MANUAL FOR SPHERE PACKING CODE

JAMES MCCLURE

1. Sphere Packing Code

1.1. **Compiling and Running.** Compiling the Sphere Packing code can be accomplished with the gcc or other c++ compiler:

```
g++ -O3 -o SpherePacking SpherePacking.cpp
```

See the g++ manual for entry for details.

In order to run the code, the input file "pack.in" must be placed in a directory where the executable can find it. All input parameters required by the sphere packing code are provided by "pack.in". These include, in the order specified:

- Number of spheres in the packing: N_s
- Variance for the normal distribution: σ^2
- Initial porosity: ϕ_{init}
- Target porosity: ϕ_{target}
- Domain length in each direction: L_x, L_y, L_z
- Number of cells in each direction: n_x, n_y, n_z
- Number of iterations to take before giving up: M_{iter}
- Factor for resizing the sphere radii: α
- Error tolerance (maximum overlap size): ε_{tol}

Of the input variables, N_s, σ^2, ϕ_{target} and L_x, L_y, L_z are most important for determining the packing structure. The remaining variables provide control numerical aspects of the code, and will be considered in a subsequent example.

1.2. **Algorithm.** The sphere packing code utilizes the algorithm used by Williams and Philipse to generate packings of sphereocylinders [1]. The algorithm has been modified to support generation of packings for systems of spheres with lognormally distributed radii, and to provide control over the final porosity of the packing. The major steps are summarized as follows:

(1) Instantiate a system of spheres, $i = 1, 2, \ldots, N_s$:
   - $c_{x,i} \sim$ Uniform$[0, L_x]$
   - $c_{y,i} \sim$ Uniform$[0, L_y]$
   - $c_{z,i} \sim$ Uniform$[0, L_z]$
   - $\log(r_i) \sim$ Normal$[\mu, \sigma^2]$

(2) Eliminate overlaps between spheres (see Williams and Philipse)

(3) Increase size of radii:

- $r_i \leftarrow \alpha r_i$
- $\mu \leftarrow \mu + \log(\alpha)$

It is easy to verify that rescaling of the radii by a constant factor α preserves the lognormal distribution variance σ^2 and that the mean μ will increase by $\log(\alpha)$ each time the radii are rescaled. This means that while σ^2 is specified as an input parameter, μ is not known until the simulation completes. Note that it is not possible to independently specify μ, σ^2 and ϕ. Given σ^2, the final value of μ can be estimated based on the target porosity ϕ_{target}. The expected volume of the lognormal packing is:

$$\frac{4\pi N_s}{3} E[r^3] = \frac{4\pi N_s}{3} \exp\left(3\mu + \frac{9}{2}\sigma^2\right) \tag{1}$$

To accelerate convergence, the spheres are divided into a collection of cells which consist of equally-sized sub-domains. This is a standard optimization for N-particle methods. The number of cells in each direction is determined from line 6 of the input file. Each cell contains a list of the spheres with centroids contained within the physical boundaries of the cell. For a given sphere, potential overlaps are considered only from the neighboring cells. The code is accelerated significantly by increasing the number of cells, thereby decreasing the length of the search path required when computing overlaps. However, this optimization can lead to oversights in the overlap computation if the maximum radius exceeds one half of the cell width, as shown in Fig. 2 (a). The maximum radius is checked after the simulation completes, and a warning message is issued if the maximum radius is too large based on the cell width:

```
. . .
SIMULATION COMPLETE
Target Porosity was 0.35
Actual Porosity is 0.349013
Mean coordination No. 5.592
FINAL DISTRIBUTION PARAMETERS:
        log(r) normally distributed with mean -2.72657 and variance 0.2
WARNING: maximum radius exceeds bin width!
```

1.3. **Generation of an Example Packing.** An example "pack.in" is provided:

```
250 0.2
0.8 0.35
1.0 1.0 1.0
2 2 2
125000
1.01
1e-10
```

The resulting packing will have $N_s = 250$, $\sigma^2 = 0.2$, and $L_x = L_y = L_z = 1.0$, with a target porosity of $\phi_{target} = 0.35$. Running the code simply requires running the executable generated from compilation from the appropriate directory. While running, the code prints out the input parameters as well as the current value of ϕ, μ and σ^2 and the error (given by the maximum overlap size) each time it finishes a loop:

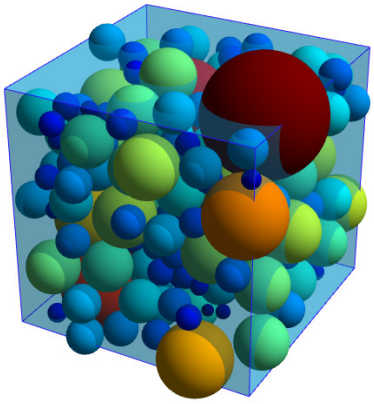
```
bash-3.2$ ./SpherePacking
Number of particles is: 300
Initial porosity is: 0.8
Target porosity is: 0.35
Domain size is: 1,1,1
Number of cells is: 2,2,2
Maximum number of iterations is: 200000
Radius scaling factor: 1.01
Error tolerance: 1e-10
Initial porosity (actual) 0.810125
Initial value for mu: -3.36521
Initial value for sigma (input): 0.547723
BEGIN ITERATIONS
        Increasing size of radii...
        log(r) now normally distributed with mean -3.34052 and variance 0.3
Finished a loop in 15 with error = 0
. . . . . .
Finished a loop in 2838 with error = 9.87316e-11
        Current porosity is: 0.355
Specified rescaling factor now being used: 1.01
        Increasing size of radii...
*****************************************
Preparing for final iteration...
        Radii will be rescaled by: 1.00258
        log(r) now normally distributed with mean -2.89348 and variance 0.3
Finished a loop in 2442 with error = 9.96838e-11
        Current porosity is: 0.35
Specified rescaling factor now being used: 1.01
SIMULATION COMPLETE
Target Porosity was 0.35
Actual Porosity is 0.35
Mean coordination No. 7.89333
FINAL DISTRIBUTION PARAMETERS:
        log(r) normally distributed with mean -2.89348 and variance 0.3
```

Note that due to the use of the random number generator, results vary slightly each time the code is executed. Fixing the seed for the random number generator will compel the code to compute identical results each time it is executed.

The output file "pack.out" is a basic (human readable) text file which lists the centroids $c_{x,i}, c_{y,i}, c_{z,i}$ and radii r_i for all spheres $i = 1, 2, \ldots, N_s$ generated by the packing code. The header lines for this file provide details of the packing including the calculations for the final distribution mean μ, porosity ϕ, and mean coordination number:

```
Number of Spheres: 250
```

(a)



(b)

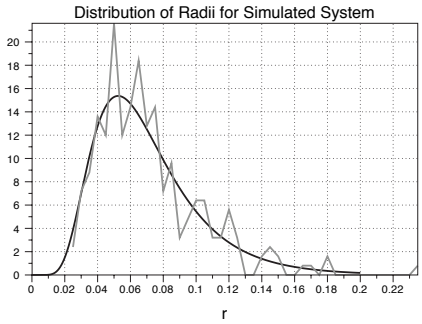


FIGURE 1. (a) Sphere packing of 250 lognormally distributed sphere with $\phi = 0.34722$ (b) Histogram of radii plotted beside the lognormal distribution with $\mu = -2.76, \sigma^2 = 0.2$.

```
Domain Length (x,y,z):  1, 1, 1
Media porosity: 0.347221
log(r) Normal with mean -2.76637, variance 0.2
Mean coordination No. 8.904
0.230307 0.767494 0.0737532 0.0610308
0.73798 0.065402 0.617662 0.0548015
0.47009 0.0336388 0.95164 0.0555246
0.430896 0.256432 0.239514 0.0811743
. . .
```

A packing generated using this output file shown in Fig. 1 (a). The code is guaranteed to preserve the specified value of σ^2. The final porosity will be less than the target porosity unless the code is unable to reduce the maximum overlap size to below ε_{tol} within M_{iter} iterations. In the latter case, the packing returned by the code corresponds with the minimum porosity packing for which overlaps were reduced below ε_{tol}.

Guidelines for the choice of ϕ_{init} and α are somewhat ambiguous: to generate a minimal random packing, ϕ_{init} must be sufficiently high and α must be sufficiently small. Physically this reflects that if ϕ_{init} is too small, the spheres may be locked into space initially and thus unable to explore the domain in a fashion that will lead to generation of a minimal packing. A similar effect may occur if α is too large. The details of this effect have not been explored for heterogeneous packings. To be safe, ϕ_{init} can be assigned a very high value (> 0.7). Since few overlaps are generated for these packings, the computational expenditure associated with high ϕ_{init} is very small compared with the computations required to obtain a minimal packing.

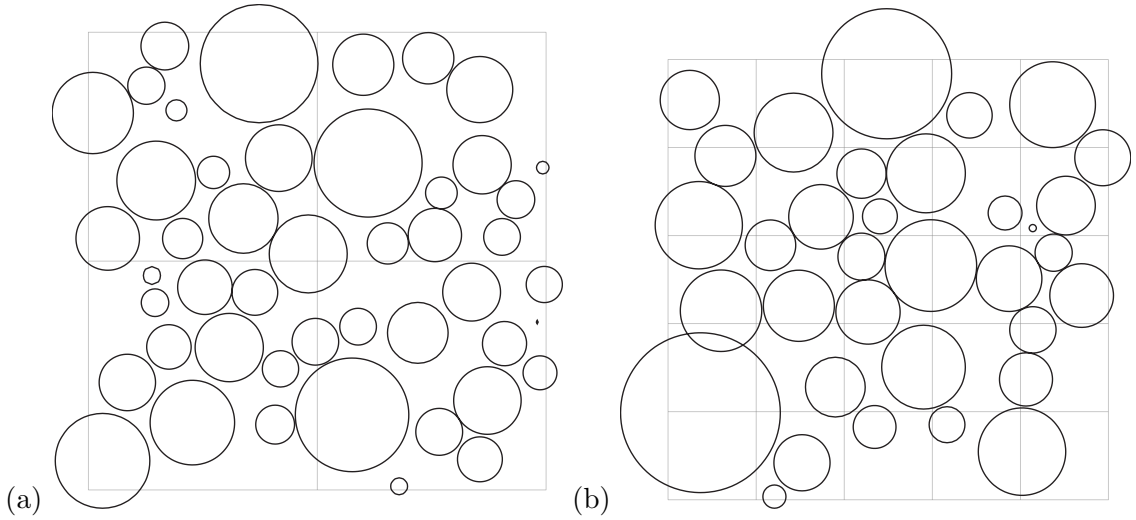


FIGURE 2. Errors may result if too many cells are requested for a given system. The maximum radius may exceed one half the cell width. Slices from two packings obtained using identical inputs with the exception that (a) $n_x = n_y = n_z = 2$ (b) $n_x = n_y = n_z = 5$.

If truly minimal packings are needed, M_{iter} must be increased significantly and α should be chosen somewhat carefully. The target porosity ϕ_{target} may be set to zero if desired. Strict limits for the minimal porosity for lognormal distribution of spheres have not been established. The well-known limit for a close packing of equal sized spheres is $\phi \approx 0.36$. Likewise, the maximum stable porosity for a homogeneous sphere packing is $\phi \approx 0.44$–0.47, with a coordination number of approximately six [2]. Each of these results has been reproduced with this packing code.

2. Digitization Procedure

Code is provided to digitize sphere packings for lattice Boltzmann simulation or other purposes. Compilation is the same as for the sphere packing code:

```
g++ -O3 -o DigitizePacking DigitizePacking.cpp
```

The input file "Digitize.in" provides the following information:

- Name of the packing file
- Name of the output (digitized) file
- Number of grid points in each direction (N_x, N_y, N_z)
- Minimum point for the digitized domain $(0, 0, 0)$
- Maximum point for the digitized domain (L_x, L_y, L_z)

A sample input file "Digitize.in" is provided:

```
pack.out
N64
```

```
64 64 64
0.0 0.0 0.0
1.0 1.0 1.0
```

In this case, the digitized file will be 64^3 and considers the full packing domain. Full periodic boundary conditions are assumed by the digitization procedure. This introduces a list of implicit spheres which are not contained in "pack.out". These spheres are written out by the digitization code into the file "pack.outBC", which follows the same conventions as pack.out, but without any header lines. Boundary spheres are shown in black in Fig. 3.

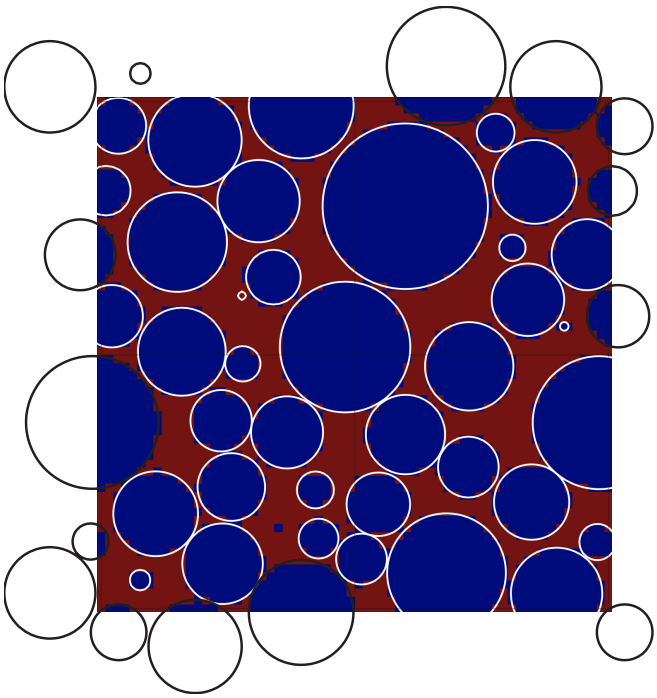


FIGURE 3. Digitization of sphere packing to 64^3. Spheres from pack.out are shown in white, boundary spheres are shown in black.

The digitized file written out by the code is a binary file with $N_x N_y N_z$ numbers of type char (8-byte integer). The 1-D index n and 3-D indices (i, j, k) are related in the standard way:

$$n = kN_xN_y + jN_x + i. \tag{2}$$

The binary files written by DigitizePacking are immediately readable by any of our lattice Boltzmann codes. Output of the digitization procedure includes the number of spheres extracted, mesh information, and the digitized porosity:

```
bash-3.2$ ./DigitizePacking
Digitizing Media
```

```
Packing file is: pack.out
Digitized media file is: N64
Mesh size is : 64x64x64
Reading the packing file...
Number of spheres extracted is: 250
Number of boundary spheres: 139
Porosity: 0.383484
```

The digitized porosity will vary slightly from the analytical value, depending on the resolution.

Since the lattice Boltzmann codes assume a constant mesh spacing in each direction, the digitization procedure can be used to stretch the domain by a constant factor in any direction by assuming a non-constant mesh spacing for the digitization. In LB world, this will "stretch" the domain so that spheres are mapped into ellipsoids. This represents a simple way to generate anisotropic packings based on the sphere packing code.

References

[1] SR Williams and AP Philipse. Random packings of spheres and spherocylinders simulated by mechanical contraction. *PHYSICAL REVIEW E*, 67(5, Part 1), MAY 2003.

[2] F. A. L. Dullien. *Porous Media: Fluid Transport and Pore Structure*. Academic Press, San Diego, CA, 1992.

[3] Alan Wouterse, Stephen R. Williams, and Albert P. Philipse. Effect of particle shape on the density and microstructure of random packings. *JOURNAL OF PHYSICS-CONDENSED MATTER*, 19(40), OCT 10 2007.