SIMULATION OF RANDOM PACKING OF HARD SPHERES USING MONTE CARLO METHOD

by

Sung-Ho Park

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

1990

APPROVAL SHEET

TITLE OF THESIS: Simulation of Random Packing of Hard Spheres
using Monte Carlo Method

NAME OF CANDIDATE: Sung-Ho Park
Master of Science in Mechanical Engineering

THESIS AND ABSTRACT APPROVED: ____________________ DATE: __________

Anthony D. Rosato
Assistant Professor
Mechanical Engineering Department

SIGNATURE OF OTHER MEMBERS ____________________ DATE: __________
OF THESIS COMMITTEE

____________________ DATE: __________

VITA

Name : Sung-Ho Park

Permanent Address :

Degree and date to be conferred : MSME, 1990

Date of Birth :

Place of Birth :

Secondary Education : Kyung-gi high school, Seoul, 1975-1977

Collegiate institutions attended	Dates	Degree	Date of Degree
Korea University	1978-82	BSME	1982
New Jersey Institute of Technology	1987-90	MSME	1990

Major : Mechnical Engineering

ABSTRACT

Title of thesis : Simulation of random packing of hard spheres
using Monte Carlo method

Sung-Ho Park, Master of Science in Mechanical Engineering, 1990

Thesis directed by : Dr.Anthony D. Rosato, Assistant Professor
Mechanical Engineering Department.

A computer based method of generating a random packing of hard spheres is described. Using a Monte Carlo method as employed in the field of Computational Statistical Physics, packing of hard spheres are generated and analyzed.

The mean packing fractions for the present assemblies of 1000 spheres are 0.555±0.015 after pouring and 0.582±0.018 after 10 cycles of shaking. These values are approximately 5 to 6 per cent lower than the experimental results of *G.D.Scott*[30], but similar with the result of *Visscher & Bolsterli*[17].

The mean coordination numbers are 5.97 and 6.33 for the pouring and shaking case, respectively. The radial distribution function was calculated and compared with other published data. The simulated results are similar with those of *G.D.Scott.*

The pouring simulations with 5 different system sizes verified that the resulting low packing density is independent of the number of particles in the system.

In an attempt to determine the reasons for the 5 to 6 per cent difference between existing experimental data of *G.D.Scott* and the simulation results, two computations were done.

The first case study measured the total void volume formed by

the gaps of the neighboring spheres. It was found that the void volume occupied approximately 0.0017 per cent of the total volume. Therefore the use of the corrected diameter cannot be a factor.

The second series of computations studied the effects of allowing the system to rapidly "cool" to an equilibrated state as opposed to incrementally reducing T^* from a value of 15.8 to 0.00211, whereby the system is allowed to equilibrium at each incremental step. The result shows that the packing density increased from 0.565 to 0.617. This can account for the 5 to 6 per cent difference between the experimental result of *G.D.Scott* and the result of current simulation.

ACKNOWLEDGEMENT

I am extremely grateful to my advisor Dr. Anthony D. Rosato for his considerable assistance and helpful discussions. This thesis would not have been completed without his help.

TO DEAR MY WIFE

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Survey of Previous Research

Random packing of hard spheres have been extensively studied, due to their importance as models for particulate systems in a wide variety of fields such as physics, chemistry, biology and engineering.

The methods used to investigate the sphere packings are broadly classified into two groups, *i.e.* mechanical packings and computer simulations.

G.D.Scott [3] carried out his experiments with 1/8 inch diameter steel balls and obtained two well-defined limits which he called "dense random packing" and "loose random packing". For the dense random packing, the balls in the container were gently shaken down for 2 minutes. For the loose packing the balls filled the container essentially by rolling down a slope of randomly-packed balls. The values for the limiting packing densities were 0.637 for dense random packing and 0.601 for loose random packing. The variation of measurements of the two limits were within ±0.2 per cent.

H.Susskind and *W.Becker* [34] packed rubber ballons with 0.118 inch diameter glass balls and 0.125 inch diameter steel balls. The beds were packed by dropping balls randomly into the rubber ballons and evacuated the air from the ballons, but in several cases the beds were vibrated for 45 minutes on a shaker before evacuating the air. The average densities of the loosely packed beds were 0.638±0.01 and 0.635±0.01 for the glass and stainless steel beds, respectively. The average density of the densely packed glass beds was 0.652±0.01.

R.K.McGeary [6] found that a maximum packing density could be obtained when the container diameter was more than about ten times the sphere diameter.

G.Mason [4,31] simulated the random packing of equal spheres on a computer, and found a limiting density of 0.63 to 0.64, close to the experimentally determined value by *G.D.Scott* [3]. The methods used by *Mason* essentially assumed a central confining force on the sphere, thereby avoiding effects due to gravity.

D.J.Adams and *A.J.Matheson* [14] generated a random close packing of hard spheres via a computer simulation. Their method placed a new sphere at the tetrahedral site nearest to the center of packing, thus producing a spherical model. The resulting packing density was 0.628. The fluctuation in the measured packing density was not specified.

C.H.Bennett [12] constructed packings of several thousand equal hard spheres by depositing each sphere, one at a time, at surface sites on a small seed cluster, placing each new sphere in contact with three already presented ones. This yielded the mean packing density of 0.61. The limiting values were bounded from 0.57 to 0.63. *Bennett* and *Matheson*'s techniques are basically the same, but the choice of sites of which to place the new sphere was different as decribed above.

Further, *W.M.Visscher and M.Bolsterli* [15] approached the problem of random packing of spheres by means of a Monte Carlo computer simulation of the physical process of dropping spheres into a bin and found a density of 0.582. *E.M.Tory et.al.*

[10,16,19,32] simulated the very slow settling of spheres from a dilute suspension into a randomly packed bed. To avoid the wall effects, the packing density was measured on the interior 5000 spheres of an assembly of 10,000 monosized spheres. An overall mean packing density of 0.58 was found.

A.J.Matheson [17] generated a homogeneous assembly of randomly closed packed spheres of packing density 0.606±0.006. He used a spherical growth method which involved the selection, from among the large list of available tetrahedral sites, of that one site which is nearest to the origin of the pile of existing spheres.

W.S.Jodrey and *E.M.Tory* [21] generated 3000 spheres in a cubic container by a relaxation method. The relaxation method eliminated the largest overlap at each step and gradually converged to an overlap-free packing. Their packing achieved density of 0.6366 and coordination number of 5.64.

J.Rodriguez et.al.[22] developed an assembly of packing under gravity, particle by particle. A new particle at a randomly chosen position above the already placed particles was dropped and allowed to roll down until it reached a stable position. The resulting packing density was 0.58±0.05. The summarized survey is presented in Table 1.1.;

Methods	Procedures	References
Experiment	Irregular packing constructed by shaking together equal steel ball bearings. These arrays are generally fixed by means of waxes and the sphere center coordinates measured by special machines. [34]	Bernal & Mason [2] Scott [3] Bernal [5,7] McGeary [6] Bernal, Mason & Knight [9] Bernal & Finney [8] Finney [13] Suskind & Becker [35]
Computer-Simulation	Computer generated sets of the expected spatial coordinates of the spheres.	Tory, Cochrane & Waddell [10] Scott & Mader [11] Bennett [12] Adams & Mathe Visscher & Bolsterli [15] Tory, Church, Tam & Ratner [16] Matheson [17] Gotoh & Finney [18] Jodrey & Tory [19] Powell [20] Rodriguez, Allibert & Chaix [22] Gotoh, Jodrey & Tory [32]

Table 1.1. The summarized previous works on random packing

1.2. Comparison of Experimental and Computer Simulated Results

Computer simulations of random packings are highly dependent on the assumptions made in the generating algorithm. In experiments, observed results also had a high dependence on the experimental procedures.

The summary described above indicates that the upper limit values of experimental and computer simulated packing densities are 0.637±0.001 and 0.6366±0004, respectively. The lower limiting densities are 0.60 and 0.58, respectively. The coordination numbers ranged from 5.45 to 6.4 at close sphere contacts in experiments. In the case of computer simulation, the coordination numbers ranged from 6.0 to 6.1 at close contacts.

Table 1.2 summarizes the results of experimental and computer simulated random packings.

Mean coordination number	Packing density	System	References
–	0.601±0.001 0.637±0.001	Steel balls in a cylinder	Scott [3]
–	0.625	Steel balls in a glass container	McGeary [6]
–	0.6366±0.0004	Steel balls in a cylinder	Finney [13]
6.1	0.59	Computer Simulation	Tory, Cochrane & Waddell [10]

-	0.628	Computer Simulation	Adams & Matheson [14]
6.0	0.61	Computer Simulation	Bennett [12]
6.4	0.582	Computer Simulation	Visscher & Bolsterli [15]
6.01	0.58	Computer Simulation	Tory, Church, Tam & Ratner [16]
6.0	0.606±0.006	Computer Simulation	Matheson [17]
6.0	0.6099 ~ 0.6472	Statistical Method	Gotoh & Finney [18]
6.0	0.59±0.01	Computer Simulation	Powell [20]
6.0	0.58±0.05	Computer Simulation	Rodriguez, Allibert & Chaix
–	0.634	Computer Simulation	Mason [31]
–	0.582	Computer Simulation	Gotoh, Jodrey & Tory [32]
5.64	0.6366	Computer Simulation	Jodrey & Tory [21]

Table.1.2 Data comparison of experimental & computer simulation

1.3 Outline of Thesis

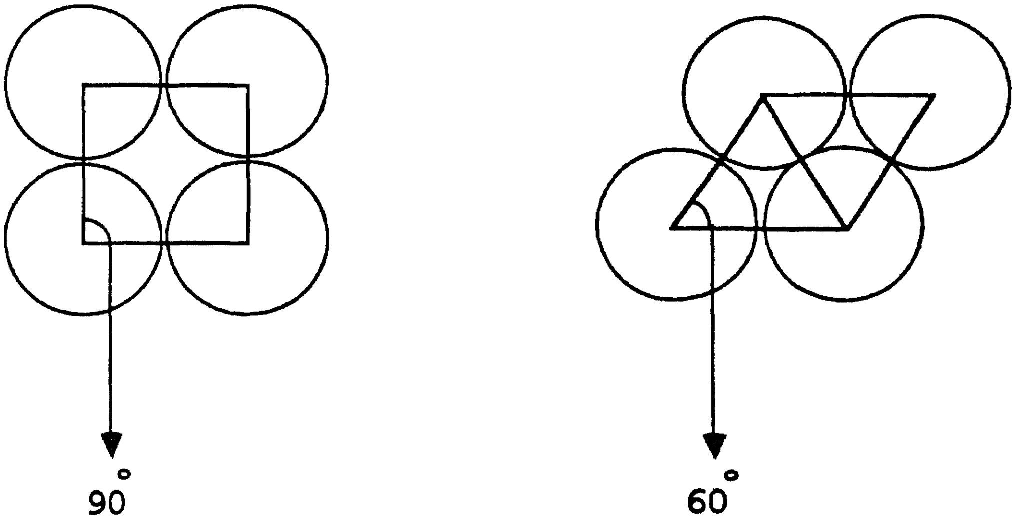
Section 2 describes the packing of monosized spheres. As a first step toward the analysis of random packing of spheres, the regular and random packing arrangements of monosized spheres are discussed in this section. In section 3, the basic algorithms for converting two dimensional code to three dimensional code are presented. The periodic boundary conditions and geometry checking subroutine are the main parts where that idea is applied. The general concepts of the Monte Carlo Method in the pouring and the shaking simulations are also introduced. Section 4 deals with the analysis of the assemblies which are obtained from the simulation code. Summary and Conclusions are presented in Section 5 with suggestions for further studies.

2. PACKING OF MONOSIZED SPHERES

2.1 Regular Packing of Spheres

A regular packing of spheres may be assembled from layers and rows. The fundamental unit is a row of contacting spheres. These rows can be arranged in the same place, parallel to each other and touching, to form a layer.

The most common packings are built from one or another of the limiting forms. These are the square layer with a 90 degree angle and the triangular or simple rhombic layer with an angle of 60 degree [24]. Those two types of layers are shown in Figure 1.1:



(a) square layer (b) simple rhombic layer

Fig. 1.1 Types of Layers

The highest over-all density in a regular packing is achieved in the face-centered cubic (F.C.C.) and hexgonal close-packed (H.C.P.) structures. The FCC structure has four spheres per unit cell and its packing density is calculated as follows:

$$\frac{V_s}{V_c} = \frac{4 \times (4 / 3 \times \pi \times r^3)}{(4 \times r / \sqrt{2})^3}$$

$$= 0.7405$$

where,

V_s : Total volume of spheres

V_c : Volume of unit cell

r : Sphere radius

In the case of HCP structure, each sphere touches three spheres in the layer below its plane, six spheres in its own plane, and three spheres in the layer above. The packing density is also found to equal 0.7405.

2.2 Random Packing of Spheres

A random packing [23,24] is formed by the haphazard positioning of spheres to form an assembly or a bed. The loose and close random packings characterize the configurations which result when an assembly of spheres is packed in an apparently random manner to its loosest and densest conditions, respectively.

In this work, Monte Carlo method [1,23,25,26] of the type from Computational Statistical Physics is applied to achieve random packings of hard spheres.

2.2.1 Random Loose Packing

This configuration is obtained by packing the spheres so that they roll individually into place over similarly placed spheres by individual random hand packing or by "dropping" the spheres into th container without bouncing.

The most probable value for the packing density of a random loose packing [2,3,10,12,15-17,20,22,32] of monosized spheres is bounded between 0.58 and 0.60.

2.2.2 Random Close Packing

A random close packing for monosized spheres corresponds to their maximum density without long range order or deformation. These are obtained when the bed is vibrated or vigorously shaken down. Most of the reported experimental values of the packing density for random close packing lies between 0.625 and 0.64 [2,3,6].

In the case of computer simulated techniques, produced packing densities [14,21,31] ranged from 0.628 to 0.6366±0.0004 for monosized spheres.

3. THE SIMULATION CODE AND PROCEDURES

This section outlines the simulation procedure and the Monte Carlo method. The algorithm for this code is presented in Appendix A.1 and the FORTRAN code listing is also found in Appendix B.1. The Monte Carlo method adapted here is commonly used in the field of Computational Statistical Physics. It was developed by *von Neumann, Ulman,* and *Metropolis* to study the diffusion of neutrons in fissionable material. The details can be found in [1], [23]-[25],[28], and [29].

3.1 Periodic Boundary Conditions

The two dimensional code is converted to three dimension mainly bymodifying the periodic boundary conditions (P.B.C.) and the geometry checking subroutine (GEOMCK). The existing dimensional code has only 6 cases of P.B.C., but 49 cases are considered in three dimension code. The basic idea for establishing the P.B.C. in three dimensions is now described:

(i) X-Y-Z coordinate system is defined in Figure 3.1. Two boundary conditions are established here. One is a hard vertical wall, X-O-Z plane and the others have periodic boundary conditions. A sphere at coordinate (X, Y, Z) reappears at (X ± Lx, Y, Z ± Lz) in a periodic boundary condition, so the packing is effectively infinite in horizontal direction.

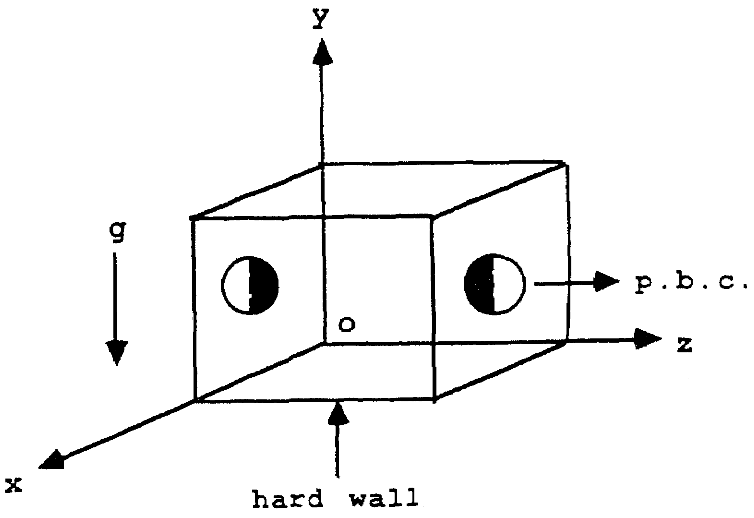
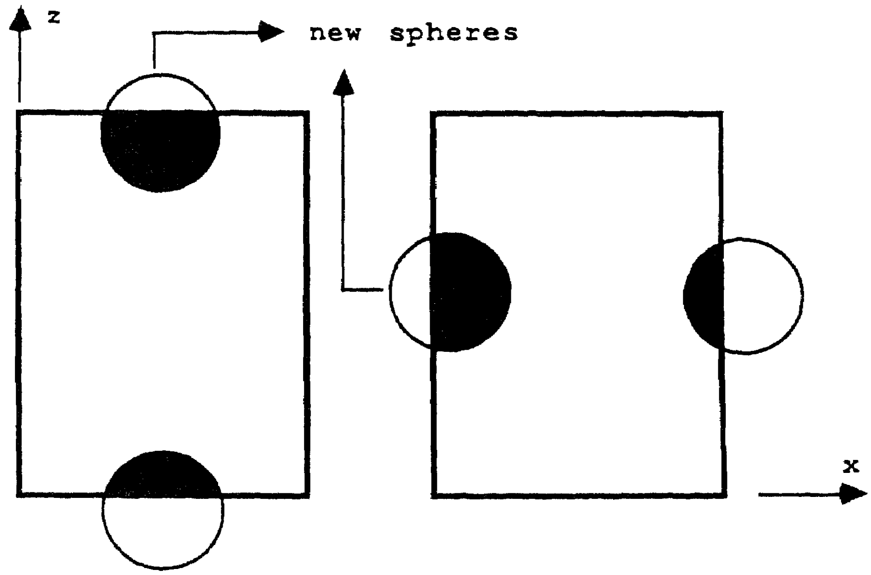


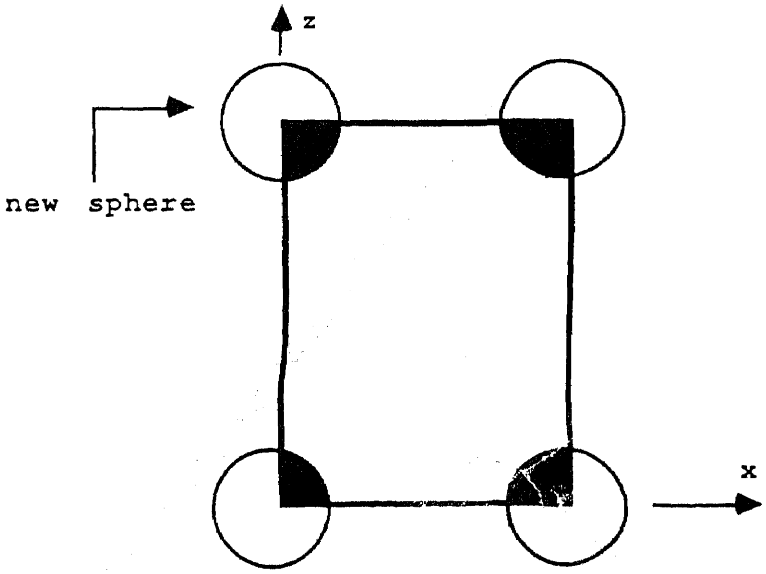
Fig.3.1 Coordinate system and periodic boundary conditions

(ii) If a new sphere is created on the side of a cell and partially included in the cell as shown in Figure 3.2.(a), the other segment of the sphere appears on opposite side of the cell.



(a) P.B.C. on the sides

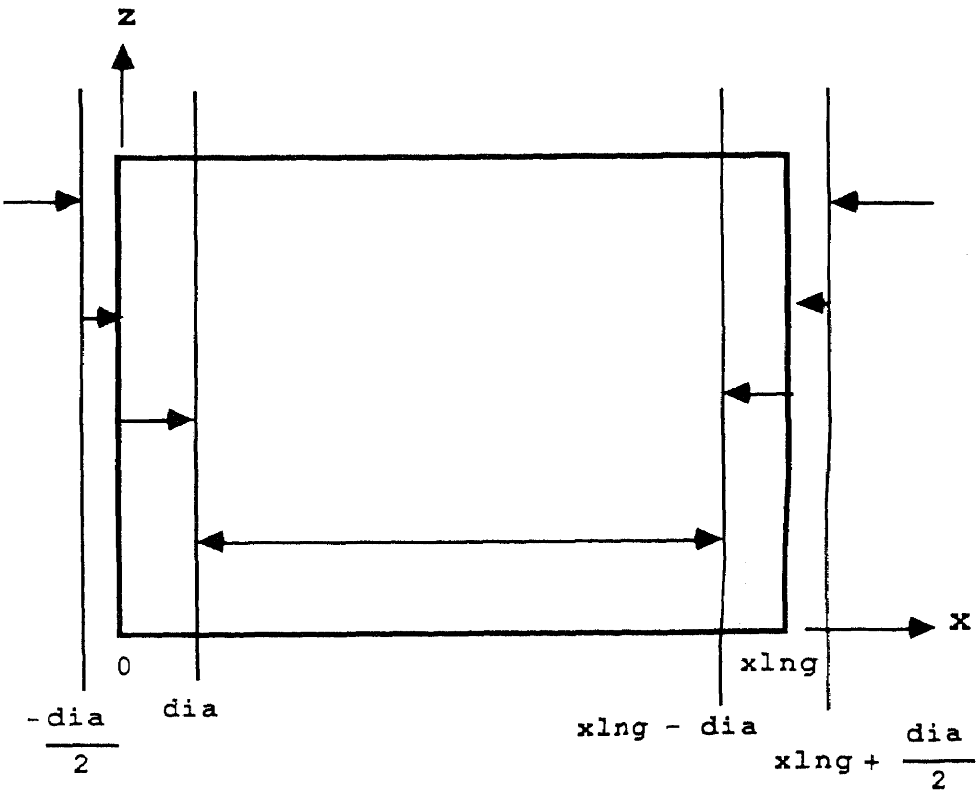
(iii) If a new sphere is created in the corner of a cell and included partially in the cell , the other segments of the sphere appears in three other corners as shown in Figure 3.2.(b).



(b) P.B.C. in the corner

Fig.3.2 Periodic Boundary Conditions in each case

(iv) In each case (ii) and (iii), the sphere can lie at seven different locations in X direction as shown in Figure 3.3. Considering the Z direction, combinations of X and Z result in 49 differnet cases of boundary conditions in this system.



xlng : Length of a cell

Fig.3.3 Possible locations of sphere in X direction

By those rules, finally 47 cases of P.B.C. are established and coding is modified to incorporate these cases. In order to check the sphere overlaps, geometry checking subroutine (GEOMCK) is used. All the cases are checked by GEOMCK whether the spheres are overlapped or not. This is effectively done to enforce the hard sphere potential, *ie.*, spheres can touch without experiencing any attractive or repulsive force, but cannot overlap.

3.2 Pouring Simulation

The "pouring" process starts with moving one sphere at a time according to the following prescription:

$$X \rightarrow X + \delta\xi_1$$
$$Y \rightarrow Y + \delta\xi_2$$
$$Z \rightarrow Z + \delta\xi_3$$

where δ is the maximum allowable displacement. ξ_1, ξ_2 and ξ_3 are the random numbers between -1 and 1. After moving a sphere, it is equally likely to be anywhere within a cubic of side 2δ centered about its original position.

A trial configuration is accepted as the new configuration based on the change of potential energy ΔE in the system. If $\Delta E < 0$, the new position is allowed by placing the trial sphere in its new position. If $\Delta E > 0$, the new position is accepted with probability $exp(-\Delta E/kT)$, *i.e.* compare a random number, and $0 \leq J \leq 1$, with $exp(-\Delta E/kT)$; move the sphere to its new position if $J < exp(-\Delta E/kT)$. Otherwise, reject the position and keep the sphere at its old location. This process is carried out for all N particles of the system thereby completing one "pass".

In this simulation, the gravitational potential is permitted only to decrease the configuration energy and no bouncing is permitted. Hence the spheres slowly settle down to the bottom of the container. As the pass number increases, the change of configuration energy becomes smaller. It requires more than a hundred thousand passes to attain an equilibrated state.

The input data for the pouring simulation is presented in Table.3.1.

sphere number	1,000
container dimensions (inch)	3.0 × 5.0 × 3.0 (Width × Height × Depth)
sphere diameter (inch)	0.3
δ in each pass	1 / 6 Dia.

Table 3.1 Initial input data for the pouring simulation

3.3 Shaking Simulation

In order to get the densest packing, a shaking procedure is necessary. The spheres are first lifted uniformly by a predefined specific amplitude and then allowed to settle down via the Monte Carlo method without bouncing as described in section 3.2.. This completes one cycle.

In this simulation, the shaking amplitude for each case is between one thirds and one sixths of the sphere diameter. Many cycles are required to obtain the "densest" packing. A cycle is halted when the change of the potential energy is less than a predefined tolerance in the input data. Table 3.2 shows the input data for shaking process.

amplitude	1 / 3 – 1 / 6 Dia.
passes for cycle	40,000
δ in each pass	1 / 6 Dia.
number of cycles	10 cycles

Table 3.2. Input data for the shaking simulation

4. RESULTS

To analyze the sphere assemblies generated, geometrical properties of the assemblies are measured and compared with the published ones. These include the packing fraction, the distribution of coordination numbers and the radial distribution function.

The mean coordination number is computed using three different tolerances, *ie.,* 1%, 5% and 10% of sphere diameter. The first one included the close contacts within 1% of the sphere diameter in separation. The second and the third one included 5% and 10%, respectively. The comparison of the results with others is based on the 5% diameter separation, because the experimental result of *Bernal et.al.* and the computer simulated result of *Matheson* are using same tolerance. The details are presented in Section 4.1.

In this work, two methods are used to calculate the packing fraction. The first one is a "Plane Growth Method" and the other one is a "Spherical Growth Method". The details are explained in Section 4.2 and 4.3.

The calculated radial distribution function is presented in Section 4.3 and compared with published results.

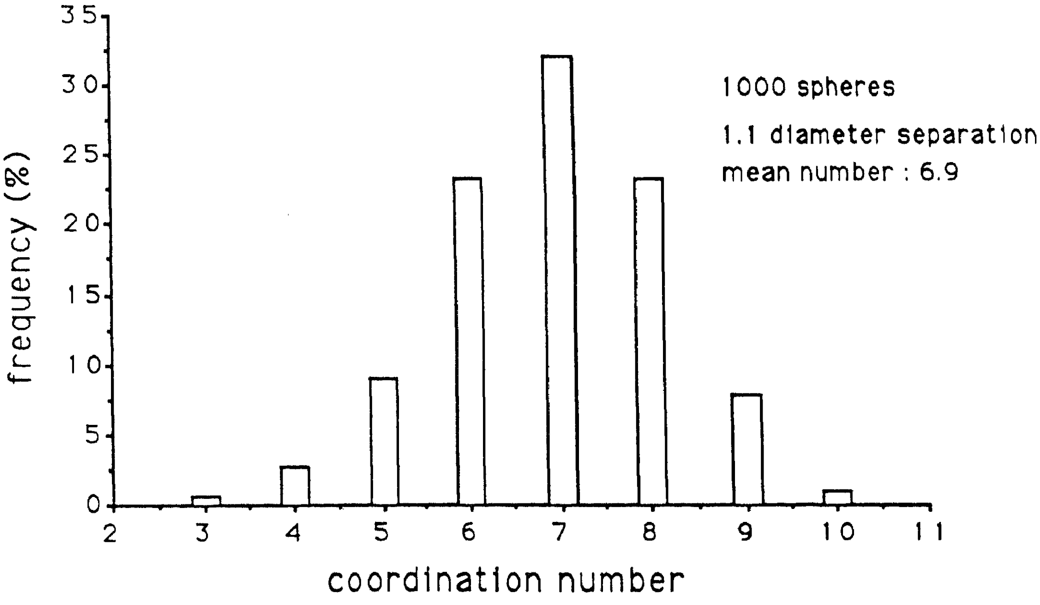
In order to obtain the possible factors that effect the low packing densities, three case studies were done and their results are presented in Section 4.4 to 4.6.

All the calculations were carried out using*VAX/VMS-8800* computer.

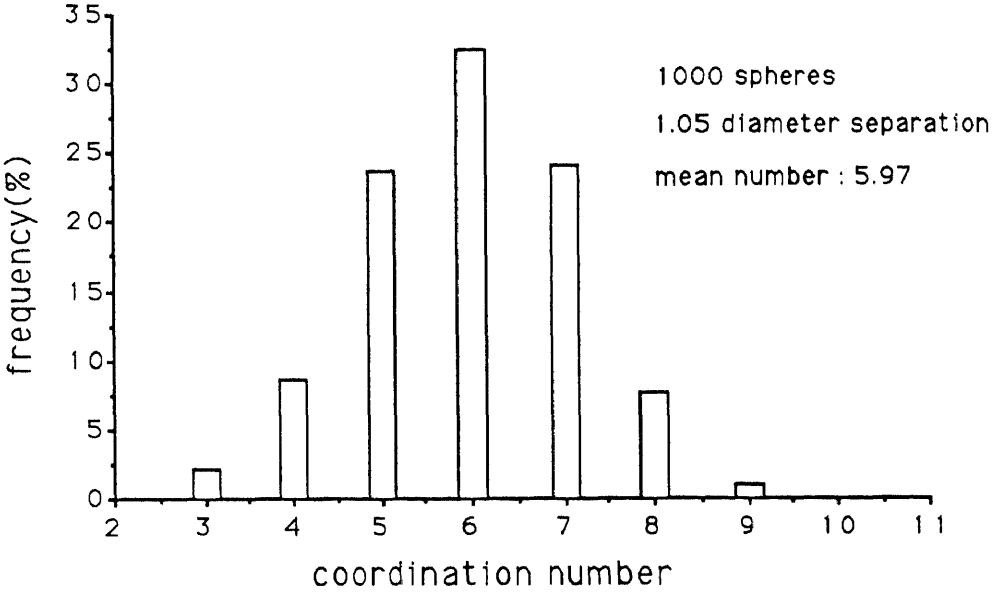
4.1 Coordination Number

The coordination number [2,21] is defined as the mean number of spheres in contact with any given sphere. The expected value of the coordination number seems to be six [2], as each sphere may be generally supported by three others and in turn to support another three spheres.

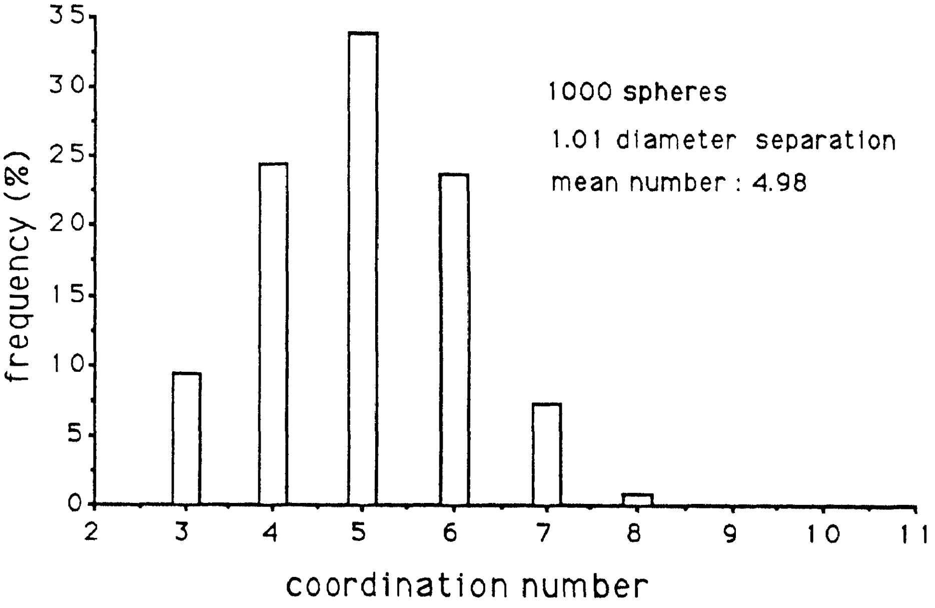
In order to include all the contacting neighbors, the coordination numbers of the central 563 spheres of the 1000 sphere assembly have been calculated. The coordination number distribution is shown in figure 4.1 for the pouring simulation. The results are computed for the sphere separations of 1.1, 1.05 and 1.01 diameters. The mean coordination numbers are 6.90 at 1.1 diameter separation, 5.97 at 1.05 diameter and 4.98 at 1.01 diameter. These values are measured using the coordination number code located in Appendix B.2. The computed values of the coordination numbers are also presented in Appendix C.1.



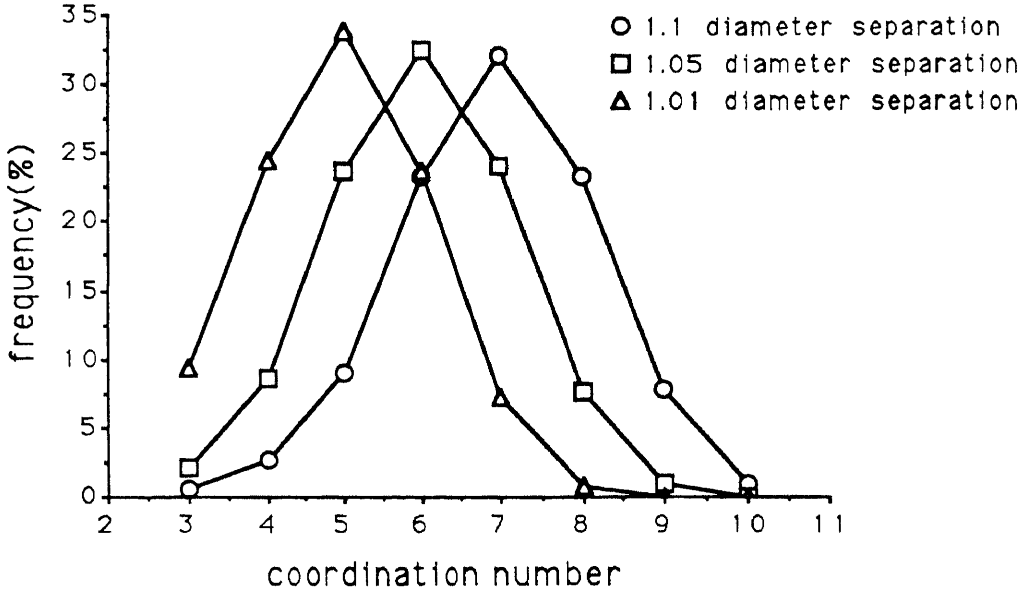
(a) 1.1 diameter separation



(b) 1.05 diameter separation



(c) 1.01 diameter separation



(d) Comparison of the results

Fig.4.1 Coordination numbers at 1.01, 1.05 and 1.1 diameter separation after pouring

The experimental result of *Bernal & Mason* and the computer simulation results of *Tory et.al., Jodrey & Tory, Matheson* and the current results of pouring simulation are compared in fig 4.2. All the results show a peak value at a coordination of six, except for the result of *Bernal et.al.*. The results of *Tory et.al.* and *Matheson* showed a similar distribution. In comparison with the experimental results by *Bernal et.al.*, the simulated distribution is shifted to the left.

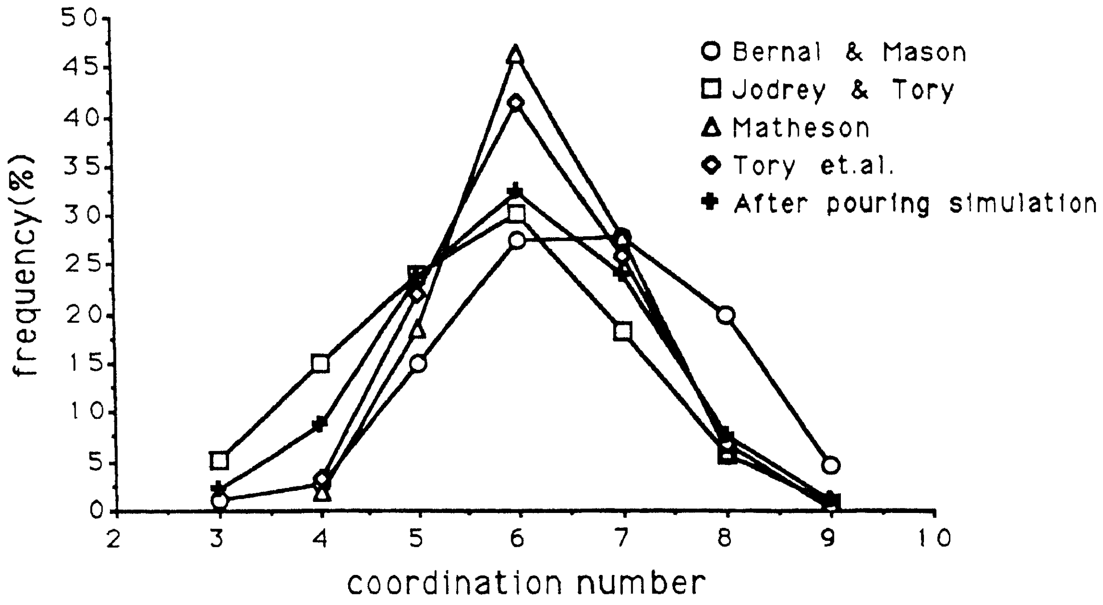
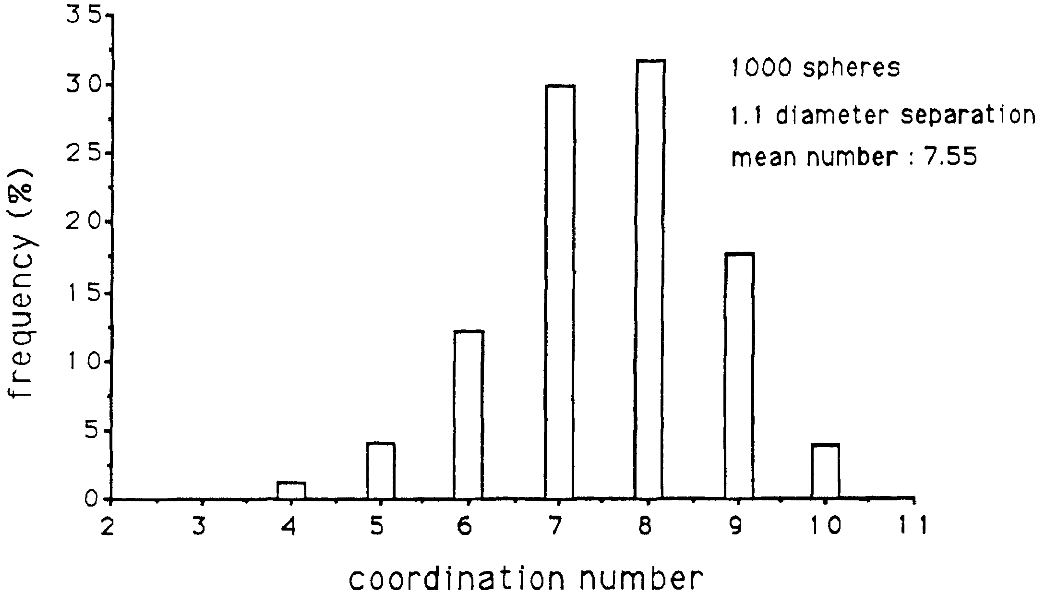


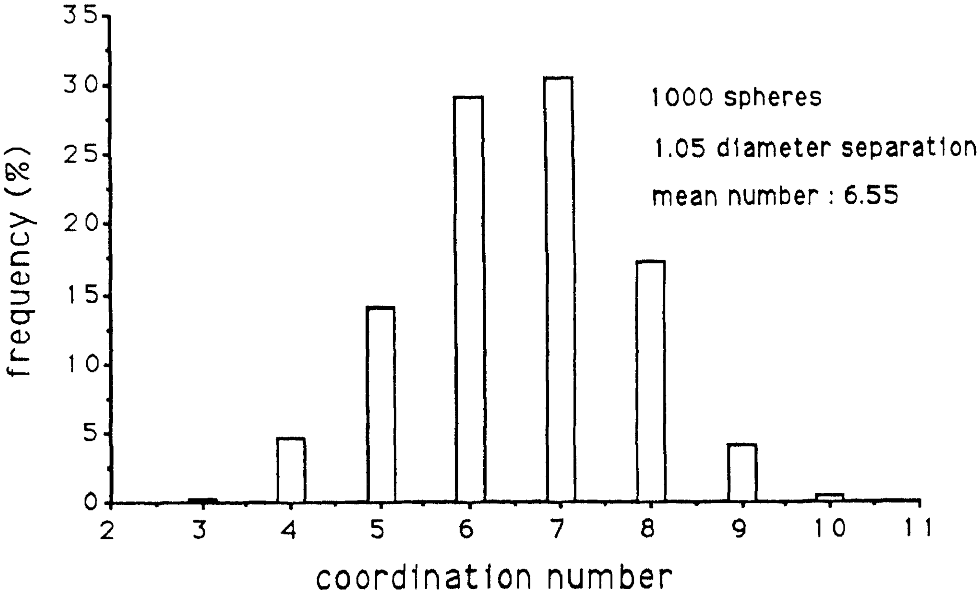
Fig. 4.2 comparison of the results between published data and pouring simulation

With an amplitude of one sixth of the sphere diameter, 10 cycles (40,000 passes per cycle) of shaking were carried out. Then coordination numbers for each case are computed. The result shows an approximate 6 to 10 per cent increase of coordination number.

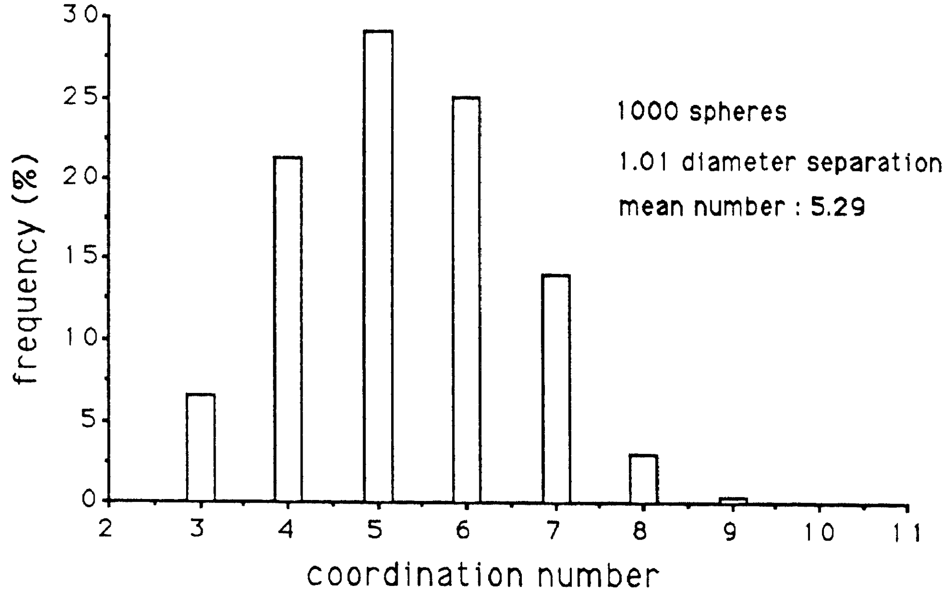
The average coordination numbers are 7.55 for 1.1 diameter separation, 6.55 for 1.05 diameter separation and 5.29 for 1.01 diameter separation. Figure 4.3 shows the coordination number histogram for the shaking case. The computed values are found in Appendix C.2.



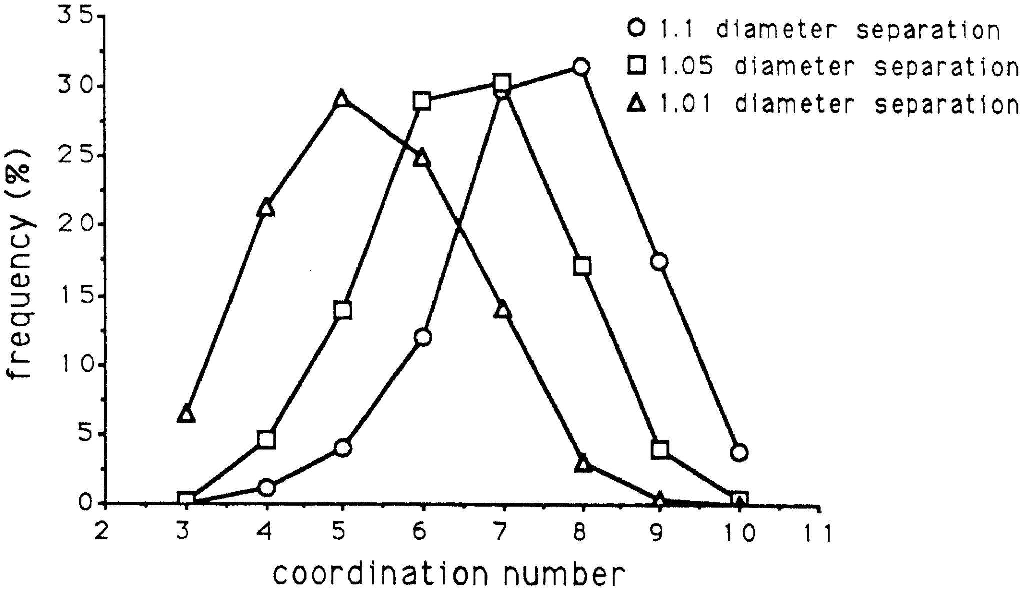
(a) 1.1 diameter separation



(b) 1.05 diameter separation



(c) 1.01 diameter separation



(d) Comparison of the results

Fig.4.3 Coordination numbers at 1.01, 1.05 and 1.1 diameter separation after 10 cycles of shaking

Figure 4.4 shows a comparison of the results between experimental results of *Bernal et.al.* and the current result after 10 cycles of shaking simulation. The mean coordination number of *Bernal & Mason*'s result is 7.99 and the current one is 6.55 for the sphere separation of 1.05 diameter.

Because of the low packing density, the present result shows a configuration shifted to left as compared with the result of *Bernal et.al.*.

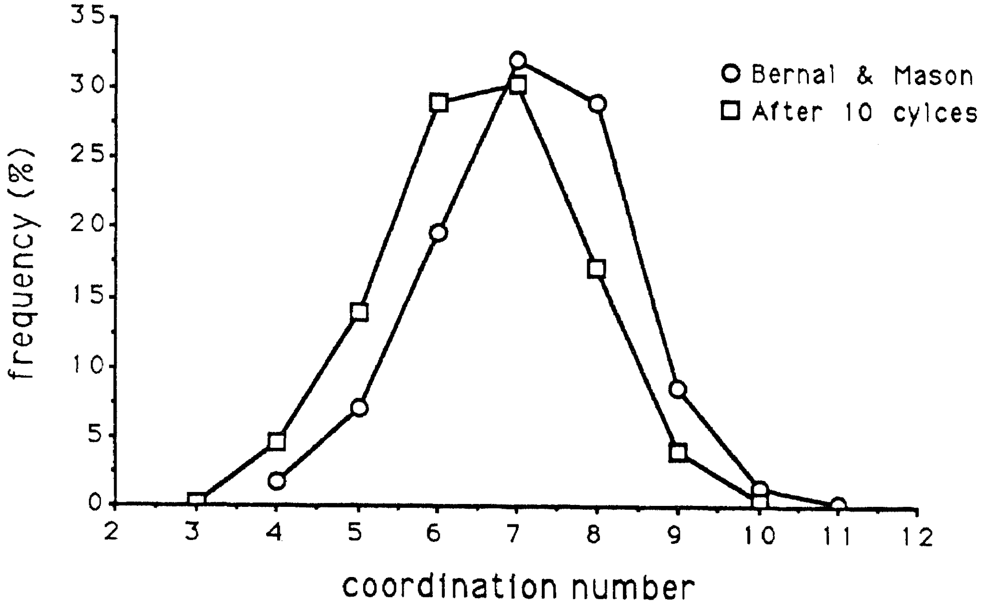


Fig. 4.4 Comparison of the results between *Bernal & Mason*'s experiment and shaking simulation.

4.2 Packing Fraction

The packing fraction [12-20,24,26,27] or solids fraction is defined as the ratio of the total volume of spheres to the volume containing them. Two methods are used to calculate the packing fraction.

4.2.1 Spherical Growth Method

This method calculated the packing fraction from the 19 spherical samples within the packing. The code may be found in Appendix B.3. The actual volume of solids within each spherical sample is determined by calculating the volume of the spheres totally within the radius plus fractional volume of those of those spheres which intersected the sampling sphere. The details are shown in Figure 4.5.

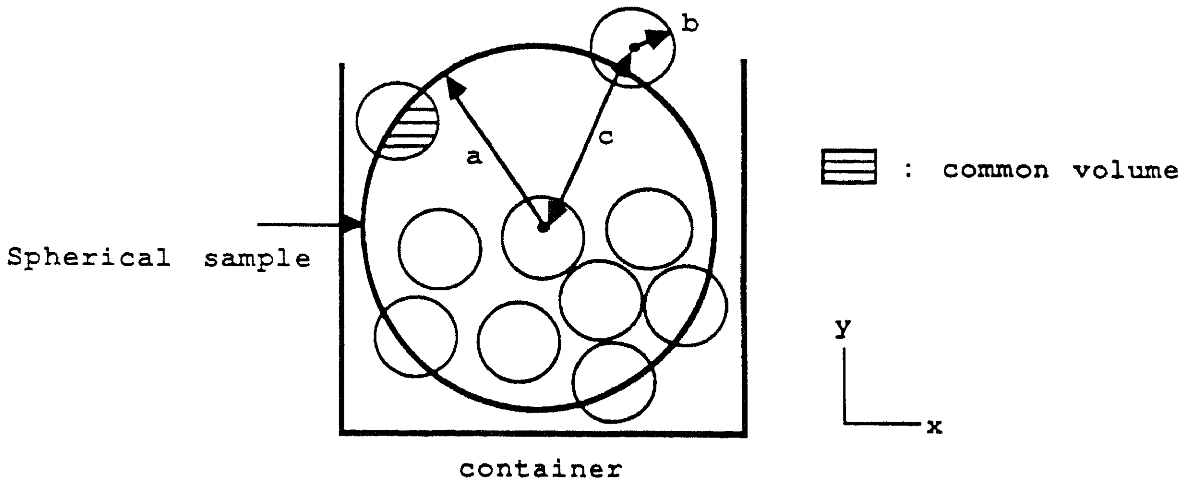


Fig.4.5 The basic algorithm of Spherical Growth Method

The volume common to two spheres [12] of radii **a** and **b**, with centers a distance **c** apart is given by:

$$V = \frac{\pi}{3} \times [2 \times a^3 + 2 \times b^3 + c^3 - 3 \times c \times (d^2 + b^2)]$$

where,

$$|a - b| \le c \le a + b$$

V : common volume

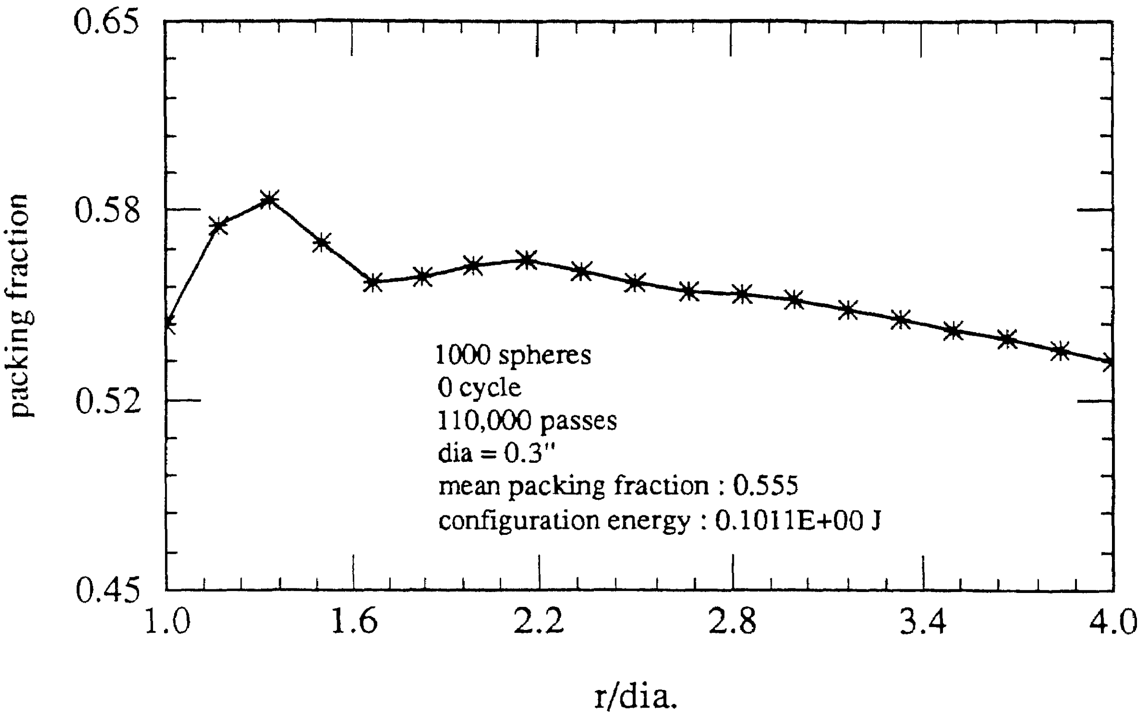
a & b : radii of two spheres

c : distance of centers between two spheres

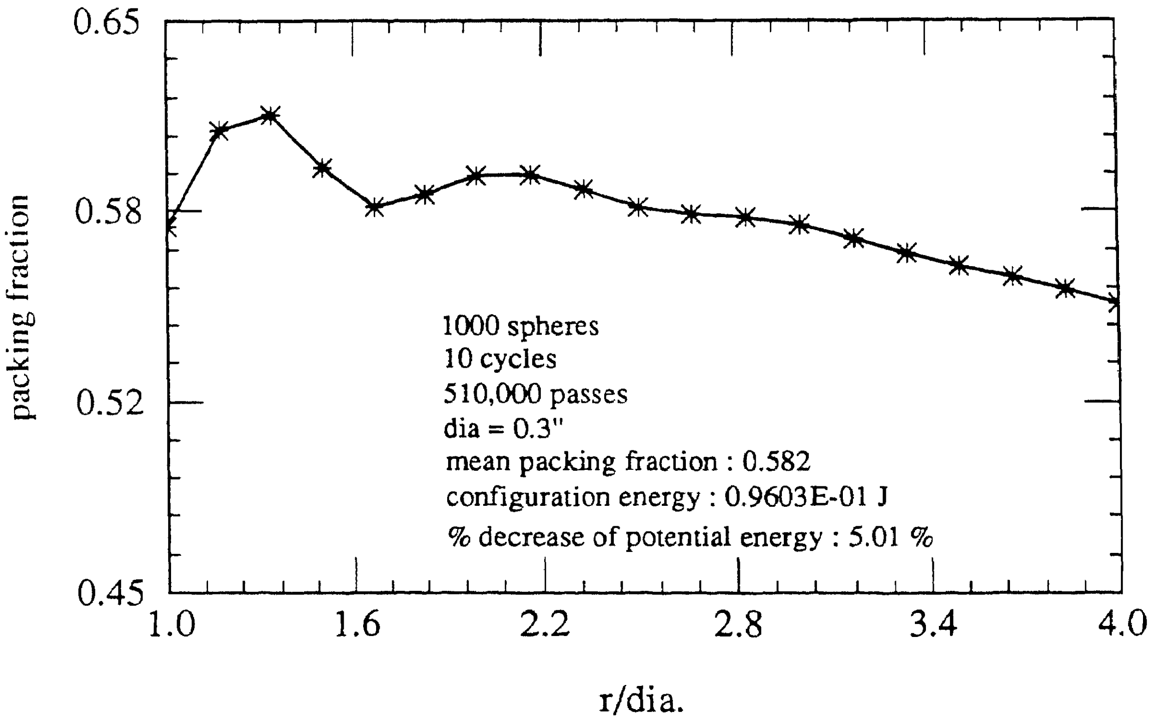
d : $\frac{a^2 + c^2 - b^2}{2 \times c}$

A spherical sample containing central 598 spheres is taken from the packing and the packing fraction is calculated for the intervals of 0.05 sphere diameters. The measured mean packing fractions are 0.555±0.015, 0.582±0.018 for the pouring and the shaking simulation, respectively.

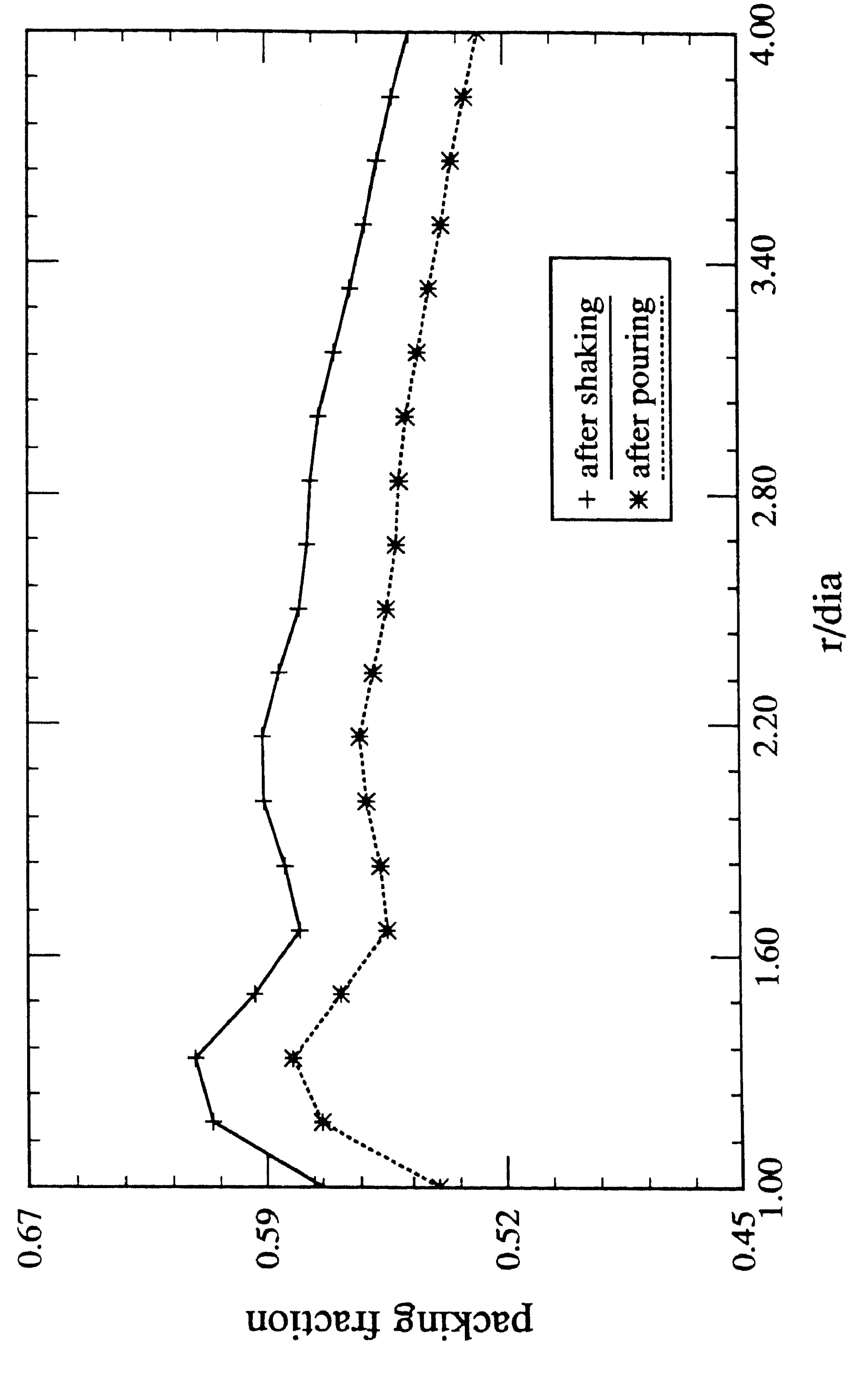
Figures 4.6 (a) and (b) show the packing fractions for the pouring and shaking cases versus r/dia. where dia. equals the diameter of the sphere and r is the radial distance measured outward from the center of the packing. There is found a small peak at 1.33 sphere diameter outward from the center of the packing and the result of *G.D.Scott*[30] shows a similar distribution of packing fraction.



(a) after pouring simulation



(b) after 10 cycles of shaking (40000 passes per cycle)



(c) Comparison of the results between after pouring and shaking

Fig.4.6 packing fractions by Spherical Growth Method

4.2.2 Plane Growth Method

This method first cuts the packing by a plane and calculates the volume of spheres bounded by that plane and the periodic "walls". The details are shown in Figure 4.7.

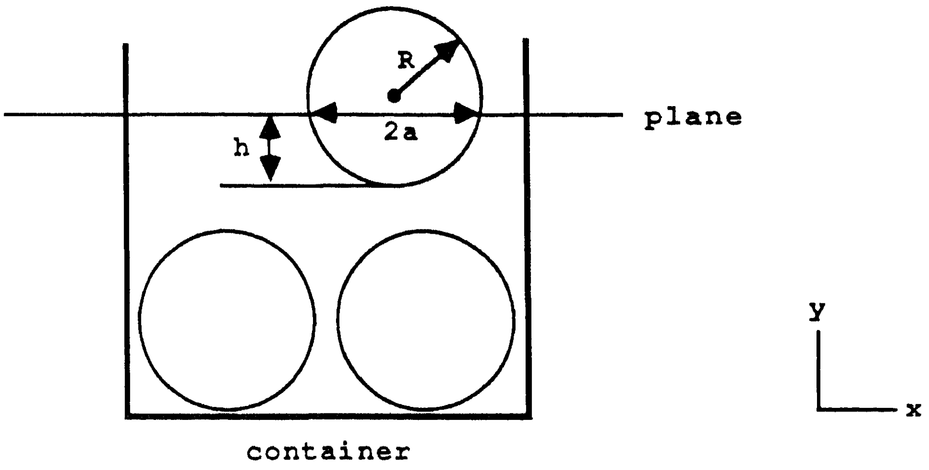


Fig. 4.7 The volume of a spherical segment

The volume of spherical segment of one base [35] is given by:

$$V_s = \frac{1}{6} \times \pi \times h \times (3 \times a^2 + h^2)$$

Where,

V_s : volume of spherical segment

h : height of a spherical segment

a : intersected distance between plane and sphere

$(= \sqrt{h \times (2 \times R - h)}$

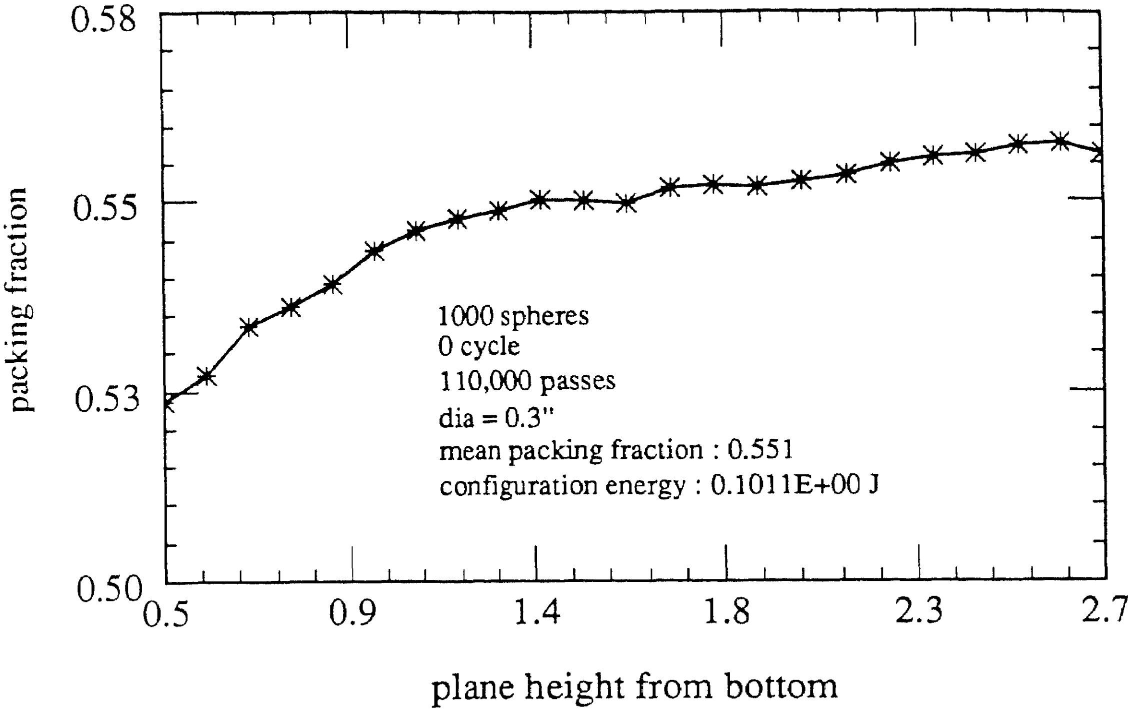
R : radius

The local packing fractions are calculated from the bottom to the top of the packing for the intervals of 0.1 inch. The resulting mean packing fractions are 0.551±0.01 for the pouring case and 0.581±0.006 for 10 cycles of shaking case, which is in a good agreement with the results obtained by spherical growth method. The mean packing fractions by spherical growth method are 0.555±0.015 and 0.582±0.018 for the pouring and shaking, respectively. The published results of *Visscher & Bolsterli, Tory et.al., Powell* and *Gotoh et.al.* show similar packing fractions with the current results.

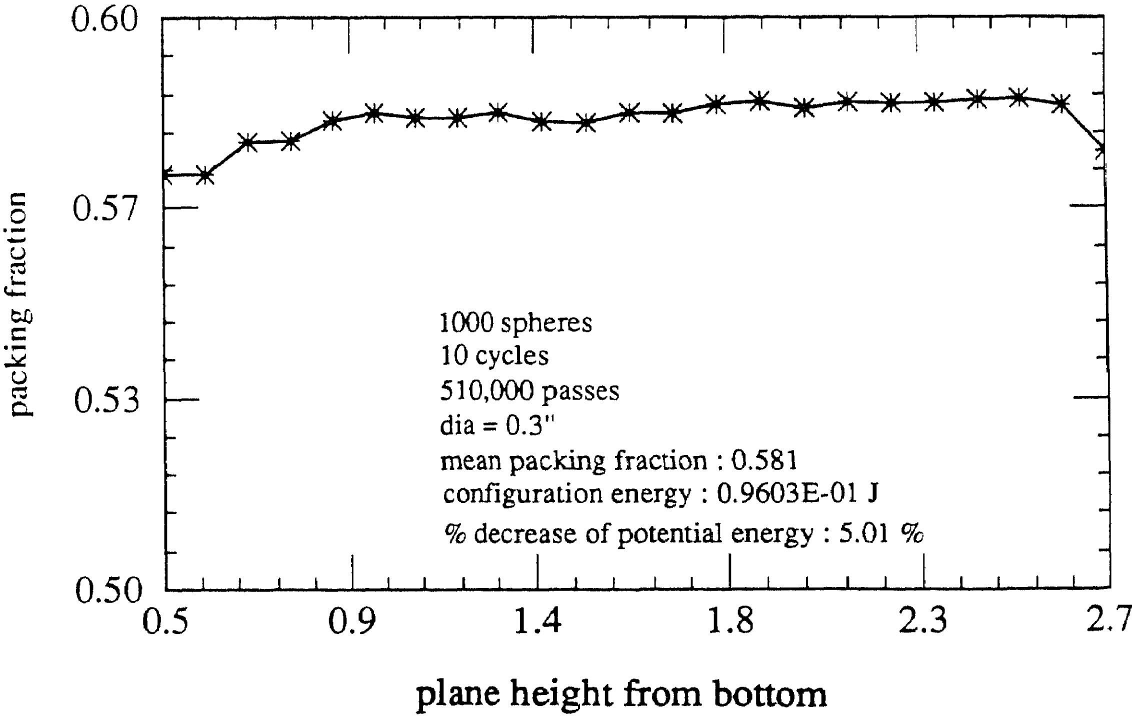
Reference	Packing fractions
Tory, Cochrane & Waddell [10]	0.59
Visscher & Bolsterli [15]	0.582
Tory et.al. [16]	0.58
Powell [20]	0.59
Gotoh et.al.[32]	0.582
Current results	0.581-0.582

Table 4.1 Comparison of the packing fractions

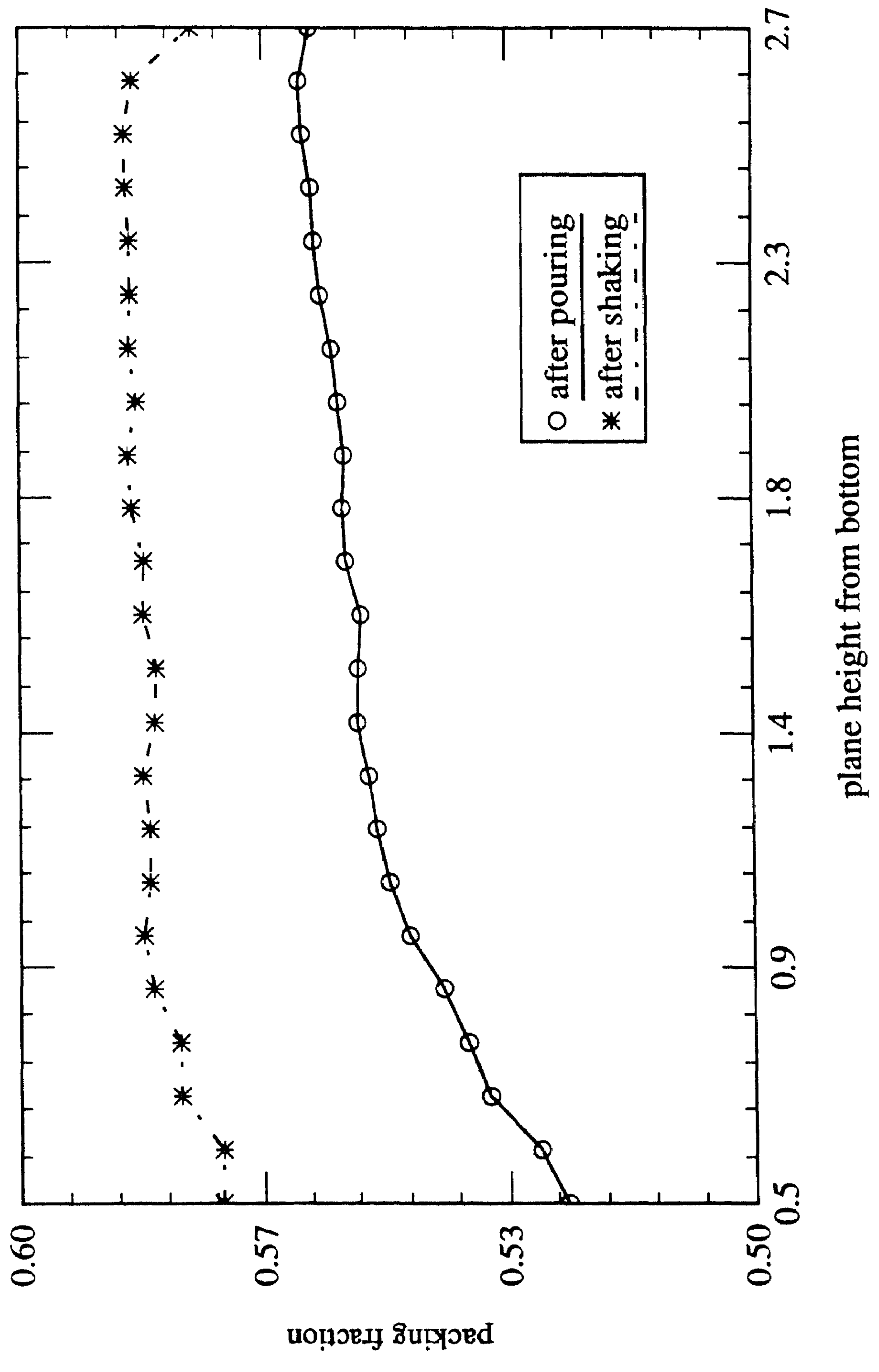
Fig 4.8 shows the distribution of local packing fractions from the bottom of the packing.



(a) After pouring case



(b) After 10 cycles of shaking (40000 passes per cycle)



(c) Comparison of the results

Fig.4.8 packing fractions by plane growth method

4.3 Radial Distribution Function

The radial distribution function [4,24,29,30] is defined as the number of spheres (or density of sphere centers) as a function of distance from the center of the packing. In other words, it is the average number of sphere centers per unit volume in a spherical shell about a central sphere. By the definition, radial distribution function *g (r/D)* is,

$$g\ (r/D) = \frac{N_{av}}{4 \times \pi \times (r/D)^2 \times \Delta(r/D)}$$

where,

N_{av} : average number of sphere centers per interval

$\Delta(r/D)$: interval (= one-fifth of sphere diameter was used)

r : radial distance

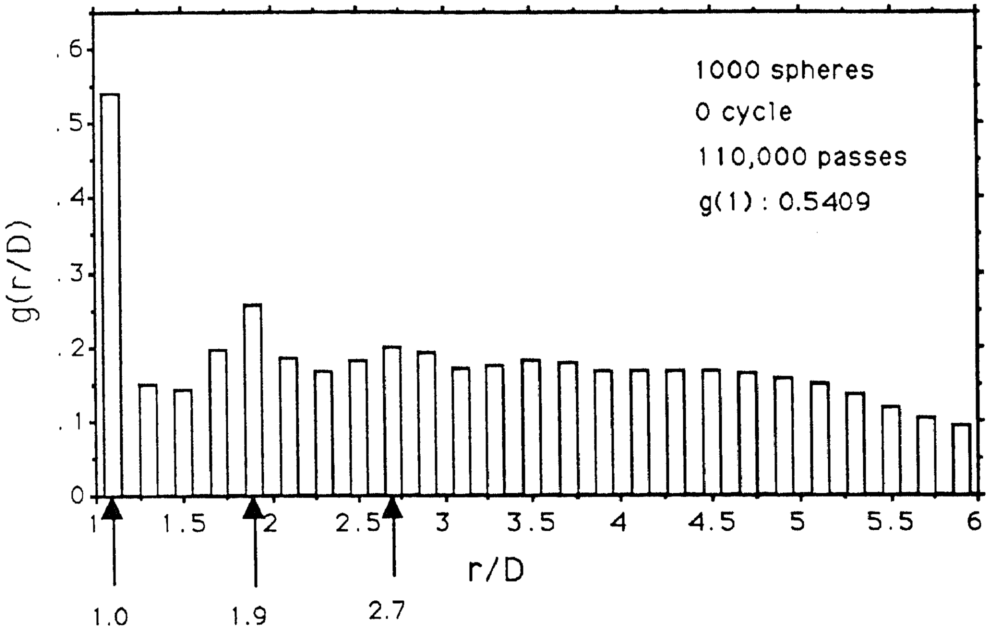
D : sphere diameter

The values of *g*(r/D) is plotted versus r/D and this is shown in Fig.4.7. The measurement was made for a cluster of 1000 spheres and the code listing is found in Appendix B.5. The computed list of data is also found in Appendix C.7 and C.8. Some published values of the radial distances of the first,second, third, fourth and fifth peaks are presented in the Table 4.2.

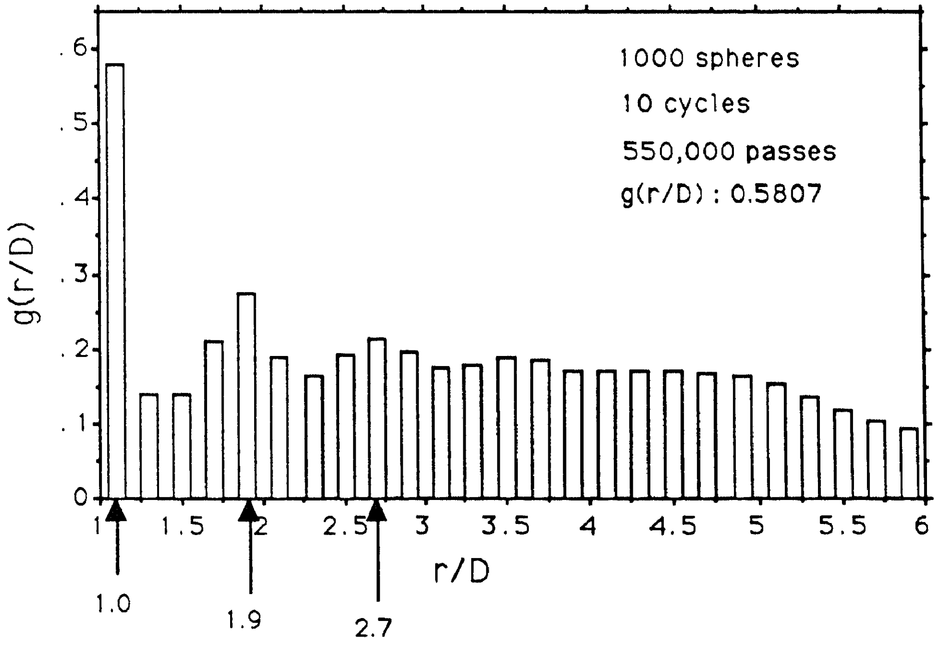
Reference	r/dia. at positions of peaks				
	first	second	third	fourth	fifth
Bennett [12]	1.00	1.73	2.68	3.53	4.38
Finney [13]	1.00	1.73	2.65	3.50	4.35
Matheson [17]	1.00	1.8	2.78	3.64	4.45
Scott [30]	1.00	1.83	2.64	3.45	-
Current result	1.00	1.9	2.7	3.5	4.5

Table 4.2 r/dia. at positions of peaks

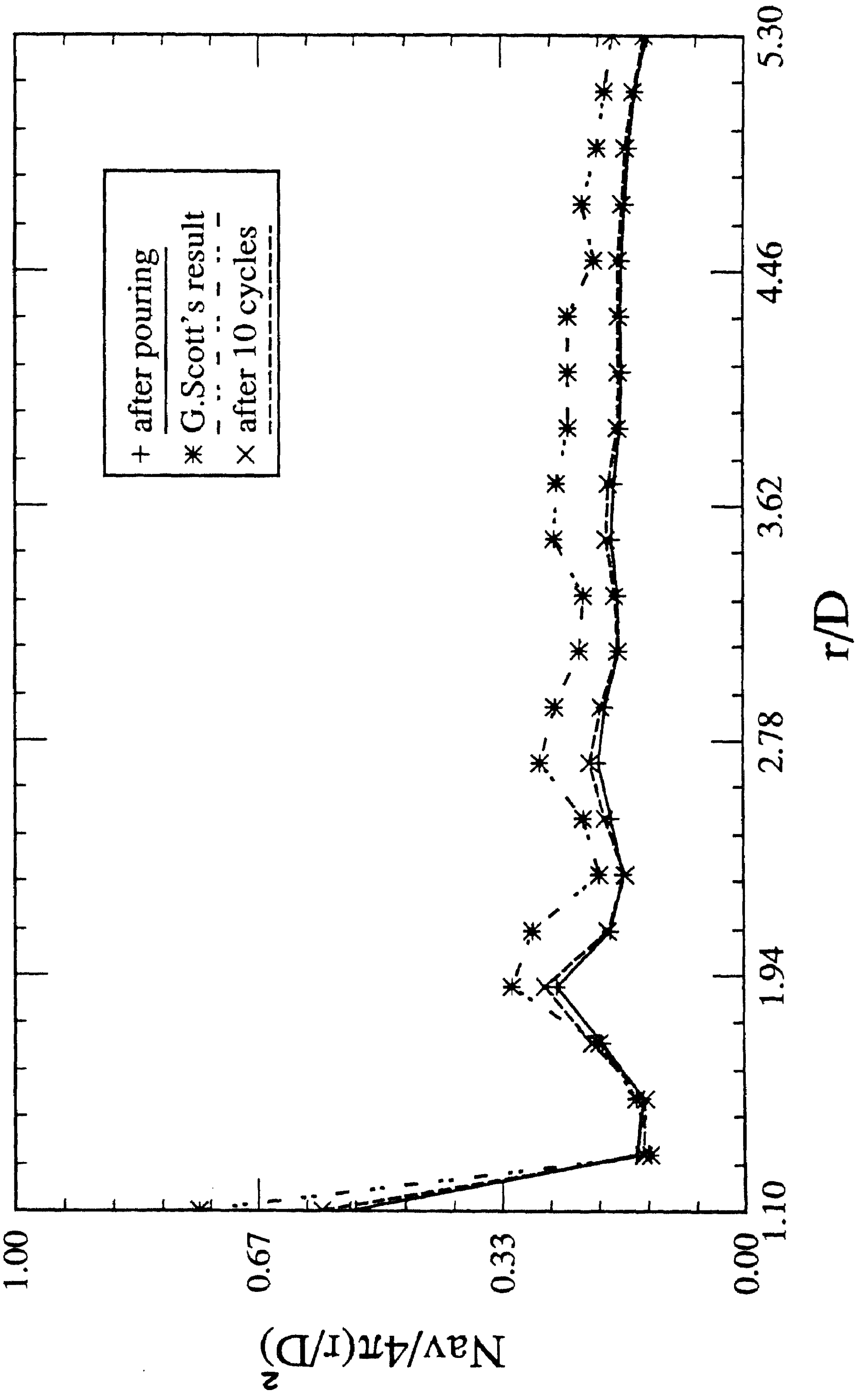
The results for the poured and the shaken assemblies are illustrated in figure 4.9. The first peak in the distribution function lies in the interval 1.0 - 1.1. Since the spheres can not overlap, values of r/D can not occur less than 1.0. The maxima of peaks 2, 3 and 4 of the assembly in pouring case occurred at 1.8, 2.6 and 3.4 sphere diameters. These values are nearly the same as *Scott* [30] and slightly larger than *Matheson*'s values. After 10 cycles of shaken, the value of g at r/D = 1 increased from 0.5409 to 0.5806 and this also appears as an increase of the coordination number. Figure 4.9.c presents the comparison of the *G.D.Scott*'s result with the simulated results.



(a) Pouring case



(b)Shaking case (10 cycles)



(c) Comparison with the published data [30]

Fig.4.9 Radial distribution function for a 1,000 sphere configurations

4.4. System Size Dependence

In order to verify that the results are independent of the number of particles, four different cases were done by varying the numbers of spheres. The cell dimension for each case was 3.0" × 3.0" (base area) × 5.0" (height).

In each case, the spheres were poured into the cell to obtain a configuration in the equilibrated state.

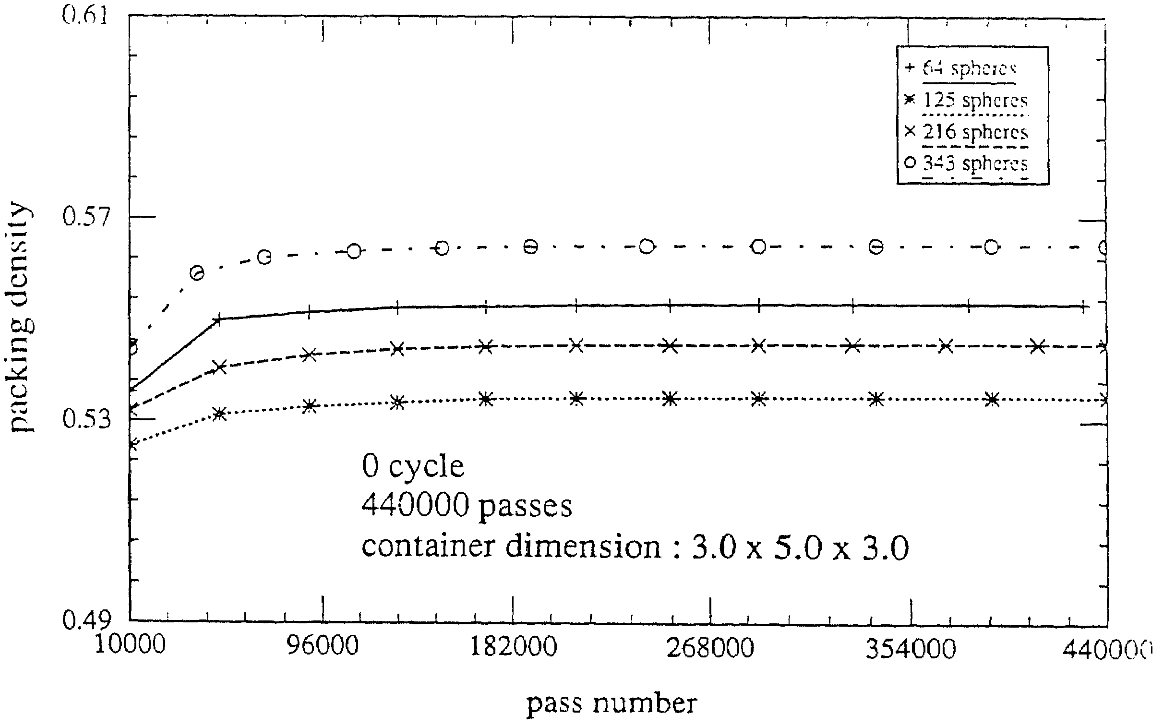
In order to measure the packing densities in a similar condition, the packing of each system was made in a similar height by varying the radius of sphere. The sphere diameters used in this simulation were 0.75" for 64 sphere system, 0.55" for 125 sphere system, 0.45" for 216 sphere system and 0.4" for 343 sphere system. Normalized configuration energy versus pass number shows the height of each system in the equilibrated state. Table 4.4 lists these energies for each size system in the equilibrium. Here Z_i denotes the location of the sphere center above the cell bottom, m_i is the sphere mass and g is the gravitational acceleration.

cases	$\sum_{i=1}^{n} m_i g Z_i$	$\sum_{i=1}^{n} m_i$	$\sum_{i=1}^{n} g Z_i$
64	0.1060	0.2858	0.3709
125	0.06475	0.2201	0.2942
216	0.05622	0.2083	0.2699
343	0.06701	0.2323	0.2885

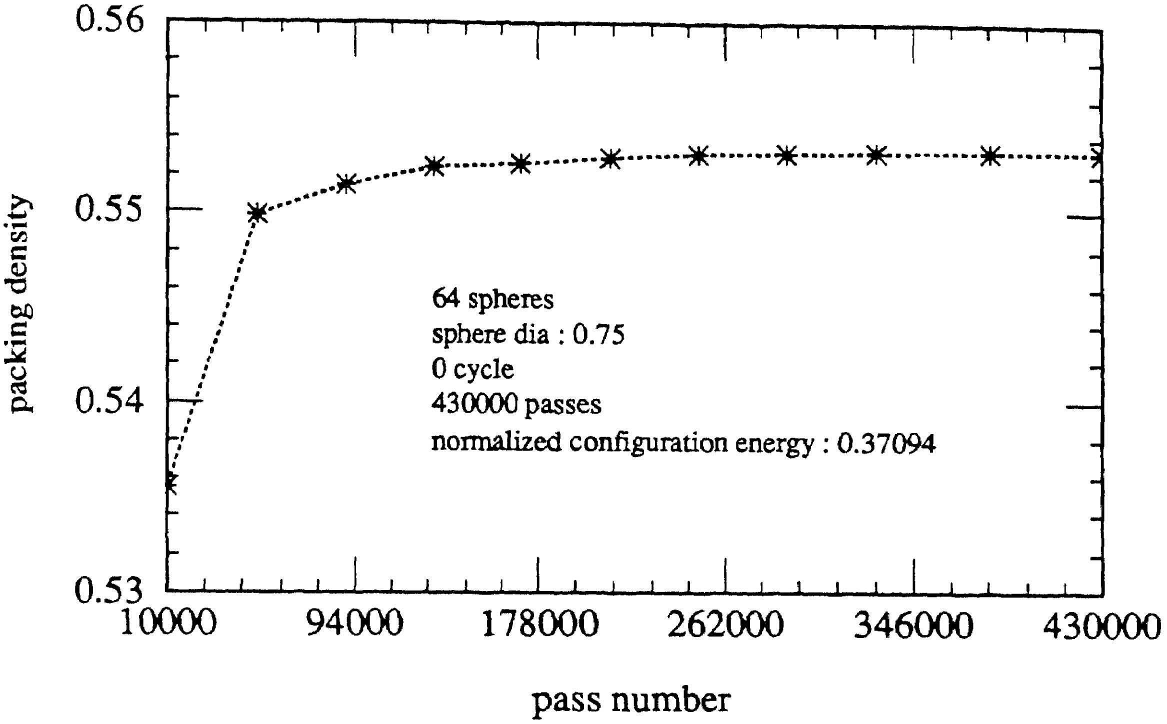
Table 4.2 Normalized configuration energy

This simulation was repeated on systems of 125, 216 and 343 spheres and run for 44,000 passes for each case. The final packing densities are 0.553±0.01 for 64 sphere case, 0.535±0.01 for 125 sphere case, 0.545±0.01 for 216 sphere case and 0.564±0.01 for 343 sphere cases. Each size case were carried 3 times to obtain an average value. Comparing these results with the result of 1000 sphere case, the mean packing density of 1000 sphere case (0.555±0.015)lies within these values. The results of four separate cases also show independence between the system size and the packing density. So the resulting low packing density of current simulation is not affected by the system size.

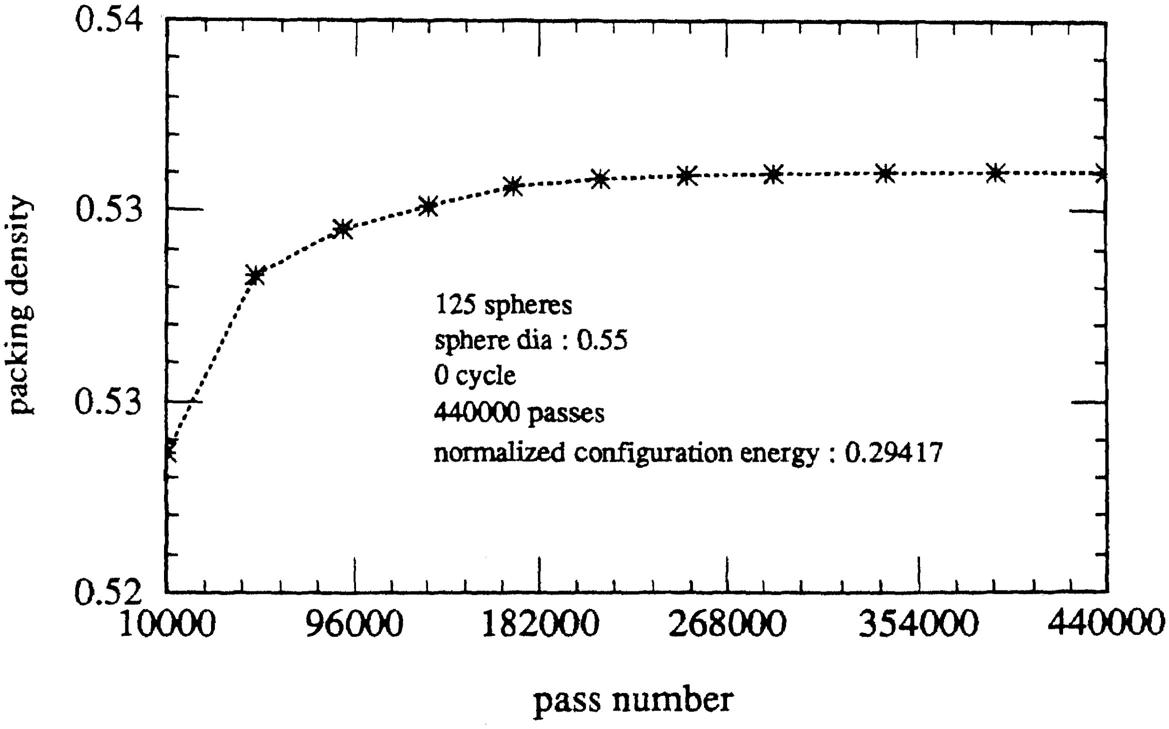
The final packing densities for each case are plotted in Figures 4.10 (a), while (b) - (e) shows the variation versus pass number.



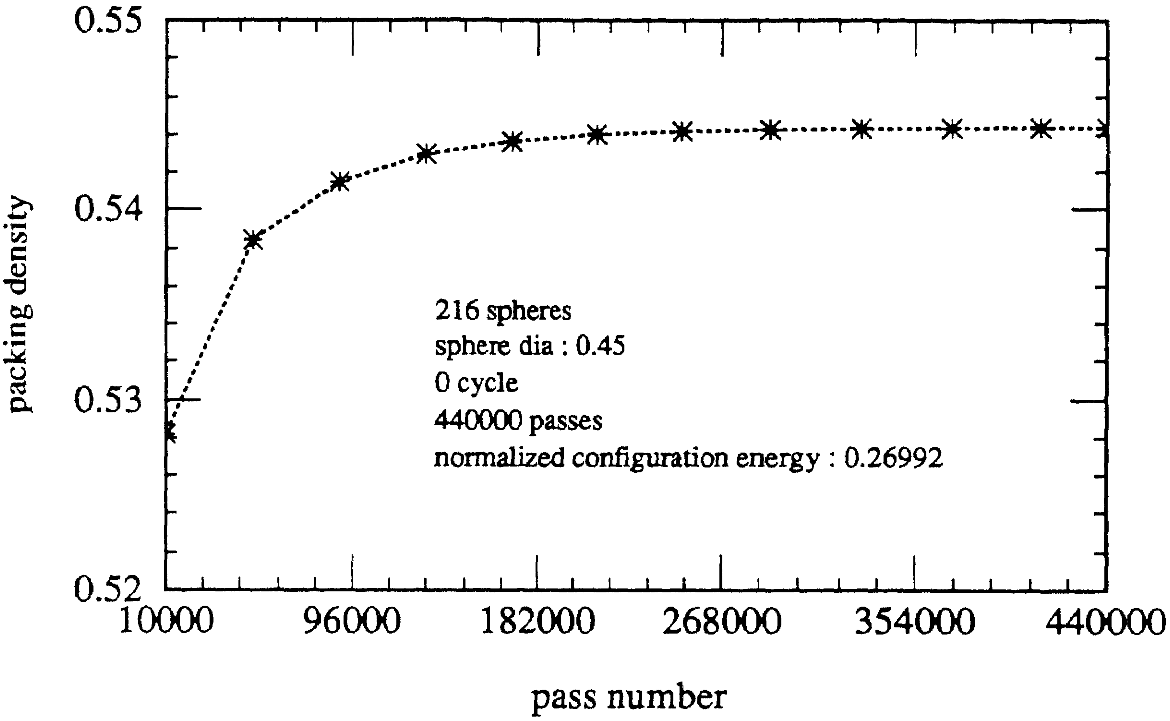
(a) packing densities in each case



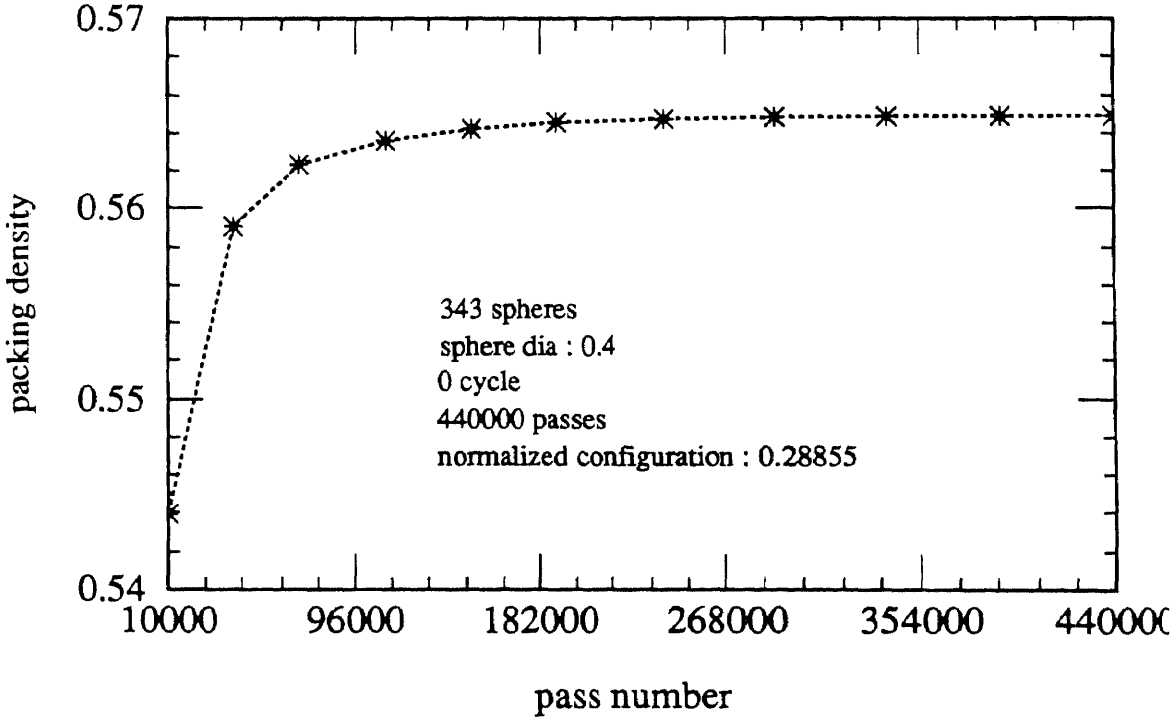
(b) 64 sphere case



(c) 125 sphere case



(d) 216 sphere case



(e) 343 sphere case

Fig.4.10 packing density in each case

4.5. Comparison of the Packing Density between Corrected & Uncorrected Values

This work was done to attempt to discover what could account for the 5 to 6 per cent lower density from published experimental data.

Let the distance between one geometric neighbor and its center sphere be **d**, and let its radii be $\mathbf{r}_1$. In this case the geometric neighbor is in contact with the center sphere if,

$$\mathbf{d} = 2 \times \mathbf{r}_1$$

In the simulation process, the equality can never be exactly obtained because of the machine error.

To locate the nearest neighbor and calculate the distance, a 1000 sphere configuration was produced, processing 415,000 passes. Since there exists only one nearest neighbor, there are 1000 nearest neighbor distances. These distances fell between 1.000000024124545 and 1.017997631992374 sphere diameter. For all practical purpose, the lowerbound is considered to be 1.0 due to machine error.

Let σ be the diameter of the spheres in the packing and P(D) be the cumulative probability [36] that the nearest neighbor is located in the range of $\sigma \le D \le \sigma + d\sigma$. Then, for a fixed packing fraction η, the median nearest neighbor distance $D_{mnn}(\eta)$ is defined by:

$$P(D_{mnn}) = \frac{1}{2}$$

The median nearest neighbors are 1.000006586507291 sphere diameter for the pouring case and 1.000096933545690 sphere diameter after 12 cycles of shaking (20,000 passes per cycle).

The cumulative probability versus normalized distance r/dia is plotted in Figure 4.11.

By using the median value R_{mnn}, to compute the sphere volume, a corrected packing density is calculated as follows:

$$V_{sp} = \frac{4}{3} \times \pi \times (R_{mnn})^3 \times N$$

$$V_{oc} = X_l \times Y_l \times Z_l$$

$$pd_{corr} = \frac{V_{sp}}{V_{oc}}$$

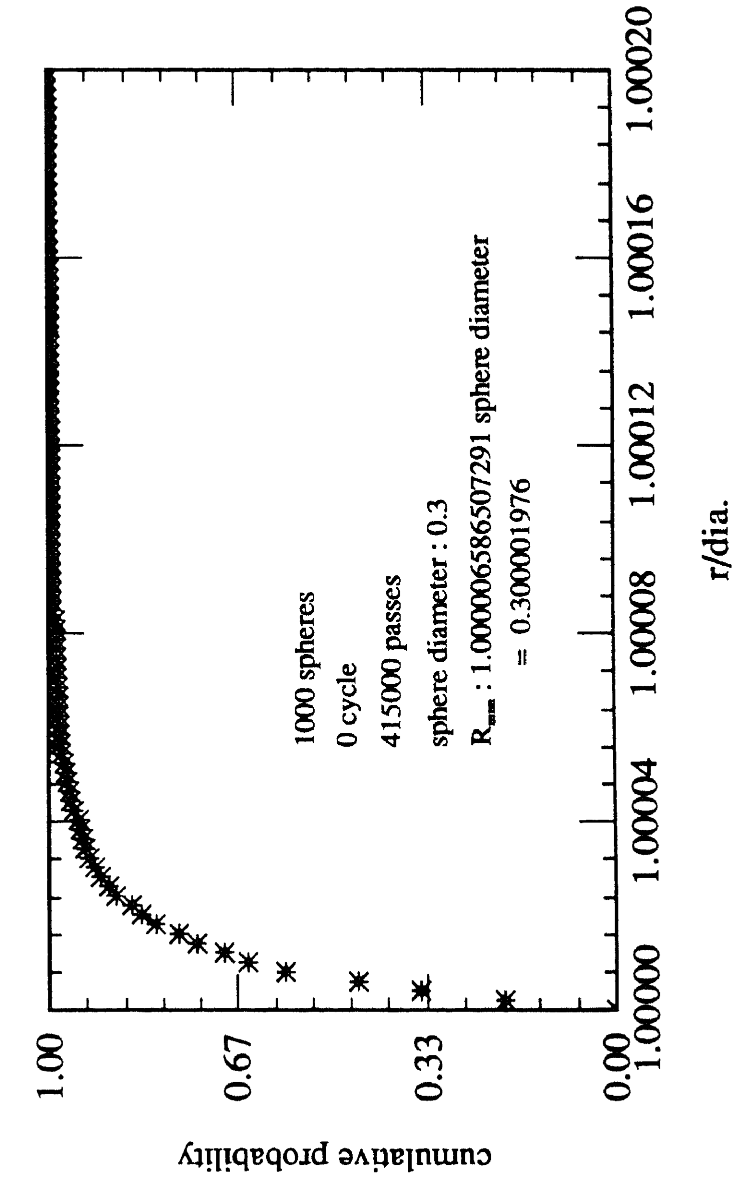
Where,

V_{sp} : total volume of spheres
V_{oc} : occupied volume
pd_{corr} : corrected packing density
N : number of spheres
R_{mnn} : $D_{mnn} / 2$

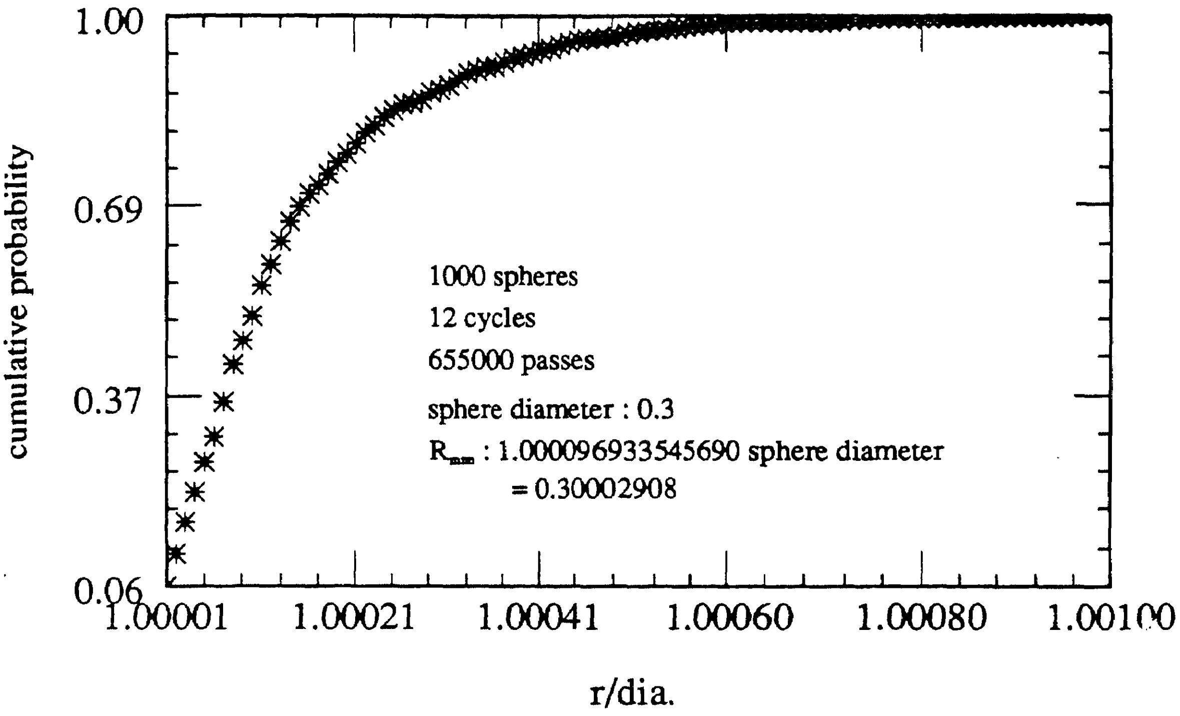
The corrected packing density was 0.5338. The difference between uncorrected and corrected packing densities is 0.002 per cent in the pouring case.

The same procedure was repeated for the shaking case and the corrected packing density was computed to be 0.57923, a very insignificant increase from the uncorrected value of 0.579. The increase was approximately 0.029 per cent.

Because the volume difference between the corrected and the uncorrected one is not significant, the use of the "corrected" sphere diameter cannot be a significant factor in accounting for the 5 to 6 per cent difference between the experimental data and the simulated results.



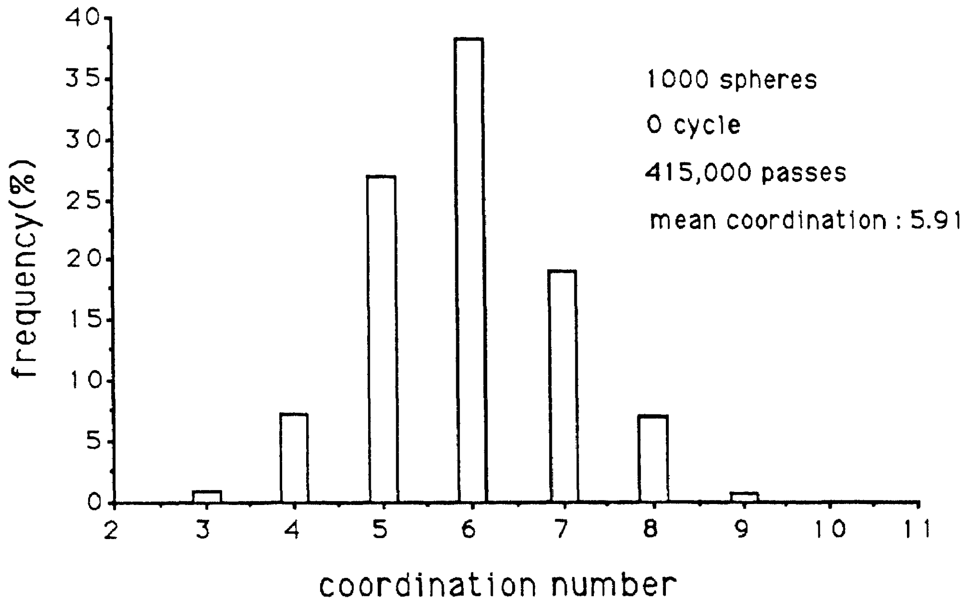
(a) After pouring (415,000 passes).



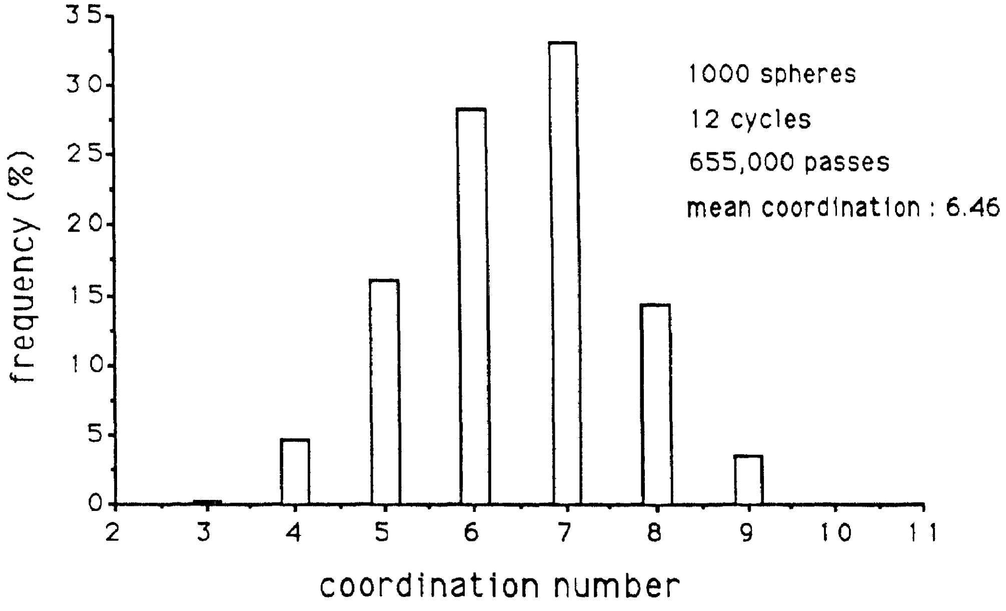
(b) After 12 cycles of shaking

Fig.4.11 The cumulative probability distribution versus the normalized distance

In order to characterize the geometry of these two packings more exactly, the coordination numbers for both cases are also calculated. The mean coordination numbers are 5.91 for the pouring case and 6.46 for the shaking case. The distributions of the coordination numbers are plotted in Figure 4.12.



(a) After pouring



(b) After 12 cycles of shaking (20,000 passes per cycle)

Fig. 4.12 Distribution of the coordination number

4.6. The Annealing Simulation

In the previous simulations, we choose a very large value for $1/k_B T$ as 1.0E+30 where k_B is a Boltzmann constant and T is a absolute temperature. This choice is equivalent to allowing only a downward movement in order to minimize the system potential energy.

Because the value of $1/k_B T$ was so large, the system cooled rapidly and possibly prevented the formation of a greater density. In order to check this factor, the "annealing simulation" was done in a 64 sphere system. The system was heated with a high temperature to an equilibrated state and then slowly cooled by decreasing the temperature of the system.

A normalized temperature, T^*, is defined as follows:

$$T^* = \frac{k_B T}{m_s g \, d_S}$$

where,

k_B : Boltzmann constant (= 1.380×10^{-23} JK^{-1})

T : Absolute temperature (K)

m_s : Mass of a sphere

g : Gravitational velocity

d_s : Diameter of a sphere

The normalization is made as a comparison with the gravitational potential energy.

The annealing simulation was started with an initial value of T^* as 15.81. When the system was brought to an equilibrated state, T^* was changed to a smaller value and again the simulation was run until the system reached another equilibrium state at T^*. Table 4.3 shows the sequence of T^*.The final configuration was obtained after 680,000 passes. The resulting packing density without using the annealing simulation was 0.565 which was computed after 550,000 passes. Comparison of these two results shows approximately 5.2 per cent difference. This gap is almost the same as the difference between the current result and the result of *G.D.Scott.* The values are 0.555 and 0.606 for the current result and the result of *G.D.Scott,* respectively.

The resulting packing densities according to T^* are also shown in Table 4.3.

T^*	pass number	packing density
1.58	120,000	0.249
0.158	280,000	0.504
0.0158	440,000	0.608
0.00316	560,000	0.617
0.00211	680,000	0.618

Table.4.3 The packing densities according to T^*

Eventually the system reached an equilibrated state and the change in the packing density became less than 0.1 per cent.

The result in the annealing simulation shows that the manner in which the system is dropped to a $T^* \simeq 0$. Therefore, this is a significant factor in accounting for the deficit between experimental data and the simulated result.

5. SUMMARY & CONCLUSION

The random packing of spheres is a process of considerable scientific interest and practical importance. A variety of simple models have been developed to obtain a better understanding of technically important processes.

In this work, a Monte Carlo simulation code [1] has been extended from two dimension to three dimension and then to investigate the properties of ranom packing of hard spheres.

With configurations of 1000 spheres obtained from the simulation code,the properties of the assemblies are calculated and compared with other results in many ways such as coordination number, packing density and radial distribution function *etc.*.

The followings are the results and the conclusions:

(1) The coordination numbers for 3 different diameter separations (*ie.,* 1.01, 1.05 and 1.1) are calculated and compared with the published data. All the comparisons are based on using the 1.05 sphere diameter separation because the experimental and computer simulated results of *Bernal et.al.* and *A.J.Matheson* uses same tolerances. In the pouring simulation, the peak value occurred at a coordination number of approximately six similar to the result obtained by *Jodrey et.al.*. In shaking simulation studies, the peak value occurred at seven coordination, but the average number is lower than the experimental results of *Bernal et.al.*. The average coordination numbers are 6.55 for the shaking simulation and 7.99 for *Bernal et.al.*.

(2) Another way of characterizing the bulk configuration of the system is the packing fraction. This quantity is measured by

two different methods called Spherical Growth Method and Plane Growth Method. A good agreement for the results is obtained by both methods. In the case of pouring simulation, the packing fractions are 0.555±0.015 by the spherical growth method and 0.551±0.01 by the plane growth method. The packing fractions obtained from the shaking simulation give a packing fraction of 0.582±0.018 using the "Spherical Growth Method" and 0.581±0.006 using the "Plane Growth Method". Both in the small and large systems, the results from the plane growth method show a good accuracy.

Comparisons with the published data shows that the resulting packing fractions are approximately 5 to 6 per cent lower than the experimental results of *G.D.Scott.*

Three case studies are done to find the significant factor in accounting for the 5 to 6 per cent deficit of the packing density between the cited experimental results. The following are the results and conclusions ofthree case studies.

(4) To check the dependence of the system size, 4 separate cases were simulated. The simulation was repeated 3 times each using an identically sized cell. Each study was allowed to run for 400,000 passes to obtain final equilibrium configurations. The final averaged packing densities for each case do not show the dependence between the system size and the packing density. Hence it is concluded, the simulation results are not system-size dependent.

(5) Comparison of the "corrected" and the "uncorrected packing densities" was done to attempt to discover what could account for the 5 to 6 per cent lower density than published

experimental results. The corrected packing density was computed using the median nearest neighbor distance and compared with the results of the uncorrected packing density. The resulting uncorrected and corrected packing densities in the pouring case were 0.5335 and 0.5338, respectively. In the shaking case, the resulting uncorrected and corrected packing densities were 0.579 and 0.57923, respectively. The differences between two cases were only 0.002 per cent and 0.029 per cent. So the use of the corrected sphere diameter cannot be a significant factor.

(6) The annealing simulation was done in an attempt to determine if the way the system was cooled effected the density. This process excluded the possibility which could prevent the formation of a greater density because of the rapid cooling of the system. The system was first heated with a high temperature then slowly cooled. The results showed approximately 5.2 per cent increase of the packing density as compared with the rapid cooling results. The packing densities are 0.565 for the rapid cooling simulation and 0.618 after annealing simulation. It is found that the method of dropping the system is a critical factor effecting the packing density and this could account for 5 to 6 per cent difference between the experimental data and the simulated results. Many case studies are necessary to verify this. However, the preliminary results cited here indicate that the claim is true.

Some aspects for further research are as follows:

(1) Find an optimized sequence of values T^* from the results of annealing simulation on various sytem sizes.

(2) The Voronoi diagram may be used in the analysis of the

results. If this diagram is applied in the analysis, the exact value of the coordination number and its distribution are obtained. In this way, the configurarion may be looked at on a local level.

REFERENCES

1. A.Rosato, F.Printz, K.J.Standdburg and R.Swendsen, "Monte Carlo Simulation of Particulate Matter Segregation," *Powder Technology,* **49,** 59-69(1986)

2. J.D.Bernal and J.Mason, "Co-ordination of Randomly Packed Spheres," *Nature,* **188,** 910-911(1960)

3. G.D.Scott, "Packing of Spheres," *Nature,* **188**, 908-909(1960)

4. G.Mason, "Radial Distribution Functions from Small Packings of Spheres," *Nature,* **217,** 733-735(1968)

5. J.D.Bernal, "A Geometrical Approach to the Structure," *Nature,* **183,** 141-146(1959)

6. R.K.McGeary, "Mechanical Packing of Spherical Particles," *J.Am.Ceram.Soc.*, **44,** 513-522(1961)

7. J.D.Bernal, "Geometry of the Structure of Monotomic Liquids," *Nature,* **185,** 68-70(1960)

8. J.D.Bernal and J.L.Finney, "Random packing of spheres in non-rigid containers," *Nature,* **214,** 265-266(1967)

9. J.D.Bernal, J.Mason and K.R.Knight, "Radial Distribution of the Random Close Packing of Equal Spheres," *Nature,* **194,** 957-958(1962)

10. E.M.Tory, N.A.Cochrane, and S.R.Waddell, "Anisotropy in Simulated Random Packing of Equal Spheres," *Nature,* **220,** 1023-1024(1968)

11. G.D.Scott and D.L.Mader, "Angular Distribution of Random Close-packed Equal Spheres," *Nature,* **201,** 382-383(1964)

12. C.H.Bennett, "Serially Deposited Amorphous Aggregates of Hard Spheres," *J.Appl.Phys.,* **6,** 2727-2733(1972)

13. J.L.Finney, "Random Packing and the Structure of Simple

Liquids. 1. The Geometry of Random Close Packing," *Proc.Roy.Soc.Lond.A.,* **319,** 479-493(1970)

14. D.J.Adams and A.J.Matheson, "Computation of Dense Random Packing of Hard Spheres," *J.Chem.Phys.* **56,** 1989-1994(1972)

15. W.M.Visscher and M.Bolsterli, "Random Packing of Equal and Unequal Spheres in Two and Three Dimensions," *Nature,* **239,** 504-507(1972)

16. E.M.Tory, B.H.Church, M.K.Tam, and M.Ratner, "Simulated Random Packing of Equal Spheres," *Can.J.Chem.Eng.,* **51,** 484-493(1973)

17. A.J.Matheson, "Computation of a Random Packing of Hard Spheres," *J.Phys.,* **7,** 2569-2576(1974)

18. Keishi Gotoh and J.L.Finney, "Statistical Geometrical Approach to Random Packing Density of Equal Spheres," *Nature,* **252,** 202-205(1974)

19. W.S.Jodrey and E.M.Tory, "Simulation of Random Packing of Spheres," *Simulation,* 1-12(Jan.1979)

20. M.J.Powell, "Computer-Simulated Random Packing of Spheres," *Powder Technology,* **25,** 45-52(1980)

21. W.S.Jodrey and E.M. Tory, "Computer Simulation of Isotropic, Homogeneous,Dense Random Packing of Equal Spheres," *Powder Technology,* **30,** 111-118(1981)

22. J.Rodriguez, C.H.Allibert, and J.M.Chaix, "A Computer Method for Random Packing of Spheres of Unequal Size," *Powder Technology,* **47,** 25-33(1986)

23. D.P.Haughey and G.S.G. Beveridge, "Structural Properties of Packed Bed - A Review," *Can.J.Chem.Eng.,* **47,** 130-140(1969)

24. D.J.Cumberland and R.J.Crawford, "The Packing of Particles," *Elsevier Science, Amsterdam, The Netherlands,* Chapter 1 - 3(1987)

25. W.W.Wood and F.R. Parker, "Monte Carlo Equation of State of Molecules Interacting with the Lennard-Jones Potential. I. A Super critical Isotherm at about Twice the Critical Temperature," *J.chem.Phys.* **27**, 720-733(1957)

26. G.D.Scott and D.M.Kilgour, "The Density of Random Close Packing of Spheres," *Brit.J.Appl.Phys.(J.Phys.D)*, **2**, 863-866(1969)

27. J.D.Bernal and S.V.King, "Random Close-Packed Hard-Sphere Model," *Discuss.Faraday Soc.*, **43**, 60-69(1967)

28. M.N.Rosenbluth and A.W.Rosenbluth, "Further Results on Monte Carlo Equation of State," *J.Chem.Phys.*, **22**, 881-884(1954)

29. B.J.Alder, S.P.Frankel, and V.A.Lewinson, "Radial Distribution Function Calculated by the Monte Carlo Method for a Hard Sphere Fluid," *J.Chem.Phys.* **23**, 417-419(1955)

30. G.D.Scott, "Radial Distribution of the Random Close Packing of Equal Spheres," *Nature*, **194**, 956-957(1962)

31. G.Mason, "General Discussion," *Discuss.Faraday Soc.*, **43**, 75-88(1967)

32. K.Gotoh ,W.S.Jodrey and E.M.Tory, "A Random Packing Structure of Equal Spheres - Statistical Geometrical Analysis of Tetrahedral Configurations," *Powder Technology*, **20**, 233-242(1978)

33. J.D.Bernal, I.A. Cherry, J.L.Finney and K.R.Knight, "An Optical Machine for Measuring Sphere Coordinates in Random Packings," *J.Phys.E*, **3**, 388-390(1970)

34. H.Susskind and W.Becker, "Random Packing of Spheres in Non-rigid Containers," *Nature*, **212**, 1564-1565(1966)

35. R.C.Weast, S.M.Selby and C.D.Hodgman, "Handbook of Mathematical Tables," *The Chemical Rubber Co., Cleveland, Ohio, U.S.A.*, 587-589(1964)

36. J.G.Berryman, "Random Close Packing of Hard Spheres and

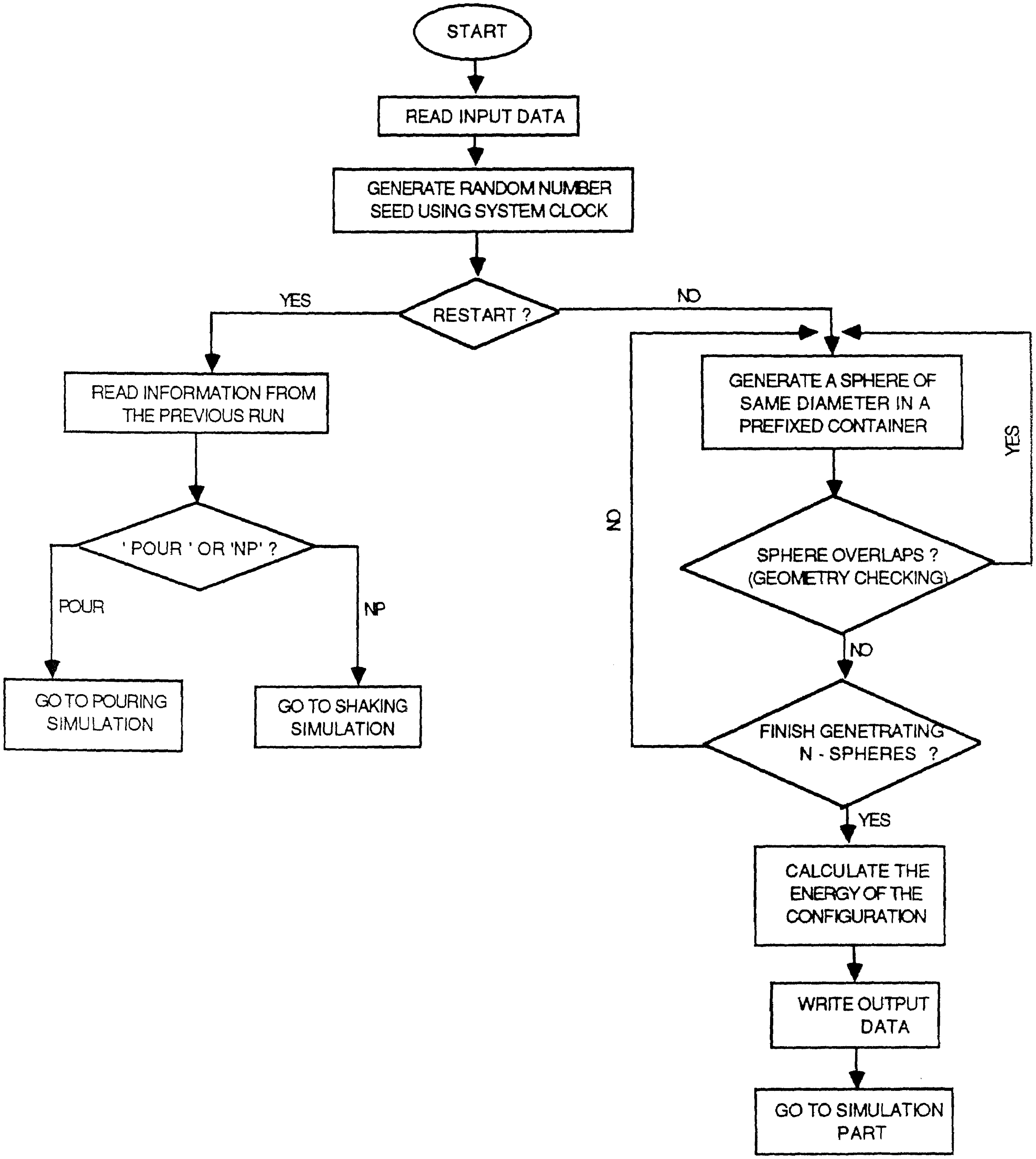
Disks," *Phys. Rev. A,* **27**, 1053-1061(1983)

37. A. Rosato, K. J. Strandburg, F. Prinz and R. H. Swendsen, "Why the Brazil Nuts are on Top: Size Segregation of Particulate Matter by Shaking," *Phys. Rev. Lett.*, **58**, 1038-1040(1987)

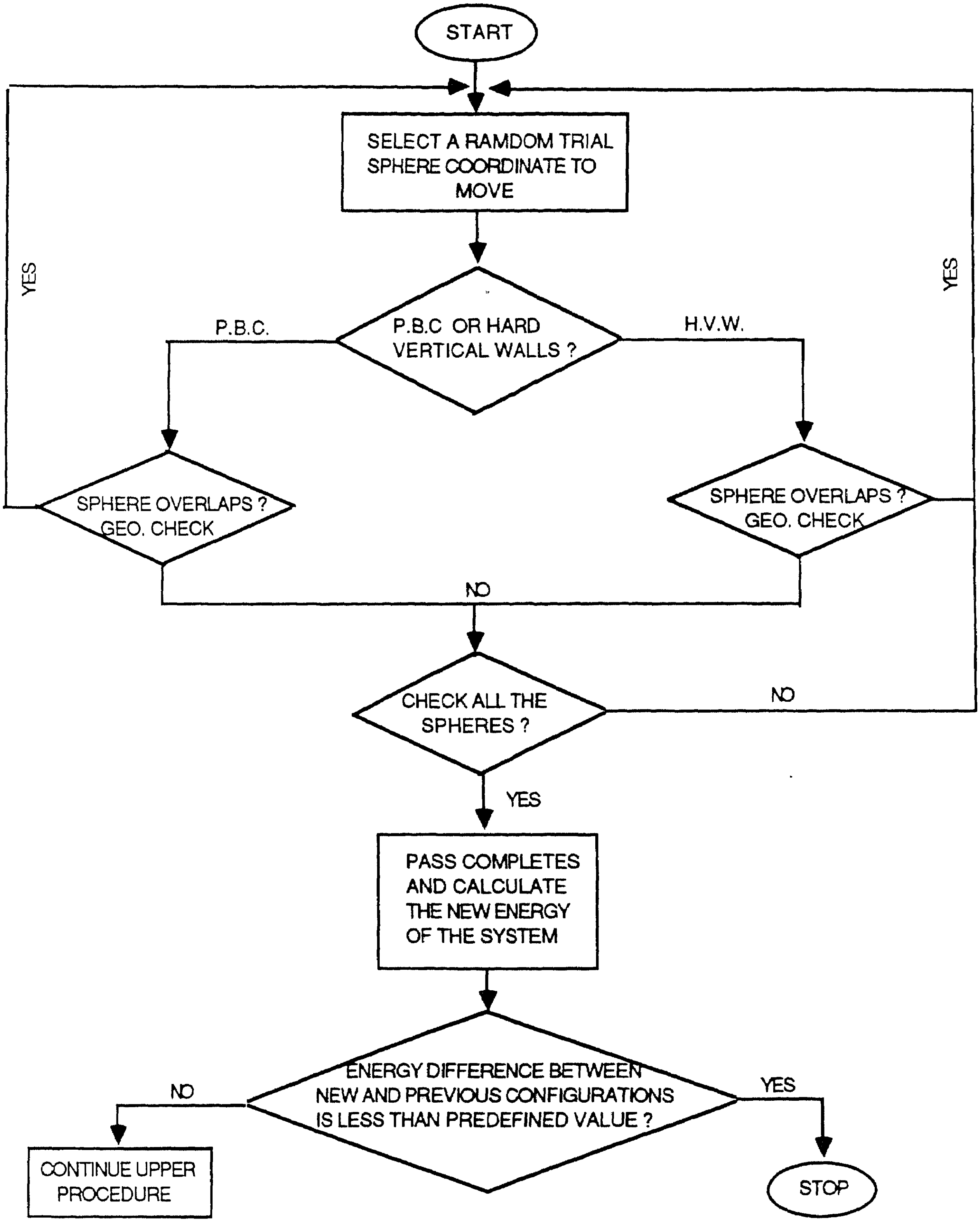
APPENDIX A :

A.1 Algorithm for Monte Carlo simulation code

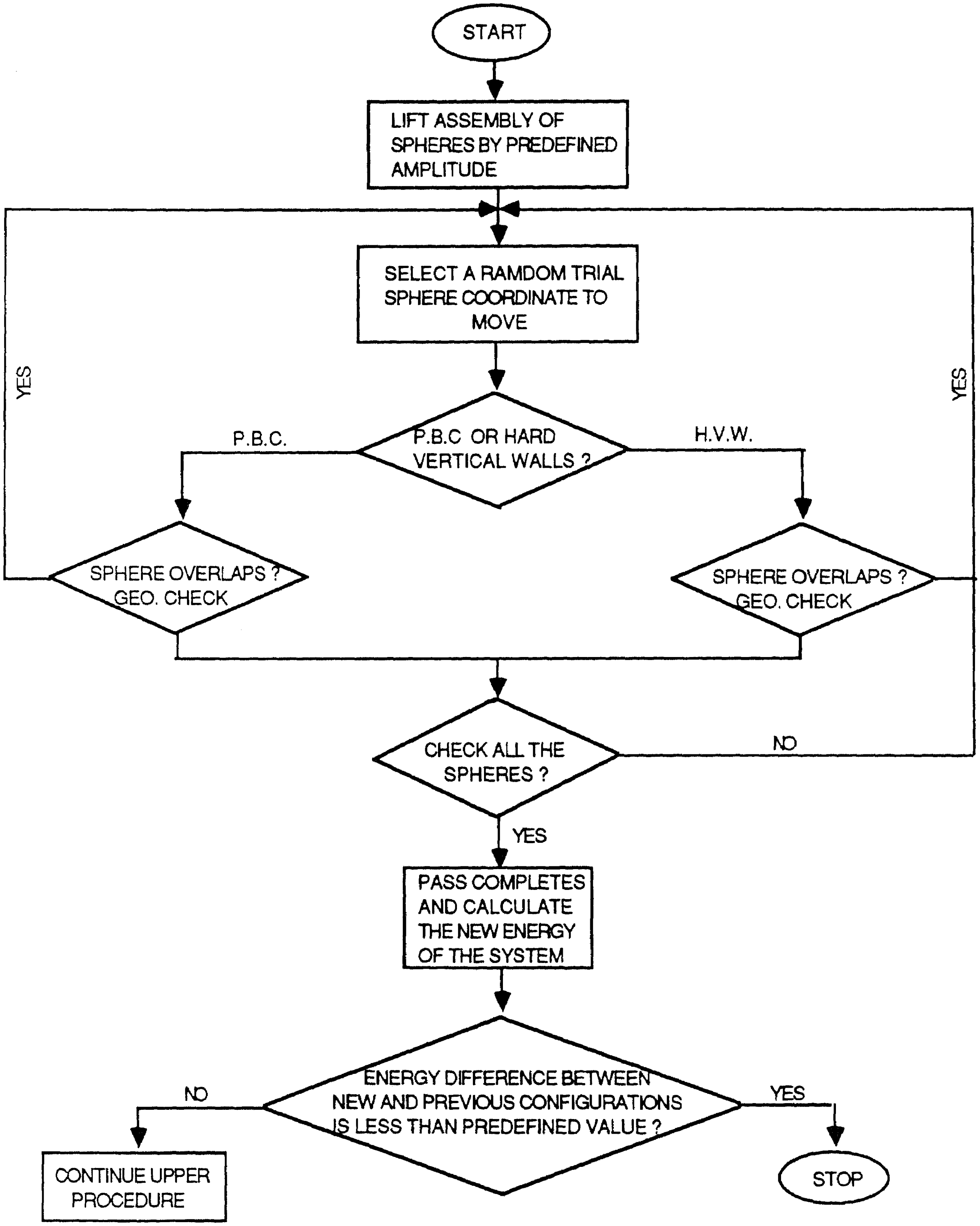
(a) Initial configuration


START
READ INPUT DATA
GENERATE RANDOM NUMBER SEED USING SYSTEM CLOCK
RESTART ?
YES
NO
READ INFORMATION FROM THE PREVIOUS RUN
' POUR ' OR 'NP' ?
POUR
NP
GO TO POURING SIMULATION
GO TO SHAKING SIMULATION
GENERATE A SPHERE OF SAME DIAMETER IN A PREFIXED CONTAINER
SPHERE OVERLAPS ? (GEOMETRY CHECKING)
YES
NO
FINISH GENETRATING N - SPHERES ?
NO
YES
CALCULATE THE ENERGY OF THE CONFIGURATION
WRITE OUTPUT DATA
GO TO SIMULATION PART

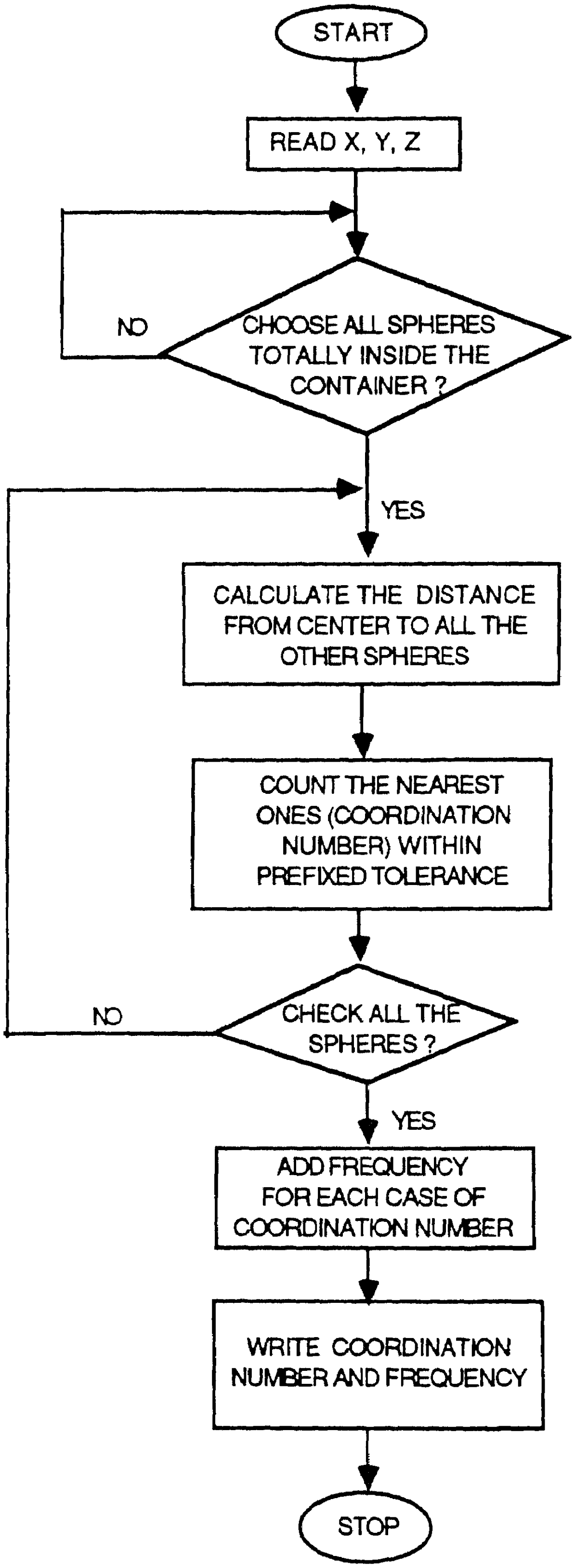
(b) Pouring simulation : begin moving spheres and generating new configurations


START
SELECT A RAMDOM TRIAL SPHERE COORDINATE TO MOVE
YES
YES
P.B.C.
P.B.C OR HARD VERTICAL WALLS ?
H.V.W.
SPHERE OVERLAPS ? GEO. CHECK
SPHERE OVERLAPS ? GEO. CHECK
NO
CHECK ALL THE SPHERES ?
NO
YES
PASS COMPLETES AND CALCULATE THE NEW ENERGY OF THE SYSTEM
NO
ENERGY DIFFERENCE BETWEEN NEW AND PREVIOUS CONFIGURATIONS IS LESS THAN PREDEFINED VALUE ?
YES
CONTINUE UPPER PROCEDURE
STOP

(c) Shaking simulation : begin moving spheres with amplitude and generating new configurations

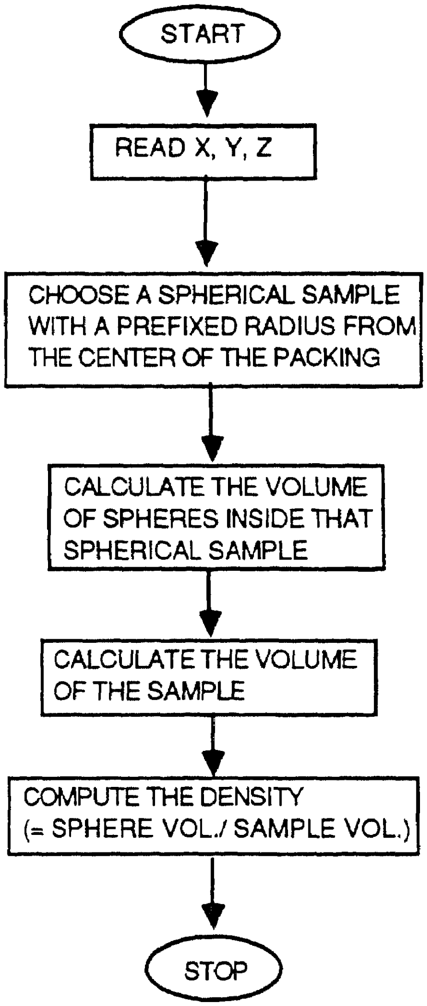

START
LIFT ASSEMBLY OF SPHERES BY PREDEFINED AMPLITUDE
SELECT A RAMDOM TRIAL SPHERE COORDINATE TO MOVE
YES
YES
P.B.C.
P.B.C OR HARD VERTICAL WALLS ?
H.V.W.
SPHERE OVERLAPS ? GEO. CHECK
SPHERE OVERLAPS ? GEO. CHECK
CHECK ALL THE SPHERES ?
NO
YES
PASS COMPLETES AND CALCULATE THE NEW ENERGY OF THE SYSTEM
NO
ENERGY DIFFERENCE BETWEEN NEW AND PREVIOUS CONFIGURATIONS IS LESS THAN PREDEFINED VALUE ?
YES
CONTINUE UPPER PROCEDURE
STOP

A.2 Algorithm for Coordination Number

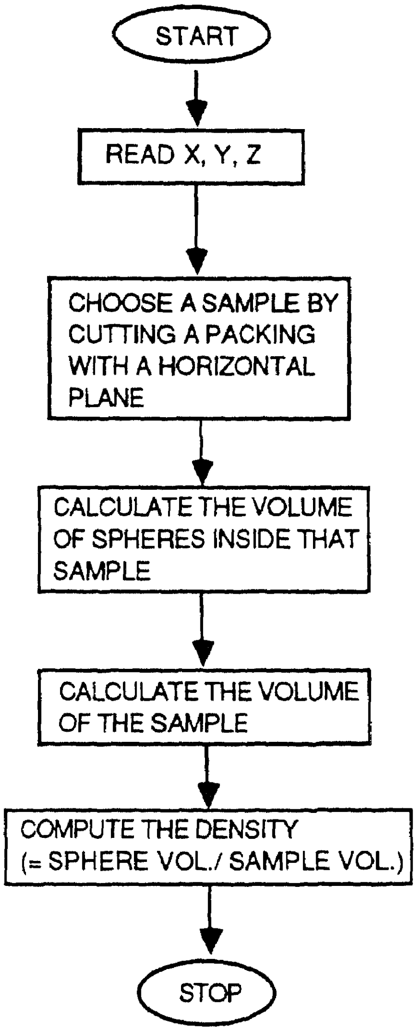

START
READ X, Y, Z
NO
CHOOSE ALL SPHERES TOTALLY INSIDE THE CONTAINER ?
YES
CALCULATE THE DISTANCE FROM CENTER TO ALL THE OTHER SPHERES
COUNT THE NEAREST ONES (COORDINATION NUMBER) WITHIN PREFIXED TOLERANCE
NO
CHECK ALL THE SPHERES ?
YES
ADD FREQUENCY FOR EACH CASE OF COORDINATION NUMBER
WRITE COORDINATION NUMBER AND FREQUENCY
STOP

A.3 Algorithm for Packing Fraction

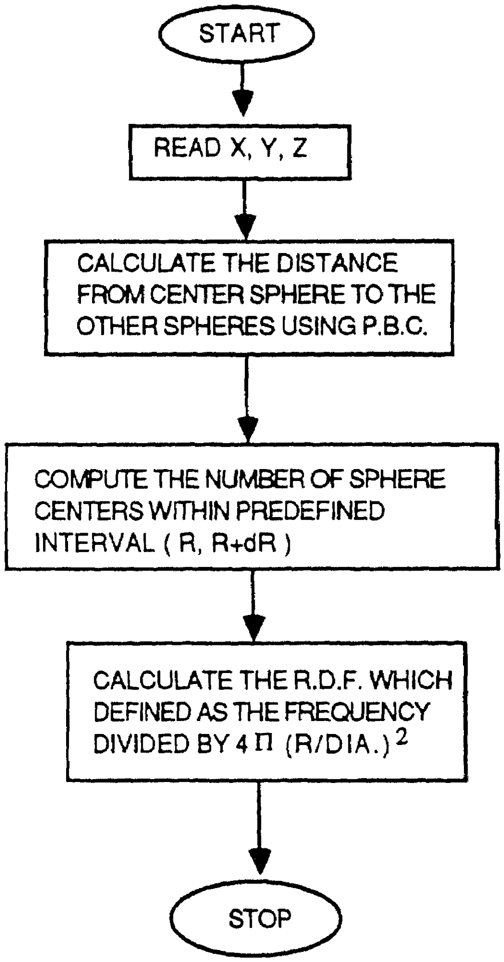
(a) Spherical Ggrowth Method


START
READ X, Y, Z
CHOOSE A SPHERICAL SAMPLE WITH A PREFIXED RADIUS FROM THE CENTER OF THE PACKING
CALCULATE THE VOLUME OF SPHERES INSIDE THAT SPHERICAL SAMPLE
CALCULATE THE VOLUME OF THE SAMPLE
COMPUTE THE DENSITY (= SPHERE VOL./ SAMPLE VOL.)
STOP

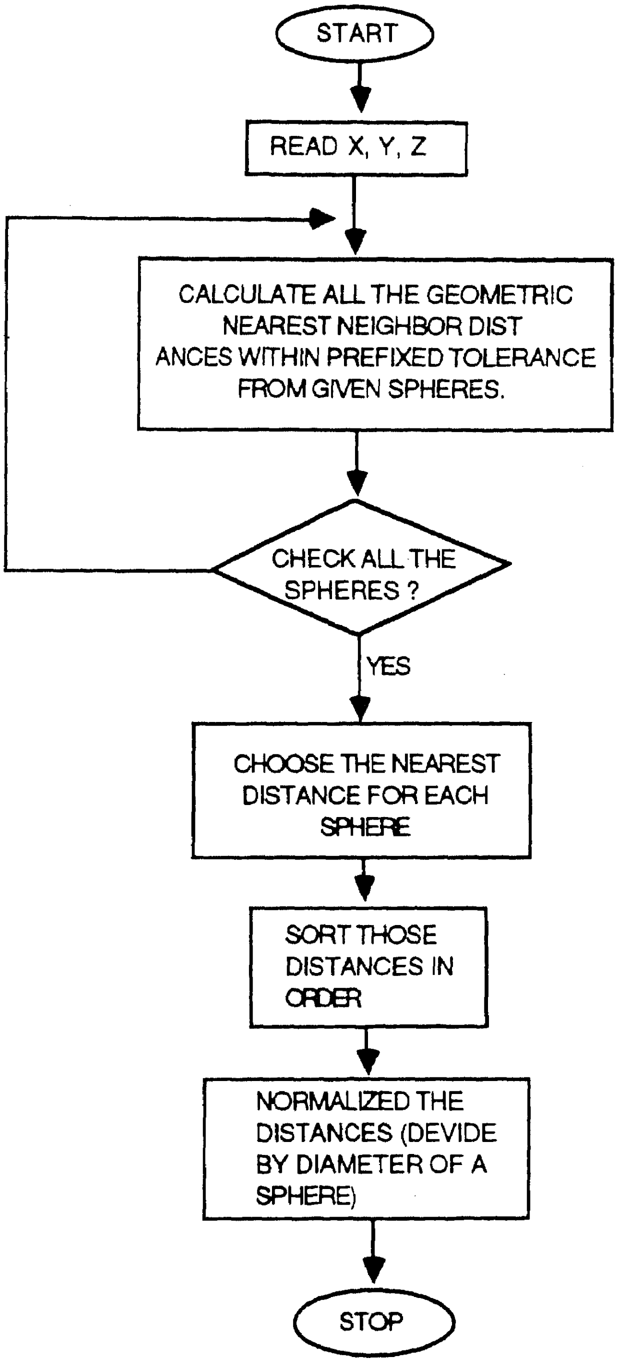
(b) Plane Growth Method


START
READ X, Y, Z
CHOOSE A SAMPLE BY CUTTING A PACKING WITH A HORIZONTAL PLANE
CALCULATE THE VOLUME OF SPHERES INSIDE THAT SAMPLE
CALCULATE THE VOLUME OF THE SAMPLE
COMPUTE THE DENSITY (= SPHERE VOL./ SAMPLE VOL.)
STOP

A.4 Algorithm for Radial Distribution Function


START
READ X, Y, Z
CALCULATE THE DISTANCE FROM CENTER SPHERE TO THE OTHER SPHERES USING P.B.C.
COMPUTE THE NUMBER OF SPHERE CENTERS WITHIN PREDEFINED INTERVAL (R, R+dR)
CALCULATE THE R.D.F. WHICH DEFINED AS THE FREQUENCY DIVIDED BY 4 Π (R/DIA.)²
STOP

A.5 Algorithm for calculating Median Nearest Neighbor


START
READ X, Y, Z
CALCULATE ALL THE GEOMETRIC NEAREST NEIGHBOR DIST ANCES WITHIN PREFIXED TOLERANCE FROM GIVEN SPHERES.
CHECK ALL THE SPHERES ?
YES
CHOOSE THE NEAREST DISTANCE FOR EACH SPHERE
SORT THOSE DISTANCES IN ORDER
NORMALIZED THE DISTANCES (DEVIDE BY DIAMETER OF A SPHERE)
STOP

APPENDIX B :

B.1 Monte Carlo Simulation Code

```
C ****************************************************************
C *                                                              *
C *         MONTE CARLO SIMULATION OF SETTLING OF SPHERES        *
C *                         AND SHAKING                          *
C *                                                              *
C ****************************************************************
C
C                               Variable List
C
C       beta     | "temperature" parameter appearing in Boltzmann
C                | distribution
C       d        | real-valued  array  of  different  diameters  in
C                | polydisperse systems
C       delt     | maximum allowable sphere displacement
C       dens     | density in (gms/cm**3)
C       dia      | array of sphere diameters
C       dmax     | diameter of largest sphere
C       dmin     | diameter of smallest sphere
C       dpe      | difference in the potential ENERGY  between  the
C                | current and a previous configurations
C       dsame    | character variable: value is 'yes'  if  all  the
C                | particles are the same  and  'no'  if  array  of
C                | diameters are to be read in
C       diam     | disc diameter when all spheres are the same size
C       eps      | tolerance used in conjunction with ebrat
C       emean    | mean value of the ENERGY over "mp1" values
C       edev     | standard deviation of ENERGY  array  over  "mp1"
C                | values
C       ebr      |  array  of  averaged  ENERGY  values;  ebr(k)  =
C                | average of 0 e(1) through e(k)
C       ebrp     | previous value of ebr (at pass k-1)
C       ebrat    | ratio of  current  average  ENERGY  to  previous
C                | average ENERGY
C       g        | 9.8 meters/(sec**2)
C       height   | height of parallelopiped (inches)
C       iaccpt   | number of accepted moves performed  to  generate
C                | one pass
C       icyc0    | cycle at which restart begins
C       icycle   | cycle counter
C       id       | integer array of size  "kn"  whose  value  id(k)
C                | represents the diameter dsort(k)
C       ipaspr   | pass print iteration counter
C       iprint   | print counter within each pass
C       iterpr   | integer designating the iteration  number  in  a
C                | specific pass at which a printout is desired
C       iterg    | integer specifying  frequency  at  which  ENERGY
C                | value is output
C       ix       | parameter used by SETRAN
C       length   | length of the parallelopiped (inches)
C       mass     | mass of the sphere in kilograms
```

```
C       maxgen  | maximum number  of  trial  to  generate  initial
C               | positions
C       maxpas  | maximum number of complete passes
C       mode    | character  variable  passed  into  GEOMCK  which
C               | designates  the  mode  in  which  GEOMCK  is  to
C               | operate (either 'generate' or 'simulate')
C       mp1     | maxpas + 1
C       n       | total number of spheres
C       nh      | number of layers (horizontal) of large spheres
C               | generated for the initial configuration (if
C               | the layer option is 'yes').
C       maxcyc  | maximum number of "shaking" cycles
C       nbig    | number of large spheres to be located in close-
C               | packed configuration on container bottom.  This
C               | initial configuration will be generated if
C               | the variable 'layer' is input as 'yes'.
C       ns      | interger array containing  distribution  of  the
C               | number of sphere sizes of diameter array d; Note
C               | that the sum of allthe  entries  of  this  array
C               | must be n
C       nsize   | number of diameters in polydisperse systems
C       nv      | number of columns of large spheres (vertical) to
C               | be generated in the initial configuration (on
C               | the container bottom) if the layer option is
C               | read in as 'yes'.
C       nw      | number of spheres along z-axis.
C       passpr  | integer designating the pass number at which  to
C               | print
C       pbc     | character variable (yes,no)  for  implementation
C               | of periodic boundary conditions or  not  (no  =>
C               | hard vertical walls)
C       pe      | potential ENERGY of the current configuration
C       peo     | potential ENERGY of the previous configuration
C      poly     |  character   variable   (yes,no)   designating
C               | polydisperse system S
C       psint1  | designates the pass number at which the  program
C               | starts. It is  nonzero  only  if  the  value  of
C               | "restrt" is yes
C       ra      | sphere radius when all spheres are the same size
C       restrt  | character variable: value is 'yes' if a  restart
C               | is to be done and 'no' if no restart required
C       xnew    | trial x-coordinate of a particular spheres.
C       x       | array containing x-coordinates of spheres.
C       ynew    | trial y-coordinate of a particular spheres.
C       y       | array containing y-coordinates of spheres.
C       z       | array containing z-coordinates of spheres.
C       znew    | trial z-coordinate of a particular spheres.
C       yjump0  | value by  which  to  displace  y-coordinates  of
C               | spheres when simulating shaking of spheres.
C       yjump   |         "
C       ymean   | average of the y array
C       ydev    | standard deviation the y array
```

```
C                  ________________________
C                  Subroutines and Functions
C                  ________________________
C
C     AINSRT   This   routine   inserts   the   newly   generated
C              coordinate   into   position   "isave"   of   a
C              real-valued array.  (4 arguments)
C     ENERGY   This function computes the potential ENERGY of the
C              current configuration based on the height of each
C              sphere above  the datum positon.  (6 argument)
C     GEOMCK   This routine searches the trial configuration  for
C              geometric  violations   (forbidden   overlap   of
C              spheres). If there is a violation, a value  of  -1
C              is returned for "ier". (9 arguments)
C     IINSRT   This routine is identical to AINSRT except that it
C              works on integer-valued arrays.
C     SEARCH   This routine searches for the location in which to
C              insert the trial  x-coordinate  (xtemp)  into  the
C              existing  x  array.It  returns  this  location  as
C              isave,  which  is  then   used   by   the   other
C              routines. (4 arguments)
C     SHELL    This   routine   does   a   shell   sort   of   a
C              one-dimensional array  into  increasing  order  (2
C              arguments).
C     SDEV     This  routine  returns  the  mean  and  standard
C              deviation of a sample (4 arguments).
C     YBARF    This function computes the  y  coordinate  of  the
C              area centroid (3 arguments).
C
C
C                  _________________________________
C                  Input and output file unit numbers
C                  _________________________________
C
C
C     Unit 9    File containing the output from the program.   It
C               is opened as '[sxp4639.mc3d]mc3d.out'
C     Unit 7    File containing the input data.  It is opened as
C               "[sxp4639.mc3d]mc3d.dat", status = "old".
C     Unit 12   File  containing  the  configuration  number  and
C               ENERGY  for  each  pass.   It  is   defined   as
C               "[sxp4639.mc3d]mc3d.erg".
C     Unit 31   File containing the restart data.  It  is  opened
C               as '[sxp4639.mc3d]mc3d.res'.
C     Unit 36   File containing  the  configuration  data  to  be
C               transferred to the DEC-20 system and then plotted
C               via     DISPLA.     It      is      opened      as
C               '[sxp4639.mc3d]mc3d.plo'
C
C   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C     B e g i n n i n g   o f   M A I N   P r o g r a m
C   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C
      Implicit Real*8 (a-h, o-z)
      Real*8 lmax, mass
```

```
      Dimension x(2000), y(2000), z(2000), e(60010), mass(2000),
     1          d(100), dia(2000), rad(2000), yt(50), ebr(5000)
      Integer freq(300), id(100), ns(100)
      Common diam, dmin, ipass
      Character * 3 restrt, dsame, poly, layer, pbc
      Character * 8 mode
      Character * 4 pour
      Integer passpr, psintl, totpas
      Integer temp(2000), trace(2000), origin(2000)
      Integer temdim, select, ip, j, ior, upbd, lwbd, jor, jtr
C
C     .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
C                  Open units  36, 9, 12 and 7
C
      open(unit=36,file='[sxp4639.mc3d]mc3d.plo',status='new')
      open(unit=9,file='[sxp4639.mc3d]mc3d.out',status='new')
      open(unit=7,file='[sxp4639.mc3d]mc3d.dat',status='old')
      open(unit=12,file='[sxp4639.mc3d]mc3d.erg',status='new')
C
C     .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
C
      read(7,*) length, height, width, diam, dens, beta, delt
      read(7,*) n, maxpas, maxgen, nsize
      read(7,*) iterpr, passpr, iterg
      read(7,*) eps
      read(7,921) yjump0
      read(7,*) maxcyc
      read(7,902) restrt
      read (7,902) poly
      read (7,902) pbc
C
C     ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
C
C         Initialization of parameters, indices and file outputs
C
      g = 9.8D0
      ndim = n + 1
      n1 = n * (n - 1) * 0.5
      mp1 = maxpas + 1
      ra = diam * 0.5D0
      eps1 = 1.0D0 - eps
      pi = 3.1415926536D0
      icycle = 0
      icyc0 = 0
      yjump = 0.0D0
      totpas = 0
      mxc1 = maxcyc + 1
      mode = 'generate'
C
C     ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..  ..
C
C         Generate the Random Number Seed (ix) using system clock
C
      rns = SECNDS(0.0)
      rns = pi * rns * 1.0D3
      ix  = IDINT(rns)
C
      if ((restrt .eq. 'YES').or.(restrt .eq. 'yes')) go to 110
      read(7,920) pour
      read(7,902) dsame
      read(7,902) layer
      if((dsame.eq.'YES').or.(dsame.eq.'yes'))  then
        dmax = diam
        dmin = diam
        atem = 4.0D0/3.0D0 * pi * (ra*ra*ra) * ((2.54D+00)**3)
        ams  = atem * dens * 1.0D-03
        do 500 ir = 1, n
               dia(ir) = diam
               mass(ir) = ams
500     continue
        close(unit=7)
        go to 2
      endif
C
C      . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C     This part of the code will generate an initial configuration
C      in which part of the container will contain "nbig" large
C     spheres.  On top of this, there will be "n - nbig" smaller
C     spheres.  (There will be 2 layers.)
C      . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
C
      IF ( ( layer .eq. 'YES' ) .or. ( layer .eq. 'yes' ) ) THEN
        read(7,*) dbig, dsmall
        read(7,*) nv, nh, nw
        dmax = dbig
        dmin = dsmall
        nbig = nv * nh * nw
        do 5550 ir = 1, nbig
                dia ( ir ) = dbig
5550    continue
        do 5560 ir = nbig + 1, n
                dia ( ir ) = dsmall
5560    continue
c
        close ( unit = 7 )
c
        rbig   = 0.5D0 * dbig
        rsmall = 0.5D0 * dsmall
        do 5581 k=1, nw
         do 5580 i = 1, nv
          do 5570 j = 1, nh
                  im1=i-1
                  jm1 = j - 1
                  km1=k-1
              Kx = i + (jm1 * nv)+(km1*nv*nh)
              x(kx) = rbig + dbig * jm1
              y(kx) = rbig+dbig*im1
              z(kx) = rbig + dbig *km1
```

```
5570          continue
5580         continue
5581        continue
            kdim = (nv * nh * nw)  + 11
            lmax = length - rsmall
            hmax = height - rsmall
            wmax = width  - rsmall
5590        xtemp = RAN ( ix ) * length
         if (( xtemp .le. rsmall ).or.( xtemp .ge. lmax )) go to 5590
            ymin = nv * dbig
5592        ytemp = RAN ( ix ) * height + ymin
         if (( ytemp .le. rsmall ).or.( ytemp .ge. hmax )) go to 5592
5593        ztemp = RAN(ix)*width
            if (( ztemp.le.rsmall).or.(ztemp.ge.wmax)) go to 5593
c
            call SEARCH ( xtemp, x, Kdim, isave )
          call GEOMCK ( x, y,z, dia, xtemp, ytemp,ztemp, dbig, dsmall,
     +                isave, Kdim, 0, mode, ier )
c
            if ( ier .eq. 0 ) then
               call AINSRT ( xtemp, x, Kdim, isave )
               call AINSRT ( ytemp, y, kdim, isave )
               call AINSRT ( ztemp, z, Kdim, isave )
               call AINSRT ( dsmall, dia, Kdim, isave )
            icount = icount + 1
            kdim = kdim + 1
            if ( icount .gt. maxgen ) go to 888
            if ( kdim .le. n ) then
               go to 5590
            else
               xpid = pi * ((2.54D+00)**3) * dens * 1.0D-03
               do 5594 i = 1, n
                       mass(i)=4.0D0/3.0D0*xpid*((dia(i)*0.5D0)**3)
5594           continue
               go to 377
            endif
            else
               icount = icount + 1
               if ( icount .gt. maxgen ) go to 888
               go to 5590
            end if
C
          ENDIF
C
C              [Either generate a polydisperse array or read in
C           distribution from unit 7]
C
503  IF (( poly .eq. 'NO' ) .or. ( poly .eq. 'no' )) THEN
            read(7,*) n2
            do 395 i = 1, n2
                   read(7,*) x(i), y(i), z(i), dia(i)
395         continue
            read(7,*) dmax, dmin
            rmin = dmin * 0.5D0
         rmax = dmax * 0.5D0
         close(unit=7)
         do 392 i = n2+1, n
                dia(i) = diam
392      continue
C           [generate the remainder of the spheres of diameter
C         'diam']
         icount = 1
         kdim = n2 + 1
393      xtemp = RAN(ix) * length
         if((xtemp.le.ra) .or. (xtemp.ge.length-ra)) go to 393
397      ytemp = RAN(ix) * height
         if((ytemp.le.ra).or.(ytemp.ge.height-ra)) go to 397
398      ztemp = RAN(ix) * width
         if((ztemp.le.ra) .or. (ztemp.ge.width-ra)) go to 398
         Call SEARCH(xtemp,x,kdim,isave)
         Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,diam,isave,
     +                kdim,0,mode,ier)
         if(ier .eq. 0) then
           Call AINSRT(xtemp,x,kdim,isave)
           Call AINSRT(ytemp,y,kdim,isave)
           Call AINSRT(ztemp,z,Kdim,isave)
           Call AINSRT(diam,dia,kdim,isave)
           kdim = kdim + 1
           icount = icount + 1
           if (icount .gt. maxgen) go to 888
           if(kdim .le. n) then
             go to 393
           else
              xpid = pi * ((2.54D+00)**3) * dens * 1.0D-3
              do 391 ia = 1, n
                     tempa=4.0D0/3.0D0*xpid*((dia(ia)*0.5D0)**3)
                     mass(ia) = tempa
391           continue
              go to 377
           endif
         else
            icount = icount + 1
            if (icount .gt. maxgen) go to 888
            go to 393
         endif
      ELSE
C
C     [Generate polydisperse system of spheres]
C
      jend = 0
      icount = 1
      kdim = 1
      read (7,*) dmax, dmin
      DO 2000 K = 1, NSIZE
              read (7, *) ns(k), d(k)
              if ( d(k) .gt. dmax ) dmax = d(k)
              if ( d(k) .lt. dmin ) dmin = d(k)
              rad(k) = 0.5D0 * d(k)
              if (k .eq. 1 ) then
                 jstrt = 1
              else
                 jstrt = jend + 1
              endif
              jend = jend + ns(k)
              do 551 j = jstrt, jend
                     dia(j) = d(k)
551           continue
501           xtemp = RAN(ix) * length
              if((xtemp.le.rad(k)).or.
     +           (xtemp.ge.length-rad(k))) go to 501
502           ytemp = RAN(ix) * height
              if((ytemp.le.rad(k)).or.
     +           (ytemp.ge.height-rad(k))) go to 502
506           ztemp = RAN(ix) * width
              if((ztemp.le.rad(k)).or.
     +           (ztemp.ge.width-rad(k))) go to 506
              if (kdim .eq. 1) then
                 isave = 1
                 go to 505
              endif
              Call SEARCH(xtemp, x, Kdim, isave)
505      Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,d(k),isave,
     +               Kdim,0,mode,ier)
              if (ier .eq. 0) then
                 Call AINSRT(xtemp, x, Kdim, isave)
                 call AINSRT(ytemp, y, kdim, isave)
                 Call AINSRT(ztemp, z, Kdim, isave)
                 Call AINSRT(d(k), dia, Kdim, isave)
                 Kdim = Kdim + 1
                 icount = icount + 1
                 if (icount .gt. maxgen) go to 888
                 if (Kdim .le. jend ) go to 501
              else
                 icount = icount + 1
                 if (icount .gt. maxgen) go to 888
                 go to 501
              endif
2000  CONTINUE
C
      xpid = pi * ((2.54D+00)**3)* dens * 1.0D-3
      do 555 k = 1, n
             tempa = 4.0D0/3.0D0 * xpid *((dia(k)*0.5D0)**3)
             mass(k) = tempa
555   continue
      close(unit=7)
      go to 377
      ENDIF
C
C     This section to generate n-spheres of diameter 'diam'
C
2     psintl = 0
      x(1) = RAN(ix) * length
      if((x(1) .le. ra).or.(x(1) .ge. length-ra))go to 2
6     y(1) = RAN(ix) * height
      if((y(1) .le. ra).or.(y(1) .ge. height-ra))go to 6
5     z(1) = RAN(ix)*width
      if ((z(1).le.ra).or.(z(1).ge.width-ra)) go to 5
3     xnew = RAN(ix) * length
      if((xnew.le.ra) .or. (xnew .ge. length-ra))go to 3
7     ynew = RAN(ix) * height
      if((ynew.le.ra) .or. (ynew .ge. height-ra))go to 7
8     znew=RAN(ix)*width
      if ((znew.le.ra).or.(znew.ge.width-ra)) go to 8
      isave = 1
      if (x(1) .le. xnew) isave = 2
      Call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,diam,isave,
     1            2,0,mode,ier)
          if(ier .eq. -1) go to 3
      Call AINSRT(xnew, x, 2, isave)
      call AINSRT(ynew, y, 2, isave)
      Call AINSRT(znew, z, 2, isave)
35    icount = 1
      Kdim = 3
96    xtemp = RAN(ix) * length
      if((xtemp .le. ra).or.(xtemp .ge. length-ra))go to 96
98    ytemp = RAN(ix) * height
      if((ytemp .le. ra).or.(ytemp .ge. height-ra))go to 98
99    ztemp=RAN(ix) * width
      if((ztemp.le.ra).or.(ztemp.ge.width-ra)) go to 99
      Call SEARCH(xtemp, x, Kdim, isave)
      Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,diam,
     1            isave,Kdim,0,mode,ier)
      if (ier .eq. 0) go to 91
C
         icount = icount + 1
         if (icount .gt. maxgen ) go to 888
         go to 96
91    Call AINSRT(xtemp,x,Kdim,isave)
      call AINSRT(ytemp,y,kdim,isave)
      Call AINSRT(ztemp,z,Kdim,isave)
      kdim = kdim + 1
      icount = icount + 1
      if ( icount .gt. maxgen ) go to 888
      if (kdim .le. n) then
         go to 96
      endif
C
377   peo = ENERGY(0, n, mass, g, y, 0.0D0)
      e(1) = peo
      if (kdim .gt. n) go to 885
C
C     [The following statments will be executed if restrt = 'Yes'.
C        File unit 31 contains the information from the previous
C      run.]
C
110    open(unit=31,file='[sxp4639.mc3d]mc3d.res',status='old')
```

```
           do 57 ir = 1, n
                 read(31,901) x(ir), y(ir),z(ir)
                 read(31,9901) dia(ir), mass(ir)
57         continue
           read (31,*) psintl, peo
           read(31,*) icyc0
           read (31,*) dmax, dmin
           read(31,*) yjump0
           read(31,919) pour
           e(1) = peo
           totpas = psintl
           icycle = 1
           yjump = yjump0
           if (yjump .ne. 0.0) pour = 'np'
           close(unit=31)
           rewind(unit=31)
           go to 885
C                       ****************************
C                       * Start of the Simulation *
C                       ****************************
C
C          Begin moving spheres and generating new configurations
C
115    if (icycle .le. maxcyc) then
          ix = ix + 2
          go to 767
       else
          go to 998
       endif
C
C         -- Lift assembly of spheres by amplitude "yjump" --
C
767    do 116 i = 1, n
             y(i) = y(i) + yjump
116    continue
C
      peo = ENERGY(0, n, mass, g, y, 0.0D0)
      e(1) = peo
C
      mode = 'simulate'
      ipass = 0
      ipaspr = 0
C
120   iaccpt = 0
      totpas = totpas + 1
      ipass = ipass + 1
      if (ipass .eq. 20) then
         Call SDEV(e, 20, ebrp, ebrdp)
         ebr(1) = ebrp
      endif
      ipaspr = ipaspr + 1
      iprint = 0
C
C  ---- Start of Random Selection Procedure -----
```

```
      temdim = n
      do 3000 J = 1, n
              Temp ( J ) = J
              Trace ( J ) = J
              Origin ( J ) = J
3000  continue
15    if ( temdim .eq. 1 ) then
        select = 1
        ip = temp ( 1 )
        i = trace ( ip )
        temdim = 0
        go to 70
      else if ( temdim .eq. 0 ) then
              go to 90
      endif
C
4005  select = IFIX(temdim * RAN(ix))
      if ( select .eq. 0 ) go to 4005
C
      ip = temp ( select )
      i = trace ( ip )
C
      if ( select .lt. temdim ) then
         do 4000 j = select, temdim
                 temp ( j ) = temp ( j + 1 )
4000     continue
      endif
C
      temdim = temdim - 1
C
70    iprint = iprint + 1
      xnew = x(i) + delt * (1.0D+00-2.0D+00*RAN(ix))
      ynew = y(i) + delt * (1.0D+00-2.0D+00*RAN(ix))
      znew = z(i) + delt * (1.0D+00-2.0D+00*RAN(ix))
      if (ynew .le. dia(i)*0.5D+00) go to 15
      if ((ynew .gt. dia(i)*0.5D+00) .and.
     1    (ynew .lt. height-dia(i)*0.5D+00)) then
         go to 65
      endif
C
C               *******************************
C               * Impose Hard Vertical Walls *
C               *******************************
C
65    if ((pbc .eq. 'no').or.(pbc .eq. 'No').or.(pbc .eq. 'nO')
     +           .or.(pbc .eq. 'NO')) then
         if (((xnew .gt. dia(i)*0.5D0).and.
     1        (xnew .lt. length-dia(i)*0.5D0)).and.
     2        ((znew .gt. dia(i)*0.5D0).and.
     3        (znew .lt. width-dia(i)*0.5D0))) then
            Call SEARCH(xnew, x, ndim, isave)
            Call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                              isave,ndim,i,mode,ier)
            if (ier .eq. -1) then
```

```
               go to 15
            else
               go to 69
            endif
         else
            go to 15
         endif
      endif
C          ***************************************
C          * Impose Periodic Boundary Conditions *
C          ***************************************
C
      if (((xnew.gt.0.0) .and. (xnew .lt. dia(i))).and.
     1    (znew.ge.width+dia(i)*0.5D0)) then
         znew=znew-width
         call SEARCH(xnew, x, ndim, isave)
         call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                               isave,ndim,i,mode,ier)
         if (ier.eq.-1) go to 15
         xnew1=xnew+length
         call SEARCH(xnew1,x,ndim,isave1)
         call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                               isave1,ndim,i,mode,ier)
         if (ier.eq.-1) go to 15
         znew1=znew+width
         call SEARCH(xnew,x,ndim,isave)
         call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                              isave,ndim,i,mode,ier)
         if (ier.eq.-1) go to 15
         xnew1=xnew+length
         znew1=znew+width
         call SEARCH(xnew1,x,ndim,isave1)
         call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                               isave1,ndim,i,mode,ier)
         if (ier.eq.-1) then
            go to 15
         else
            go to 69
         endif
C
      else if (((xnew.gt.0.0).and.(xnew.lt.dia(i))).and.
     1    ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                    isave,ndim,i,mode,ier)
              if (ier .eq. -1) go to 15
              xnew1=xnew+length
              call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier .eq. -1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
              znew1=znew-width
              call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 znew=znew1
                 go to 69
              endif
C
      else if (((xnew.gt.0.0).and.(xnew.lt.dia(i))).and.
     1        ((znew.gt.width-dia(i)).and.(znew.lt.width))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1 = xnew+length
                 call SEARCH(xnew1, x, ndim, isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew,x,ndim,isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    go to 69
                 endif
C
      else if (((xnew.gt.0.0).and.(xnew.lt.dia(i))).and.
     1      ((znew.ge.dia(i)).and.(znew.le.width-dia(i)))) then
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier .eq. -1) go to 15
              xnew1 = xnew+length
              call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
              if (ier .eq. -1) then
                 go to 15
              else
```

```
                 go to 69
              endif
      else if (((xnew.gt.0.0) .and. (xnew.lt.dia(i))).and.
     1          ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                        isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1 = length + xnew
                 call SEARCH(xnew1, x, ndim, isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                       isave1,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew,x,ndim,isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                      isave,ndim, i,mode, ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                       isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    go to 69
                 endif
C
      else if (((xnew.gt.0.0).and.(xnew.lt.dia(i))).and.
     1           ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                        isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                        isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                       isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                       isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    znew=znew1
                    go to 69
                    endif
      else if (((xnew.gt.0.0).and.(xnew.lt.dia(i))).and.
     1         (znew.le.-dia(i)*0.5D0)) then
              znew=znew+width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier .eq. -1) go to 15
              xnew1=xnew+length
              call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                     isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1         (znew.ge.width+dia(i)*0.5D0)) then
              znew=znew-width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier .eq. -1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                     isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1    ((znew.ge.width).and.(znew.lt.width+dia(i)*0.5D0))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                      isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew, x, ndim, 'isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                      isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    znew=znew1
                    go to 69
                 endif
C
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1        ((znew.gt.width-dia(i)).and.(znew.le.width))) then
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1 = xnew-length
                 call SEARCH(xnew1, x, ndim, isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew,x,ndim,isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                      isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                      isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    go to 69
                 endif
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1    ((znew.ge.dia(i)).and.(znew.le.width-dia(i)))) then
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1 = xnew - length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1         ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1 = xnew-length
                 call SEARCH(xnew1, x, ndim, isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                  isave1,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew,x,ndim,isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                 isave,ndim, i,mode, ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    go to 69
                 endif
C
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1         ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                  isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                  isave,ndim,i,mode,ier)
                if (ier .eq. -1) go to 15
                xnew1=xnew-length
                call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                  isave1,ndim,i,mode,ier)
                if (ier.eq.-1) go to 15
                xnew1=xnew-length
                znew1=znew+width
                call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
                if (ier.eq.-1) then
                  go to 15
                else
                  znew=znew1
                  go to 69
                endif
C
      else if (((xnew.gt.length-dia(i)).and.(xnew.le.length)).and.
     1        (znew.le.-dia(i)*0.5D0)) then
             znew=znew+width
             call SEARCH(xnew, x, ndim, isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                isave,ndim,i,mode,ier)
             if (ier .eq. -1) go to 15
             xnew1=xnew-length
             call SEARCH(xnew1, x, ndim, isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
               znew1=znew-width
               call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                  isave,ndim,i,mode,ier)
               if (ier.eq.-1) go to 15
               xnew1=xnew-length
               znew1=znew-width
               call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
             if (ier.eq.-1) then
               go to 15
             else
               go to 69
             endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1        (znew.ge.width+dia(i)*0.5D0)) then
             znew=znew-width
             xnew=xnew+length
             call SEARCH(xnew, x, ndim, isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                    isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              znew1=znew-width
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1    ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                        isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 znew1=znew-width
                 xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     +                                      isave1,ndim,i,mode,ier)
                 if(ier.eq.-1) go to 15
                 xnew1=xnew+length
                 call SEARCH(xnew1, x, ndim, isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                     isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                        isave,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    znew=znew1
                    xnew=xnew1
                    isave=isave1
                    go to 69
                 endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1         ((znew.gt.width-dia(i)).and.(znew.le.width))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                      isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                      isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                       isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                      isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    xnew=xnew1
                    isave=isave1
                    go to 69
                 endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1    ((znew.ge.dia(i)).and.(znew.le.width-dia(i)))) then
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1 = xnew + length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 xnew = xnew1
                 isave = isave1
                 go to 69
              endif
C
     else if (((xnew.gt.-dia(i)*0.5D0) .and. (xnew .le. 0.0)).and.
     1         ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                       isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim, i,mode, ier)
                if (ier.eq.-1) go to 15
                znew1=znew+width
                call SEARCH(xnew, x, ndim, isave)
               call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                               isave, ndim, i, mode, ier)
                if (ier.eq.-1) go to 15
                znew1=znew+width
                xnew1=xnew+length
                call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
                if (ier.eq.-1) then
                   go to 15
                else
                   xnew=xnew1
                   isave=isave1
                   go to 69
                endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1        ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
                call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
                if (ier .eq. -1) go to 15
                xnew1=xnew+length
                znew1=znew+width
                call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     +                                  isave1,ndim,i,mode,ier)
                if (ier.eq.-1) go to 15
                xnew1=xnew+length
                call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                             isave1, ndim, i, mode, ier)
                if (ier.eq.-1) go to 15
                znew1=znew+width
                call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
                if (ier.eq.-1) then
                   go to 15
                else
                   xnew=xnew1
                   znew=znew1
                   isave=isave1
                   go to 69
                endif
C
      else if (((xnew.gt.-dia(i)*0.5D0).and.(xnew.le.0.0)).and.
     1        (znew.le.-dia(i)*0.5D0)) then
             znew=znew+width
```

```
              xnew=xnew+length
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                 isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              znew1=znew-width
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1          (znew.ge.width+dia(i)*0.5D0)) then
              znew=znew-width
              xnew=xnew-length
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                  isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew, x, ndim, isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                    isave,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
```

```
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1  ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                        isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     +                                      isave1,ndim,i,mode,ier)
                 if(ier.eq.-1) go to 15
                 xnew1=xnew-length
                 call SEARCH(xnew1, x, ndim, isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                  isave1, ndim, i, mode, ier)
                 if (ier.eq.-1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew,x,ndim,isave)
                call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                        isave,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    xnew=xnew1
                    znew=znew1
                    isave=isave1
                    go to 69
                 endif
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1       ((znew.gt.width-dia(i)).and.(znew.le.width))) then
                 call SEARCH(xnew, x, ndim, isave)
               call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                       isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                       isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
                 znew1=znew-width
                 call SEARCH(xnew, x, ndim, isave)
               call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                    isave, ndim, i, mode, ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                        isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) then
                    go to 15
                 else
                    xnew=xnew1
                    isave=isave1
                    go to 69
                 endif
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1       ((znew.ge.dia(i)).and.(znew.le.width-dia(i)))) then
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1 = xnew - length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 xnew=xnew1
                 isave=isave1
                 go to 69
              endif
C
      else if (((xnew.gt.length).and.
     1          (xnew.lt.length+dia(i)*0.5D0)).and.
     2          ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
                 call SEARCH(xnew, x, ndim, isave)
                 call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                          isave,ndim,i,mode,ier)
                 if (ier .eq. -1) go to 15
                 xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                    isave1,ndim, i,mode, ier)
                 if (ier.eq.-1) go to 15
                 znew1=znew+width
                 call SEARCH(xnew, x, ndim, isave)
            call GEOMCK(x, y, z,dia,xnew,ynew,znew1,dmax,dia(i),
     +                                 isave, ndim, i, mode, ier)
                 if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
               if (ier.eq.-1) then
                  go to 15
               else
                  xnew=xnew1
                  isave=isave1
                  go to 69
               endif
C
```

```
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1         ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
                call SEARCH(xnew, x, ndim, isave)
               call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                           isave,ndim,i,mode,ier)
                if (ier .eq. -1) go to 15
                znew1=znew+width
                xnew1=xnew-length
                call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     +                                           isave1,ndim,i,mode,ier)
                if (ier .eq. -1) go to 15
                xnew1=xnew-length
                call SEARCH(xnew1, x, ndim, isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                      isave1, ndim, i, mode, ier)
                if (ier.eq.-1) go to 15
                znew1=znew+width
                call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                           isave,ndim,i,mode,ier)
                if (ier.eq.-1) then
                   go to 15
                else
                   xnew=xnew1
                   znew=znew1
                   isave=isave1
                   go to 69
                endif
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(i)*0.5D0)).and.
     1        (znew.le.-dia(i)*0.5D0)) then
             xnew=xnew-length
             znew=znew+width
             call SEARCH(xnew,x,ndim,isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                         isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             xnew1=xnew+length
             call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                          isave1,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             znew1=znew-width
             call SEARCH(xnew,x,ndim,isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                         isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             xnew1=xnew+length
             znew1=znew-width
             call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                         isave1,ndim,i,mode,ier)
             if (ier.eq.-1) then
```

```
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1        (znew.ge.width+dia(i)*0.5D0)) then
              xnew=xnew+length
              znew=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                    isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                     isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              znew1=znew+width
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1  ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
              xnew=xnew+length
              znew=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                    isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                     isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                    isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                    isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
c
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1        ((znew.gt.width-dia(i)).and.(znew.le.width))) then
              xnew=xnew+length
              call SEARCH(xnew,  x,  ndim,  isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
              if (ier .eq.  -1) go to 15
              xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
                call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                    isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.le.-dia(i)*0.5D0).and.((znew.ge.dia(i)).and.
     1         (znew.le.width-dia(i)))) then
              xnew = xnew + length
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
```

```
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1         ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
              xnew=xnew+length
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                    isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1         ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
              xnew=xnew+length
              znew=znew+width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                    isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew-length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
```

```
                 go to 69
              endif
C
      else if ((xnew.le.-dia(i)*0.5D0).and.
     1         (znew.le.-dia(i)*0.5D0)) then
              xnew=xnew+length
              znew=znew+width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew-length
              znew1=znew-width
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     1         (znew.ge.width+dia(i)*0.5D0)) then
              xnew=xnew-length
              znew=znew-width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                               isave,ndim,i,mode,ier)
              xnew1=xnew+length
              znew1=znew+width
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
```

```
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     1    ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
              xnew=xnew-length
              znew=znew-width
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     1         ((znew.gt.width-dia(i)).and.(znew.lt.width))) then
              xnew=xnew-length
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew-width
                 call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
c
      else if ((xnew.ge.length+dia(i)*0.5D0).and.((znew.ge.dia(i))
     1         .and.(znew.le.width-dia(i)))) then
              xnew = xnew - length
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
              call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                   isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15.
              else
                 go to 69
              endif
c
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     1         ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
              xnew=xnew-length
              call SEARCH(xnew, x, ndim, isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              xnew1=xnew+length
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                    isave1,ndim,i,mode,ier)
                 if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
                 xnew1=xnew+length
                 znew1=znew+width
                 call SEARCH(xnew1,x,ndim,isave1)
               call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                     isave1,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
c
```

```
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     1        ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
             xnew=xnew-length
             znew=znew+width
             call SEARCH(xnew, x, ndim, isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             xnew1=xnew+length
                call SEARCH(xnew1,x,ndim,isave1)
              call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     +                                 isave1,ndim,i,mode,ier)
                if (ier.eq.-1) go to 15
             znew1=znew-width
             call SEARCH(xnew,x,ndim,isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
                xnew1=xnew+length
                znew1=znew-width
                call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                 isave1,ndim,i,mode,ier)
             if (ier.eq.-1) then
                go to 15
             else
                go to 69
             endif
c
      else if ((xnew.ge.length+dia(i)*0.5D0).and.
     +        (znew.le.-dia(i)*0.5D0)) then
             xnew=xnew-length
             znew=znew+width
             call SEARCH(xnew, x, ndim, isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     +                                isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             xnew1=xnew+length
             call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(i),
     *                                isave1,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             znew1=znew-width
             call SEARCH(xnew,x,ndim,isave)
             call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
             if (ier.eq.-1) go to 15
             xnew1=xnew+length
             znew1=znew-width
             call SEARCH(xnew1,x,ndim,isave1)
             call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(i),
     *                                 isave1,ndim,i,mode,ier)
             if (ier.eq.-1) then
                go to 15
              else
                 go to 69
              endif
c
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i)))
     *                 .and.(znew.ge.width+dia(i)*0.5D0)) then
              znew=znew-width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1=znew+width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
c
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
     1   ((znew.gt.width).and.(znew.lt.width+dia(i)*0.5D0))) then
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1 = znew - width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 znew = znew1
                 go to 69
              endif
c
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
     1        ((znew.gt.width-dia(i)).and.(znew.le.width))) then
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                  isave,ndim,i,mode,ier)
              if (ier.eq.-1) go to 15
              znew1 = znew - width
              call SEARCH(xnew,x,ndim,isave)
              call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                                   isave,ndim,i,mode,ier)
              if (ier.eq.-1) then
                 go to 15
              else
                 go to 69
              endif
C
```

```
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
     1     ((znew.gt.0.0).and.(znew.lt.dia(i)))) then
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) go to 15
           znew1 = znew + width
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) go to 15
           znew2=znew-width
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew2,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) then
              go to 15
           else
              go to 69
           endif
C
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
     1     ((znew.gt.-dia(i)*0.5D0).and.(znew.le.0.0))) then
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) go to 15
           znew1 = znew + width
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) then
              go to 15
           else
              znew = znew1
              go to 69
           endif
C
      else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
     1     (znew.le.-dia(i)*0.5D0)) then
           znew = znew + width
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) go to 15
           znew1=znew-width
           call SEARCH(xnew,x,ndim,isave)
           call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
     *                         isave,ndim,i,mode,ier)
           if (ier.eq.-1) then
              go to 15
           else
              go to 69
           endif
      else
         call SEARCH(xnew,x,ndim,isave)
         call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
     *                                isave,ndim,i,mode,ier)
         if (ier .eq. -1) then
            go to 15
         else
            go to 69
         endif
      endif
C
C
C        ***************************************
C        *Calculating new ENERGY of the system*
C        ***************************************
C
69    pe = ENERGY(i, n, mass, g, y, ynew)
      dpe = pe - peo
      if (dpe .gt. 0.0D0) go to 80
75    iaccpt = iaccpt + 1
      peo=pe
C
C  ---- Core section of random selection -----
C
      tempd = dia ( i )
      tempm = mass ( i )
C
      if ( isave .gt. i ) isave = isave - 1
C
      IF ( i .lt. isave ) THEN
c
c ---- Update arrays x, y, z, dia, &  mass -----
c
         jr = isave - i
         do 9100 j = 1, jr
                 ij = i + j
                 ijm1 = ij - 1
                 x ( ijm1 ) = x ( ij )
                 y ( ijm1 ) = y ( ij )
                 z ( ijm1 ) = z ( ij )
                 dia ( ijm1 ) = dia ( ij )
                 mass ( ijm1 ) = mass ( ij )
9100     continue
c
c----- Update arrays trace and origin -----
c
        ior = Origin ( i )
        upbd = ior
        lwbd = ior
        do 5000 j = i+1, isave
                jor = Origin ( j )
                if ( jor .gt. upbd ) then
                   upbd = jor
                   go to 5000
                endif
                if ( jor .lt. lwbd ) lwbd = jor
5000    continue
C
        do 6000 j = lwbd, upbd
                jtr = trace ( j )
                if ( jtr .le. i ) go to 6000
                if ( jtr .gt. isave) go to 6000
                trace ( j ) = jtr - 1
                origin(trace(j)) = j
6000    continue
C
        trace ( ip ) = isave
        origin ( isave ) = ip
C
      else if ( i .gt. isave ) then
c
c----- Update arrays x, y, z, dia, and mass -----
c
              jr = i - isave
              do 9200 j = 1, jr
                      ij = i - j
                      ijp1 = ij + 1
                      x ( ijp1 ) = x ( ij )

                     y ( ijp1 ) = y ( ij )
                      z ( ijp1 ) = z ( ij )
                      dia ( ijp1 ) = dia ( ij )
                      mass ( ijp1 ) = mass ( ij )
9200          continue
c
c----- Update arrays trace and origin -----
c
              ior = origin ( isave )
              upbd = ior
              lwbd = ior
              do 7000 j = isave + 1, i
                      jor = origin ( j )
                      if ( jor .gt. upbd ) then
                         upbd = jor
                         go to 7000
                      endif
                      if ( jor .lt. lwbd ) lwbd = jor
7000          continue
C
              do 8000 j = lwbd, upbd
                      jtr = trace ( j )
                      if ( jtr .ge. i ) go to 8000
                      if ( jtr .lt. isave ) go to 8000
                      trace ( j ) = jtr + 1
                      origin ( trace ( j ) ) = j
8000          continue
C
              trace ( ip ) = isave
              origin ( isave ) = ip
      ENDIF
c
      x ( isave ) = xnew
      y ( isave ) = ynew
      z ( isave ) = znew
      dia ( isave ) = tempd
      mass ( isave ) = tempm
c
c----- end of random selection -----
c
      if ( iprint .ne. iterpr ) go to 15
      iprint = 0
      write(9,205) ipass
      write(9,206)
      write(9,203)
c      do 305 Kp = 1, n
c              write(9,204) Kp, x(Kp), y(kp), z(Kp)
c305   continue
      go to 15
C
80    bd = beta * dpe
      if (DABS(bd) .ge. 170.0D+00) go to 15
       pr = DEXP(-bd)
       if (RAN(ix) .le. pr) then
          go to 75
       else
          go to 15
       endif
C
90    e(ipass+1) = peo
      perctg = DFLOAT(iaccpt)/DFLOAT(n) * 100.0D+00
C
C     After 10,000 passes, Set the value of delta as 0.06E-02
C
C     if (perctg .lt. 3.0D+00) then
C        delt = delt * 0.75D+00
C     endif
C
C          ******************
C          *Output pass data*
C          ******************
C
      IF (ipaspr .eq. passpr) THEN
         ipass1 = ipass + psintl
         ipaspr = 0
         icy = icycle + icyc0
         write(9,220) yjump
         if (pour .eq. 'pour') icy=0
         write(9,199) icy
         write(9,208) ipass1
         write(9,209) e(ipass+1)
         write(9,210) perctg
         write(9,206)
         write(9,203)
c         do 317 jp = 1, n
c         write(9,204) jp, x(jp), y(jp), z(jp), dia(jp), mass(jp)
c317      continue
         write(36,*) icy
         write(36,*) n
         write(36,*) ipass1
         write(36,*) beta
         write(36,*) e(ipass+1)
         write(36,*) dmin
c         do 403 jp = 1, n
c               write(36,*) x(jp), y(jp), z(jp), dia(jp)
c403        continue
      ENDIF
C
C         Compute ENERGY averages from e(1) every 100 passes
C
      if ( MOD(ipass,100) .eq. 0.0 ) then
         ip1 = ipass + 1
         ip2 = ipass/100
         Call SDEV(e, ip1, emean, edev)
         ebr(ip2) = emean
         ebrat = ebr(ip2)/ebrp
c        if (icycle .eq. maxcyc) ebrat = 1.0D0
         ebrp = ebr(ip2)
      endif
C
C            Check if cycle has been completed
C
      IF ((ipass .eq. maxpas) .or.
     +    ((ebrat.ge.1.0D0-eps).and.(ebrat.le.1.0D0+eps))) THEN
         yjump = yjump0
         mp1 = ipass + 1
         do 912 i2 = 1, mp1, iterg
                i3 = i2 + psintl
                write(12,903) i3, e(i2)
912      continue
         ipass1 = totpas
         psintl = totpas
         icy = icycle + icyc0
         write(9,220) yjump
         if (pour .eq. 'pour') icy = 0
         write(9,199) icy
         write(9,208) ipass1
         write(9,209) e(mp1)
         write(9,210) perctg
         write(9,206)
         write(9,203)
c         do 318 jp = 1, n
c         write(9,204) jp, x(jp), y(jp), z(jp), dia(jp), mass(jp)
c318      continue
         write(36,*) icy
         write(36,*) n
         write(36,*) ipass1
         write(36,*) beta
         write(36,*) e(mp1)
         write(36,*) dmin
c         do 319 jp = 1, n
c                write(36,*) x(jp), y(jp), z(jp), dia(jp)
c319      continue
C
C          Compute mean and standard deviation of ENERGY array
C               and store values in "emean" and "edev".
C
         Call SDEV(e, mp1, emean, edev)
         write(9,907) emean, edev
         write(9,916)
         icycle = icycle + 1
         ebrat =  0.0D0
         ebrp =   0.0D0
         ebrpdp = 0.0D0
         nerg = MOD(maxpas,100) + 5
         do 713 inr = 1, nerg
                ebr(inr) = 0.0D0
713      continue
         if (pour .ne. 'pour') then
C           [Shaking Cycle completed:  Begin new cycle.]
            go to 115
         else
C           [Pouring completed.]
            go to 998
         endif
C
      ELSE
C        [Cycle not completed.]
C        [Continue until (1-eps) <= ebrat <= (1+eps) ]
         go to 120
      ENDIF
C
C     .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
C            Initial Data and Output formats
C
885   Call IDATE(month,iday,iyear)
      write(9,201)
200    format(1h ,'MAIN: THE MAXIMUM NUMBER OF TRIALS TO GENERATE
     *        THE INITIAL DISTRIBUTION =',I7,' HAS BEEN EXCEEDED.')
201   format(16X,'MONTE CARLO SIMULATION SHAKING SEGREGATION',/)
      write(9,219) month, iday, iyear
      write(9,220) yjump
      write(9,199) icyc0
      write(9,202) psintl
202   format(11X,'STARTING COORDINATES OF SPHERES,   PASS NO: ',I8
     +,/,6X,'_______________________________________________________
     +_____________')
      if (psintl .eq. 0) then
         write(36,*) ratio
         write(36,*) amp, ecut
         write(36,*) n
```

```
          write(36,*) beta
          write(36,*) peo
          write(36,*) dmin
c          do 405 ij = 1, n
c                  write(36,*) x(ij), y(ij), z(ij), dia(ij)
c405       continue
        endif
C
        write(9,203)
203     format(4X,'Sphere',4x,'X',10x,'Y',10X,'Z',6x,'Diameter',
      +         3x,'Mass',/)
c        do 300 K = 1, n
c                write(9,204) k, x(K), y(k), z(K), dia(K), mass(K)
c300     continue
        write(9,209) peo
        write(9,211) delt
204     format(2X,I5,1x,3(D11.5,1X),2x,D10.4,1x,D11.5)
199     format(///,26X,'CYCLE NUMBER:',I7)
205      format(///,9X,'SPHERE COORDINATES OF A CONFIGURATION FOR
      +         PASS NO.:',I6)
206     format(7X,'_______________________________________________________
      +___________')
208     format(/,9X,'SPHERE COORDINATES AT PASS NUMBER:',I9)
209     format(/,9X,'Configuration ENERGY =',D20.7,1x,
      +         'Newton-meters')
210     format(9X,'Percentage of moves accepted: ',D11.4)
211     format(9X,'maximum absolute value of particle displacement:'
      +   ,D14.6)
213     format(5(I5,6x))
218     format(1h ,'Pass Number = ',I9,/)
219     format(23X,'Run Date:  ',I2,'-',I2,'-',I2)
220     format(11X,'Amplitude of Shaking:  ',D11.5)
901     format(1X,3(D22.16,1X))
9901    format(1X,2(D22.16,1X))
902     format(10X,A3)
903     format(I8,5x,D19.13)
904     format(1X,'= C / ',D11.5,',',D11.5,',',D11.5)
905     format(1X,//)
907     format(1X,'The Mean Energy = ',D13.5,5x,'Standard deviation
      +          =' D13.5)
908     format(6X,'The approximate packing height = ',D15.8,/,6X,
      +         'The standard deviation is ',D15.8)
909     format(11X,'The mean + standard deviation = ',D15.8,/,11X,
      +         'The mean - standard deviation = ',D15.8)
910     format(6X,'The approximate packing center = ',D15.8)
911     format(11X,'Upper limit = ',D15.8,/,11X,'Lower limit = ',
      +         D15.8)
913     format(6X,'The mean of the entire y-array = ',D14.7)
914     format(11X,'Its standard deviation = ',D14.7)
915     format(6X,'The maximum height attained by some sphere =',
      +         D14.7)
916     format(4X,'====================================================',
      +         '========================',/)
917     format(1X ,D12.6,1X,D12.6,1X,I1)
```

```
918     format(1h ,'ShaKing-Segregation Parameters:  ',/,26X,
     +   'yjump = ',D10.4,/,26X,'ENERGY cut-off ratio = ',D12.6,)
     +      /,26X,'Number of Cycles = ',I5,'#')
919     format(1X,A4)
920     format(10X,A4)
921     format(10X,D9.3)
        go to 115
C
888     write(9, 200) maxgen
        go to 999
C
C       ********* write the restart file to unit 31 **********
C
998    open(unit=31,file='[sxp4639.mc3d]mc3d.res',status='new')
       open(unit=33,file='[sxp4639.rdf]radist3.dat',status='new')
C
       do 900 i1 = 1, n
              write(31,901) x(i1), y(i1), z(i1)
              write(31,9901) dia(i1), mass(i1)
900    continue
              write(31,*) totpas, e(mp1)
              if (pour .eq. 'pour') icy = 0
              write(31,*) icy
              write(31,*) dmax, dmin
              write(31,*) yjump
              write(31,919) pour
C
C      ******* write the r.d.f. data file to unit 33  **********
C
              write(33,1902) n
              write(33,1901) diam
              write(33,*)
              write(33,*)
1902   format(I4)
1901   format(D7.1)
       do 1900 i1=1, n
               write(33,1903) x(i1), y(i1), z(i1)
1903   format(3(D22.16, 1X))
1900   continue
C
C      ***************************************************************
C      Shell sort the y-array to estimatethe average pacKed height.
C      Also compute mean and standard deviation of entire y array.
C      ***************************************************************
C
       Call SHELL(y, n)
       Call SDEV(y, n, ymean, ydev)
       nav = IDINT(length/diam)
       nava = n - nav
       do 400 K = 1, nav
              j = nava + K
              yt(K) = y(j)
400    continue
       Call SDEV(yt, nav, ytmean, ytdev)
```

```
      ycen = ytmean * 0.5D0
      ytmax = ytmean + ytdev
      ytmin = ytmean - ytdev
      ytmaxc = ytmax * 0.5D0
      ytminc = ytmin * 0.5D0
        write(9,905)
        write(9,908) ytmean, ytdev
        write(9,909) ytmax, ytmin
        write(9,910) ycen
        write(9,911) ytmaxc, ytminc
        write(9,913) ymean
        write(9,914) ydev
        write(9,915) yt(nav)
        write(9,905)
C
999   close(unit=12)
      close(unit=9 )
      close(unit=31)
      close(unit=33)
      close(unit=36)
      stop
      end
C
C
C -----------------------------------------------------------------
C              E n d   o f   m A I N   P r o g r a m
C -----------------------------------------------------------------
C
C
C      ****************************************************
C      * B I N A R Y   S E A R C H   S U B R O U T I N E *
C      ****************************************************
C
C
      SUBROUTINE SEARCH (xtemp, x, kdim, isave)
C
C
      Implicit Real*8 (a-h, o-z)
      Dimension x(kdim)
      Integer high, low, mid
C
C     [SEARCH for location "isave" where xtemp < x(low)]
C
      low = 1
      high = Kdim-1
50    mid = 0.5 * ( low + high )
      if(xtemp .lt. x(mid)) high = mid - 1
      if(xtemp .gt. x(mid)) low = mid + 1
      if(xtemp .eq. x(mid)) go to 60
      if(low .le. high) go to 50
      isave = low
      go to 99
60    isave = mid
99    return
```

```
      end
C
C    *************************************************************
C    *              REAL-VALUED INSERT SUBROUTINE               *
C    *************************************************************
C
      SUBROUTINE AINSRT(xtemp, x, kdim, isave)
C
      Implicit Real*8 (a-h, o-z)
      Dimension x(kdim)
C
      if (isave .eq. kdim) go to  70
      temp1 = x(isave)
      x(isave) = xtemp
      isp1 = isave + 1
      temp2 = x(isp1)
      x(isp1) = temp1
      j = isave + 2
      if (j .gt. Kdim) go to 100
80    temp1 = x(j)
      x(j) = temp2
      jp1 = j + 1
      if (jp1 .gt. Kdim) go to 100
      temp2 = x(jp1)
      x(jp1) = temp1
      j = j + 2
      if (j .gt. Kdim) go to 100
      go to 80
70    x(isave) = xtemp
100   return
      end
C
C           **********************************
C           * INTEGER VALUED INSERT ROUTINE *
C           **********************************
C
      SUBROUTINE IINSRT(ntemp, nx, kdim, isave)
C
C     Insert xtemp in position "isave" and move other elements
C    appropriately.This is identical to AINSRT except the argument
C    are of type integer.
C
      Implicit Real*8 (a-h, o-z)
      Dimension nx(Kdim)
      Integer temp1, temp2, ntemp
C
      if (isave .eq. Kdim) go to  70
      temp1 = nx(isave)
      nx(isave) = ntemp
      isp1 = isave + 1
      temp2 = nx(isp1)
      nx(isp1) = temp1
      j = isave + 2
      if (j .gt. Kdim) go to 100
```

```
80     temp1 =  nx(j)
       nx(j) = temp2
       jp1 = j + 1
       if (jp1 .gt. Kdim) go to 100
       temp2 = nx(jp1)
       nx(jp1) = temp1
       j = j + 2
       if (j .gt. Kdim) go to 100
       go to 80
70     nx(isave) = ntemp
C
100    return
       end
C
C       ****************************************************
C       *                                                  *
C       *          GEOMETRY CHECKING SUBROUTINE            *
C       *                                                  *
C       ****************************************************
C
C      This subroutine checKs for forbidden overlap of spheres and
C     returns a value of -1 in ier if overlap occurs.
C
C      mode      Character variable designating whether the routine
C                   is to be used in a 'generation' mode or in the
C               'simulation' mode.
C      ipos      array index of the particle that has been moved to
C                a new position (xtemp, ytemp,ztemp).  This is used
C               used only in the'simulate' mode.
C     Kdim      dimension of the x, y, and z arrays (plus 1).
C     dmax      maximum sphere diameter in the system of spheres.
C     diam      diameter of sphere to be inserted in 'generate' mode
C       isave       array index, returned by SEARCH, at which the
C               particle is to be placed after its move. ('simulate'
C               mode)
C     ier       error flag returned as -1 if any discs overlap with
C                 the location of the displaced sphere. ('simulate'
C               mode)
C     x         array of x-coordinates of the spheres.
C     y         array of y-coordinate of the spheres.
C     z         array of z-coordinates of the spheres.
C               array of diameters of the spheres.
C      xtemp       tentative new x-coordinate of the sphere whose
C               x-coordinate is given by x(ipos).
C      ytemp     tentative new y-coordinate of the sphere of which
C               y-coordinate is given by y(ipos).
C      ztemp       tentative new z-coordiante of the sphere whose
C               z-coordinate is given by z(ipos).
C
C
       SUBROUTINE GEOMCK(x,y,z,d,xtemp,ytemp,ztemp,dmax,diam,
      *                               isave,kdim,ipos,mode,ier)
C
       Implicit Real*8 (a-h, o-z)
      Dimension x(kdim), y(kdim), z(kdim), d(kdim)
      Character *8 mode
C
      if (mode .eq. 'generate') then
         rtemp = 0.5D0 * diam
      else
         rtemp = 0.5D0 * d(ipos)
      endif
C
      j = 0
9     if (isave+j .eq. ipos) j = j + 1
      if ( isave+j .gt. Kdim-1 ) then
         go to 50
      endif
      isj = isave + j
      dx = DABS(xtemp - x(isj))
      dy = DABS(ytemp - y(isj))
      dz = DABS(ztemp - z(isj))
      dist = rtemp + d(isj)*0.5D0
C
10    if (dx .ge. dmax) then
         go to 50
      endif
      if (dx .ge. dist) then
         j=j+1
         go to 9
      else if (dy .ge. dist) then
              j=j+1
              go to 9
      else if (dz .ge. dist) then
              j=j+1
              go to 9
      else
         dst =DSQRT(dx*dx + dy*dy +dz*dz)
         if (dst .ge. dist) then
            j = j + 1
            go to 9
         else
            ier = -1
            return
         endif
      endif
C
50    j = -1
51    if (isave+j .eq. ipos) j = j - 1
      if (isave+j .lt. 1) go to 70
      isj = isave + j
      dx = DABS(xtemp - x(isj))
      dy = DABS(ytemp - y(isj))
      dz = DABS(ztemp - z(isj))
      dist = rtemp + d (isj)*0.5D0
C
60    if (dx .ge. dmax) then
         go to 70
      endif
      if (dx .ge. dist) then
         j = j-1
         go to 51
      else if (dy .ge. dist) then
              j = j-1
              go to 51
      else if (dz .ge. dist) then
              j = j-1
              go to 51
      else
         dst =DSQRT(dx*dx + dy*dy +dz*dz)
         if (dst .ge. dist) then
            j = j-1
            go to 51
         else
            ier = -1
            return
         endif
      endif
C
70    ier = 0
      return
      end
C
C       ***************************************
C       *           ENERGY FUNCTION           *
C       ***************************************
C
C      This function calculates the potential ENERGY of the system
C       in Joules, or Newton-meters.  The y coordinates are
C        converted to meters. The mks system is used in this
C     calculation.(1 J = 1 Nm)
C
      REAL*8 FUNCTION ENERGY(k, n, mass, g, y, ynew)
C
      Implicit Real*8 (a-h, o-z)
      Real*8 mass
      Dimension y(n), mass(n)
C
      temp = 0.0D0
      temp1 = 0.0D0
      do 10 j = 1, n
            temp = temp + mass(j) * g * y(j) *0.0254D0
 10   continue
      energy = temp
      if (k .ne. 0) then
         temp1 = mass(k) * g * y(k) * 0.0254D0
         energy = temp - temp1 + mass(k) * g * ynew * 0.0254D0
      endif
      return
      end
```

```
C          ****************************************
C          *        Y-CENTROID FUNCTION           *
C          ****************************************
C
C               This function returns the y-coodinate of area
C           centroid.
C
           REAL*8 FUNCTION YBARF(n, y, dia)
C
           Implicit Real*8 (a-h, o-z)
           Dimension y(n), dia(n)
C
           sum1 = 0.0D0
           sum2 = 0.0D0
           do 10 K = 1, n
                sum1 = sum1 + (dia(k)**2) * y(k)
10              sum2 = sum2 + (dia(k)**2)
           ybarf = sum1/sum2
           return
           end
C
C          ****************************************
C          *       SHELL SORT SUBROUTINE          *
C          ****************************************
C
C This subroutine taKes an array "v" of dimension "n" and sorts it
C       into increasing order.
C
 SUBROUTINE SHELL(v, n)
C
        Implicit Real*8 (a-h, o-z)
 Dimension v(n)
 Integer gap
 gap = n * 0.5
20      do 10 i = gap, n
       j = i - gap
15          if (j .le. 0) go to 10
       if( v(j) .le. v(j+gap) ) go to 10
       temp = v(j)
       v(j) = v(j + gap)
       v(j + gap) = temp
       j = j - gap
       go to 15
10      continue
 gap = gap * 0.5
 if (gap .gt. 0) go to 20
99      return
 end
```

```
C               ***********************************
C               *        SUBROUTINE SDEV          *
C               ***********************************
C
C     This routine computes the mean and standard deviation of the
C        array "x" and returns them in "xmean" and "xdev"
C     respectively.
C
      SUBROUTINE SDEV(x, n, xmean, xdev)
C
      Implicit Real*8 (a-h, o-z)
      Dimension x(n)
C
      n1 = n - 1
      xsum = 0.0D0
      sum = 0.0D0
      do 10 i = 1, n
10          xsum = xsum + x(i)
      if (n1 .eq. 0) n1 = 1
      xmean = xsum / DFLOAT(n1)
      do 20 i = 1, n
20          sum = sum + (x(i) - xmean)**2
      var = sum / DFLOAT(n1)
      xdev =DSQRT(var)
      return
      end
C
C              *********************************************
C              *             HISTOGRAM SUBROUTINE          *
C              *********************************************
C
C           This subroutine taKes an array "v" which has been
C         presorted by the routine SHELL, and returns the integer
C         array "freq" consisting of "nintv" elements. The latter
C         arrary contains the frequency distribution of "v" which
C        can be used to plot its histogram.  The parameter "dmax"
C         is the supremum of the elements of "v".  The routine in
C        effect does a binary SEARCH on the array "v".
C
        SUBROUTINE HSTOGm(v, dmax, deltar, n, nintv, freq)
C
        Implicit Real*8 (a-h, o-z)
        Dimension v(n)
        Integer freq(300), high, low, mid
C
        do 10 i = 1, nintv + 1
10            freq(i) = 0
        isaveo = 1
        vtemp = deltar
        i = 1
80      if (vtemp .gt. dmax) go to 100
           low = isaveo
           high = n
50         mid = (high + low) * 0.5
```

```
           if (vtemp .lt. v(mid)) high = mid - 1
           if (vtemp .gt. v(mid)) low = mid + 1
           if (vtemp .eq. v(mid)) then
              Km = mid + 1
              mid = Km
30            vt1 = DABS(v(Km) - vtemp)
              if (vt1 .gt. 1.0D-06*vtemp) go to 60
                 Km = Km + 1
                 mid = Km
                 go to 30
           endif
           if (low .le. high) then
                 go to 50
           else
              isave = low
              go to 99
           endif
60         isave = mid
99         freq(i) = isave - isaveo
           isaveo = isave
           if (isaveo .gt. n) go to 100
           i = i + 1
           vtemp = vtemp + deltar
           go to 80
100     return
        end
```

B.2 Coordination Number Code

```
C   ==============================================================
C                         Co-ordination Number
C   ==============================================================
C
C     Variables
C
C     freq        Integer array containing histogram of intersphere
C                 distances.
C     (x, y, z)   Arrays of sphere center coordinates.
C     n           Number of spheres.
C     xlng        Length of "box" containing spheres.
C     zlng        Width of "box" containing spheres.
C     dx          Difference between x coordinates of two spheres.
C     dy          Difference between y coordinates of two spheres.
C     dz          Difference between z coordinates of two spheres.
C     dia         Diameter of sphere.
C       eps           Torelance of the close contact = (1 + eps)
C                 * Diameter.  Input by user.
C     dst         Intersphere distance.
C
C     ............................................................
C     Input and  Output Files
C
C     Unit 51      Defined as [SXP4639.codnum]cod_num.dat.  This is
C                 the file from which the input data is read.
C     Unit 52      Defined as [SXP4639.codnum]cod_num.out.  This is
C                 the file to which output is written.
C     ............................................................
C
C     Description
C        This program calculates the coordination number which
C      is defined as the number of spheres in con contact with a
C       given sphere.  The coodinates of spheres are readed from
C         the input file, then the coordination numbers within
C         predefined tolerances of the diameter separation are
C      calculated.
C
C     ============================================================
C                        Beginning of Program
C     ============================================================
C
      Implicit real*8 (a-h,o-z)
      Integer freq(1000), sum(20), freq1(1000), tot
      Dimension x(1000), y(1000), z(1000)
      Real perctg(20)
C
     open(unit=51,file='[SXP4639.codnum]cod_num.dat',status='old')
     open(unit=52,file='[SXP4639.codnum]cod_num.out',status='new')
C
      read(51,*) n
      read(51,*) dia
      read(51,*) xlng, ylng, zlng
C
```

```
      do 3 i = 1, n
           read(51,*) x(i), y(i), z(i)
3     continue
C
      do 4 i = 1, n
           freq(i) = 0
4     continue
      do 5 i = 1, n
           freq1(i) = 0
5     continue
C
      do 6 l = 1, 13
            sum(l) = 0
6     continue
C
      do 7 l = 1, 13
            perctg(l) = 0.0
7     continue
C
      eps = 0.05D+00
      xlngf = xlng*0.5D+00
      ylngf = ylng*0.5D+00
      zlngf = zlng*0.5D+00
      i = 0
      k = 0
C
110   k = k + 1
111   if (((x(k).gt.(xlngf-0.8D0)).and.(x(k).lt.(xlngf+0.8D0)))
     1  .and.((y(k).gt.0.7D0).and.(y(k).lt.1.8D0)).and.
     2((z(k).gt.(zlngf-0.8D0)).and.(z(k).lt.(zlngf+0.8D0)))) then
         go to 112
      else
         k = k + 1
         if (k .gt. n) then
            go to 220
         else
            go to 111
         endif
      endif
C
112   i = i + 1
      n1 = i
      do 38 j = 1, n
            dx = DABS(x(k) - x(j))
            dy = DABS(y(k) - y(j))
            dz = DABS(z(k) - z(j))
            if (dx .gt. xlngf) then
               if (x(j) .gt. xlngf) then
                  xjp = x(j) - xlng
               else
                  xjp = x(j) + xlng
               endif
               dx = DABS(x(k) - xjp)
            endif
C
```

```
             if (dy .gt. ylngf) then
                if (y(j) .gt. ylngf) then
                   yjp = y(j) - ylng
                else
                   yjp = y(j) + ylng
                endif
                dy = DABS(y(k) - yjp)
             endif
C
             if (dz .gt. zlngf) then
                if (z(j) .gt. zlngf) then
                   zjp = z(j) - zlng
                else
                   zjp = z(j) + zlng
                endif
                dz = DABS(z(k) - zjp)
             endif
             dst2 = dx**2 + dy**2 + dz**2
             if ((dst2 .ge. ((1-eps)*dia)**2).and.
     1           (dst2 .le. ((1+eps)*dia)**2)) then
                 freq(k) = freq(k) + 1
                 freq1(i) = freq(k)
             endif
38     continue
       go to 110
C
220    do 43 i = 1, n1
          do 42 l = 1, 12
                if (freq1(i) .eq. l) then
                   sum(l) = sum(l) + 1
                endif
42        continue
43     continue
C
       tot = 0
       do 45 k = 1, 12
             tot = tot + sum(k)
45     continue
C
       do 46 i = 1, 12
      tr = (FLOAT(sum(i)))/(FLOAT(tot))
             perctg(i) = tr * 100.0
46     continue
C
       write(52,55)
       write(52,56)
       write(52,57)
       do 44 i = 1, 12
             write (52,58) i, sum(i), perctg(i), n1
44     continue
C
55     format(10X, '=============================================')
56     format(10X,'CO.NUM.',5X,'FREQ.',5X,'PERCTG',5X,'INNER SP.')
57     format(10X, '=============================================')
58     format(10X,I3,6X,I5,7X,F7.4,5X,I4)
```

```
C
      close(unit=51)
      close(unit=52)
      stop
      end
C
C     =============================================================
C                            End of Program
C     =============================================================
```

B.3 Packing Fraction code using Spherical Growth Method

```
C   ===============================================================
C           Packing volume fraction by spherical growth method
C   ===============================================================
C
C     Variables
C
C     (x, y, z)  Arrays of sphere center coordinates.
C     n          Number of spheres.
C     xlng       Length of "box" containing spheres.
C     ylng       Height of "box" containing spheres.
C     zlng       Width of "box" containing spheres.
C     dx         Difference between x coordinates of two spheres.
C     dy         Difference between y coordinates of two spheres.
C     dz         Difference between z coordinates of two spheres.
C     dia        Diameter of sphere.
C     rad        0.5 * Diameter
C     dst        Radius of the container.
C     dist       Distance of interspheres.
C     tvol       Volume of the spherical sample.
C     svol       Volume of the spheres.
C     .............................................................
C
C     Input and  Output Files
C
C     Unit 51    Defined as [SXP4639.fractn]frac.dat.  This is the
C                file from which the input data is read.
C     Unit 52    Defined as [SXP4639.fractn]frac.out.  This is the
C                file to which output is written.
C     .............................................................
C
C     Description
C        This program calculates the packing fraction from the
C       several spherical samples.   The packing fraction is
C       determined by the ratio of the total volume of spheres
C      (svol) to the volume of spherical sample (tvol).
C
C     ============================================================
C                        Beginning of Program
C     ============================================================
C
      Implicit real*8 (a-h,o-z)
      Dimension x(2000), y(2000), z(2000), sum(110), tsum(100)
C
      open(unit=51,file='[SXP4639.fractn]frac.dat',status='old')
      open(unit=52,file='[SXP4639.fractn]frac.out',status='new')
C
      read(51,*) n
      read(51,*) dia
      read(51,*) xlng, ylng, zlng
C
      do 3 j = 1, n
           read(51,*) x(j), y(j), z(j)
```

```
3      continue
C
       rad = dia * 0.5D+00
       pi = 3.1415926536D+00
       xlngf = xlng*0.5d0
       zlngf = zlng*0.5d0
C
       do 4 l = 1, 100
            sum(l) = 0.0d0
4      continue
C
       do 5 l1 = 1, 100
            tsum(l1) = 0.0D0
5      continue
C
       i = 0
       k = 0
       k1 = 1
C
110    k = k + 1
111    If (((x(k).gt.(xlngf-1.2d0)).and.(x(k).lt.(xlngf+1.2d0)))
      1.and.((y(k).gt.0.16302D0).and.(y(k).lt.2.56302d0))
      2.and.((z(k).gt.(zlngf-1.2d0)).and.(z(k).lt.(zlngf+1.2d0))))
      3 then
          go to 112
       else
            k = k + 1
            if (k .gt. n) then
               go to 220
            else
               go to 111
            endif
       endif
C
112    i=i+1
       n1=i
       dst = dia
C
11     vol1=0.0d0
       vol2=0.0d0
       do 38 j = 1, n
             dx = DABS(x(k)-x(j))
             dy = DABS(y(k)-y(j))
             dz = DABS(z(k)-z(j))
             if (dx.gt.xlng*0.5d0) then
                if (x(j).gt.xlng*0.5d0) then
                   xjp = x(j)-xlng
                else
                   xjp = x(j) + xlng
                endif
                    dx = DABS(x(k)-xjp)
             endif
             if (dy .gt. ylng*0.5d0) then
                if (y(j).gt.ylng*0.5d0) then
                   yjp = y(j)-ylng
               else
                  yjp = y(j) + ylng
               endif
                   dy = DABS(y(k) - yjp)
            endif
            if (dz .gt. zlng*0.5d0) then
               if (z(j) .gt. zlng*0.5d0) then
                  zjp = z(j) - zlng
               else
                  zjp = z(j) + zlng
               endif
                   dz = DABS(z(k)-zjp)
            endif
            dist=DSQRT(dx*dx + dy*dy + dz*dz)
C
               if (dist .le. (dst-rad)) then
                  vol1= vol1 + 4.0d0/3.0d0*pi*rad**3
C
               else if ((dist .gt. (dst-rad))
     1                     .and.(dist .lt. (dst+rad))) then
                     d = (dst**2 + dist**2 - rad**2)/(2.0d0*dist)
                     vol2 = vol2 + pi/3.0d0*(2*(dst**3)+2*(rad**3)
     2                     + (dist**3) - 3*dist*(d**2+rad**2))
               endif
C
38    continue
C
      svol = vol1 + vol2
      tvol = 4.0D0/3.0D0*pi*(dst**3)
      pf = svol/tvol
      sum(k1) = sum(k1) + pf
C
      dst = dst + 0.05D0
      if (dst .gt. 1.2D0) then
         k1 = 1
         go to 110
      else
         k1 = k1 + 1
         go to 11
      endif
C
220   do 39 k1 = 1, 19
            tsum(k1) = sum(k1)/n1
            write(52,58) tsum(k1), n1
39    continue

58    format(7x, F7.4, 10x,I5)
      close(unit=51)
      close(unit=52)
      stop
      end
C
C     ==============================================================
C                                  End of Program
C     ==============================================================
```

B.4 Packing Fraction code using Plane Growth Method

```
C    ============================================================
C               Packing volume fraction by plane growth method
C    ============================================================
C
C      Variables
C
C      (x, y, z)  Arrays of sphere center coordinates.
C      n          Number of spheres.
C      xlng       Length of "box" containing spheres.
C      yhigh      Heith of the packing sampled.
C      zlng       Width of "box" containing spheres.
C      dx         Difference between x coordinates of two spheres.
C      dy         Difference between y coordinates of two spheres.
C      dz         Difference between z coordinates of two spheres.
C      dia        Diameter of sphere.
C      rad        Diameter * 0.5
C      tvol       Volume of the container.
C      svol       Volume of the spheres.
C
C      ..........................................................
C      Description
C       This method cuts the packing by a plane and calculates the
C        volume of spheres bounded by that plane and the periodic
C      "walls".  The packing fraction is determined by the ratio of
C        the volume of spheres (svlo) to the volume of container
C      (tvol) containing them.
C      ..........................................................
C
C      ==========================================================
C                          Beginning of Program
C      ==========================================================
C
      Implicit real*8 (a-h,o-z)
      Dimension x(2000), y(2000), z(2000)
C
      open(unit=51,file='[SXP4639.dens]dnsty.dat',status='old')
      open(unit=52,file='[SXP4639.dens]dnsty.out',status='new')
C
      read(51,*) n
      read(51,*) dia
      read(51,*) xlng, yhigh, zlng
C
      do 3 j = 1, n
           read(51,*) x(j), y(j), z(j)
3     continue
C
      rad = dia * 0.5D+00
      pi = 3.1415926536D+00
      nu1 = 0
      nu2 = 0
      nu3 = 0
      vol = 0.0D0
      vol1 = 0.0D0
      vol2 = 0.0D0
      vol3 = 0.0D0
C
C     Volume of Spherical segment of one base
C
      do 4 k = 1, n
           if ((y(k).lt.(yhigh+rad)).and.(y(k).ge.yhigh)) then
                   nu1 = nu1 + 1
                   h = rad - y(k) + yhigh
                   vol1 = vol1 + 1.0D0/3.0D0*pi*h*h*(3*rad-h)
         else if ((y(k).lt.yhigh).and.(y(k).gt.(yhigh-rad))) then
                   nu2 = nu2 + 1
                   h = y(k) + rad - yhigh
                   vol2 = vol2 + 4.0D0/3.0D0*pi*rad*rad*rad
     1                    - 1.0D0/3.0D0*pi*h*h*(3*rad-h)
           else if (y(k) .le. (yhigh - rad)) then
                   nu3 = nu3 + 1
                   vol3 = vol3 + 4.0D0/3.0D0*pi*rad*rad*rad
           endif
4     continue
C
      tvol = xlng * yhigh * zlng
      svol = vol1 + vol2 + vol3
      pd = svol/tvol
C
      write(52,*) tvol, svol, pd
C
      close(unit=51)
      close(unit=52)
      stop
      end
C
C     ==============================================================
C                            End of Program
C     ==============================================================
```

B.5 Radial Distribution Function code

```
C  ================================================================
C                       RADIAL DISTRIBUTION FUNCTION
C  ================================================================
C
C    Variables
C
C    freq       Integer array containing histogram of intersphere
C               distances.
C    g          Array containing distribution function.
C    (x, y, z)  Arrays of sphere center coordinates.
C    n          Number of spheres.
C    delbin     Bin width in units of sphere diameter.
C               It is also redefined as "delbin * dia".
C    xlng       Length of "box" containing spheres.
C    zlng       width of "box" containing spheres.
C    dx         Difference between x coordinates of two spheres.
C    dy         Difference between y coordinates of two spheres.
C    dz         Difference between z coordinates of two spheres.
C    dia        Diameter of sphere.
C    rad        Radius of sphere.
C    dmax        Maximum distance for which distribution function
C               is to be calculated
C      eps          = 1.0D-07 : error parameter to account for
C               machine accuracy
C    dst        Intersphere distance.
C    kbmax      Total number of bins.
C    low         Integer used in binary search to locate correct
C               bin
C    high        Integer used in binary search to locate correct
C               bin
C    mid         Integer used in binary search to locate correct
C               bin
C    ..............................................................
C
C    Input and  Output Files
C
C    Unit 51    Defined as [SXP4639.rdf]radist3.dat.  This is the
C               file from which the input data is read.
C    Unit 52    Defined as [SXP4639.rdf]radist3.out.  This is the
C               file to which output is written.
C    ..............................................................
C
C    Description
C
C         This Fortran 77 code computes the radial distribution
C       function from the coordinates of the center of the sphere
C       (i.e. the configuration).  The configuration is read from
C       unit 51.  The code computes and returns (to unit 52) the
C        histogram of intersphere distances and the distribution
C          function.  This is defined as the number of sphere
C        centers in (r, r+dr) divide by 4 * pi * (r/dia)**2 Note
C         that the ring width, "dr", is not included here in the
```

```
C           definition of the distribution function.  Also, the
C        distribution function is computed at the midpoint of each
C        bin.
C
C     =============================================================
C                       Beginning of Program
C     =============================================================
C
      Implicit real*8 (a-h,o-z)
      Integer freq(600), low, high, mid
      Dimension x(2000), y(2000), z(2000), g(600)
C
      open(unit=51,file='[SXP4639.rdf]radist3.dat',status='old')
      open(unit=52,file='[SXP4639.rdf]radist3.out',status='new')
      eps=1.0D-07
C
      read(51,*) n
      read(51,*) dia
      read(51,*) delbin
      read(51,*) xlng, ylng, zlng
C
      do 3 i = 1, n
           read(51,*) x(i), y(i), z(i)
3     continue
C
      do 4 i = 1, 600
           freq(i) = 0
4     continue
C
      dmax = xlng
      rad = dia * 0.5D0
      delbin = delbin * dia
      kbmax = dmax/delbin
C
      k = 1
5     j = k + 1
6     if (j .gt. n) then
         k = k + 1
         if (k .gt. n) then
            go to 100
         endif
         go to 5
      endif
C
      dx = DABS(x(k) - x(j))
      dy = DABS(y(k) - y(j))
      dz = DABS(z(k) - z(j))
      if (dx .gt. xlng*0.5D0) then
         if (x(j) .gt. xlng*0.5D0) then
            xjp = x(j) - xlng
         else
            xjp = x(j) + xlng
         endif
         dx = DABS(x(k) - xjp)
      endif
```

```
C
      if (dy .gt. ylng*0.5D0) then
         if (y(j) .gt. ylng*0.5D0) then
            yjp = y(j) - ylng
         else
            yjp = y(j) + ylng
         endif
         dy = DABS(y(k) - yjp)
      endif
C
      if (dz .gt. zlng*0.5D0) then
         if (z(j) .gt. zlng*0.5D0) then
            zjp = z(j) - zlng
         else
            zjp = z(j) + zlng
         endif
         dz = DABS(z(k) - zjp)
      endif
C
      dst = DSQRT(dx*dx  + dy*dy + dz*dz)
C
      low = 1
      high = kbmax
50    mid = (low + high)/2
      bmid = FLOAT(mid) * delbin
      if (dst .lt. bmid) high = mid - 1
      if (dst .gt. bmid) low = mid + 1
      if ((dst .le. bmid+eps).and.(dst .ge. bmid-eps)) then
         freq(mid) = freq(mid) + 1
         j = j + 1
         go to 6
      endif
      if (low .le. high) then
         go to 50
      else
         isave = low
         freq(isave) = freq(isave) + 1
         j = j + 1
         go to 6
      endif
C
100   write(52,200)
      write(52,201)
      write(52,202)
C
      pi = 3.1415926536D0
      rs = delbin
C
      do 110 j = 1, kbmax
         rsd1 = (rs - 0.5D0*delbin)/dia
         g(j) = 2.0D0*FLOAT(freq(j))/(4.0D0*pi*rsd1*rsd1*FLOAT(n))
         write(52,203) rsd1, freq(j), g(j)
         rs = rs + delbin
110   continue
200   format(1h ,10x,'RADIAL DISTRIBUTION FUNCTION',/,1h ,10x,
     1       'FOR PERIODIC BOUNDARY CONDITIONS',//)
201   format(1h ,5x,'r/diameter     ',5x,'Frequency     ',3x,
     1       'Distribution Function')
202   format(1h ,5x,'===========================================',
     1       '==========',/)
203   format(1h ,5x,D10.3,10x,I7,9x,D13.6)
C
120   close(unit=51)
      close(unit=52)
      stop
      end
C
C     ============================================================
C                            End of Program
C     ============================================================
```

B.6 Cumulative Probability of the Normalized Nearest Neighbor Distance r/dia.

(a) Calculating the nearest neighbor distances

```
C   ==============================================================
C                   The nearest neighbour distace
C   ==============================================================
C
C     Variables
C
C     (x, y, z)  Arrays of sphere center coordinates.
C     n          Number of spheres.
C     xlng       Length of "box" containing spheres.
C     ylng       Heith of "box" containing spheres.
C     zlng       Width of "box" containing spheres.
C     dx         Difference between x coordinates of two spheres.
C     dy         Difference between y coordinates of two spheres.
C     dz         Difference between z coordinates of two spheres.
C     dia        Diameter of sphere.
C     rad        Diameter * 0.5
C     dst        Distance of interspheres.
C
C     ...........................................................
C     Input and Output files
C
C     Unit 51    Defined as [adr7805.park.dist]dist.dat.  This is
C                the file from which the input data is read.
C
C     Unit 52    Defined as [adr7805.park.dist]dist.out.  This is
C                the file from which the output data is written.
C     ............................................................
C     Description
C        This program chooses the nearest distance between two
C     spheres.  [dist.dat] is the coordinates of the spheres.
C     ............................................................
C
C     ============================================================
C                        Beginning of Program
C     ============================================================
C
C
      Implicit real*8 (a-h,o-z)
      Integer low, high, mid
      Dimension x(1100), y(1100), z(1100), dst(1100)
C
    open(unit=51,file='[ADR7805.PARK.dist]dist.dat',status='old')
    open(unit=52,file='[ADR7805.PARK.dist]dist.out',status='new')
C
      read(51,*) n
      read(51,*) dia
      read(51,*) xlng, ylng, zlng
C
```

```
      do 3 i = 1, n
           read(51,*) x(i), y(i), z(i)
3     continue
C
      do 4 i =1, n
           dst(i) = 0.0D0
4     continue
C
      k = 1
      j = 1
      j1 = 1
C
6     if (j .gt. n) then
         L = 1
         do 66 i1 = 2, 999
               if (dst(i1) .ge. dst(L)) then
                  go to 66
               else
               L = i1
               endif
66       continue
         write(52,*) dst(L)
C
         k = k + 1
         if (k .gt. n) then
            go to 100
         endif
         j = 1
         j1 = 1
         go to 67
      endif
C
      if (k-j .eq. 0) then
         j = j + 1
      endif
C
67    dx = DABS(x(k) - x(j))
      dy = DABS(y(k) - y(j))
      dz = DABS(z(k) - z(j))
C
      if (dx .gt. xlng*0.5D0) then
         if (x(j) .gt. xlng*0.5D0) then
            xjp = x(j) - xlng
         else
            xjp = x(j) + xlng
         endif
         dx = DABS(x(k) - xjp)
      endif
C
      if (dy .gt. ylng*0.5D0) then
         if (y(j) .gt. ylng*0.5D0) then
            yjp = y(j) - ylng
         else
            yjp = y(j) + ylng
         endif
         dy = DABS(y(k) - yjp)
      endif
C
      if (dz .gt. zlng*0.5D0) then
         if (z(j) .gt. zlng*0.5D0) then
            zjp = z(j) - zlng
         else
            zjp = z(j) + zlng
         endif
         dz = DABS(z(k) - zjp)
      endif
C
         dst(j1) = DSQRT(dx*dx  + dy*dy + dz*dz)
C
      j = j + 1
      j1 = j1 + 1
      go to 6
C
100   close(unit=51)
      close(unit=52)
      stop
      end
C
C     ==============================================================
C                             End of program
C     ==============================================================
```

(b) Sorting the nearest distances obtained from code (a)

```
C   ==============================================================
C                       Exchange sort method
C   ==============================================================
C
C        Variables
C
C        a(i)      Arrays of distances obtained from code (a).
C        n         Number of distances.
C        dia       Diameter of sphere.
C
C     ..........................................................
C     Input and Output files
C
C     Unit 5     Defined as [adr7805.park.dist]sort.dat.  This is
C                the file from which the input data is read.
C
C     Unit 6     Defined as [adr7805.park.dist]sort.out.  This is
C                the file from which the output data is written.
C     ............................................................
C     Description
C
C     This program sorts the distances obtained from code (a) and
C     normalize them by diameter of a sphere.
C     ............................................................
C
C     ============================================================
C                       Beginning of Program
C     ============================================================
C
C
       implicit real*8 (a-h, o-z)
       dimension a(1000)
C
     open (unit=5,file='[adr7805.park.dist]sort.dat',status='old')
     open (unit=6,file='[adr7805.park.dist]sort.out',status='new')
C
       read (5,*) n
       read (5,*) dia
C
       do 3 i = 1, n
            read (5,*) a(i)
3      continue
C
       last = n - 1
C
       do 4 j = 1, n
            l = j
            j1 = j + 1
            do 110 k = j1, n
                   if (a(l) .le. a(k)) then
                      go to 110
                   else
```

```
                      l = k
                   endif
110         continue
C
       temp = a(l)
       a(l) = a(j)
       a(j) = temp
C
       if ((a(j)-a(j-1)) .ne. 0.0D0) then
       write (6,*) a(j)/dia
       endif
4      continue
C
       close (unit = 5)
       close (unit = 6)
C
       stop
       end
C
C      =============================================================
C                             End of program
C      =============================================================
```

(c) Cumulative probability of the normalized distances.

```
C   ==============================================================
C          Cumulative probability of the normalized distances
C   ==============================================================
C
C     Variables
C
C     dist        Arrays of normalized distances from code (b).
C     n           Number of distances.
C     dia         Diameter of sphere.
C     delbin      Bin width in unit of sphere diameter.
C     kbmax       Total number of bins.
C
C     ...........................................................
C     Input and Output files
C
C     Unit 51     Defined as [adr7805.park.dist]search.dat.  This
C                 is the file from which the input data is read.
C
C     Unit 52     Defined as [adr7805.park.dist]search.out.  This
C                 is the file from which the output data is written
C     .............................................................
C
C     Description
C     This program computes the cumulative probability of the nor-
C     malized nearest distances using the result of code (b).
C
C     ============================================================
C                             Beginning of Program
C     ============================================================
C
C
      Implicit real*8 (a-h,o-z)
      Integer freq(2000), sum
      Dimension dist(1100)
C
   open(unit=51,file='[adr7805.park.dist]search.dat',status='old')
   open(unit=52,file='[adr7805.park.dist]search.out',status='new')
C
      read(51,*) n
      read(51,*) delbin
C
      do 3 i = 1, n
           read(51,*) dist(i)
3     continue
C
      do 4 i = 1, 2000
           freq(i) = 0
4     continue
C
      kbmax = 0.001D0/delbin
      i = 1
```

```
110    if (i .eq. 360) then
          write(52,*) dist(i)
          endif
       if (i .gt. n) then
          go to 220
       endif
       ad = dist(i)-1.0D0
       call SEARCH (kbmax, delbin, ad, freq)
       i = i + 1
       go to 110
C
220    sum = 0
       rs = 1.0D0 + delbin
       do 40 j =1, kbmax
             sum = sum + freq(j)
             cum = FLOAT(sum)/FLOAT(n)
             write(52,53) rs, sum, cum
             rs = rs + delbin
40     continue
C
53     format(5X,F12.6,5X,I5,5X,F12.5)
       close(unit=51)
       close(unit=52)
C
       stop
       end
C
C      =============================================================
C                                End of Main Program
C      =============================================================
C
C
          Subroutine SEARCH (kbmax, delbin, ad, freq)
C
          Implicit Real*8 (a-h, o-z)
          Integer freq(2000), high
          eps1 = 1.0D-09
C
          low = 1
          high = kbmax
50        mid = (low + high)/2
          bmid = FLOAT(mid) * delbin
C
          if (ad .lt. bmid) then
             high = mid - 1
          endif
          if (ad .gt. bmid) then
             low = mid + 1
          endif
          if ((ad .le. bmid+eps1).and.(ad .ge. bmid-eps1)) then
             freq(mid) = freq(mid) + 1
             return
          endif
          if (low .le. high) then
             go to 50
```

```
   else
      isave = low
      freq(isave) = freq(isave) + 1
      return
   endif
end
```

APPENDIX C

C.1 Coordination number of pouring simulation

NO. OF CONTACTS	SPHERE SEPARATION BY DIAMETER		
	1.1	1.05	1.01
3	0.654	2.064	9.455
4	2.748	8.644	24.57
5	8.974	23.71	**33.78**
6	23.29	**32.58**	23.78
7	**32.08**	24.18	7.232
8	23.38	7.701	0.797
9	7.896	1.044	0.0
10	0.972	0.0	0.0
MEAN NUMBER	6.9	5.97	4.98

C.2 Coordination number of shaking simulation

NO. OF CONTACTS	SPHERE SEPARATION BY DIAMETER		
	1.1	1.05	1.01
3	-	0.178	6.595
4	1.07	4.635	21.39
5	4.10	13.90	**29.23**
6	12.12	29.06	25.13
7	**29.77**	**30.48**	14.08
8	31.55	17.29	3.030
9	17.65	4.099	0.357
10	3.743	0.357	-
MEAN NUMBER	7.55	6.55	5.29

C.3 Packing fraction by spherical growth method after pouring

spherical distance from the center	packing fraction		
	centers within		
	2 sphere dia.	3 sphere dia.	4 sphere dia.
0.3	0.538	0.543	0.535
0.35	0.573	0.578	0.566
0.4	0.584	0.587	0.574
0.45	0.571	0.572	0.558
0.5	0.558	0.558	0.542
0.55	0.560	0.560	0.542
0.6	0.566	0.564	0.544
0.65	0.569	0.566	0.543
0.7	0.568	0.562	0.539
0.75	0.565	0.558	0.532
0.8	0.564	0.555	0.528
0.85	0.565	0.554	0.526
0.9	0.565	0.552	0.523
0.95	0.563	0.549	0.519
1.0	0.562	0.545	0.514
1.05	0.560	0.541	0.510
1.1	0.558	0.538	0.506
1.15	0.556	0.534	0.502
1.2	0.553	0.530	0.498
average	0.563	0.555	0.532

C.4 Packing fraction by spherical growth method after shaking

spherical distance from the center	packing fraction centers within 2 sphere dia.	packing fraction centers within 3 sphere dia.	packing fraction centers within 4 sphere dia.
0.3	0.574	0.578	0.560
0.35	0.609	0.612	0.590
0.4	0.615	0.617	0.593
0.45	0.597	0.599	0.572
0.5	0.584	0.585	0.555
0.55	0.590	0.589	0.555
0.6	0.598	0.596	0.558
0.65	0.600	0.596	0.556
0.7	0.597	0.591	0.549
0.75	0.593	0.585	0.542
0.8	0.593	0.582	0.538
0.85	0.595	0.581	0.535
0.9	0.596	0.579	0.532
0.95	0.593	0.574	0.527
1.0	0.590	0.568	0.521
1.05	0.587	0.564	0.516
1.1	0.585	0.560	0.512
1.15	0.582	0.556	0.508
1.2	0.579	0.551	0.504
average	0.592	0.582	0.543

C.5 Packing fraction by plane growth method after pouring

r/dia.	packing fraction
2.9	0.5401192109310428
2.8	0.5533563455982388
2.7	0.5600851403260435
2.6	0.5615047610326287
2.5	0.5611386025702016
2.4	0.5599272515142990
2.3	0.5595471876137151
2.2	0.5585678923456342
2.1	0.5570778976261102
2.0	0.5562386420548790
1.9	0.5554193648712017
1.8	0.5556645800907506
1.7	0.5552448467437003
1.6	0.5530973602248455
1.5	0.5534938907057115
1.4	0.5536318213528416
1.3	0.5520493410288450
1.2	0.5509418419623947
1.1	0.5492418546867472
1.0	0.5464236879526412
0.9	0.5417447435574421
0.8	0.5385738956187314
0.7	0.5358313991082953
0.6	0.5289553649019375
0.5	0.5252080193302946

C.6 Packing fraction by plane growth method after shaking

r/dia.	packing fraction
2.8	0.5606383468702942
2.7	0.5766495711363379
2.6	0.5846561458142102
2.5	0.5857439493292930
2.4	0.5855629952082822
2.3	0.5850296185058922
2.2	0.5849551228148686
2.1	0.5851321397674422
2.0	0.5841343563632578
1.9	0.5853026573358648
1.8	0.5848086754926802
1.7	0.5831994237058778
1.6	0.5832984725730151
1.5	0.5815191561614208
1.4	0.5817628050301373
1.3	0.5833592809004963
1.2	0.5823838955573167
1.1	0.5824365402864338
1.0	0.5832531305759836
0.9	0.5819327194935322
0.8	0.5783872140324019
0.7	0.5782444496803609
0.6	0.5725269548270809
0.5	0.5726105533288898

C.7 Radial distribution after pouring

r/D	Radial distribution function
1.1	0.540864
1.3	0.149267
1.5	0.140551
1.7	0.195392
1.9	0.256588
2.1	0.183299
2.3	0.165834
2.5	0.181920
2.7	0.199566
2.9	0.190853
3.1	0.172073
3.3	0.172922
3.5	0.181047
3.7	0.177326
3.9	0.168625
4.1	0.166190
4.3	0.167453
4.5	0.166276
4.7	0.162159
4.9	0.157418
5.1	0.148973
5.3	0.133386

C.8 Radial distribution after shaking

r/D	Radial distribution function
1.1	0.580587
1.3	0.139755
1.5	0.136803
1.7	0.209765
1.9	0.272812
2.1	0.187088
2.3	0.163367
2.5	0.190731
2.7	0.211617
2.9	0.196455
3.1	0.173282
3.3	0.177219
3.5	0.188284
3.7	0.184894
3.9	0.172203
4.1	0.170800
4.3	0.170603
4.5	0.171321
4.7	0.165971
4.9	0.161886
5.1	0.151402
5.3	0.136503

C.9 **Cumulative probability of the normalized nearest neighbour distance r/dia. for 1000 sphere after pouring**

r/dia.	number	cumulative probability
1.000000	0	0.00000
1.000002	139	0.19577
1.000004	244	0.34366
1.000006	322	0.45352
1.000008	413	0.58169
1.000010	461	0.64930
1.000012	490	0.69014
1.000014	525	0.73944
1.000016	548	0.77183
1.000018	577	0.81268
1.000020	595	0.83803
1.000022	607	0.85493
1.000024	627	0.88310
1.000026	636	0.89577
1.000028	646	0.90986
1.000030	653	0.91972
1.000032	660	0.92958
1.000034	665	0.93662
1.000036	668	0.94085
1.000038	671	0.94507
1.000040	673	0.94789
1.000042	679	0.95634
1.000044	683	0.96197
1.000046	684	0.96338
1.000048	687	0.96761
1.000050	689	0.97042
1.000052	691	0.97324
1.000054	694	0.97746
1.000056	695	0.97887
1.000058	696	0.98028
1.000060	696	0.98028
1.000062	697	0.98169
1.000064	697	0.98169
1.000066	697	0.98169
1.000068	698	0.98310
1.000070	699	0.98451
1.000072	699	0.98451
1.000074	700	0.98592
1.000076	700	0.98592
1.000078	700	0.98592
1.000080	700	0.98592
1.000082	701	0.98732
1.000084	705	0.99296
1.000086	705	0.99296

1.000088	705	0.99296
1.000094	706	0.99437
1.000124	707	0.99577
1.000160	708	0.99718
1.000166	709	0.99859
1.000198	710	1.00000

C.10 Cumulative probability of the normalized nearest neighbour distance r/dia. for 1000 sphere after shaking

r/dia.	number	cumulative probability
1.000010	44	0.06162
1.000020	82	0.11485
1.000030	119	0.16667
1.000040	154	0.21569
1.000050	189	0.26471
1.000060	219	0.30672
1.000070	260	0.36415
1.000080	304	0.42577
1.000090	332	0.46499
1.000100	361	0.50560
1.000110	397	0.55602
1.000120	422	0.59104
1.000130	449	0.62885
1.000140	472	0.66106
1.000150	490	0.68627
1.000160	505	0.70728
1.000170	515	0.72129
1.000180	528	0.73950
1.000190	542	0.75910
1.000200	552	0.77311
1.000210	564	0.78992
1.000220	577	0.80812
1.000230	585	0.81933
1.000240	595	0.83333
1.000250	603	0.84454
1.000260	610	0.85434
1.000270	612	0.85714
1.000280	616	0.86275
1.000290	624	0.87395
1.000300	627	0.87815
1.000310	632	0.88515
1.000320	640	0.89636
1.000330	647	0.90616
1.000340	650	0.91036
1.000350	654	0.91597
1.000360	655	0.91737
1.000370	660	0.92437
1.000380	665	0.93137
1.000390	667	0.93417
1.000400	670	0.93838
1.000410	673	0.94258
1.000420	676	0.94678
1.000430	680	0.95238
1.000440	683	0.95658

1.000450	686	0.96078
1.000460	687	0.96218
1.000470	688	0.96359
1.000480	689	0.96499
1.000490	690	0.96639
1.000500	693	0.97059
1.000510	695	0.97339
1.000520	696	0.97479
1.000530	697	0.97619
1.000540	698	0.97759
1.000550	699	0.97899
1.000560	701	0.98179
1.000570	702	0.98319
1.000580	703	0.98459
1.000590	704	0.98599
1.000600	706	0.98880
1.000610	706	0.98880
1.000620	706	0.98880
1.000630	706	0.98880
1.000640	706	0.98880
1.000650	706	0.98880
1.000660	706	0.98880
1.000670	706	0.98880
1.000680	706	0.98880
1.000690	706	0.98880
1.000700	706	0.98880
1.000710	706	0.98880
1.000720	707	0.99020
1.000730	709	0.99300
1.000740	709	0.99300
1.000750	710	0.99440
1.000760	711	0.99580
1.000880	712	0.99720
1.000960	713	0.99860
1.000970	714	1.00000
1.001000	714	1.00000