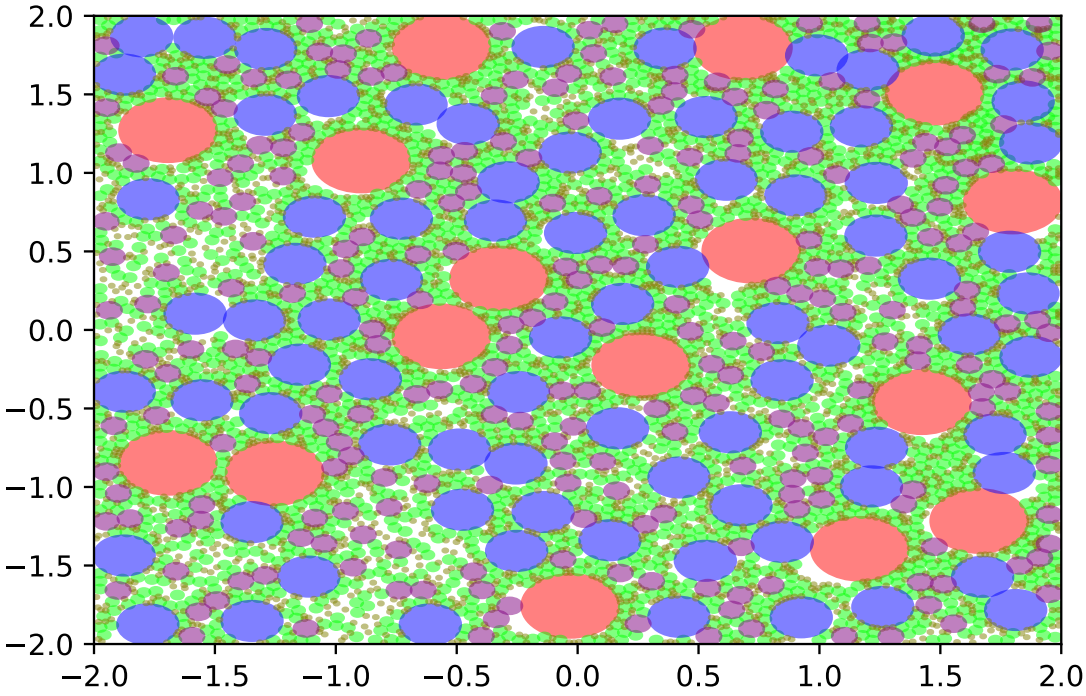
MesoMake

Truman M. Russell


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Contents

List of Figures

List of Tables

Chapter 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%.

Chapter 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.

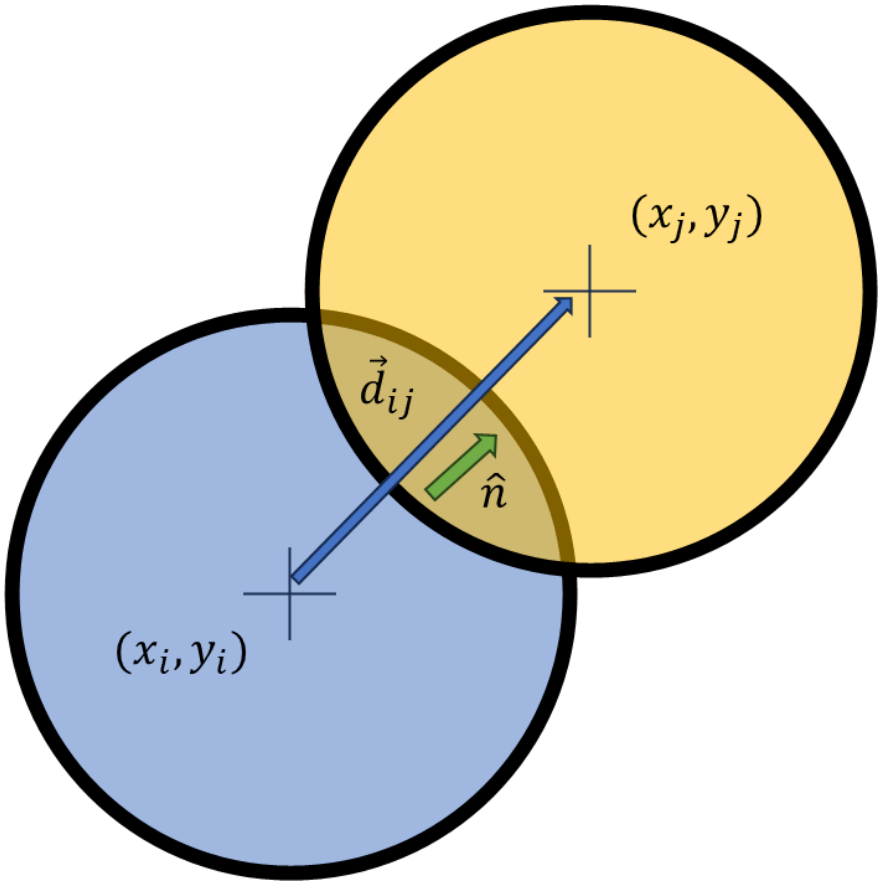


Figure 2.1: Two Overlapping Circles. Vector Deffinitions

In Figure 2.1, 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:

$$\vec{d_{ij}} = \begin{bmatrix} x_j - x_i \\ y_j - y_i \end{bmatrix} \tag{2.1}$$

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:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.2}$$

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:

$$\vec{x}_j^{n+1} = \vec{x}_j^n + err * \hat{n} \tag{2.3}$$

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 $n + 1$.

$$\vec{x}_j^{n+1} = \vec{x}_j^n + \frac{r_i^2}{r_i^2 + r_j^2} err * \hat{n} \tag{2.4}$$

$$\vec{x}_i^{n+1} = \vec{x}_i^n - \frac{r_j^2}{r_i^2 + r_j^2} err * \hat{n} \tag{2.5}$$

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. A

system of two particles is easy to resolve because there can only one overlap, and it can always be resolved in one step. A system of thousands of particles is significantly more difficult as the number the problem becomes a nonlinear problem due to the nonlinear nature of contact, where the over lap is zero before contact, and linear after contact. This means that the problem can no longer be solved in a single step.

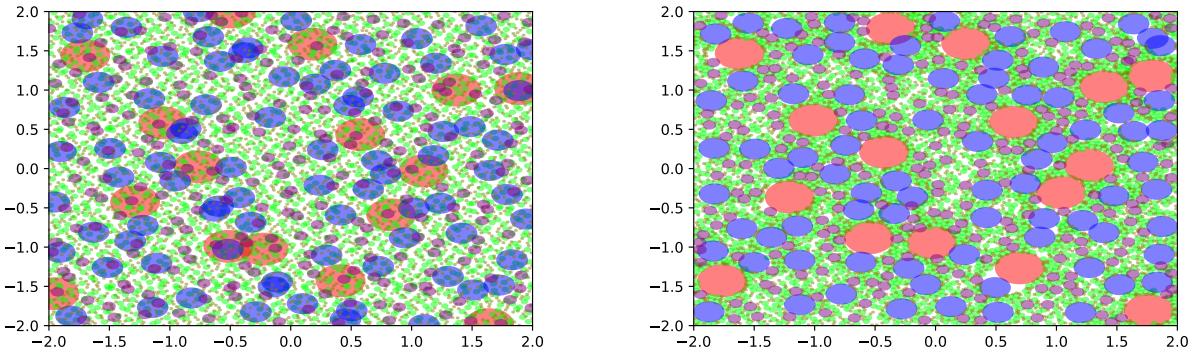


Figure 2.2: Example system of points of varying radius. Before (a) and after 10 steps (b).

The number of steps required varies, and freqnently, dense systems can not be completly resolved completly. But, even solving an rare system efficiently is difficult. Dense and rare systems are visualy very different, but the difference can be difficult to describe numericaly. It is certainly true that if less than 30% of the problem volume is filled up that the problem is under dense, and if greater than 60% if full, that the problem is dense. But there are a range of problems in between which could go either way.

While one solver can be set up to solve dense or rare problems, the philosiphy behind the solver options is different. For example, in an under dense problem, most particles will only ever overlap one other particle at a time. When this is true, the problem becomes the easy linear contact between two spheres, and can be solved in one step. However, in dense systems, one particle may overlap with six or more particles simultaneously, and each particle it overlaps overlaps with another six or more particles, and so on and so fourth, until the entire system is interconnected. This problem is much more difficult to resolve as moving any particle will create a whole new set of contacts.

2.1.1 Relaxation Paramater

As a result, the relaxation parameter (h) is used to vary the step size. For rare systems, a complete step ($h = 1$) can resolve the system in one step. For dense systems, attempting complete steps can cause instability. To resolve this, the parameter is default to 1, but can be adjusted. If the user knows their system is dense, the relaxation parameter can be

reduced manually. The progeram had been set up such that any time the amount of overlap increases during a time step, the relasation parameter will be reduced to prevent a cascade of instability.

2.1.2 Boundary Conditions

To apply a bounding box to the system, simple boundry conditions have been applied. The simple boundry condition is that if a particle lies outside of the domain at any time step, it is simply moved back to the domain such that its most extreem point lies directly on the boundry. An alternative would be the periodic boundry condition, where when a particle leaves the domain on one side, it reenters on the other side. The simple condition can result in a stack up of particle on the boundry, but this is less of a concern when the particle size varies. The stack up can happen when enough particles are outside the domain, that upon moving these back to the edge of the domain, these particles line up all the way along the boundry. If the centers of the particles are verticaly aligned, when the overlap is removed on the next time step, they will only be seperated verticaly, and will not seperate horizontaly. This effect can be observed (to some extent) in Fig. 4.1. This can be avoided by using a variety of particle sizes, or by underrelaxing the step size (h). Periodic boundries have not been implemented in mesoMake.

Chapter 3. Zoning

A severe inefficiency in this program thus far is the fact that for each particle, the distance to each other particle must be computed. Therefore, the computational cost of one loop of the solver is on the order of N^2 calculations, where N is the number of particles. To address this problem, versions 4 and 5 use zoning to decompose the domain. To do this, the domain is broken up into Nx by Ny subdomains, and each particle is assigned to a domain. Once particles have been assigned, each only needs to be checked against particles in the same or neighboring domains. This reduces the complexity of the eularian update to $N^2 \frac{3^D}{\Pi_i^D(n_i)}$, where D is the number of dimensions, and n_i is the number of subdomains in direction i. However, there is an additional lagrangian update that is required after the particles are moved; if a particle has crossed the boundry of a subdomain, it needs to be transfered to the new subdomain. This additional step adds a computational cost on the order of $N\Pi_i^D(n_i)$, that zoning might not save any time in the end; however, to save some time, a shortcut can be taken, where this lagrangian update is not performed every step. Since the system is relatively stable after the first few steps, it is unlikely that a particle will cross a subdomain boundry thereafter; therefore, this update only needs to be performed every so often.

The number of subdomains is limited by the largest diameter particle in the problem. Since each particle is only checked against particles in the same and neighboring subdomains, no particle should be allowed to overlap a particle a subdomain that is two steps away. Figure 3.1, shows an example, where the diameters of two particles are larger than the subdomains, as a result, these two particles are overlaping but are not in neighboring subdomains and this overlap will not be resolved. Therefore, the minimum side length for any subdomain is equal to the radius of the largest particle in the problem. However, if the zone size is set to the minimum the lagrangian update needs to be performed more frequently (and at a greater computational cost). This is because there is a greater chance that if a particle crosses a domain boundry that it can overlap a particle that is now in a neighboring subdomain, but until the particle is reasigned, this overlap may not be checked. When the remap finally occurs, this can cause an sudden instability that can ruin the progress of the solver.

There is no magic number that maxaimzes the efficency of the solver, but the subdomains should be 2-16 times the minimum size. This ratio can be set by the user with the *ratio* keyword, but the default is 2. The lagrangian update by default is performed on the first 5 iterations and then at the tenth, twentieth and so on. This can be modified by the user as well. A benchmark was set up for a two-dimensional domain, with 4776 particles

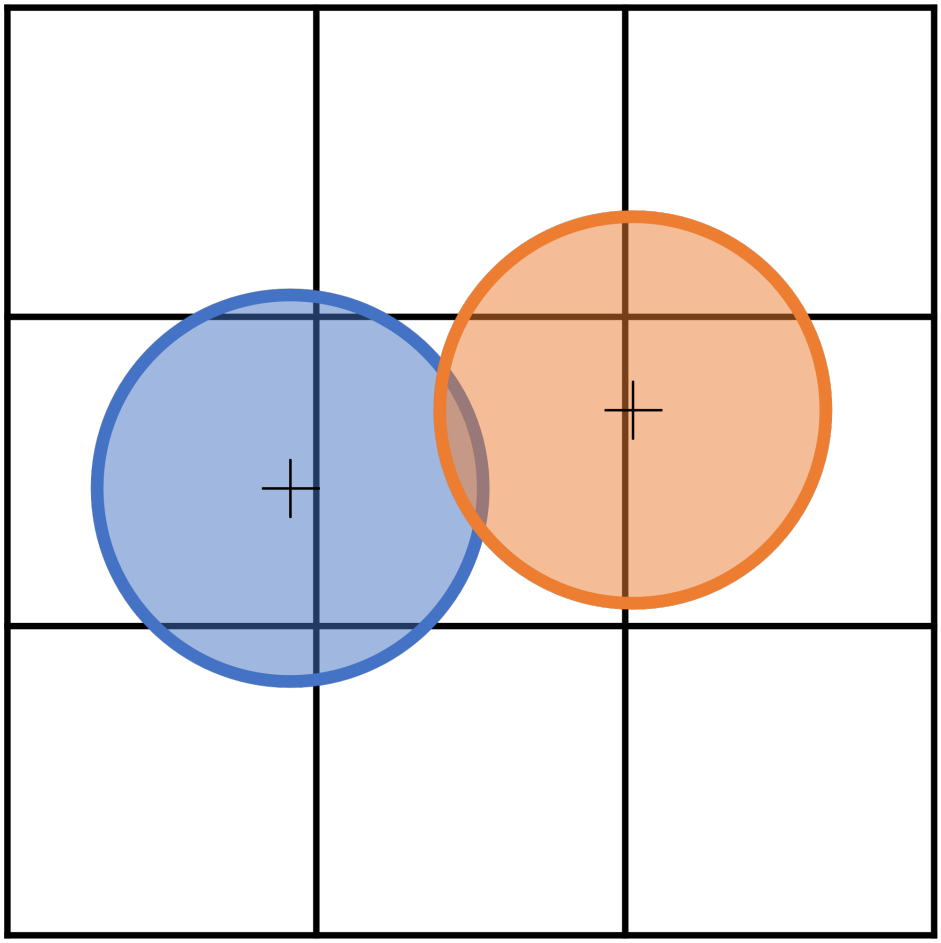


Figure 3.1: Improper Zoning

and the ratio was varied. The solver was run for 10 iterations, where remapping was carried out on every iteration. Runtimes can be found in Table 3.1. These results show a sweet spot around a ratio of 4, but these trends might change in more complex problems.

Table 3.1: Zoning: Ratio Benckmark

ratio	Number of Subdomains	Solve Time [s]
1	2500	354.7
2	600	114.5
4	150	88.1
8	36	148.7

To improve the efficiency of the remap step, the subroutine has been modified to first check if a given particle is in the same subdomain as the previous step. If the particle is not, then the particle is remapped. This is effective especialy late in the solving routine since most particles will remain in the same zone for the duration of the process. In a benchmark

test with 4776 particles in 600 subdomains, this modification reduced the solve time from 94 seconds to 89 seconds (a 5% reduction).

While zoning greatly improves the efficienfy of the update step, the introduction of the remap step adds a lot of computational cost. This domain decomposition is effective for serial evaluation, but would greatly benefit from parallel execution, where several subdomains are allocated to each processor. Since a subdomain has no effect on a subdomain far away they can be resolved concurrently. Descrepancies may arise at the boarders between each processor's region, where the result could depend on which processor gets to the boundry first. However, these descrepancies would be acceptable since these problems have to true solution. Parallel processing has yet to be implemented, but should greatly speed up runtimes. Furthermore, with parallel processing, the trade offs between large and small ratios mat shift the sweet spot back toward one or two.

Chapter 4. Sobol Sampling

While zoning helps the efficiency of the solver, it doesn't improve the effectiveness; if a problem takes 100 steps to resolve without zoning, it takes the same number of steps to resolve with zoning (possibly more). In fact, some problems simply can not be resolved, and there is very little the solver can do to improve this. One potential cause of this is that the initial locations are randomly generated from a uniform distribution. Unfortunatly, this does not produce a distribution that uniformly fills the domain. Quasi monte-carlo sampling methods are designed to address this problem. Two examples of this are Latin Hyper Cube (LHC) and the Sobol Method. A LHC sampling method divides each dimension into bins– one bin for each sample. For each sample, a bin is randomly selected for each dimension, and each bin may only contain one sample. This method mantains a the random aspect of the sample, and introduces some uniformity by limiting the number of points in close proximity. However, a sample could be created where all points could sit on a diagonal, and this would satisfy the LHC requirements. This counter example is a case of low entropy (as it would be perfectly ordered) and is unlikely from a large number of samples; as a result, the method can be optimized by pulling several samples and choosing the best result according a some heuristic. However, this can be time consuming and inefficient. The Sobol sequence presents a better alternative, that maintains the quasi-random aspect. The Sobol sampling method is formulated to create a highly uniform distribution, and has been implemented to do so very quickly.

The mesoMake program has set up so that the Sobol method can be used to create the initial distribution. In an example problem, this reduced the number of overlaps in the initial sample by 25% when compared to a uniform random distribution. Forthermore, since the sobol sample eliminates any clupms of high density from the initial sample, the solver was able to resolve a greater number of overlaps, and after 10 steps the soble sample had reduced the number of overlaps by 75% compared to the uniform distribution after 10 steps.

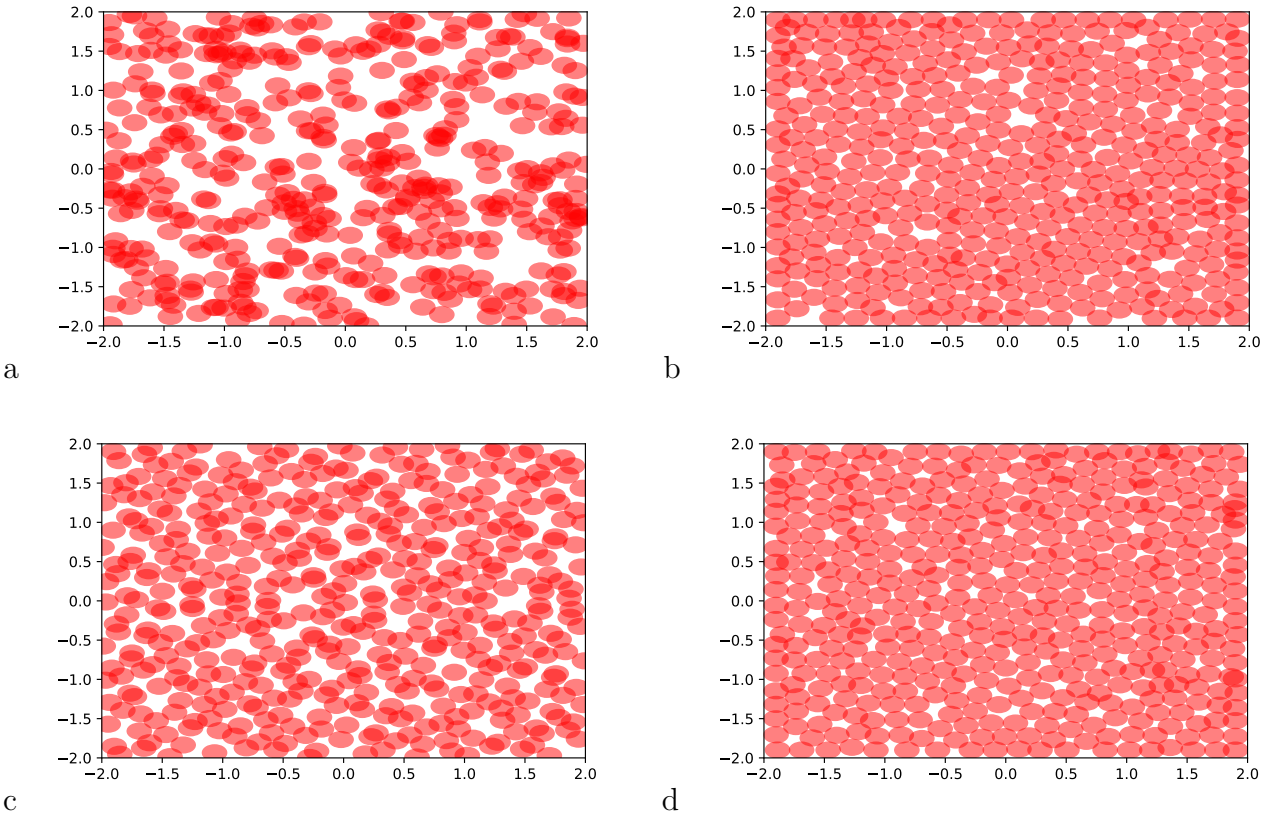


Figure 4.1: Random sample before (a) and after 10 steps (b) compared to sobol sample before (c) and after 10 steps (d)

Chapter 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.

5.0.1 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 2.4 and 2.5. This simple process is repeated some number of times or until some convergence criteria is met.

5.0.2 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.

5.0.3 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]
```

If the user wants to see the system, mesoMake creats a plot of the system on every iteration or only on the first and final iteration. These are by default saved as png files, but the user may change this.

5.1 Example Input

This input file generates the system seen in the inside cover.

```
vfrac=[0.125,0.25,0.25,0.0625,0.125] #   volume fraction of each species
R=[0.2,0.025,0.125,0.01,0.05]   #   radius of each species
id=[2,1,3,4,5]       #   identity of each species

X=[-2,2]        #   X limits
Y=[-2,2]         #   Y limits

'''
    Optional information
'''
color=[[1,0,0],[0,1,0],[0,0,1],[0.5,0.5,0],[0.5,0,0.5]]
#   user defined colors, must be in proper color formats
```

```
#   if you dont define your own colors, it will default to random ugly colors

n_iter_max=10
# maximum number of solver iterations, this is a recommended input, but default is 10

earlyExit=False # if True, solver will exit if convergence stops
seed=0 # you may define a random seed for reproducable results

sobol=True
# if you do not want sobol sampling, set to False. default is True
# sobol sampling prevents clumps at the initiliazation,
# It is most helpful when constituents are similarily sized

resave=True
# if you want a meso.csv file saved on every loop, set to true. default is false

ratio=1 # initial relaxation parameter

imgFMT='pdf' # set ploting format
```