Crystallization and Glassy Behaviour in Short-Range Attractive Square-Well Fluids

Tristan Hartskeerl
Soft Condensed Matter group
Debye Institute for Nanomaterials Science
Utrecht University

June 12, 2009

Contents

Abstract

Square-well systems with attractive ranges of over 25% of the particle diameter are thoroughly studied in simulations. Most of these simulations are performed with monodisperse or binary systems while experiments are most frequently performed with polydisperse systems. In systems with attractive ranges significantly shorter than the particle diameter the gas-liquid transition becomes metastable. This metastable coexistence region plays a vital role in the formation of gels and is therefore particularly interesting. Using Molecular Dynamics we measure the metastable gas-liquid coexistence and the diffusion of a polydisperse square well system near the attractive glass transition. The attractive range was chosen to be 3% of the mean particle diameter. The polydispersity was chosen to be 10 percent to avoid the formation of crystals and to allow measurements near close packing. We apply two methods for computing the densities of the two bulk phases during phase separation and compare the results. Based on the gas-liquid coexistence measurements we can make a rough estimate of the critical temperature and critical density of this system. We compare the shape of the coexistence curve to previous results for systems with longer-ranged potentials and conclude that it is unlikely the critical behavior is classically quadratic or cubic and that the critical temperature is lower for shorter attractive ranges. The long time diffusion seems to reduce to very small values near the glass transition, in agreement with previous results on a similar system with a binary mixture. We interpret our failure to equilibrate the system at attraction strengths above the glass transition predicted by MCT and the rapid decay of diffusion in the direction of the glass line as a confirmation of a dynamic glass transition near the glass transition as obtained from MCT.

Chapter 1

Introduction

1.1 Introduction

As shown by Bolhuis and Frenkel [1, 2] the phase behavior of systems with attractive potentials is strongly dependent on the range of the attraction. It is agreed upon that for systems with very short-ranged attraction relative to the particle diameter the stable gas-liquid transition disappears. This phenomenon was first noted in theoretical work by Gast, Russel and Hall [3] and further investigated by Lekkerkerker *et al* [4]. Evidence comes from both simulations [5, 6] and experiments [7]. The disappearance of the stable gas-liquid transition is not sensitive to the precise form of the potential. It is therefore easy observable experimentally. Noro and Frenkel [8] define the range of an attractive potential to be the range of a square-well potential that behaves similarly. More precisely, the range of a potential is defined as the range of a square-well potential that yields a system with the same reduced second virial coefficient at the same reduced temperature. This definition enables the comparison of results with different potentials and seems to work best for attractive potentials with no repulsive barrier and where three-body (or more) interactions play no role. Using this definition, they state that the gas-liquid transition becomes metastable when the range of the attraction becomes less than approximately 0.14 [8].

Dynamic light-scattering experiments, Mode Coupling Theory calculations and simulations of systems with short-ranged attractions by Poon *et al* [9] show that these systems can exhibit glassy behaviors. Their measurements, performed on colloidal dispersions in which polymer concentration controls the attractive range of the particles, were performed ofer a wide range of densities and attractive ranges and show two types of glassy states. If the density of an equilibrium fluid, where particles can freely diffuse over long distances, approaches the glass transition the system gets dynamically arrested. Although the particles retain some freedom of motion they are essentially trapped locally. The effect that causes this seems to be the formation of cages around particles by their neighboring particles [10]. If the density is low enough the particles can easily escape from these cages but if the density gets higher it becomes increasingly unlikely that a particle is able to escape its cage. This mechanism seems to play an important role for low attraction strengths where the attractive part of the potential is low with respect to the kinetic energy of the particles. For higher attraction strengths the attractive part of the potential produces clusters of particles at short distances, a different mechanism that causes dynamic arrest. From a practical point of view the diffusion time scale has an exponential dependence on the distance (temperature and density wise) to the glass transition [11].

The diffusion dynamics of short-range square-well fluids in the reentrant regime near the glass transition curve have been studied using Molecular Dynamics by Zacarelli *et al* [12]. Theoretical predictions and comparison to simulations of the glass line for this type of system have been made in detail by [13]. MD simulations near the attractive glass line have been performed by Foffi *et al* [14]. Gas-liquid coexistence curves of square-well fluids with medium-to-long range have been measured by [15].

The theoretical predictions by [13] of the glass-glass and glass-liquid transitions of a $\lambda = 0.03$

square-well fluid is shown in Figure 1.1 along with isodiffusivity curves of the same system obtained by simulation by [12] and gas-liquid coexistence curves for fluids with longer-ranged interactions. We also indicated roughly the location of the coexistence curve for a short-range fluid following the trends of the longer-ranged curves from [15]. Simulations performed recently use a binary mixture of particles where the species differ in particle size. The simulations used to measure the coexistence curves mentioned before use a monodisperse system.

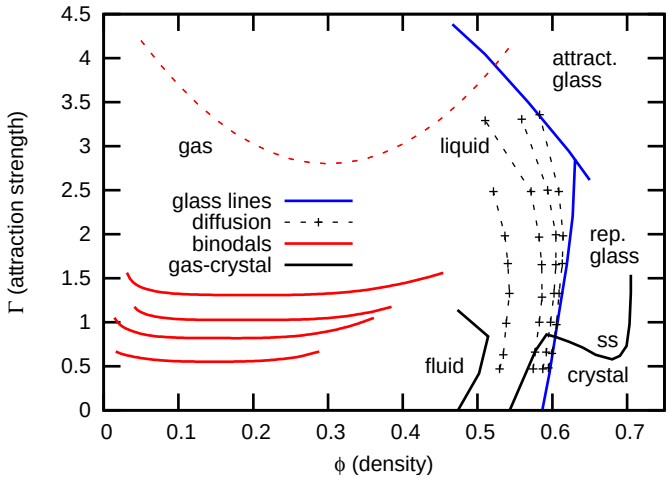


Figure 1.1: *Phase diagram of a $\lambda = 0.03$ square-well fluid (unless noted otherwise). Blue: theoretical MCT calculations using the Percus-Yevic approximation of the structure factor for liquid-glass and glass-glass transitions from [13]. Red: Measurements of gas-liquid coexistence curves for square-well systems with a varying well width $\lambda \neq 0.03$ by [15] (see Figure 3.7). The dashed line is an indication of where one would expect to find the gas-liquid coexistence curve. Black: Isodiffusivity curves measured by [12] (see Figure 3.10). The labels **attrac. glass** and **rep. glass** denote the regions where the interactions in a glass are dominated by attraction and repulsion respectively. **gas** and **liquid** at the top indicate the gas and liquid branches of the coexistence curve. The solid black line shows gas-crystal and solid-solid coexistence data from [2] where **fluid** and **solid** indicate the fluid and solid phase. In between the lines there is fluid-crystal (FCC) coexistence. **ss** indicates the solid-solid binodal where there is coexistence between a high and low density FCC crystal.*

As can be seen in the isodiffusivity measurements by [12] (shown as black dashed lines in Figure 3.10) for a certain density the diffusion increases and then decreases again if the attraction strength is increased. This reentrant behavior can be explained by two competing mechanisms. At low attraction strengths the interactions are dominated by the cage mechanism (hard-core repulsion). By slightly increasing the attraction strength the particles tend to stick closer, thereby increasing the diffusion. Further increasing the attraction strength results in the particles getting stuck close together and forming clusters because they lack the energy to overcome the square-well barrier, again decreasing diffusion. Plotted in blue are the two (attractive and repulsive) glass lines. They intersect at some point and for higher densities a glass-glass transition remains, eventually terminating in what is called the A_3 singularity in MCT language. Also shown are the data for melting, freezing and solid-solid binodal (coexistence between two FCC crystals with different densities, strongly dependent on attractive range) from [2], labeled **gas-crystal**.

This study will focus on systems with a log-normal distribution in particle size with short-range square-well particles with an attractive range of 3% of σ. We have selected a polydispersity of 10% which helps preventing crystallization and allows us to do diffusion measurements near the glass transition at high density. We will focus on the attractive glass region (see Figure 1.1) where the interactions between particles will be dominated by attractions (as opposed to repulsion). We will investigate the gas-liquid coexistence curve for these systems as well, also looking for signs of the glass transition line crossing the coexistence curve.

1.2 Model System

1.2.1 Hard Sphere Potential

The hard sphere (HS) potential is a very simple pair potential that models an infinitely strong repulsive potential by not allowing two spherical particles to overlap. Because of its simplicity it is widely used in theoretical work and simulations. The hard sphere potential of two particles at position $\vec{r}_1$ and $\vec{r}_2$ is given in Equation 1.1. $\bar{\sigma}$ is the smallest distance the two particles can be away from each other without overlap ($\bar{\sigma} = \frac{1}{2}(\sigma_1 + \sigma_2)$ where σ_1 and σ_2 are the particle diameters). The left graph in Figure 1.2 shows the HS potential, where we define $r = |\vec{r}_1 - \vec{r}_2|$ as the center of mass distance.

$$V(r) = \begin{cases} \infty & \text{if} \quad r < \bar{\sigma} \\ 0 & \text{if} \quad r \geq \bar{\sigma} \end{cases} \tag{1.1}$$

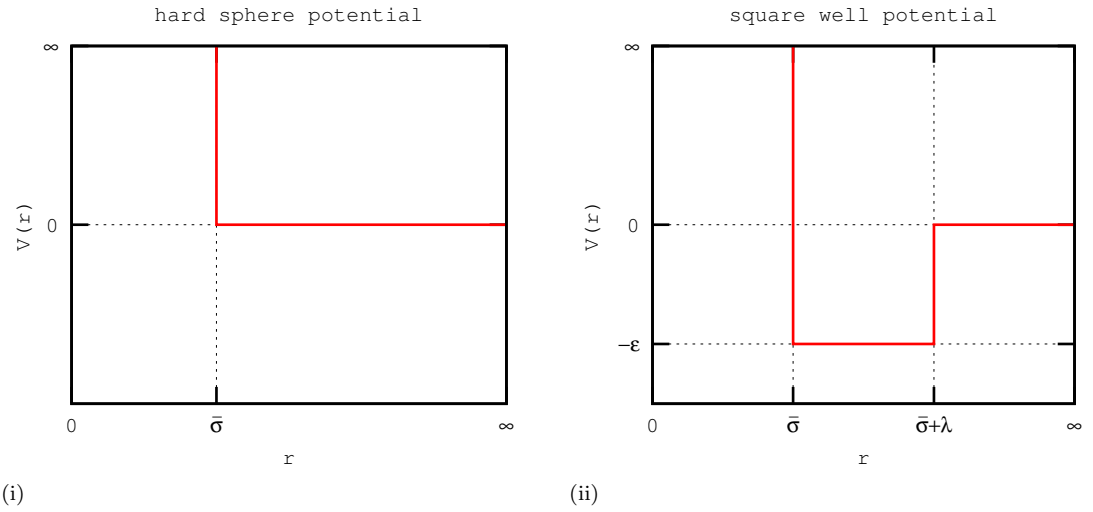


Figure 1.2: *(i) the HS potential and (ii) the SW potential. In both graphs $\bar{\sigma}$ is the smallest distance two particles can be away from each other without causing overlap and λ and ϵ are the width and the depth of the attractive well, respectively.*

1.2.2 Square Well Potential

The square well (SW) potential has an attractive region in addition to a strongly repulsive part. It models particles with a repulsive core and a typically short-to-medium range attraction. The strength of the attraction is tunable.

Figure 1.2 shows the square well potential on the right. The name *square well* is derived from the fact that the attractive region is rectangular. Although the potential is slightly more complex than the hard sphere potential, it is still used in many theoretical predictions. The width of the well is λ, the depth of the well is ϵ.

The SW potential is defined in Equation 1.2. Note that a more common way to specify the width of the attractive region is to express it as a fraction of $\bar{\sigma}$.

$$V(r) = \begin{cases} \infty & \text{if} \quad 0 & \leq r < & \bar{\sigma} \\ -\epsilon & \text{if} \quad \bar{\sigma} & \leq r < & \bar{\sigma} + \lambda \\ 0 & \text{if} \quad \bar{\sigma} + \lambda & \leq r < & \infty \end{cases} \tag{1.2}$$

1.2.3 Parameters

We perform event-driven Molecular Dynamics simulations on a system of 200 to 1000 square well particles in a box periodic in all directions. We use a log-normal distribution for the particle diameters with standard deviation $\sigma = 0.1$ and mean $\mu = -\frac{1}{2}\sigma^2$ yielding a distribution very similar to a normal distribution with parameters $\sigma = 0.1$ and $\mu = 1$. The reason for employing a log-normal distribution is that we do not want particles with radius < 0 but we also did not want to cut off the tail of the normal distribution. Regardless of being polydisperse all particles have the same mass in order to simplify the equations of motion. Parameter σ is kept as low as possible because it benefits the simulation (see Section 2.13). For convenience we define attraction strength $\Gamma = \epsilon(k_B T)^{-1}$ with k_B Boltzmann's constant and T the temperature.

Chapter 2

Step Potential Molecular Dynamics Algorithm

2.1 Molecular Dynamics

Molecular dynamics basically involves solving the many-body problem for particles with a certain potential. Usually the potential energy function consists of a sum of pair potentials and an external field which are described by a certain potential energy function. In this thesis we do not apply an external field.

The idea is to start with a set of particles with initial velocities and solve their equations of motion using numerical integration e.g. the verlet or the Runge-Kutta method.

Due to the discontinuities in the square well potential numerical integration of the equations of motion does not work as the numerical integration uses the first and/or second derivatives (gradients) of the pair potential. Since our pair potential is piecewise constant its derivatives are all zero except at the discontinuities. Numerical integration schemes only sample the derivatives at discrete points - they assume the derivatives are smooth - and thus they miss the discontinuities with a very high probability. Because the discontinuities are the only parts of the potential that cause interaction there is no point in using this method to evolve the simulation; it will not work.

Instead we need a method that is driven by the discontinuities in the potential. Since all derivatives of the potential are zero almost everywhere the particles move in straight lines most of the time. When they do interact the interaction is instantaneous changing only the velocity, or more precisely the energies and momenta but not the total momentum, of the two interacting particles.

There are two distances at which particles interact. The most obvious one is at the hard-sphere discontinuity at a distance of $\frac{1}{2}(\sigma_1+\sigma_2)$, the second one is at the square-well discontinuity at a distance of $\frac{1}{2}(\sigma_1+\sigma_2)+\lambda$. We will refer to the moments in time of these interactions as events. This results in is an event-driven method that does not have a fixed time step but rather advances the simulation by a series of discrete steps (events).

2.2 The Algorithm

Because the particles move in straight lines in between collisions, determining whether or not and at what moment in time a collision occurs is straightforward. We will refer to this as checking or detecting collisions. The properties of a particle we need to keep track of are listed in Table 2.2.

The basic event-driven algorithm, shown in Algorithm 1 is very simple. Without any optimizations, however, it is impractically slow since we are checking all possible pairs of particles for collision at every event. The algorithm can be made faster using simple optimizations. We

Algorithm 1: The basic algorithm

```
while time < length of simulation do
    Check all possible pairs of particles for collision;
    Pick the first one;
    Advance the simulation to the time of the first collision;
    Update the velocities of the particles involved;
end
```

are going to introduce the *event calendar*, a structure that stores the results of collision checks so we can partially avoid re-checking for collisions in Section 2.3. We are also going to introduce cell lists that enable us to perform the collision checks more efficiently in Section 2.4.

Detecting Collisions

In order to determine when a collision between two particles will take place we express their separation in terms of time and equate that to their separation when they touch. Let particle 1 be defined by its radius r_1, its position $\vec{x}_1$ and velocity $\vec{v}_1$ at time t_1 and particle 2 by r_2, $\vec{x}_2$, $\vec{v}_2$ and t_2. Furthermore let $\vec{a} = \vec{v}_1 - \vec{v}_2$ and $\vec{b} = \vec{x}_1 - \vec{x}_2 - \vec{v}_1 t_1 + \vec{v}_2 t_2$. The time dependent distance $d_{a,b}$ between the two particles can be expressed as

$$d_{a,b} = \sqrt{\langle \vec{a}t + \vec{b}, \vec{a}t + \vec{b} \rangle} \tag{2.1}$$

where t is the simulation time. Equating this to $(r_1 + r_2)$ and squaring on both sides yields the quadratic equation

$$\langle \vec{a}, \vec{a} \rangle t^2 + 2\langle \vec{a}, \vec{b} \rangle t + \langle \vec{b}, \vec{b} \rangle = (r_1 + r_2)^2. \tag{2.2}$$

If we define $A = \langle \vec{a}, \vec{a} \rangle$, $B = 2\langle \vec{a}, \vec{b} \rangle$ and $C = \langle \vec{b}, \vec{b} \rangle - (r_1 + r_2)^2$ we can write the (two) solutions as

$$t_\pm = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \ . \tag{2.3}$$

If $B^2 - 4AC < 0$ there will be no collision. In case of a hard sphere collision we check if $t_- > t$ where t is the current time for the simulation. If this is true, we have found a future collision. We do not look at past collisions and we are not interested in t_+ because that solution is exactly the moment in time that the two particles stop overlapping.

In case we are checking for a square-well collision, we need to replace $r_1 + r_2$ by $r_1 + r_2 + \lambda$ where λ is the well width. If the particles are close, meaning $d_{a,b} < r_1 + r_2 + r$, t_+ is the solution we are looking for because it is the moment in time the particles leave the well. Otherwise t_- is the correct solution.

Collision Response

All collisions, hard-core or square-well collisions alike, preserve energy and momentum. In case of a hard-core collision both particles must reverse direction because the energy barrier is too high to overcome. We are going to explain how the velocities of the particles are changed in the zero-impulse system of the two particles.

Let $\vec{dr} = \vec{x}_2 - \vec{x}_1$ be the vector that points from particle 1 to particle 2. The velocity of the particles will change only in the direction of $\vec{dr}$. Let

$$a = \frac{\langle \vec{v}_1, \vec{dr} \rangle}{\sqrt{\langle \vec{dr}, \vec{dr} \rangle}} \quad \text{and} \quad b = \frac{\langle \vec{v}_2, \vec{dr} \rangle}{\sqrt{\langle \vec{dr}, \vec{dr} \rangle}}. \tag{2.4}$$

a and b are the velocities along $\vec{dr}$ that the particles have. We want to find the new values for a and b, called c and d respectively, that satisfy conservation of energy and momentum. The particles, although not of equal size, all have the same mass.

Hard-core collisions

We can express the conservation of energy and momentum in the variables defined above; $a^2+b^2 = c^2 + d^2$ (energy) and $a + b = c + d$ (momentum). Solving the energy conservation equation for c by substituting $d = a+b-c$ from the momentum conservation equation we get $c^2 - c(a+b) + ab$ which is again a quadratic formula. Let $A = 1$, $B = -(a + b)$ and $C = ab$. The solutions to the above equation are $c = \frac{1}{2}[(a + b) \pm \sqrt{(a + b)^2 - 4ab}]$ yielding $c = a$ or b. The correct solution is b because the other solution does not change anything. Now that we know c we can deduce, and probably guess, $d = a + b - c = a$.

Square-well collisions

In a square-well collision particles either lose or gain energy depending on whether they move away or approach each other, respectively, assuming ϵ is negative. This means we need to add a term for the potential jump in the energy conservation equation.

First we will handle the case when two particles are approaching each other and consequently gain energy by adding ϵ to the left side of the energy conservation constraint. We now need to solve $a^2 + b^2 + \epsilon = c^2 + d^2$ and $a + b = c + d$. Solving the first equation to c by substituting $d = a + b - c$ yields

$$c = \frac{1}{2}[a + b \pm \sqrt{(a + b)^2 - 4ab - 2\epsilon)}]. \tag{2.5}$$

We can pick the correct solution by letting ϵ go to zero. If the particles are moving towards each other and have no interaction because ϵ is zero $c = a$ so we pick

$$c = \frac{1}{2}[a + b + \sqrt{(a - b)^2 - 2\epsilon}]. \tag{2.6}$$

If we substitute c back into $d = a + b - c$ we find that

$$d = \frac{1}{2}[a + b - \sqrt{(a - b)^2 - 2\epsilon}]. \tag{2.7}$$

Second we handle the case when to particles move away from each other. If the total energy of the two particles is lower than ϵ the energy barrier is too high to overcome and the particles just bounce like in a hard-core collision. Otherwise, solve the same problem as for approaching particles but we add ϵ to the right side of the energy conservation equation instead. This effectively flips the sign of ϵ so we can basically use the same solutions as before, yielding

$$c = \frac{1}{2}[a + b + \sqrt{(a - b)^2 + 2\epsilon}] \quad \text{and} \quad d = \frac{1}{2}[a + b - \sqrt{(a - b)^2 + 2\epsilon}]. \tag{2.8}$$

2.3 Event Calendar

It is computationally expensive to check the same pairs of particles for collision each time even though their trajectories have not changed. Therefore we will store all the collisions we find in a structure called the event calendar. When searching for the first collision that is going to occur we find many potential collisions. Some will not happen because the particles involved have their trajectories changed before these collisions can actually take place. Some of them *will* happen. We can make the simulation faster by storing all of the collisions we found and, at each collision, remove the ones involving particles whose trajectories changed.

The event calendar would ideally be able to quickly report the first event it contains as well as quickly delete all of the events involving a specific particle. To this end the events are stored in *event nodes* that are both in a binary search tree and two circularly linked lists - the former allows for quick sorting on time, the latter allows for quick deletions.

Event Types

So far we have only seen one type of event; a collision. Since all our algorithm does is evaluate the events in chronological order we can insert events of any kind into the event calendar. For example, we can add an event type that triggers the measurement of some quantity of the system. The cell list method (see Section 2.4) introduces a type that triggers the code to handle a cell boundary crossing which is used for faster collision detection as well as periodic boundaries.

Each event is described by four properties; the time at which it occurs (*time*), the type (*type*) of the event and the (up to) two particles involved (*idA* and *idB*). An event node contains the properties of an event as well as the pointers for the binary search tree and the circularly linked lists. An overview of the structure of an event node is listed in Table 2.1.

Field	Description
Event properties	
`time`	Time of event
`type`	Type of event
`idA`	ID of particle A
`idB`	ID of particle B / flags
Event calendar	

`left`
`right`
`up`
	Binary Tree data

`circAL`
`circAR`
`circBL`
`circBR`
	Circular List data

Table 2.1: *The fields in the event node structure.*

Field		Description
`x`	(vector)	Position
`v`	(vector)	Velocity
`xmoved`	(vector)	Total distance moved
`t`		Time
`r`		Radius
Cell list		
`ncell`		Index into cell list
`cell[3]`		Coordinates of cell

Table 2.2: *Properties of a particle.*

2.4 Cell Lists

Most particles in our simulation will have a very high probability of colliding with particles that are in their neighbourhood. Therefore it will pay off to check those particles first when searching for the first collision of a given particle. Later we will see that we only need to check for collisions with the particles directly surrounding the particle we are checking. An example is shown in Figure 2.1 - the gain of using cell lists can be considerable.

We use a event type to handle cell crossings so the cell lists will be updated whenever a particle moves from one cell to an adjacent cell, this involves removing the particle from the cell

list it belongs to and inserting it into the cell list belonging to the cell the particle is moving to. When the event is handled, we re-check the particle for collisions. This ensures we miss no collisions in case the first collision for a particle is outside of its adjacent cells. It also paves the way for periodic boundaries because all we need to do is move the particle to the other side of the box if it crosses the boundaries of the simulation box (which are also cell boundaries).

In order to speed up collision queries we subdivide the simulation box in cells just large enough to hold the largest particle, including all of the discontinuities of the potential, such that if two particles are more than one cell away in a particular direction they can not interact. Associated with each of these cells is a list (a *cell list*) containing all of the particles whose center lies in the box. In case we want to find the first collision candidate for a given particle we only need to check for collisions with particles in the same or neighbouring cells. Since we recompute events for possible collisions when a particle crosses a cell boundary we do not have to check any other particles.

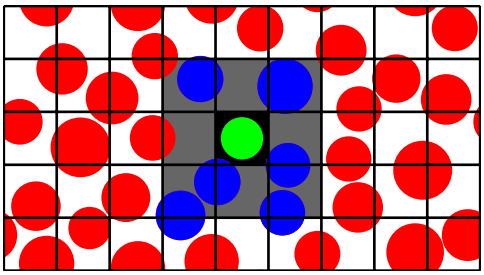


Figure 2.1: *Cell lists in action. The center green particle, in the black cell, is being checked for collisions. The blue particles are either in the same or an adjacent cell. Adjacent cells are gray. The red particles are omitted from the search.*

The cell lists can be either linked lists, singly-linked lists, binary search trees or arrays. The maximum number of particles overlapping a cell is bounded (but the exact bound depends on the size of the cell, the particles and polydispersity) which means an operation on any of those structures is a constant time operation. We used a singly linked list.

2.5 Binary Tree

A binary tree is a structure in which nodes are stored in an ordered way. Each node points to two other nodes; its left and right child nodes. There is also an ordering on the nodes. The tree is always constructed and kept such that $time(leftchildren(n)) < n < time(rightchildren(n))$ for all nodes n. The functions $left(n)$, $right(n)$ and $up(n)$ give pointers to the left and right child nodes and to the parent node, respectively of node n, $time(n)$ is the time which the event to which node n corresponds takes place. We chose to include the up pointer for convenience. During insertions and deletions we need to take special care that the ordering relation holds for all nodes.

There is one special node; the *root node*. The root node does not describe an actual event but its left child (but we could as well have picked the right child) points to the "first" node in the tree. Using a node instead of just a pointer to the first node makes our work easier when implementing deletions.

Insertion

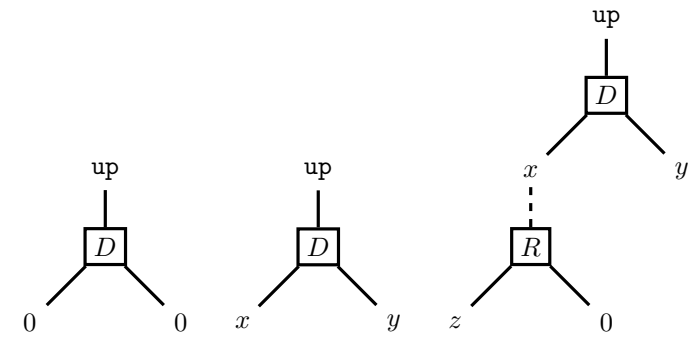
In order to insert an element into the binary tree we start at the root node of the tree. We keep picking either the left or right child of the node based on $time(n)$ of the current node n (we pick left if $time(n) \leq time(p)$ else we pick right) until the child we just picked does not yet exist. The last node we found will be the parent of the new node. The pseudo code for this operation is listed in Algorithm 2.

Algorithm 2: Binary tree insertion

```
Input: n - The node to be inserted
p ← left(root);
while left(p) ≠ 0 or right(p) ≠ 0 do
    if time(left(p)) < time(n) then
        p ← left(p);
    else
        p ← right(p);
    end
end
up(n) ← p;
if time(n) < time(p) then
    left(p) ← n
else
    right(p) ← n
end
```

Deletion

Deleting a node from the binary tree is a little more complicated due to the fact that the node to be deleted may have child nodes that need to remain attached to something. We can divide the problem in three cases of increasing difficulty. Figure 2.2 shows the three cases labeled A, B and C. Suppose we want to delete node D. In Case A, node D has no children and can be removed easily. In Case B, node D has exactly one child (that is, either x or y is 0) so we can remove D as long as we update its child too.



(i) Case A (ii) Case B (iii) Case C

Figure 2.2: *Binary tree deletions (see text)*

Case C is the most complicated since node D has two children (both x and y are not zero). D is going to be replaced by a node (R) that can replace D without breaking the binary tree's properties. We need to find a node that splits up the left and right children of D just as D does. It is easy to find such a node; we pick the rightmost node of the left child nodes of D. It has the property that no other left child is smaller. The pseudo code for finding the replacement node is shown in Algorithm 3. We know the initial $left(n)$ statement is valid because we assumed both children of n exist. The pseudo code for the Case C deletion is in Algorithm 4. Note how it first unlinks the rightmost left child, then copies the child pointers of D, updates the pointers of the nodes that D points to, then unlinks D.

We tried to implement a *self-balancing* binary tree but after a few attempts decided to give up as it was not worth the time. In Ref. [16] it was stated that the tree is not likely to grow very deep and unnecessarily enforcing that by writing a piece of fairly complex code is probably not going to make a significant improvement on the efficiency of the algorithm. The amount of operations needed to insert or delete anything from the tree is proportional to its depth. In the case of a balanced binary tree, its depth is proportional to $\log n$ where n is the number of nodes.

Algorithm 3: *GetRightmostLeftChild* algorithm

Input: n - the node we want the rightmost left child of
$n \leftarrow left(n)$;
while $right(n) \neq 0$ **do**
  $n \leftarrow right(n)$;
end
return n

Algorithm 4: Case C Deletion

Input: D - the node to be deleted
$R \leftarrow GetRightmostLeftChild(D)$;
if $right(up(R)) = R$ **then**
  $right(up(R)) \leftarrow 0$;
else
  $left(up(R)) \leftarrow 0$;
end
$\{up(R), right(R), left(R)\} \leftarrow \{up(D), right(D), left(D)\}$;
$\{up(left(R)), up(right(R))\} \leftarrow \{R, R\}$;
if $right(up(R)) = D$ **then**
  $right(up(R)) = R$;
else
  $left(up(R)) = R$;
end

2.6 Circular List

A typical event involves one or two particles. In case the event becomes invalid due to one or both of these particles having their velocity changed or another event causing a particular particle to be re-checked for collision all of the other events involving either of those particles need to be removed from the event calendar. In order to quickly do this, each event node involving one or two particles is in one or two circular lists. If we want to delete the event nodes involving some particles that have become invalid we remove them by removing all of the nodes from the corresponding list. The data members of an event node that are used for these circular lists are shown in Table 2.1.

Associated with every particle is one circular list. All of the events that directly affect a particle are then chain-linked to one of those two lists belonging to a particle depending on whether they refer to the particle by their idA or idB pointer. For each particle there is a dummy event node that will serve as a root node for that particle's circular list by using its circular list pointers. A circular list consists of two chains; chain A and B. If an event involves a particle pointed to using its idA member the event will be linked to the A chain of the particle, otherwise it will be linked to the B chain.

Insertion

Insertion of a node into a single chain is pretty straightforward. We do it by updating $L(R(r)) = n$ and $R(n) = R(r)$ thus linking the new event to the right node of the root by the right side. We then fix the other two links by doing $R(r) = n$ and $L(n) = r$ such that the root node points to the new node on the right side (where L and R indicate the left and right pointers, respectively, of the chain we are inserting the node in). In case an event involves two particles we insert the node in two lists. Algorithm 5 lists the pseudo code for an insertion.

Algorithm 5: Circular List Insertion

Input: n - the event node to be inserted
$root(idA).circAR.circAL = n$;
$n.circAR = root(idA).circAR$;
$root(idA).circAR = n$;
$n.circAL = root(idA)$;
if *n.idB refers to a particle* **then**
  repeat the above for idB and circBL/circBR;
end

Deletion

We always remove all of the event nodes associated with a particular particle, never single nodes. As such we can walk the chain of a root node and remove all of the nodes we encounter, making sure the pointers to them from their neighbours are updated to point to each other before we discard the node. The operations we perform are similar to what we do for an insertion but in reverse. We can stop walking the chain if we reached the root node again since that indicates we walked across all of the nodes in a circular list. After we delete all nodes in the chain all that remains is a root node pointing to itself. Algorithm 6 lists the pseudo code for this deletion. Note that we leave all the nodes we want to remove linked by their *circAR* pointers due to the way we manage nodes that are free to be reused by the node pool allocator (see Section 2.7).

Algorithm 6: Circular List Deletion

Input: id - the node to delete
`# walk the A chain`
$idd \leftarrow id.circAL$;
while $idd \neq id$ **do**
  $idd.circBL.circBR = idd.circBR$;
  $idd.circBR.circBL = idd.circBL$;
  remove idd from the binary tree;
  $idd \leftarrow idd.circAL$;
end
`# used for memory pool`
`# poolnode is the root node used for the free node pool`
$id.circAL.circAR \leftarrow poolnode.idA$;
$poolnode.idA \leftarrow id.circAR$;
`# make the A chain root point to itself`
$id.circAL, id.circAR \leftarrow id$;
`# walk the B chain`
$idd \leftarrow id.circBL$;
while $idd \neq id$ **do**
  $idd.circAL.circAR \leftarrow idd.circAR$;
  $idd.circAR.circAL \leftarrow idd.circAL$;
  remove idd from the binary tree;
  $idd.circAR \leftarrow poolnode.idA$;
  $poolnode.idA \leftarrow idd$;
  $idd \leftarrow idd.circBL$;
end
`# make the B chain root point to itself`
$id.circBL, id.circBR \leftarrow id$;

2.7 Event Node Pool

We are creating and discarding a huge number of nodes during a simulation. Dynamic memory allocation slows down the simulation and fragments memory. Instead, at the start of the simulation we allocate a large array of nodes, large enough so that we never need *more* nodes than there are in the array ($4n_{particles} + 100$ will suffice). The nodes are then put in a large chain so that each free node points to the next free node using the *circAR* member (which is an arbitrary choice). *circAR* is zero for the last free node in the chain and the first free node is pointed to by the *circAR* pointer of the binary tree root node. The amount of nodes we need is fixed because any particle requires at most 2 nodes (one associated with a collision event and one with a cell boundary crossing event). A few extra nodes are required for the event nodes that do not describe changes in the paths of particles.

Allocation

Obtaining a pointer to an unused node is simple provided the *circAR* pointer of the binary tree root node is not zero. In that case the new node n is $root.circAR$ and we update $root.circAR = n.circAR$. This automatically takes care of the case where $\texttt{circAR}(n) = 0$. If $root.circAR$ is zero there are no free nodes and we must either allocate new memory or abort the simulation with an error. If enough nodes are allocated at the start, however, neither will be necessary.

Deletion

There is no exceptional case when committing a node d back to the free node pool. It is done by setting $d.circAR = root.circAR$ to add the node to the top of the list and updating the root node to reflect this change by setting $root.circAR = d$.

2.8 Nearness Queries

After they have been moved very close ($|\hat{r_1} - \hat{r_2}| = \sigma + \lambda$) it is impossible to reliably tell on which side of the potential discontinuity two particles are. Checking for new collisions is done after the particles trajectories (and energies) have been updated, but at this point the collision detection routine does not know if the two particles should interact or not. A solution is to keep track of all of the pairs of particles that are close to each other. We only have to compute this once before the start of the simulation. During the simulation the collision events are responsible for updating the nearness information.

We used a binary tree composed of pairs of numbers referring to the particles with lexicographical ordering (we used `std::set` instead of implementing it ourselves) to save time. The lowest number is always the first number in the pair so that pairs (a, b) and (b, a) refer to the same thing. If a pair is present in the tree it means the particles in it are close. A better way would be to keep a list for each particle containing pointers to all the particles that are close to each other, because that method runs in constant time (independent on the number of particles) whereas the binary tree method runs in $\mathcal{O}(\log n)$ time. In retrospect we dealt with simulations with less than a thousand particles. A big $n * n$ array of bytes indicating whether or not a pair is close would do the trick faster than the binary tree would, has constant look up time, requires no insertions and/or allocations and uses no more than 1 MB of RAM.

2.9 Initialization

Before the simulation can start it must be populated with particles. We show the two methods we used to initialize the particle positions and their initial velocities. Initial particle placement for monodisperse systems and polydisperse systems at low densities can be generated by putting

the particles in an FCC lattice and running the simulation for a while with hard-core interactions only. Dense polydisperse systems are impossible to generate this way because the particles will overlap before the system is dense enough. This is obvious if one considers all particles to be as large as the largest particle in such a system. After the starting positions and velocities have been set the event calendar is populated by performing a collision and cell-boundary check for each particle.

Initial velocity

During equilibration of the system as well as with the help of the thermostat (see Section 2.10) the distribution of the particle velocities will automatically converge to the Maxwell-Boltzmann velocity distribution. It is therefore not strictly necessary to spend any effort on starting out with the correct distribution. We like to mention this because the thermostat uses the same method (and distribution) to re-set the particle velocities and because hard-sphere systems do not need a thermostat as the kinetic energy, and hence the temperature, does not change in these systems.

Let T be the temperature we want the system to be at and $normal(s)$ a function that returns random numbers with a normal distribution with mean 0 and standard deviation s. For each component of the velocity of the new particles we pick $normal(\sqrt{k_B T/m})$. Note that we pick $k_B = 1$ and $m = 1$. The distribution of $|v|$ will be the Maxwell-Boltzmann velocity distribution with temperature T.

FCC lattice

An FCC lattice is a cubic crystal structure with 4 particles per unit cell. The coordinates of the four particles in a unit box are

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}, \begin{pmatrix} \frac{1}{2} \\ 0 \\ \frac{1}{2} \end{pmatrix} \; and \; \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{pmatrix}. \tag{2.9}$$

If we give the particles a diameter of 1 (radius $\frac{1}{2}$) we need to stretch the box so that the particles can not overlap. The smallest box that does not cause overlap has edges of length $\sqrt{2}\sigma$ and thus a volume of $2\sqrt{2}\sigma^3$ where we use σ as out unit of length. The number density of the densest FCC crystal is $\phi\sigma^3 = \frac{4}{\sqrt{2}^3} = \sqrt{2} \approx 1.41$. The corresponding volume fraction is $\eta = \frac{\pi}{6}\sigma^3\phi \approx 0.74$.

Glassy initialization

A method for increasing the density of any overlap-free system is to gradually increase the size of the particles until the system jams. For this we modify the simulation to use a time-dependent monotonically increasing particle size. Collision detection needs to account for this since the right hand term of Equation 2.2 now also depends on time (quadratically). We pick the time dependent sphere radius to be $r(t) = r(0)(1 + gt)$ where g is the growth speed factor. Incorporating this into equation 2.2 yields $A = \langle\vec{a}, \vec{a}\rangle - r^2 g^2$, $B = \langle\vec{a}, \vec{b}\rangle - rg$ and $C = \langle\vec{b}, \vec{b}\rangle - r$, where $r = r_1 + r_2$. Collision response needs to be modified such that particles are guaranteed to be overlap-free after a collision. In order to do this we add a portion of the growth speed to the particle's velocities after collision in the direction pointing away from the other particle. The growth speed is gr so we add half that plus δ (a small value to ensure the particles will separate) to the velocities of the particles.

In order to make a high density polydisperse system we start with a dilute hard-sphere system with the particles positioned on an FCC lattice at arbitrary temperature. After the system has equilibrated we let the particles grow. It is important to re-check all particles for collision at the moment particle growth is being enabled. Because the system gets denser over time the amount

of collisions per time unit will increase to unacceptable levels quickly, grinding the simulation to a halt and indicating that we have a dense system. This is essentially the method described in [17] that we modified to allow for polydisperse mixtures. We easily got systems with a volume fraction of 0.64 using this method which is sufficient for our purposes.

2.10 Thermostat

During a simulation in the NVE ensemble, i.e. the number of particles N, the volume V and the energy E are kept fixed, the total amount of energy in the system should not change. However, the temperature (and thus the kinetic energy) fluctuates during a run, as can be seen in Figure 2.3. If we want the temperature to remain constant (within the statistical fluctuations), we have to implement a thermostat. The kinetic energies of the particles in the system should follow the Maxwell-Boltzmann distribution.

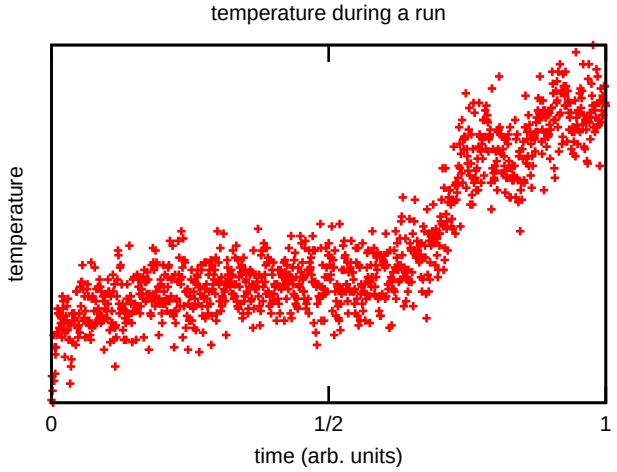


Figure 2.3: *The temperature of a square-well particle system fluctuates heavily during a simulation run in the NVE ensemble.*

We implemented an Andersen thermostat in order to keep the system at the required temperature. At some interval - the sampling interval - we pick a number of random particles and sample new velocities from the Maxwell-Boltzmann velocity distribution. We use the same interval for performing measurements on the system. These particles then need to have their events removed and should be re-checked for collision, analogous to what happens when they are involved in a collisions (the thermostat can be thought of as causing random collisions with an infinitely big reservoir). If we replace the particle velocities with new velocities from the Maxwell-Boltzmann distribution frequently enough the system has no time to diverge from the initial situation and thus the distribution of velocities in the system will converge to the Maxwell-Boltzmann velocity distribution corresponding to a fixed temperature. When done subtly enough, applying the thermostat to as low a number of particles as possible, the average temperature of the system will remain constant. If we then turn off the thermostat once the system has equilibrated, the temperature should not change further.

At a fixed interval we insert an event node into the event calendar that causes the thermostat to be applied. All of the particles that had their velocities changed and all of the other particles involved with them get their events removed and recomputed. For this reason, the thermostat is computationally relatively expensive. When we turn off the thermostat we make the total momentum of the system equal to zero by subtracting from each particle velocity the average particle velocity. We then need to re-check all particles for cell crossings.

2.11 Mean Squared Displacement

In order to measure the mean squared displacement (MSD) of a system, needed to calculate the diffusion coefficient, we need to record the displacement of each particle during the simulation. Dedicated to this is the *xmoved* member of the particles (see Table 2.2). We initialize *xmoved* to zero for each particle and each time we move a particle add the displacement to *xmoved*. The mean squared displacement is then the average of the square of the length of *xmoved* for all particles. We can increase the accuracy of the MSD measurement by picking several times from which we calculate *xmoved*;

$$MSD(t) = \frac{1}{nk}\Sigma_{i,j=0}^{i=n,j=k}|\vec{x}moved_i(t + j\Delta t) - \vec{x}moved_i(j\Delta t)|^2 \tag{2.10}$$

where $\vec{x}moved_i(t)$ is the accumulated *xmoved* for particle i at time t, Δt is the time interval at which we measure the MSD, n is the number of particles and k the number of snapshots of *xmoved* we (are willing to) take. This only works if the slope of the MSD is constant, or rather $\vec{x}moved_i(t - hj) \approx \vec{x}moved_i(h(k - j))$ which is generally only valid for large values of t, in an equilibrated system.

The mean squared displacement is related to the diffusion constant D by the Einstein expression [18];

$$D = \lim_{t\to\infty} \frac{1}{6t} MSD(t) \tag{2.11}$$

2.12 Pressure

It is possible to use Molecular Dynamics to obtain the pressure of a hard-sphere system [19]. To that end the virial needs to be computed during a run which is, apart from a constant factor, the average momentum transfer per unit time due to the collisions. The relation between the compressibility factor Z_{NVT} and pressure P_{NVT} of a hard sphere system is

$$Z_{NVT} = \frac{P_{NVT}V}{Nk_BT} \tag{2.12}$$

with N the number of particles and V the volume of the box. Z and the virial W are related the following way:

$$Z_{NVT} - 1 = -\left(1 - \frac{1}{N}\right)\left\langle \bar{W}(t)/\hat{E}\right\rangle_{NVE} \tag{2.13}$$

where $\hat{E}$ is the average kinetic energy of the particles in the system in its center-of-mass frame of reference. Energy is related to temperature by $E = \frac{3}{2}Nk_BT$ according to the equipartition theorem. The virial can be computed as follows:

$$\bar{W}(t) = -\frac{m}{t}\Sigma_{\gamma=1}^{c(t)}\ \vec{\sigma}_{ij}(t_\gamma) \cdot \Delta\vec{v}_i(t_\gamma) \tag{2.14}$$

where γ refers to the collisions, t_γ is the time of the γ^{th} collision. $c(t)$ is the number of collisions up to time t, $i = i(\gamma)$ and $j = j(\gamma)$ are the pair of colliding particles at time t_γ, $\vec{\sigma}_{ij}$ is the line that connects the centers of the colliding pair and $\Delta\vec{v}_i(t_\gamma)$ is the velocity change of particle i for collision γ. We update W each collision and every time a reading of pressure is needed we divide it by the length of time we have been collecting W to obtain $\bar{W}(t)$.

2.13 Running Time Analysis

To recap, every collision or cell crossing involves the following steps:

- Remove the events involving 1 or 2 particles
- Compute new events for those particles and the particles that they were previously involved with
- Optionally update the nearness flags for a number of particles
- In case of a cell crossing, move a particle from one cell to another

The removing of an event e involves unlinking k events and removing them from a binary tree, where k is the number of particles involved in e. This number has an upper bound that depends on the particle size distribution and cell size (no more than k_{upper} particles can be in one cell). This is then also the upper bound on the amount of events involving any particular particle. So now k has an upper bound that does not depend on n we will treat it as being a constant. Unlinking k events from a circular list takes $\mathcal{O}(k)$ time and deleting them from the binary tree (assuming its depth is limited, as in [16]) takes $\mathcal{O}(k \log n)$ time, totalling $\mathcal{O}(k \log n)$ because the constant term is not significant.

Computing new events is a matter of checking k pairs of particles for collision and inserting up to k new events into the event calendar which takes $\mathcal{O}(k \log n)$ time (note that again the circular list insertions are insignificant). Updating the nearness flags for k particles takes $\mathcal{O}(k \log n)$ time because we used a binary tree internally to do that. Using an array or linked lists to keep track of nearness this can be reduced to $\mathcal{O}(k)$. Moving a particle from one cell to another is an $\mathcal{O}(k)$ operation because the number of particles in a cell is bounded and therefore removing or inserting a particle in whatever structure used to do so has a limited depth. Using linked lists (and having each particle contain a pointer to its node in the list) the cost does not depend on the number of particles in a cell.

Each of these steps takes at most $\mathcal{O}(k \log n)$ where k depends on some geometric terms. This means that for a given simulation, each event costs $\mathcal{O}(\log n)$ time to process. The amount of time spent per unit simulation time is also inversely proportional to the collision rate of the system, which depends on the density and number of particles.

This analysis tells us that the running time of the simulation scales well with the number of particles. For a lower number of particles, like the 800-900 we used, it is still important to make the code as fast as possible in order to reduce the constant factor.

Conclusion The algorithm costs $\mathcal{O}(\tau_c^{-1} n \log n)$ per unit simulation time for a given particle size distribution, where n is the number of particles and τ_c is the average time between collisions. What we did not cover in the analysis is the thermostat and all of the measurement routines.

Chapter 3

Measurements

3.1 Testing the Simulation

In order to test the simulation we compare its output to known literature. If we manage to reproduce the measurements from literature we have reason to be confident that the simulation code produces correct results. The first test checks the pressure of a hard-sphere system, the second test checks the square-well part of the simulation code.

3.1.1 Hard-Sphere Test

The simplest thing we can measure is the pressure of a hard-sphere system. The pressure in the fluid phase can accurately be described in terms of the packing fraction $\eta = \frac{\pi}{6}\sigma^3 \frac{N}{V}$ by the Carnahan-Starling equation of state [20]:

$$\frac{PV}{Nk_BT} = \frac{1 + \eta + \eta^2 - \eta^3}{(1-\eta)^3} \tag{3.1}$$

For higher densities, where the system has crystallized, we compare our results to the empirical equation of state from a paper by R. J. Speedy [21]. In the following equation, $z = \frac{\sigma^3}{\sqrt{2}}\left(\frac{N}{V}\right)$ is the density relative to close packing. Constants a, b and c are in Table 3.1.1 for HCP and FCC crystals. Our simulation gets initialized as FCC so we use the constants from the second row.

$$\frac{PV}{Nk_BT} = \frac{3}{1-z} - a\left(\frac{z-b}{z-c}\right) \tag{3.2}$$

Crystal	a	b	c
HCP	0.5935	0.7080	0.601
FCC	0.5921	0.7072	0.601

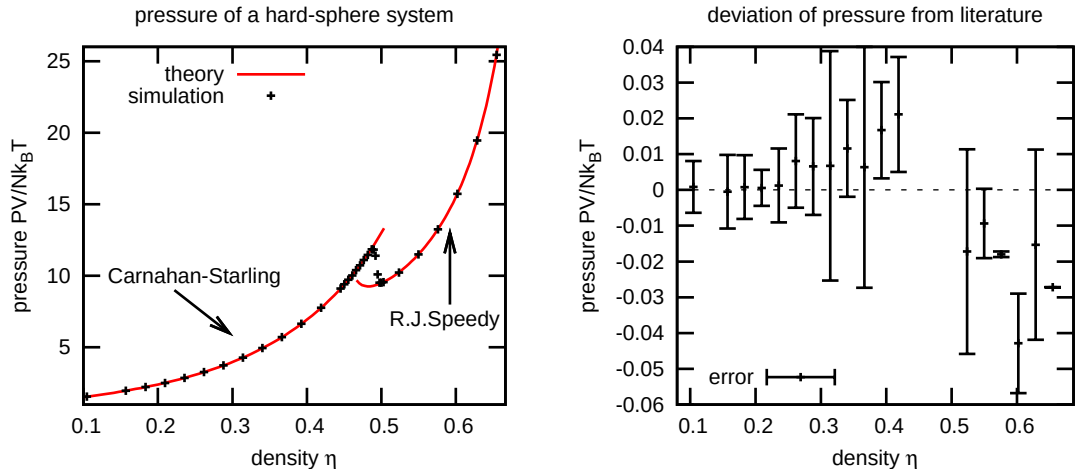
Table 3.1: *Constants used in Equation 3.2*

Inserting the constants for FCC of Table 3.1.1 into Equation 3.2 and substituting $z = \frac{\eta}{\sqrt{2}}\frac{6}{\pi}$ results in the following equation for the pressure, we find for high packing fractions ($\eta > 0.5$):

$$\frac{PV}{Nk_BT} = \frac{0.5921(\eta - 0.438136)(\eta + 2.92579)}{(\eta - 0.74048)(\eta - 0.445029)} \tag{3.3}$$

We performed simulations for several densities and measured the resulting pressure. The system was initialized with the particles in an FCC lattice and left to equilibrate for a while after the pressure measurements started. Figure 3.1(i) shows the pressure we measured together with

plots of Equations 3.1 and 3.3 and they appear to be a good fit. We only find deviations with theory close to the fluid-solid transition at $0.493 < \eta < 0.545$, which is to be expected as the system phase-separates in a fluid and solid phase.



(i) (ii)

Figure 3.1: *Comparison of theory and measurements on the pressure of a hard-sphere system. (i): equations of state from Equations 3.1 and 3.3 with measurements. (ii): difference between equations of state and measurement.*

Figure 3.1(ii) shows the difference between Equation 3.1 and 3.3 and the measured pressure of our simulation. There are some significant errors that may be caused by system size effects because of the low number of particles (864) used or the simulation time, which may not be long enough. In any case the error is small compared to the pressure itself. Note that the data points close to the fluid-solid transition were omitted in Figure 3.1(ii).

3.1.2 Square-Well Test

The second test we perform is meant to test the correct behavior of the square-well part of the simulation. We compare the densities of two coexisting gas and liquid phases to values obtained from literature [15].

In Ref [15] simulations were performed for monodisperse systems using Gibbs ensemble Monte Carlo simulations yielding far greater accuracy than we can manage with our MD simulation. The temperature-density gas-liquid coexistence curve for a $\lambda = 0.25$ square well fluid that was obtained with this method is shown in Figure 3.2 along with the results of [15].

For our initial configuration we first run our simulation with only hard-sphere interactions before enabling the square-well interactions so that the particles are not in an FCC lattice anymore. After enabling square-well interactions an interface will form between the two phases. Because an interface costs energy and complicates measuring the densities of the gas and liquid phases we stretch the simulation box in one direction so that it becomes elongated. After sufficiently long simulation time for forming the interface the thermostat is turned off and the simulation continues to collect a sufficient number of snapshots of the system for determining the two coexisting densities (see Section 3.3 for details).

In addition to the gas-liquid coexistence curve from [15] Figure 3.2 shows our measurement of two points on the binodal of a system with a well width of $\lambda = 0.25$ and a temperature of $k_BT = 0.7$ ($\Gamma = 1/0.7 \approx 1.43$). The error bars in η show the uncertainty in the density, the error in Γ is caused by the fact that T has also an uncertainty. Our measurement agrees with the values obtained from Ref [15], thereby providing confidence in the simulation code.

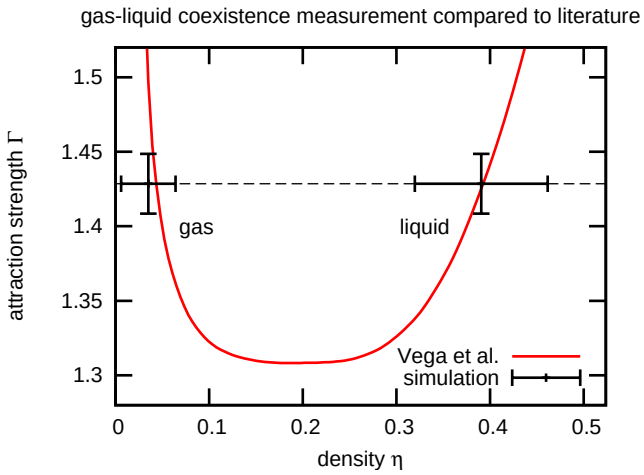


Figure 3.2: *The gas-liquid coexistence curve for a $\lambda = 0.25$ square well fluid as measured by [15] using a Gibbs ensemble Monte Carlo simulation (red) and the two data points obtained from our Molecular Dynamics simulation (black). Our measurements are in agreement with Ref [15].*

3.2 Crystallization

During a simulation the system can crystallize as can be seen in Figure 3.2, where we plot the fraction of particles that is crystalline as a function of time. This is a problem when we want to measure the diffusion of systems at high density in the glass phase. The crystallization can be avoided by using size polydisperse systems.

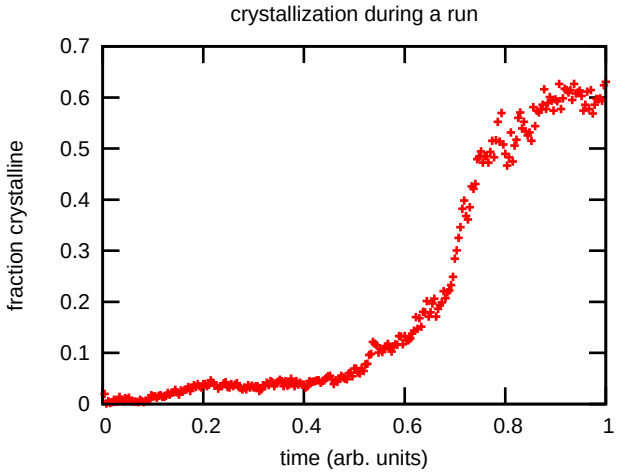


Figure 3.3: *Fraction of particles that are considered crystalline by the Q6 bond orientational order parameter (see Section 3.2) plotted against time for a particular system that crystallizes A significant portion of the system has crystallized during the simulation.*

The definition of our square well potential slightly deviates from the one used traditionally but if the system is monodisperse the two definitions are equivalent. If we keep the system as monodisperse as possible it resembles previous work more closely, making comparisons easier. The cell list method used by the simulation benefits from low polydispersity because cells can only be as small as the largest particle. Smaller cells are favorable because a smaller number particles fits within them, thus speeding up the algorithm. It is therefore in our interest to find a particle size distribution as monodisperse as possible that still prevents nucleation. As explained in Section 1.2.3 the particle size is log-normally distributed with a mean diameter of 1. We then find an acceptable value for the standard deviation by testing several values on a representative system and picking the lowest value that does not cause nucleation.

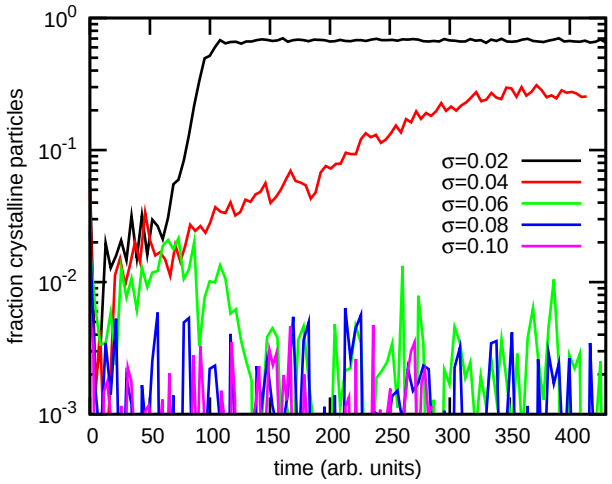


Figure 3.4: *Fraction of crystalline particles in time of systems with several different polydispersities at* $\Gamma = 4.5$ *and* $\eta = 0.51$ *as indicated by the Q6 order parameter.*

We ran simulations of several $\lambda = 0.03$ square well systems ($\Gamma = 4.5$ and $\eta = 0.51$) for polydispersities of $\sigma \in \{0.06, 0.08, 0.10, 0.12\}$ and $\mu = -\frac{1}{2}\sigma^2$ where σ and μ are the standard deviation and the mean of the Gaussian distribution used for the log-normal distribution respectively. Figure 3.4 shows the fraction of crystalline particles as determined from the Q6 bond-order parameter as a function of time for varying σ. At $\sigma = 0.08$ crystallization seemed to have been successfully prevented. To be on the safe side 0.10 was picked which yields a distribution that closely resembles a Gaussian distribution with $\sigma = \frac{1}{10}\mu$.

Q6 Order Parameter

For the algorithm that detects crystal nucleation we rely on the assumption that the neighbours of a particle in the crystal lattice are arranged in a particular way around the particle. The neighbours should share this property. The local bond order parameter $q_6(i)$ of particle i, based on the spherical harmonics Y_{lm}, is a measure of the bond orientation of the surrounding particles. If the q_6 parameters of two neighbouring particles are correlated (meaning their dot product is high enough, > 0.6 in our case) we call them joined by a solid-like bond. Particles that are closer than $1.4\bar{\sigma}$ are neighbours. If a particle has over 7 solid-like bonds we call it crystalline and if two crystalline particles are correlated by at least 0.9 they belong to the same cluster. We do not use the cluster information but it can be used for the visualisation of typical configurations.

Following Ref. [22], we define

$$q_{lm}(i) = \frac{1}{N_b(i)} \sum_{j=1}^{N_b(i)} Y_{lm}(\hat{\mathbf{r}}_{ij}) \tag{3.4}$$

where $\hat{\mathbf{r}}_{ij}$ is the normalized direction vector between particles i and j, particles i are neighbours of j of which there are $N_b(i)$. We pick $l = 6$ [23]. The correlation between two particles i and j is

$$\mathbf{q}_6(i) \cdot \mathbf{q}_6(j) = \sum_{m=-6}^{6} \hat{q}_{6m}(i)\ \hat{q}^*_{6m}(j) \tag{3.5}$$

where $\hat{q} = \frac{q}{|q|}$ and q^* is the complex conjugate of q.

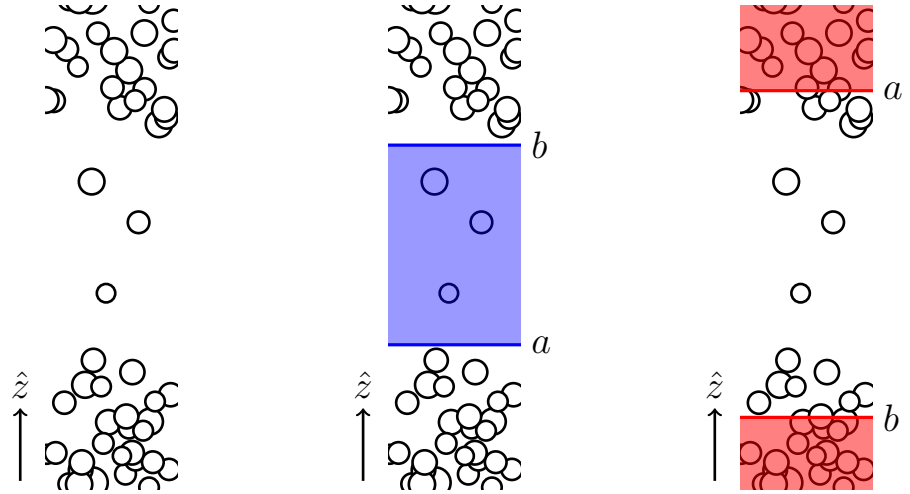
3.3 Spinodal Decomposition

We measured the densities of two metastable phases, gas and liquid, to obtain the temperature-density gas-liquid coexistence curve (where $T = \Gamma^{-1}$). We expected the critical point to lie at $\eta \approx 0.28$ so we measured the densities of the two bulk phases of a phase separated system with a mean density of 0.28 with several values for Γ.

The measurements were done by running the simulation in a box stretched in one direction to reduce the energy barrier that has to be overcome to form an interface. After equilibration we used two different methods, a manual method and one using Voronoi cells, to measure the densities of the coexisting phases. The particles were initialized by running without attractive interactions for a suitably long period of time. At the same time that the interactions were enabled, the thermostat was activated and the system forms an interface between the coexisting phases. After equilibration the thermostat was turned off and the total momentum was set to zero. Snapshots of the system where then collected in order to determine the densities of both phases. Below we discuss the methods we used to determine the coexisting densities.

Manual method

We stretched the box in one direction in order to minimize the surface area of the interfaces due to phase separation. It is therefore extremely likely that the interfaces are perpendicular to the longest dimension of the box. We will refer to this direction as $\hat{\mathbf{z}}$. Figure 3.5(i) shows a typical scenario where this is indeed the case. If the interface is indeed perpendicular to this (or any other) axis we can pick two planes perpendicular to $\hat{\mathbf{z}}$ that separate the two phases. The drawback of this method is that it is requires manual determination of the location of the interface which is prone to error. In order to negate any effect the presence of an interface may cause we ignore the region in which the interface lies. We can easily compute the volume V_b of the region confined by the two clipping planes and the volume V_p that is occupied by the particles whose centers reside in volume V_b. Dividing the two volumina yields the packing fraction $\eta = V_p/V_b$ between the planes. The interface is likely to remain stationary for short time intervals so we can pick the two cutting planes again and use them for multiple successive snapshots in order to increase the accuracy of the measurement.



(i) A sample system

(ii) The two planes envelop the shaded blue region containing a significant portion of the gas phase

(iii) Like Figure 3.5(ii) but for the liquid phase. Note that the liquid phase above a is connected to the liquid below b due to periodic boundary conditions.

Figure 3.5: *A projection of a sample configuration showing gas-liquid coexistence of an arbitrary square-well fluid. The $\hat{z}$-direction is indicated with an arrow. It is apparent that it is hard to give an exact location for the interface because the transition between the two phases is gradual.*

Let N be the number of particles in the region of which we wish to determine the packing fraction. We use a rectangular box of dimensions $L_x \times L_y \times L_z$ with $L_x = L_y$. The area of the box in the xy direction is denoted $A_{xy} = L_x \times L_y$. As we use periodic boundary conditions, we can determine η in the region between the two planes or in the complementary region. Defining z_a and z_b in the z-coordinate of the a and b plane, we can calculate η using

$$\eta = \frac{V_p}{V_b} \quad \text{where} \quad V_b = A_{xy} \begin{cases} z_b - z_a & \text{If } z_b > z_a \\ z_a - z_b + L_z & \text{If } z_b < z_a \end{cases} \quad \text{and} \quad V_p = \frac{4}{3}\pi \sum_i^N r_i^3 \tag{3.6}$$

where r_i denotes the radius of particle i. We wrote a **Mathematica** script to visualise the z-dependent density of a snapshot of the simulation which facilitates defining the locations of the a and b planes. We divide the box into slabs in the z-direction. The total volume of the particles in the slab and the volume of the slab itself are used to calculate the density. The resulting graph is susceptible to fluctuations. The graphs of snapshots at several times can be played back in an animation using **Mathematica**.

Voronoi method

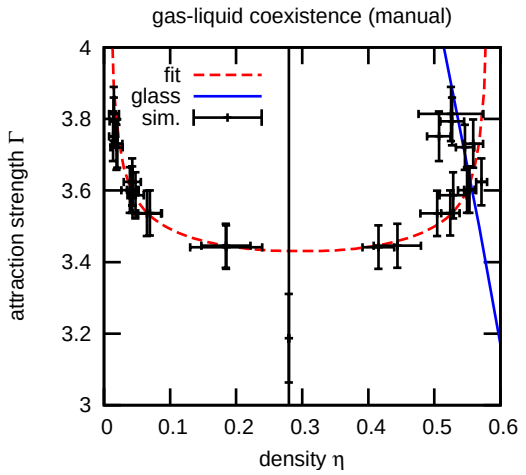
The manual method we discussed above depends strongly on the precise location of the planes and is time-consuming. In an attempt to alleviate these problems we tried an automatic method to determine the density. The method works by identifying the two phases based on the connectivity between the particles and uses the Voronoi cells around the particles belonging to a phase to determine the volume it occupies. For every particle the number of neighbours is determined. Particles are neighbours if their centers are closer than $\bar{\sigma} + \lambda$. We attribute a particle to the gas phase if it has at most one neighbour and to the fluid phase if it has 5 or more neighbours. The particles with 2 to 4 neighbors are ignored. The reason for this is that, even in a homogeneous system, particles with less neighbours tend to have more space around them and hence a larger Voronoi cell.

Measurement

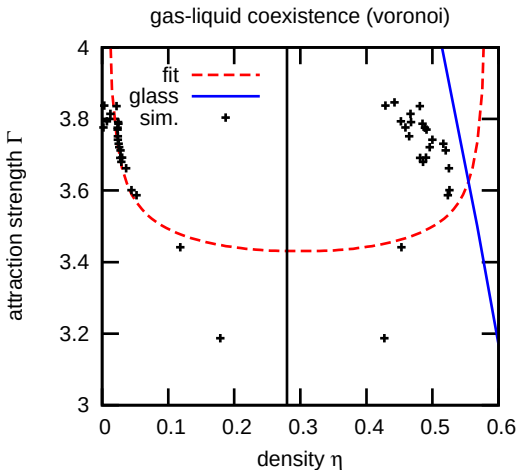
Figures 3.6(i) and (ii) show our results for the gas-liquid coexistence curve for a $\lambda = 0.03$ square well fluid using the two different methods to measure the coexisting densities of both phases. We have fit the gas-liquid coexistence curve in Figure 3.6(i) by manually picking parameters $c_1, \cdots, c_3$ of the function $\Gamma = c_3/(\eta + c_2) + c_3$ as a guide to the eye. The Voronoi method seems to confirm our doubts as it still detects spinodal decomposition at attraction strengths where it should not occur. For higher attraction strengths the low-density binodal agrees well with the manual measurements (because both lie on the fit).

As shown in Figure 3.6 we found gas-liquid coexistence of a short-range $\lambda = 0.03$ square-well fluid with a critical point in the range $3.18 < \Gamma < 3.4$ and $0.18 < \eta < 0.42$. Mode Coupling Theory calculations [13] predict a glass transition line that intersects the gas-liquid coexistence curve which is confirmed by computer simulations [12]. We indeed find that the density of the liquid phase, as determined by our simulations, is much lower than the fit of the liquid binodal at sufficiently high attractions, which may be explained by the intersection of the glass line with the gas-liquid coexistence curve, shown as the blue line in Figure 3.6.

Gibbs ensemble Monte Carlo simulations from [15] show that the binodal gets wider and more square-like when the attractive range λ of a square-well fluid is decreased. In addition the critical temperature decreases with decreasing λ, i.e. the critical point moves to higher Γ. We plot the results in Figure 3.7. For comparison, we plot the binodal obtained from our simulations and we find that the shape of the binodal and the location of the critical point is consistent with the trend that can be deduced from previous work [15].



(i) Gas-liquid coexistence curve for a $\lambda = 0.03$ polydisperse square-well mixture computed using the manual method (see text).



(ii) Same as (i) but computed using the Voronoi method. The high-density branch is not in agreement with the manual measurement.

Figure 3.6: *Gas-liquid coexistence curve obtained by both methods measured with our MD simulation.*

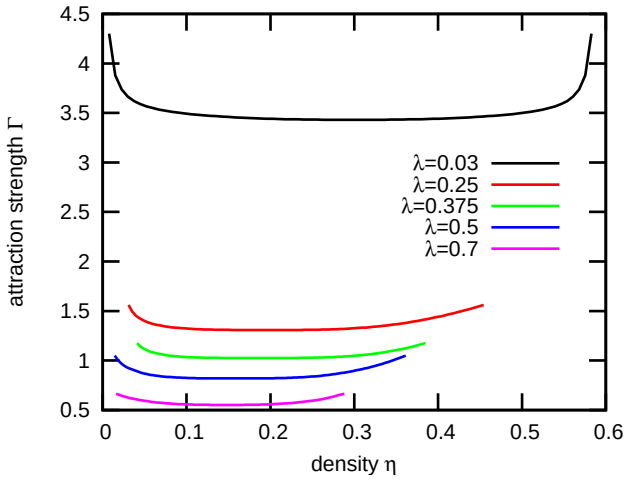


Figure 3.7: *Measurements of the gas-liquid coexistence curve for square well fluids with several well widths measured by [15] (red,green,blue,purple) compared with our measurement for a $\lambda = 0.03$ SW fluid (black).*

3.4 Isodiffusivity lines

Additionally, we measure the diffusion coefficient for several $\lambda = 0.03$ short-range square-well systems and we construct isodiffusivity lines. We calculate the mean square displacement of polydisperse square-well systems for varying η and Γ. Crystallization was prevented by using a polydisperse system (see Section 3.2) and the Q6 bond order parameter was measured to check this. The method used to measure the diffusion coefficient (from the mean square displacement) of a particle system is discussed in Section 2.11.

Method

Several simulations were run with parameters $\Gamma \in [2.5 \ldots 5.0]$, $\eta \in [0.5 \ldots 0.6]$, $\lambda = 0.03$, $T = \epsilon/\Gamma = \Gamma^{-1}$ and a polydispersity as explained in Section 3.2. For each simulation the long-time mean square displacement was measured. We first equilibrate the system before we determine the mean-square displacement, from which we determine the diffusion coefficient for each system

(see Section 2.11). Independent of a thermostat the temperature of a system can change slightly during equilibration. Following [12], we correct the diffusion coefficient D and attraction strength Γ by

$$\Gamma^* = \Gamma/\sqrt{T} \quad \text{and} \quad D = \frac{1}{6}\frac{MSD}{\sqrt{T}} \tag{3.7}$$

In Figure 3.8 we plot the corrected values for Γ and η as denoted by the red crosses. As a result of the correction, the values are not on a rectangular grid anymore in the $\eta - \Gamma^*$-plane. By interpolating the diffusion between the measurements with identical values for η, on the vertical lines of Figure 3.8, we generate a rectangular grid of points with interpolated values for the diffusion coefficient. Figure 3.9(i) shows the interpolated functions obtained this way. Each vertical line in Figure 3.8 corresponds with one of these functions. The points of this grid are shown as blue dots in Figure 3.8. Using this grid we interpolate the diffusion coefficient between measurements that have the same density, meaning we interpolate diffusion coefficients for parameters on the horizontal lines of Figure 3.8 that intersect the blue dots.

As Figure 3.9(i) shows all of the measurements turned out to have a corrected attraction strength Γ^* of between 2.5 and 3.5 and a density between 0.5 and 0.6. This is thus the range in which we can reliably reconstruct the isodiffusivity lines. We construct the isodiffusivity line for $D = d$ using the horizontal interpolation functions D_γ for several values of $\gamma \in [2.5, 3.5]$ that represent an axis with $\Gamma = \gamma$ by solving

$$D_\gamma(\eta_\gamma) - d = 0 \tag{3.8}$$

where η_γ is the density at which the isodiffusivity line intersects the horizontal $\Gamma = \gamma$ line. By connecting these (η, Γ) pairs we obtain the isodiffusivity line for a diffusion coefficient of d. Functions D_γ are obtained by using the vertical interpolation functions to obtain samples to be used to built a horizontal interpolation. This means we are using interpolations of interpolations.

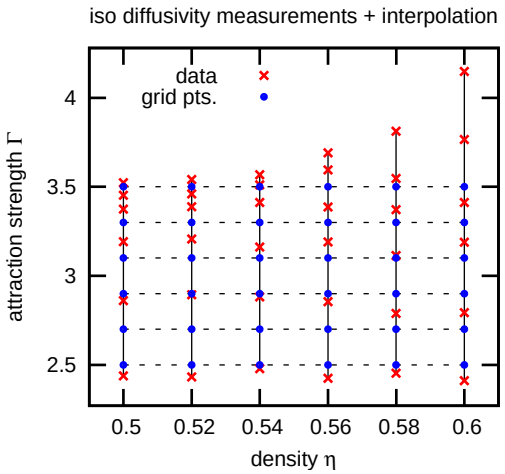
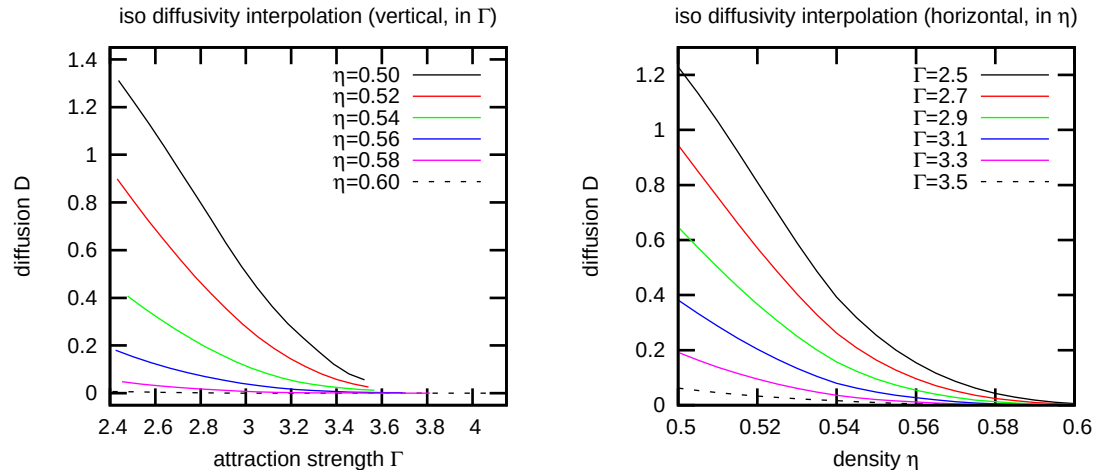


Figure 3.8: *Corrected parameters (η,Γ^*) of the system are represented as red crosses. The blue dots are in an equispaced grid and their diffusion values have been interpolated using the red crosses. The lines represent points at which we interpolate the diffusion values (see text).*

Measurement

The result of measuring the diffusion coefficient for several systems, interpolating them and plotting the isodiffusivity lines is shown in Figure 3.10.

Figure 3.10 shows that the isodiffusivity lines obtained from our measurements are parallel to the ones from [12]. Our diffusion coefficient is about 2.0 times too high for the results to agree;



(i) (ii)

Figure 3.9: *Interpolated values for the diffusion coefficient on the (i) vertical and (ii) horizontal lines of Figure 3.8. (i) shows the diffusion coefficient D in terms of Γ for several values of η whereas (ii) shows D in terms of η for several values of Γ.*

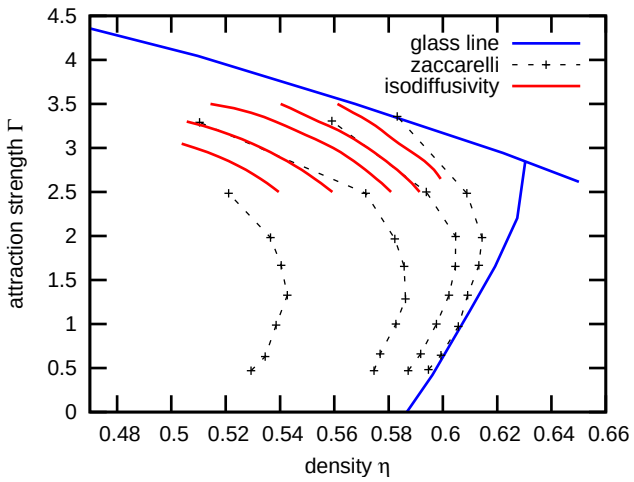


Figure 3.10: *Red: isodiffusivity lines obtained by interpolation for $D = \{5 \cdot 10^{-3}, 2 \cdot 10^{-2}, 5 \cdot 10^{-4}, 2 \cdot 10^{-4}, 5 \cdot 10^{-5}\}$. from left to right. Black, dashed: isodiffusivity lines from [12] for $D = \{5 \cdot 10^{-3}, 5 \cdot 10^{-4}, 5 \cdot 10^{-5}, 5 \cdot 10^{-6}\}$. Blue: glass line from theoretical predictions by [13]*

we are unable to explain this difference. Due to equilibration issues we were unable to obtain measurements for higher values of Γ.

Chapter 4

Discussion

In this study we have implemented the event-driven molecular dynamics algorithm as described in [16]. We have extended the algorithm to simulate particles interacting with a square-well potential (two discontinuities) that could be expanded to a potential with a larger number of discrete discontinuities straightforwardly. We have tried to improve the algorithm by implementing self-balancing binary trees but did not push that further due to time constraints.

We have measured the gas-liquid coexistence curve for a $\lambda = 0.03$ SW fluid using Molecular Dynamics, while previous results were obtained by using the Gibbs ensemble Monte Carlo method [15]. This complicated the measurement of the densities of the two bulk phases. Of the two methods we tried the method using Voronoi cells ended up performing poorly (at lower attraction strengths) so we decided to use the manual method for our measurements. If another algorithm is used in the Voronoi method to identify to which phase particles should be attributed the method may perform acceptably.

As stated in [15], for $\lambda = 2$ the gas-liquid coexistence curve shows classical critical behavior whereas for shorter well widths the behavior is closer to cubic than quadratic. Due to the very sharp onset of phase separation near the critical temperature we see in our measurements (where $T = \epsilon\ (\Gamma k_B)^{-1}$) we conclude that the critical behavior for $\lambda = 0.03$ may behave differently even than quadratic or cubic. Based on the measured curve we estimate the critical temperature and critical density to be in the range $T_c = 0.30 \pm 0.013$ and $\eta_c = 0.3 \pm 0.13$. This agrees with the statement that the system is unstable with respect to gas-liquid phase separation for temperatures below $T = 0.32$ mentioned by Foffi *et al* in [14]. A more careful study is needed for a more precise figure, for example using the Monte Carlo technique employed in [15]. We interpret our inability to obtain samples for attraction strengths higher than the point at which the glass transition line intersects the metastable gas-liquid coexistence curve as evidence for the presence of the glass transition at that location.

We used an algorithm that uses the Q6 bond orientational order parameter to detect crystallization to find a suitable distribution of particle sizes that does not crystallize at high densities. The simulations we performed using this size distribution allowed us to measure the diffusion coefficient for square-well systems near the attractive glass region. From this data we constructed isodiffusivity lines. The diffusion constant appears to show exponential decay close to the glass transition, in agreement with simulations from Zacarelli *et al* [12] and the theoretical Mode Coupling theory calculations from Sperl *et al* [13].

4.1 Acknowledgements

This thesis could not have been realised without the guidance, inspiration and help from Michiel Hermes and Marjolein Dijkstra, my supervisors. I was very happy with Michiel's help when debugging my simulation code with particularly nasty bugs on several occasions. I thank them both for proof-reading, correcting and making suggestions for my thesis.

Joost de Graaf's help with my presentation was invaluable. I am still amazed by his willingness to help. Thanks to all of the students in the student room for the conversations. It would have been pretty lonely if I were alone in that room all the time.

Bibliography

[1] P. Bolhuis and D. Frenkel. Prediction of an expanded-to-condensed transition in colloidal crystals. *Phys. Rev. Lett.*, 72(14):2211–2214, Apr 1994.

[2] P. Bolhuis, M. Hagen, and D. Frenkel. Isostructural solid-solid transition in crystalline systems with short-ranged interaction. *Phys. Rev. E*, 50(6):4880–4890, Dec 1994.

[3] A. P. Gast, C. K. Hall, and W. B. Russel. Polymer-induced phase separations in nonaqueous colloidal suspensions. *Journal of Colloid and Interface Science*, 96(1):251–267, 1983.

[4] H. N. W. lekkerkerker, W.C.-K. Poon, A.Stroobants P. N. Pusey, and P. B. Warren. Phase behaviour of colloid + polymer mixtures. *Europhys. Lett*, 20:559–564.

[5] M. H. J. Hagen and D. Frenkel. Determination of phase diagrams for the hard-core attractive yukawa system. *J. Chem. Phys.*, 101(5):4093–4097, 1994.

[6] E. J. Meijer and F. ElAzhar. Novel procedure to determine coexistence lines by computer simulation. application to hard-core yukawa model for charge-stabilized colloids. *J. Chem. Phys.*, 106(11):4678–4683, 1997.

[7] S. M. Ilett, A. Orrock, W. C. K. Poon, and P. N. Pusey. Phase behavior of a model colloid-polymer mixture. *Phys. Rev. E*, 51(2):1344–1352, Feb 1995.

[8] M. G. Noro and D. Frenkel. Extended corresponding-states behavior for particles with variable range attractions. *J. Chem. Phys.*, 113(8):2941–2944, 2000.

[9] K. N. Pham, A. M. Puertas, J. Bergenholtz S. U. Egelhaaf, A. Moussaid, P. N. Pusey, A. B. Schofield, M. E. Cates, M. Fuchs, and W. C. K. Poon. Multiple glassy states in a simple model system. *Science*, 296(5565):104–106, 2002.

[10] K. Dawson, G. Foffi, M. Fuchs, W. Götze, F. Sciortino, M. Sperl, P. Tartaglia, Th. Voigtmann, and E. Zaccarelli. Higher-order glass-transition singularities in colloidal systems with attractive interactions. *Phys. Rev. E*, 63(1):011401, Dec 2000.

[11] K. A. Dawson. The glass paradigm for colloidal glasses, gels, and other arrested states driven by attractive interactions. *Current Opinion in Colloid & Interface Science*, 7(3-4):218–227, 2002.

[12] E. Zaccarelli, G. Foffi, K. A. Dawson, S. V. Buldyrev, F. Sciortino, and P. Tartaglia. Confirmation of anomalous dynamical arrest in attractive colloids: A molecular dynamics study. *Phys. Rev. E*, 66(4):041402, Oct 2002.

[13] M. Sperl. Dynamics in colloidal liquids near a crossing of glass- and gel-transition lines. *Phys. Rev. E*, 69(1):011401, Jan 2004.

[14] G. Foffi, K. A. Dawson, S. V. Buldyrev, F. Sciortino, E. Zaccarelli, and P. Tartaglia. Evidence for an unusual dynamical-arrest scenario in short-ranged colloidal systems. *Phys. Rev. E*, 65(5):050802, May 2002.

[15] L. Vega, E. de Miguel, L. F. Rull, G. Jackson, and I. A. McLure. Phase equilibria and critical behavior of square-well fluids of variable width by gibbs ensemble monte carlo simulation. *J. Chem. Phys*, 96(3):2296–2305, 1992.

[16] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, second edition, 2005.

[17] B. D. Lubachevsky and F. H. Stillinger. Geometric properties of random disk packings. *J. Stat. Phys.*, 60(5):561–583, 1990.

[18] D. A. Quarrie. *Statistical Mechanics*. Harper and Row, 1976.

[19] W. W. Wood. J. J. Erpenbeck. Molecular dynamics calculaions of the hard-sphere equation of state. *J. Stat. Phys.*, 35(3-4):321–340, 1984.

[20] C. Kittel and H. Kroemer. *Thermal Physics*. Second edition, 1980.

[21] R. J. Speedy. Pressure and entropy of hard-sphere crystals. *J. Phys.: Cond. Mat.*, 10(20):4387–4391, 1998.

[22] S. Punnathanam and P. A. Monson. Crystal nucleation in binary hard sphere mixtures: A monte carlo simulation study. *J. Chem. Phys.*, 125(2):024508, 2006.

[23] S. Auer and D. Frenkel. Numerical prediction of absolute crystallization rates in hard-sphere colloids. *J. Chem. Phys.*, 120(6):3015–3029, 2004.