4 Monte Carlo simulation of Hard spheres in the NVT ensemble

In this exercise we have a couple of assignment that have to be done. The web app used for plotting is found at `https://webspace.science.uu.nl/~herme107/viscol/`.

4.1

Here we had to make code that tiled the space whit spheres in a cubic lattice formation. The code for the generation of this lattice is found in Appendix A.1.
On the web app for plotting this lattice, Figure 1 was made using the file that was generated.

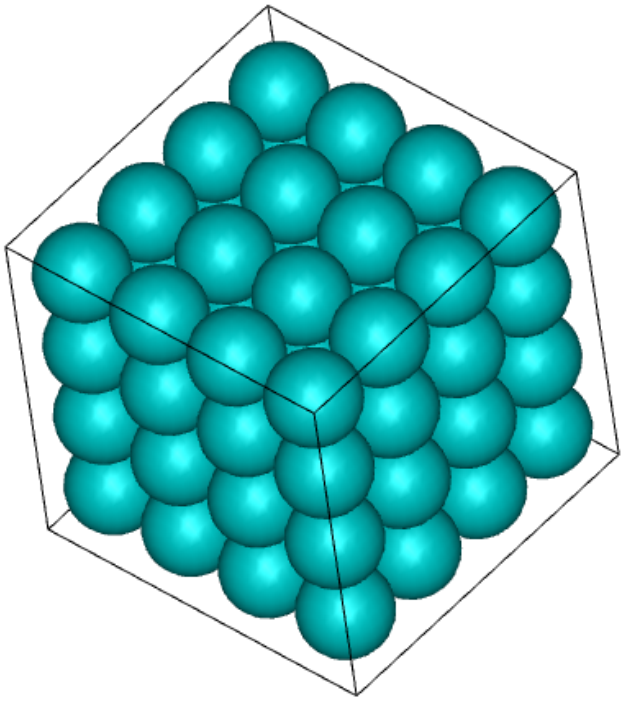

Figure 1: Here the cubic lattice generated by the code is graphed

4.2

We want to know the maximum packing density for spheres in a cubic lattice.
To get this first we need the lattice vector equation

$$\vec{R} = N_1\vec{a_1} + N_2\vec{a_2} + N_3\vec{a_3}. \tag{1}$$

Here the $N_i \in \mathcal{Z}$ is the counting number and $\vec{a_i}$ are the primitive translation vectors.
For the cubic case the vectors are unit vectors times the radius of the atoms. Dividing this up into unit cells gives us that only one atom may exist in the unit cell. meaning that the occupied fraction

$$f_o = \frac{V_p}{V_{uc}} = \frac{\frac{4}{3}\pi(\frac{a}{2})^3}{a^3} = \frac{\pi}{6} \tag{2}$$

Here V_p is the volume of particles occupying the unit cell, V_{uc} is the volume of the unit cell and a is the diameter of the particle.

4.3

Here we had to make code that tiled a space with spheres in a face-centered cubic (FCC) lattice. The code for the generation of this lattice is found in Appendix A.2.
On the web app for plotting this lattice, Figure 2 was made using the file that was generated.

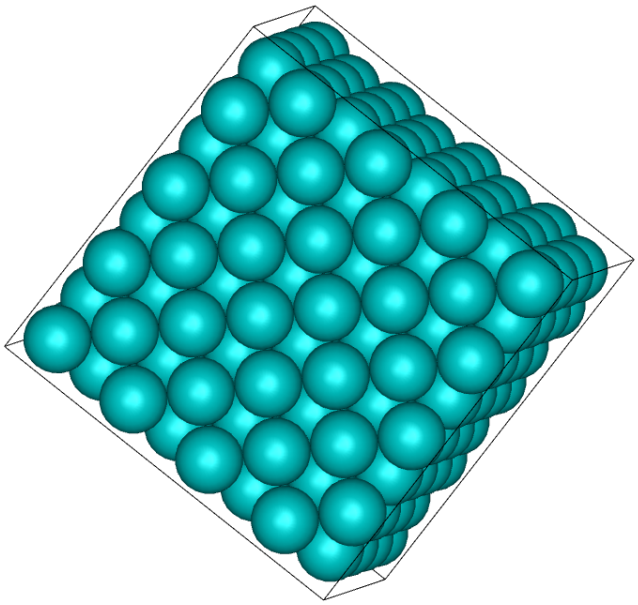

Figure 2: Here the cubic lattice generated by the code is graphed

4.4

We want to know the maximum packing density for spheres in a FCC lattice.
to get this we will use the same process as in 4.2. We know that our primitive translation vectors are

$$a_1 = \frac{a}{\sqrt{2}}\begin{pmatrix}0\\1\\1\end{pmatrix},\ a_2 = \frac{a}{\sqrt{2}}\begin{pmatrix}1\\0\\1\end{pmatrix},\ a_3 = \frac{a}{\sqrt{2}}\begin{pmatrix}1\\1\\0\end{pmatrix}$$

If we do this we have a problem, choosing the unit cell with length a in each direction leads to a non-translation symmetric unit cell. To get a translation symmetric unit cell we have to chose the length of the unit cell to be $\sqrt{2}a$.
Looking at the unit cell we can see that for each corner 1/8th of a sphere exists and we get 1/2 for each face of the cube, meaning the cube contains $8 * 1/8 + 6 * 1/2 = 4$ spheres
From this we know that if we look at how much particles would be in a unit cell we see that this would be

$$f_o = \frac{4V_p}{V_{uc}} = \frac{\frac{16}{3}\pi(\frac{a}{2})^3}{(\sqrt{2}a)^3} = \frac{\pi}{3\sqrt{2}} \tag{3}$$

4.5

A Code Exercise 4

A.1 Code 4.a)

```c
#include <stdio.h>
#include <math.h>
// in this file we will make the cubic latice

int main(){
  int N = 4; // The number of particles in each dirrection
  float d = 1.0; // the distance between two spheres
  float a = 1.0; // the radius of an sphere

  // Make a file where we can save the position data
  FILE *print_coords; // inititialises a file variable
  print_coords = fopen("cubic_xyz.dat","w"); // defining the file variable to be the opening
    of some file cubic.xyz

  // Let us print some initial coordinates
  fprintf(print_coords, "%i\n", N*N*N); // the total number of particles
  fprintf(print_coords, "%lf\t%lf\n", -0.0, 1.0*d*N); // The ocupied space in the x direction
  fprintf(print_coords, "%lf\t%lf\n", -0.0, 1.0*d*N); // The ocupied space in the y direction
  fprintf(print_coords, "%lf\t%lf\n", -0.0, 1.0*d*N); // The ocupied space in the z direction

  // we first initialise the particle possision saving arrays
  float x[N*N*N], y[N*N*N], z[N*N*N], r[N*N*N];


  // now we start generating particle possitions and radiuses
  int n = 0; // this is our counting variable, it wil index which particle we will consider

  /*
  The latice points are described by
  R= a_x n_x + a_y n_x + a_z n_z
  a_x = i a
  a_y = j a
  a_z = k a
  */


  // sweeping over the N_x particles
  for(int i=0; i<N; i++){
    // sweeping over the N_y particles
    for(int j=0; j<N; j++){
      // sweeping over the N_z particles
      for(int k=0; k<N; k++){
        // generating the possition for i,j,k latice cite, also the radius of the particle
        x[n]= (i+0.5)*d;
        y[n]= (j+0.5)*d;
        z[n]= (k+0.5)*d;
        r[n]= a;

        // saving the x,y,z possition and radius of the particle
        fprintf(print_coords, "%lf\t%lf\t%lf\t%lf\n", x[n], y[n],z[n],r[n]);

        n++;
      }
    }
  }


  fclose(print_coords);
  return 0;
}
```

A.2 Code 4.c)

```c
#include <stdio.h>
#include <math.h>
// in this file we will make the cubic latice

int main(){
  int N = 4; // The number of particles in each dirrection
  float d = 1.0; // the distance between two spheres
  float a = 1.0; // the radius of an sphere

```

```
// creating an distance variable that makes les typing
float l = sqrt(2.0)*d;

// defining the size of the box that will be spanned
float x_max = N*l;

// definging a variable such that the outline of the box aligns with the border of the particles
float s = 0.5*d;

// Make a file where we can save the position data
FILE *print_coords; // inititialises a file variable
print_coords = fopen("FCC_xyz.dat","w"); // defining the file variable to be the opening of some file cubic.xyz

// Let us print some initial coordinates
fprintf(print_coords, "%i\n", 4*N*N*N); // the total number of particles
fprintf(print_coords, "%lf\t%lf\n", -s, x_max-sqrt(2)*s+0.5*d); // The ocupied space in the x direction
fprintf(print_coords, "%lf\t%lf\n", -s, x_max-sqrt(2)*s+0.5*d); // The ocupied space in the y direction
fprintf(print_coords, "%lf\t%lf\n", -s, x_max-sqrt(2)*s+0.5*d); // The ocupied space in the z direction

// we first initialise the particle possision saving arrays
float x[4*N*N*N], y[4*N*N*N], z[4*N*N*N], r[4*N*N*N];

// now we start generating particle possitions and radiuses
int n = 0; // this is our counting variable, it wil index which particle we will consider

/*
The latice points are described by
R= a_1 n_x + a_2 n_x + a_3 n_z
a_1 = a/2 (j + k)
a_2 = a/2 (i + k)
a_3 = a/2 (i + j)
i, j, k are the unit vectors in x, y and z directions respectively (not the counts)
*/



// sweeping over the N_x particles
for(int i=0; i<N; i++){
  // sweeping over the N_y particles
  for(int j=0; j<N; j++){
    // sweeping over the N_z particles
    for(int k=0; k<N; k++){
      // generating the possition for i,j,k latice cite, also the radius of the particle

      // first we start on the base vector because we know this patern reapeats every 2*unit vector in each direction
      x[n]= (i)*l;
      y[n]= (j)*l;
      z[n]= (k)*l;
      r[n]= a;

      // saving the x,y,z possition and radius of the particle
      fprintf(print_coords, "%lf\t%lf\t%lf\t%lf\n", x[n], y[n],z[n],r[n]);

      n++;

      // here we will add the a_1 vector and make the same spacing
      x[n]= (i)*l;
      y[n]= (j+0.5)*l;
      z[n]= (k+0.5)*l;
      r[n]= a;

      // saving the x,y,z possition and radius of the particle
      fprintf(print_coords, "%lf\t%lf\t%lf\t%lf\n", x[n], y[n],z[n],r[n]);

      n++;

      // here we will add the a_2 vector and make the same spacing
      x[n]= (i+0.5)*l;
      y[n]= (j)*l;
      z[n]= (k+0.5)*l;
```

```
        r[n]= a;

        // saving the x,y,z possition and radius of the particle
        fprintf(print_coords, "%lf\t%lf\t%lf\t%lf\n", x[n], y[n],z[n],r[n]);

        n++;

        // here we will add the a3 vector and continue the same spacing
        x[n]= (i+0.5)*l;
        y[n]= (j+0.5)*l;
        z[n]= (k)*l;
        r[n]= a;

        // saving the x,y,z possition and radius of the particle
        fprintf(print_coords, "%lf\t%lf\t%lf\t%lf\n", x[n], y[n],z[n],r[n]);

        n++;









      }
    }
  }


  fclose(print_coords);
  return 0;
}
```