assigns points to zones. The domain is responsible for initializing a randomly generated set of points, as well as plotting the system along the way.

The domain contains the system. The system is a simple class that contains the collection of points, and has a few methods that delegate various tasks to the points– plotting for example.

The points are contained within the system, as an array. Each point simply contains it's location, radius, identity, and plotting color.

In zoned problems, the domain holds an ordered array of subdomains. The subdomain hold the list of the identities points inside of the subdomain, which function as pointers to the array of points held in the system. In problems with zoning.

5.1 Solver

In non-zoning problems, the solver goes to each point and one-by-one goes to each point, finds the center-to-center distance and determines if there is overlap. If there is the points are moved according to Equation 4 and 5.

5.2 Input File Formating & Defaults

The input file is a python script that the main program imports. The main imports the input after the defaults, so any defaults can be overwritten by the user. The user must specify a few things in the input deck, and there are several optional inputs as well ,which all have defaults set in the default deck. The user must specify the limits of the domain, as well as the size, identity, and volue fraction of each species. The user can specify several additional optional keywords, including a color for each species (mesoMake will randomly generate colors if not specified). The user may speciy if mesoMake should plot images by setting *plotting* to True or False. The user may determin if mesoMake should use the sobel sampling sampling method by setting the *sobol* keyword to True or False. For zoned problems, the user can change the size of the zones with the *ratio* keyword. There needs to be atleast 2 zones, but the ratio must be greater than one. However, if there are only two zones per direction, the program will work the same as an unzoned problem.

5.3 Output File Formatting

The output from mesoMake is a csv file including the locations of the particles, as well as radii and identities. The name of the file can be specified by the user, but defaults to: $meso.csv$. The program writes the output file only at the conclusion, but the user may set $resave = True$ to make the program save on every iteration. This can be helpful if the user is impatient.

The output file is written such that the user can easily read it into a pandas dataframe: $data = pd.read_csv("meso.csv")$. Then the position and radius of particle i can be accessed by: $x = data.x[i]; y = data.y[i]; r = data.r[i]$.