Agent based simulations and optimization with uniform grid applied to colloidal jamming transitions

Jimmy-Xuan Shen[1]
Department of Physics, University of California, Santa Barbara, California 93106-9530, USA

I. INTRODUCTION

A. Agent bases simulations

Agent based simulation (ABS) is a class of computational techniques used to simulate the behavior of autonomous individual agents with the intent to study the behavior of the collective. The technique is used to study a wide range of problems ranging from soft-matter physics and biology to social simulations and economics. A key feature of these simulations is that the interactions between the individual agents give rise to non-trivial behavior of the entire system. A common bottle-neck of ABS is iterating over all of the agents in a system and accessing the data stored for each agent to detect for interacting, for a system of N agents will there will be $\mathcal{O}(N^2)$ such interactions checks. However, in most cases it is only possible for a given agent to interact with a relatively small number of other agents at any given time. However, the iteration over all particles to check for interactions cannot be skipped using standard data storage techniques since we have no prior knowledge of the agent before accessing the that information in the computer's memory. To avoid the unnecessary memory access, we will have to implement the memory storage of the agents in an organized way so that only a small set of possible interaction candidate will be accessed each time. In this report, a particular example of this principle applied to colloidal jamming transitions will be discussed to provide a solid example. The example we are using is quite simple in-terms of offering physical understanding of realistic colloids. However, the concepts can be used for any system where many individual elements interact on short distances.

B. Jamming transitions

The particular system we will be studying is a simple two dimensional colloid consisting entirely of hard-disks. For simplicity the particles in the system will be treated as hard-disks with elastic collision as the only interaction between them, we will also ignore all of the properties of the suspending liquid such as viscosity and temperature.

Jamming transitions[1,2] are a physical process found in colloids and other complex materials like glasses and foams, where the complex fluids become rigid with increasing density. While one might expect the transition from colloidal liquid to a jammed rigid state to occur at a well-defined density, recent studies have shown that this transition is not well defined but rather exists over a continuous range.[3]

For our simple system, we expect the particles to be able to move around freely, while the close packed system will not be able to move at all. The system will evolve in discrete time dt, and the interaction finding problem in this case is the detection of collisions between all of the particles in the system. And speed-up of simulations will come from knowing some information about the position of the particle before the stored data of that given particle is actually accessed.

The only physically relevant quantity is the filling fraction(ϕ) of total space occupied by the particles in the system — a unit-less parameter. Thus we can use arbitrary units to describe the radial size (r) and velocity (v) of the particles in the system. With the requirement that $v \times dt$ in arbitrary units is small enough that possible collision will not be skipped during any given time-step. For simplicity, the mass of all the particles will be set to 1 in arbitrary units.

II. DESCRIPTION OF SIMULATION

A. Initialization

The particles are initialized in fixed positions in a bounded square box. When they are also initialized with random velocities sampled from uniform disk, with maximum velocity of 1 (in reduced units). Because the number of particles in the system does not change during the course of a simulation, we can designate a continuous block of memory to store the physical properties of the particles (position and velocity) using a standard c++ array. The speed-up of our simulation is based on organizing the data of the particles into groups that are dependent on the position of the particles. The groups will be determined by the position of the centers of the particles on a uniform $k \times k$ grid, we will call each square created by the grid a neighborhood. Since the particles can move freely between the neighborhoods, we expect to constantly move elements between the groups. The most efficient data structure for frequent data relocations is a linked list,[4] where the each piece of data stored in the list form a node and all of the nodes are connected by memory references (or pointers) to other nodes. The pointers are variables in c++, like an integers or floats, which take on the value of physical memory addresses in the computer. The addition/deletion of a node simply requires changing the pointers in the list to accommodate for the new/removed node, Figure 1.

Because the N particles in the system will always use the same locations in memory to store their physical attributes, we only need to organize their memory references into linked lists based on which neighborhood the particle is in. Doing this allow us to utilize the compliers optimized array data access for tasks like writing the particle data to file, while use the linked list data structures to optimized collision detection.

B. Update particle positions

To update the positions of the particles, we will move the particles one by one at each time step and check for collisions with other particles. To better simulate the geometry of each collision,

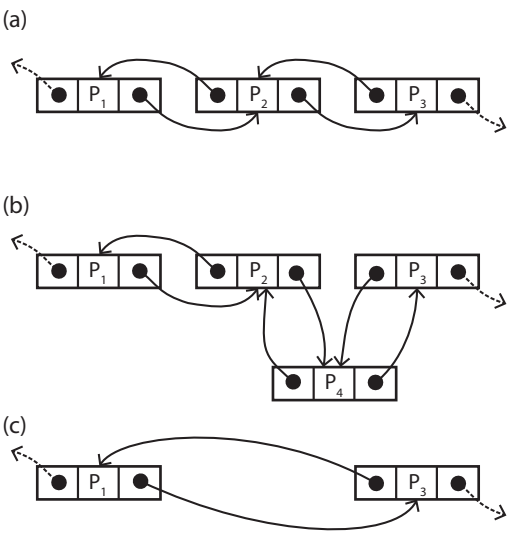

Figure 1. (a) Basic structure of the linked list, the arrow indicate pointer references to where the another nodes is stored in memory. (b) Reorganization of the pointers to insert an additional node to the list. (c) Reorganization of the pointers to delete a node to the list.

we will advance each particle over five sub time-steps, moving by $v \times \frac{dt}{5}$ a total of five times. Normally, we will have to check for collision of the advancing particle with all of the other $N-1$ particles in the system, which requires $N-1$ distance calculations. However, we can take advantage of our storage structure and only access a subset of the remaining particles in the system for possible collision candidates. Since the advancing particle cannot travel too far from the initial position in dt, we will restrict ourselves to a 3×3 block of neighborhoods (Figure 2) to look for colliding particles. Thus instead of checking $N-1$ particles of collision, we will be checking approximately $\frac{9}{k^2}(N-1)$, assuming a uniform distribution. If a collision has been found, meaning that advancing the particle by $v \times dt/5$ will result in overlap with another particle, the advancing particle is halted (skipping the remainder of the sub-steps) and the velocities of both the advancing particle and the colliding particles are modified according to elastic collision of hard disks.

At the end of each time-step, we check to see if the particle has moved to a new neighborhood. If the particle is in a new neighborhood, we will update the linked lists for the old and new neigh-

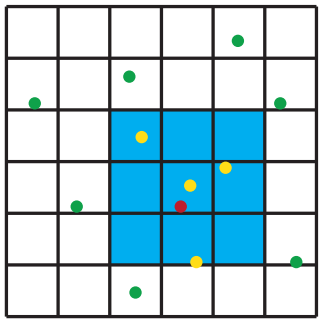

Figure 2. For a given particle (red), only collisions with the yellow particles will be consider possible in our simulation, the data of the green particles are not accessed during collision detection.

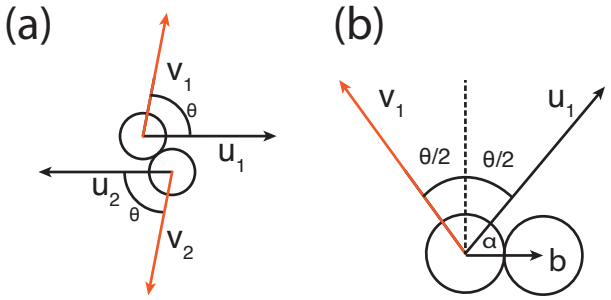

Figure 3. (a) Collision in the CM frame, the initial velocities u_1 and u_2 and the final velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ all have the same magnitude. (b) $\mathbf{u}_1$ and $\mathbf{v}_1$ must form an isosceles triangle with angle θ between them, and α is the angle beween $\mathbf{u}_1$ and $\mathbf{b}$.

borhoods accordingly. Note that doing this in a linked list only requires us to modify the pointers of at most two other elements in the list and will not disturb the rest of the data. Doing the same operation with an array will sometime require the entire array to be rewritten.

C. Treatment of collision

To treat the elastic collisions, we first shift the velocities of the incident particles to center of mass (CM) frame so that the incident particle will have equal but opposite momentums, same with the particles after collision, Figure 3 (a). And since the kinetic energy must be conserved in the center of mass frame, the magnitudes of the initial and final velocities are all equal. The only remaining unknown is the relative angle θ between the incident and outgoing particles. To determine this angle, we must use the fact that the impulse of the collision is entirely along the impact vector $\mathbf{b}$—displacement vector between the two colliding particles. Let alpha be the angle between the incident particle velocity $\mathbf{u}_1$ and the impact vector $\mathbf{b}$. Since the resulting velocity $\mathbf{v}_1$ must have the same magnitude as $\mathbf{u}_1$, and the component perpendicular to the impact vector cannot change (assuming frictionless collisions), we must have that $\alpha + \theta/2 = \pi/2$, as shown in Figure 3 (b). Once we have θ, we can solve for $\mathbf{v}_1$ and $\mathbf{v}_2$ in the CM frame and translate back to the rest frame.

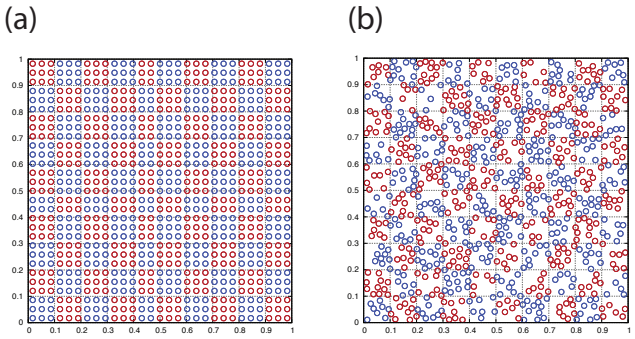

Figure 4. (a) Initial configuration of the particles, the uniform grid is shown and the particles in adjecent neighborhoods are colored differently. (b) The configuration of the particles after 100 time-steps.

The resulting snapshots of our simulations with $k = 10$ and $N = 841$ is shown in Figure 4.

III. DISCUSSION

A. Grid size dependence

To test the effectiveness of the uniform grid method, we can tested the speed of the simulation using different grid sizes. For each grid size, we run a series of simulations with the same particle size and velocity distribution with different number of particles. The time taken for each simulation run to complete 100 time-steps is shown in Figure 5. For each grid size, the systems with more particles take more time to simulate. How-

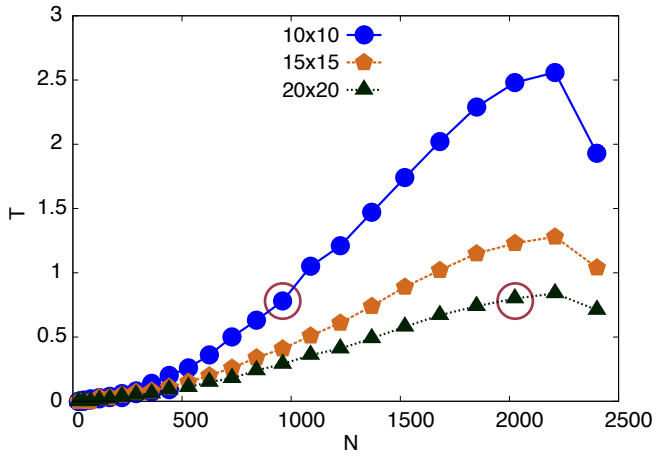

Figure 5. Plot of time taken to complete 100 time-steps vs. the number of particles in the simulated system, using a 10×10, 15×15 and 20×20 uniform grids. The two red circles have centers of the same y-value, so while the system size more than doubled, there was little additional computation time.

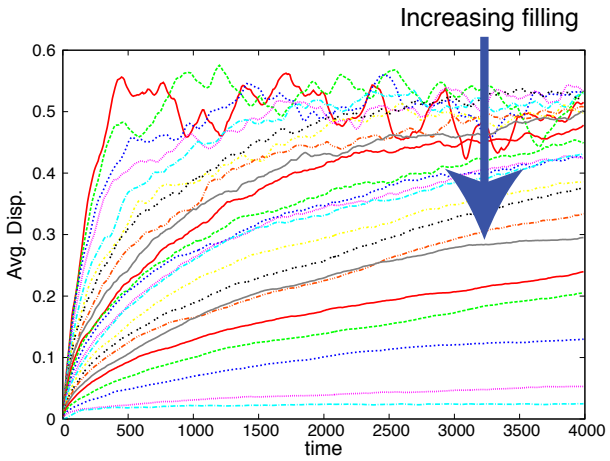

Figure 6. Time-series of the average displacement (as a fraction of the simulation system size) of particles in the system. The average displacement curves all go down as a filling fraction increases.

ever, the relationship is not quite quadratic since particles in the dense system are not likely to complete all five sub-steps before encountering another particle and halting. A key thing to note is that the simulation of 1000 particles using 10×10 grid takes the same amount of time as a 2000 particle simulation on a 20×20 grid. Note that the total number of collision checks is $\mathcal{O}(9N^2/k^2)$ so scaling k and N together will not increase computational time. This is primary benefit of the uniform grid method, as more particles can be included in the simulation with no additional computational cost (only need additional storage).

B. Where is the Jamming transition?

To quantify the mobility of the particles in our simulations, we store the initial position of all the particles and calculate the average distance of the particles to that initial position. As we increase the density of particles in our simulations the particles are less likely to move freely without collision. And for completely close packed, the particles will not move at all. When we observe the average displacement of the particles in the system as a function of time (Figure 6), we see that the simulation exhibits some degree of mobility at almost all particles densities. This agrees with previous studies[3] which found that the Jamming transition occurs over a continuous range and we cannot define an exact transition point. Although they modify the initial conditions by starting at different volumes and compressing the systems to a jamming state. The fact that our simulation shows some degree of sustained mobility for almost all particle densities is due to us treating all collisions as purely elastic. Our result cannot be directly compared with previously studies which used a potential for inter-particle interactions and not just purely hard-sphere collisions. In Figure 7 we plot the average displacement after 100,000 time-steps against the filling fractions of the systems, for both square and triangular lattice of initial particle positions. The triangular lattices exhibits lower mobility at the high filling because each particle in the triangular initial arrangement has *six* nearest neighbors making it less likely to have enough of them moving out of the way for the given particles to move at each time-step. At lower filling fractions both arrangements gives average displacement of 0.5.

C. Additional applications

Since this study is focused on the implementation of the uniform grid method, it is worth-

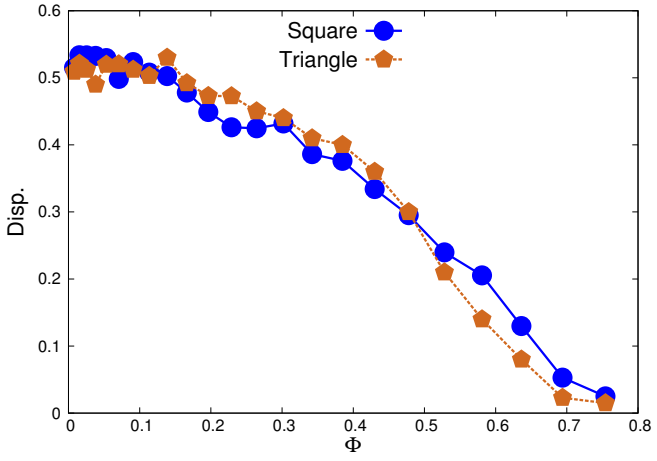

Figure 7. Plot of the average displacement of particles in the system against the filling fraction for both square and triangular initial arrangements.

while to discuss other potential application of the method. One important aspect of linked list that we have not fully utilized is the the fact that addition and deletion of nodes in the list does not require any kind of array resizing (which require many memory operations). This is important in simulating systems were the total number of agents is not conserved, for example reaction of molecules adsorbing on the surface of a catalyst. For these kinds of system, we will not be able to store all of the agents in a fixed array as we have done for the jamming transition. We will store the full data of each agent in a linked list instead of storing just just the pointer to its memory address. This allows for not only frequent relocation data between the linked lists but also for permanent addition and deletion of data the whole system, which frees up memory so that new particle can be created later (avoiding memory leaking in long simulations).

IV. CONCLUSION

Using the simple example of hard sphere collisions, we have demonstrated the power and effectiveness of the uniform grid method, which is able to scale up the size of the simulation without using additional CPU time. Using our simulations we found that the jamming transitions is difficult to identify due to the fact that jamming depends on the initial configuration of the system. This agrees in spirit with previous theoretical studies. A more practical definition of where the jamming occurs could be where the average displacement of the particles is below 10% the size of the systems after a sufficiently long time.

REFERENCES

[1]C. O'Hern, L. Silbert, A. Liu, and S. Nagel, Phys. Rev. E **68**, 011306 (2003).
[2]G. Biroli, Nat Phys **3**, 222 (2007).
[3]M. Ozawa, T. Kuroiwa, A. Ikeda, and K. Miyazaki, Phys. Rev. Lett. **109**, 205701 (2012).
[4]J. L. Antonakos and K. C. Mansfield, *Practical Data Structures Using C/C++*, 1st ed. (Prentice Hall, Upper Saddle River, N.J, 1998).