CSCI 7000: Software Engineering for Scientists

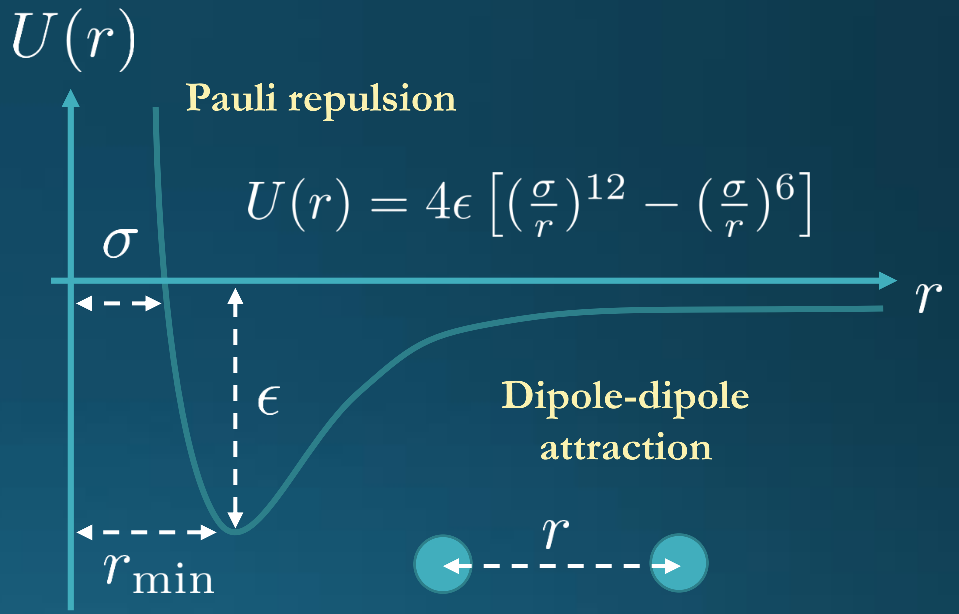
Monte Carlo Simulation of Lennard-Jones Fluids in Canonical Ensembles

Final Project Presentation

Presenter: Wei-Tse Hsu and Chi-Ju Wu

Lecturer: Dr. Ryan Layer

Date: 12.05.2019



Be Boulder.

University of Colorado Boulder

Department of Chemical and Biological Engineering, University of Colorado Boulder

Outline

Scientific backgrounds

- System of interest
- Lennard-Jones potential
- Metropolis-Hastings algorithm
- Monte Carlo simulation

Applications of software engineering

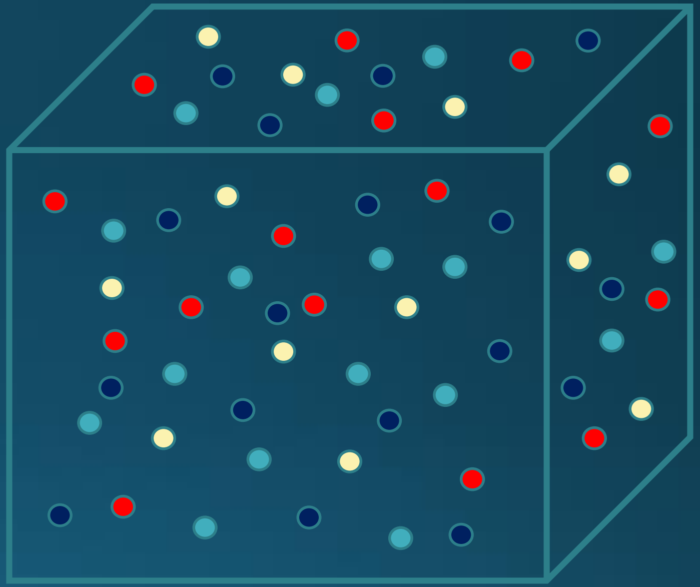
- Requirements fulfilled
- Enhancement of code efficiency
- Object-oriented programming

Results

- Total potential of the system
- Molecular visualization
- Animated trajectory

Goal: Develop a software package able to perform a Monte Carlo simulation of Lennard-Jones fluids

- 500 Lennard-Jones particles inside a box
- Simulation will be performed in a canonical ensemble (constant N, V, and T)
- 10,000,000 MC steps will be carried out.
- Task: Calculate the potential energy of the system and compare it with the NIST benchmark.



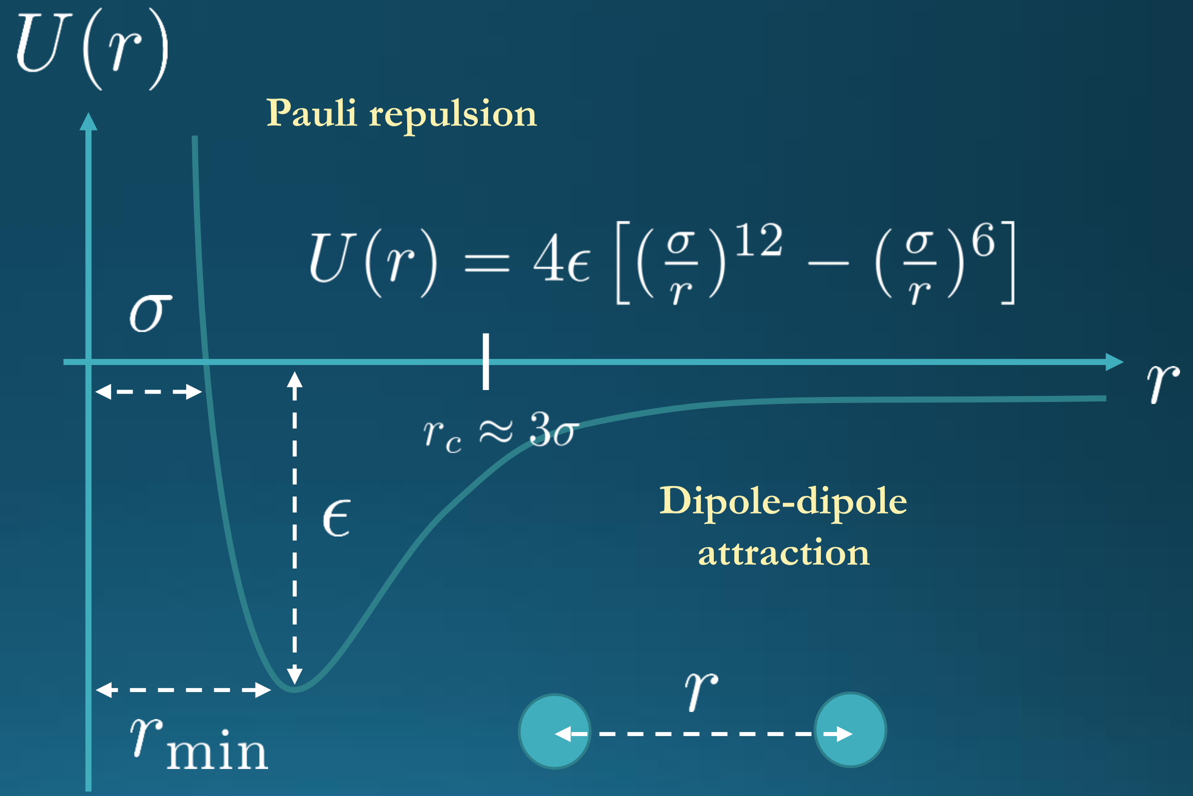
Be Boulder.

University of Colorado Boulder

Lennard-Jones potential is a simple model which describes the van der Waals interactions between a pair of neutral particles

- r^{-6} term: attractive term
- r^{-12} term: repulsive term
- Tail correction to correct the effect of energy truncation

$$U_{\text{tail}} = \frac{8\pi N^2}{3V}\epsilon\sigma^3\left[\frac{1}{3}\left(\frac{\sigma}{r_c}\right)^9 - \left(\frac{\sigma}{r_c}\right)^3\right]$$



$$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^6\right]$$

Metropolis-Hastings algorithm drives the energy of a system to its local minima

- Boltzmann distribution $P_k \propto \exp(-U_k/k_B T)$
- Metropolis-Hastings algorithm
  1) Calculate the initial energy U
  2) Choose a particle at random
  3) Propose a move for the particle
  4) Calculate the new/proposed energy U'
  5) Decided if the move should be accepted

$$P_{acc} = \exp(-\Delta U/k_B T), \text{if} \begin{cases} r \leq P_{acc} : & \text{accept} \\ r > P_{acc} : & \text{reject} \end{cases}$$

Why Metropolis-Hastings algorithm?

- If U' < U, the move will always be accepted.
- If U' > U, the move will not always be accepted.
- Therefore, by using Metropolis-Hastings algorithm, the system will be driven to a lower energy, which is more stable.

With Monte Carlo simulations, we can simulate the motion of the particles

Initialization (`class SystemSetup`)

- Parameters setup:
  - ✓ 500 particles, 10 million steps
  - ✓ $T^* = 0.9, \rho^* = 0.9, r_c = 3\sigma$
- Initial configuration: random placement

Energy calculation methods (`class Energy`)

- Periodic boundary condition
- Lennard-Jones potential
- Total pair energy
- Tail correction

Metropolis algorithm (`class MonteCarlo`)

- Calculate energy differences
- Accept/reject Monte Carlo moves
- Displacement adjustment

```
if acc_rate < 0.38:
    max_displacement *= 0.8
elif acc_rate > 0.42:
    max_displacement *= 1.2
```

Data generation

- Trajectory file (`.xyz` file)
- Molecular visualization by VMD
- Energy as a function of MC steps
- Final configuration at 3D space

We applied the knowledge of software engineering when developing the software package

- ✓ Best practices of GitHub workflow and PEP8 coding style
- ✓ Unit tests and functional tests following TDD
- ✓ Continuous integration using Travis CI
- ✓ Application of hash tables according to the result of profiling
- ✓ Object-oriented programming and design patterns

Object-oriented programming allows extensibility of the code

- Application of an abstract factory in `energy.py`
- Provides an interface for creating families of related or dependent objects without specifying their concrete classes.

Advantages of using an abstract factory

- Isolation of concrete classes
- Exchanging product families easily
- Promoting consistency among products

```python
from abc import ABC, abstractmethod


class EnergyModel(ABC):

    @abstractmethod
    def calc_energy(self):
        pass

    @abstractmethod
    def cutoff_correction(self):
        pass
```

Be Boulder.
University of Colorado Boulder

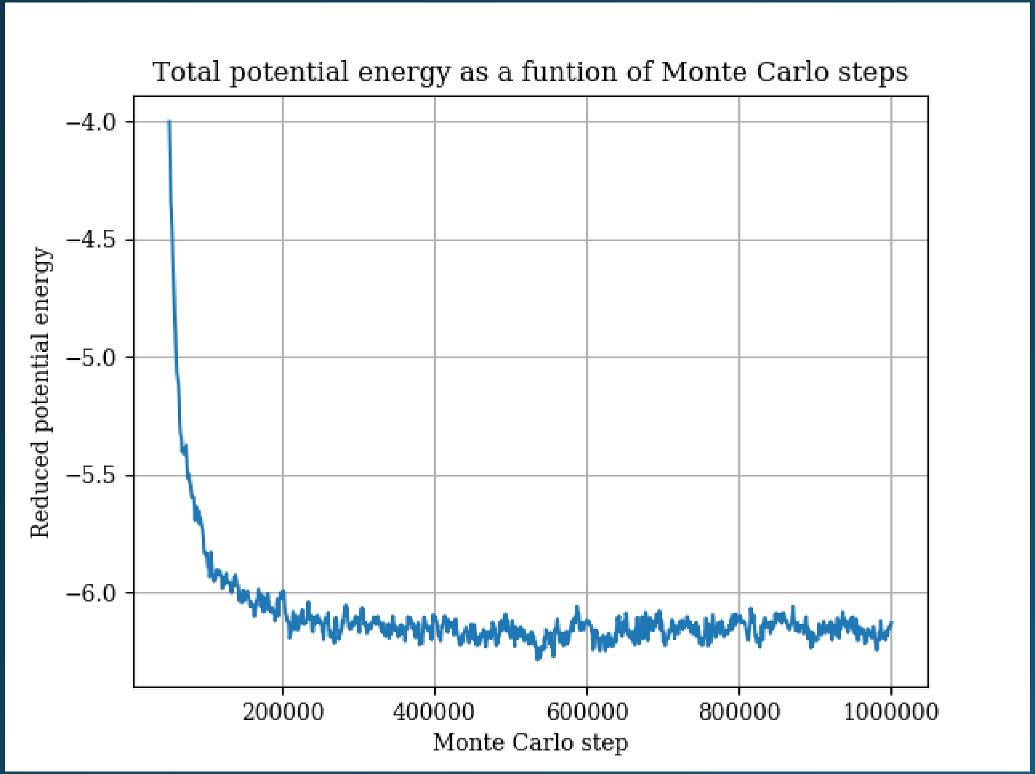
`class MonteCarlo` takes instance objects of other classes as inputs

```python
class MonteCarlo:
    def __init__(self, system: object = None, energy: object = None,
                 args: object = None):

        # get parameters from the class SystemSetup
        self.N_particles = system.N_particles
        self.coordinates = system.coordinates
        self.box_length = system.box_length

        # get parameters from the class Energy
        self.init_ener = energy.calc_init_ener(
            self.coordinates, self.box_length)
        self.tail = energy.calc_tail(self.N_particles, self.box_length)
        self.energy = energy    # to extract the attributes in Energy class

        # get parameters from the method initialize()
        self.args = args
```

We used hash tables to enhance the efficiency of the code

- Result of profiling
  - ✓ Simulation: 20 particles, 100000 steps
  - ✓ `calc_energy`, which took 20.09 seconds (26.4%), needed to be improved.
  - ✓ Reason of high computational cost:
    - ➢ Number of pairs is quadratically proportional to the number of particles.
    - ➢ The neighbors of each particle update every Monte Carlo step.
- Improvement made
  - ✓ Python dictionary: implementation of hash tables
  - ✓ Result: decrease the computer time of `calc_energy` to 13.86 seconds, which is 1.45 times faster

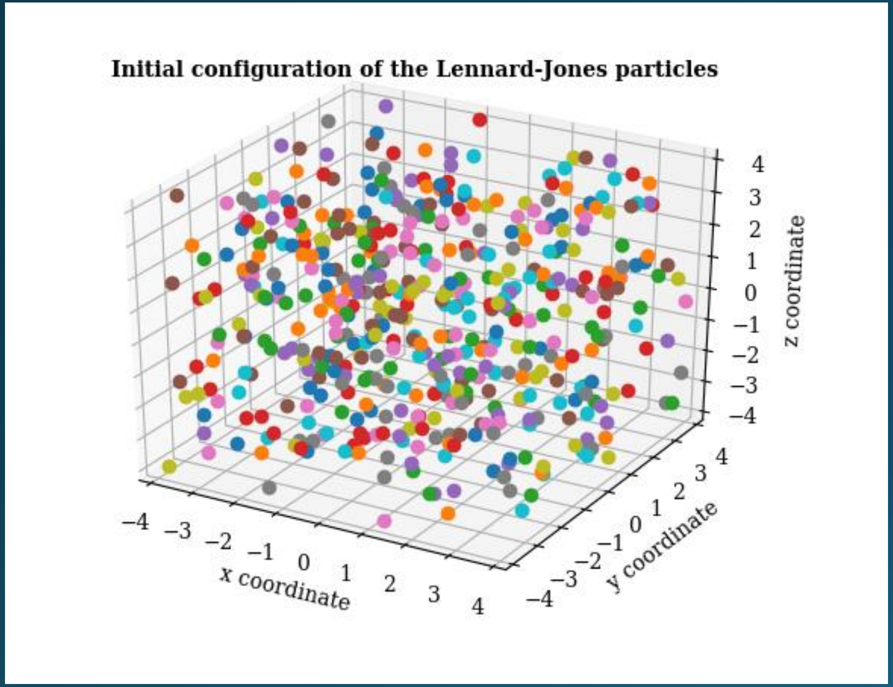
The prediction of the total potential energy was satisfactory

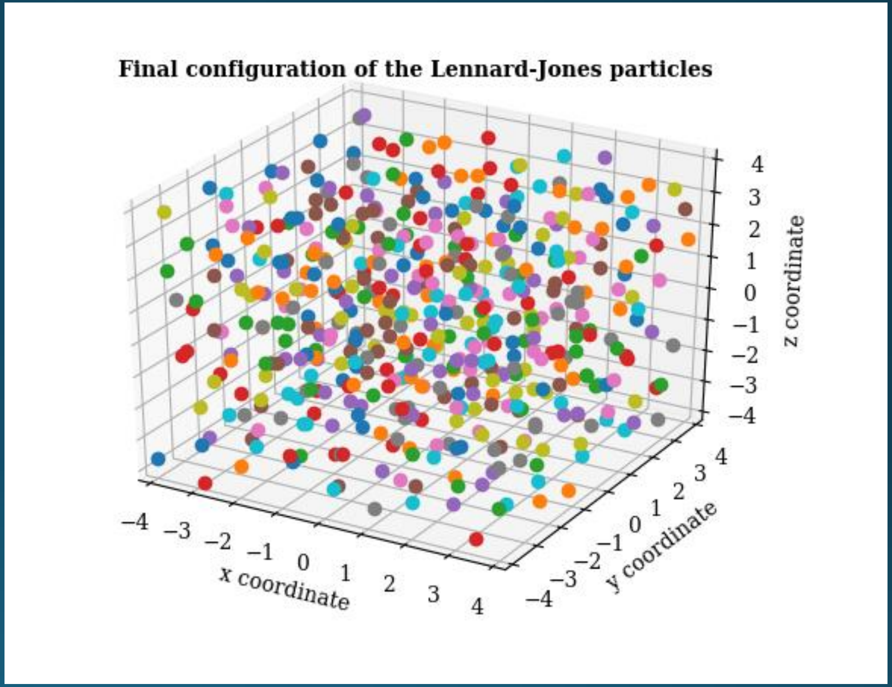


- After 1 million Monte Carlo steps, the total potential of the system averaged over the last 100,000 steps is -6.1616, which is pretty close to the NIST benchmark (-6.1773).
- The total reduced potential energy decreased very rapidly and converged to values around 6.1 within few steps.

Particles had a tendency to distribute more uniformly in the box

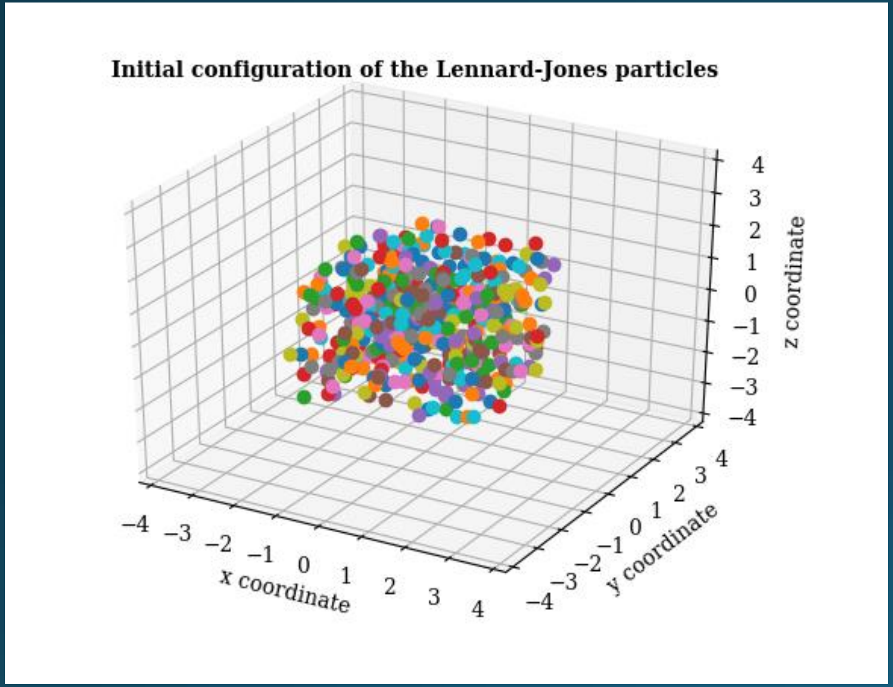
- Group A: Simulation starting with particles randomly placed in the entire box

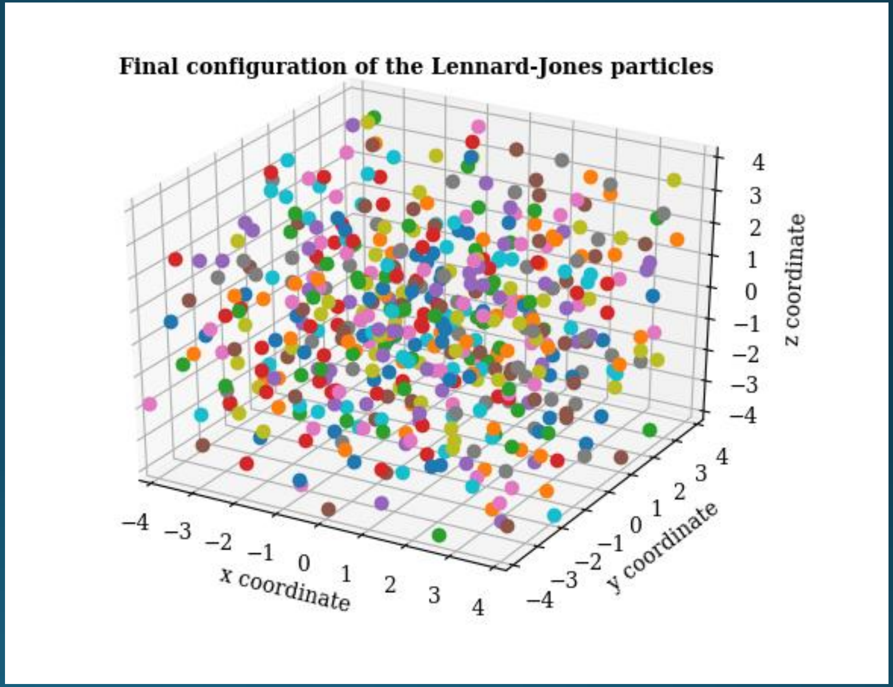




Particles had a tendency to distribute more uniformly in the box

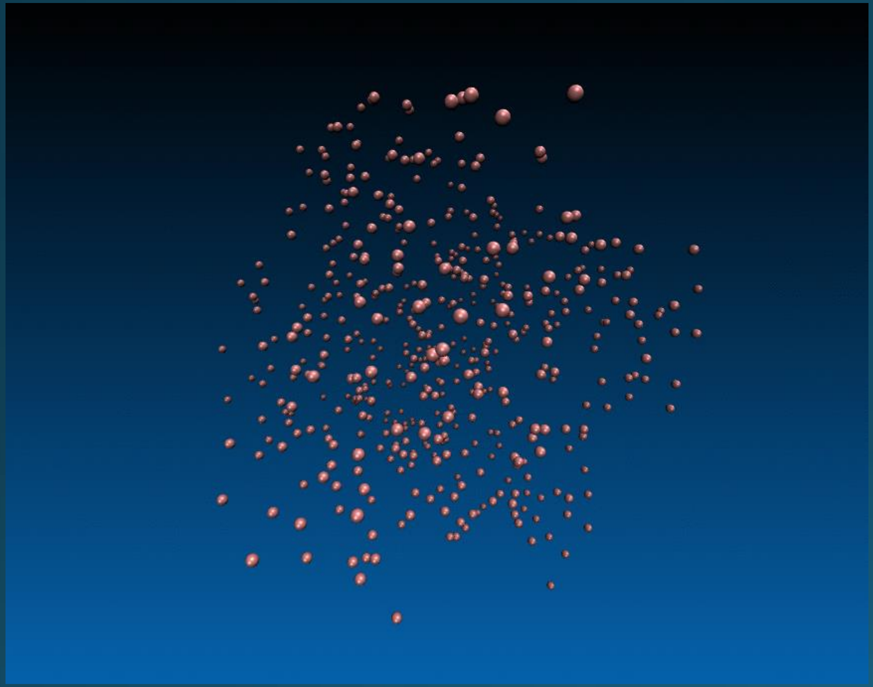
- Group B: Simulation starting with particles randomly placed in a smaller space in the box



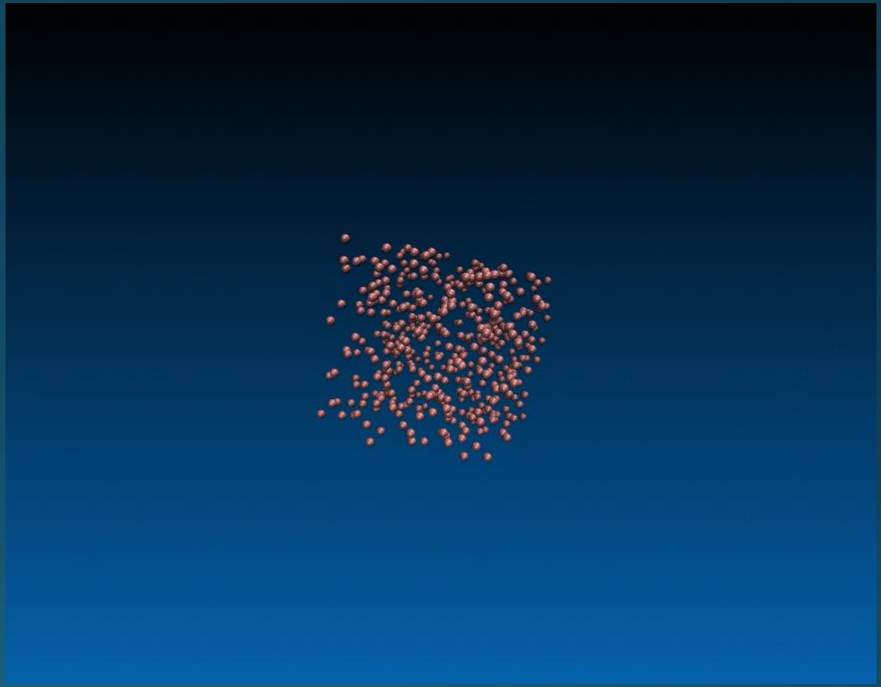


Visual Molecular Dynamics allows us to examine the trajectory easily

Group A



Group B



The model could be useful in simple systems but still has many limitations

- For simple systems like our case, our model is able to provide satisfactory prediction of the total potential energy of the system.
- This model is only applicable to
  - ✓ NVT ensemble simulation
  - ✓ Particles with only Lennard-Jones potentials in between