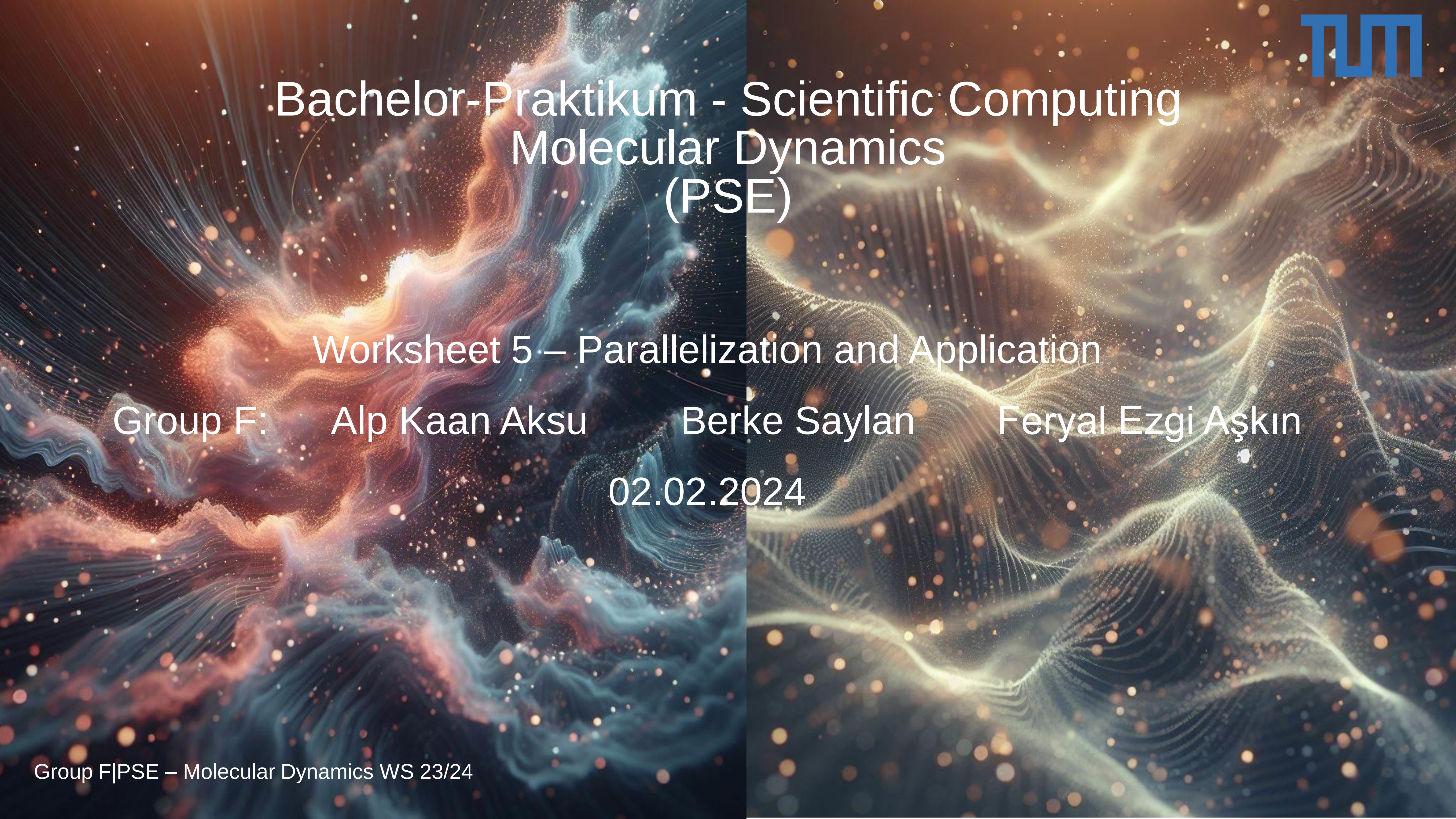


Bachelor-Praktikum - Scientific Computing
Molecular Dynamics
(PSE)

Worksheet 5 – Parallelization and Application

Group F: Alp Kaan Aksu Berke Saylan Feryal Ezgi Aşkın

02.02.2024

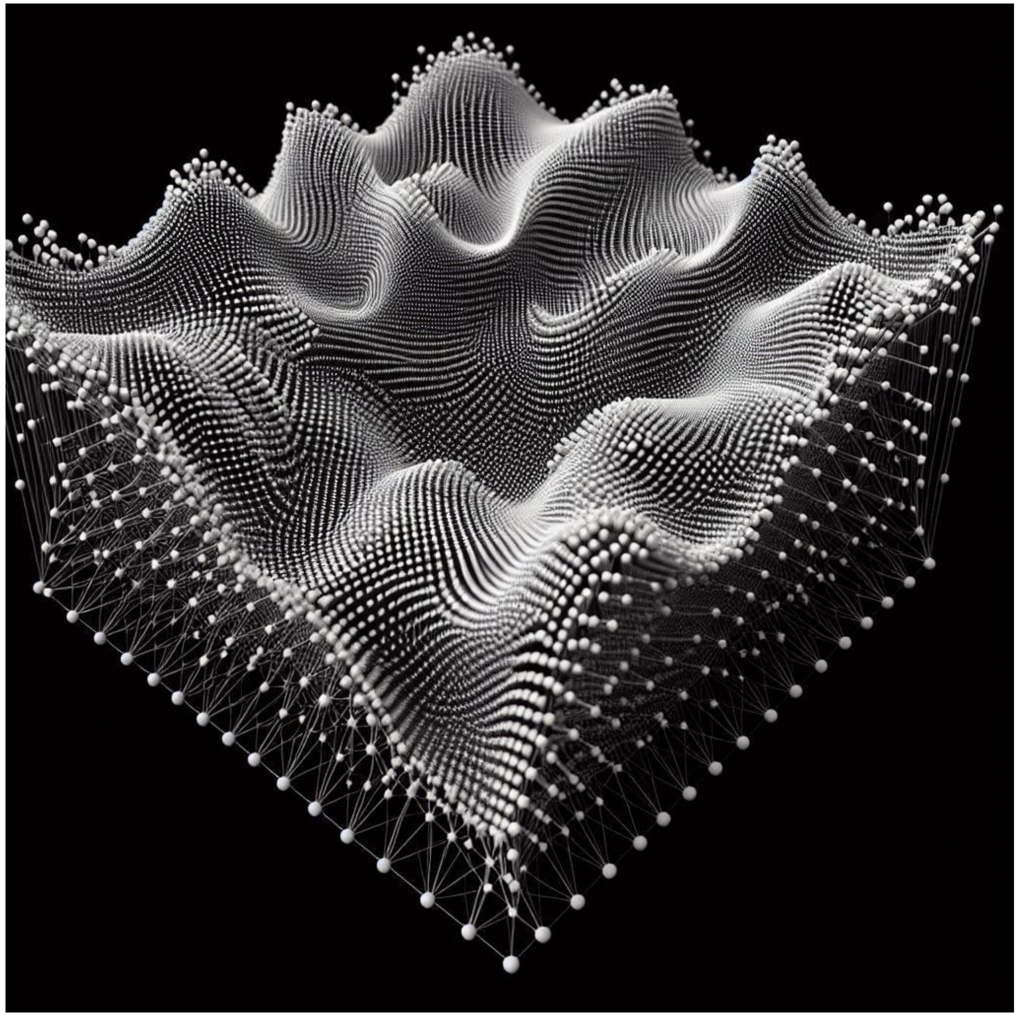
Simulation of a Membrane

Generator Class

- ‘membrane’ method is responsible for creation of a cuboid filled with particles that have the id attribute and have correct neighbors set as their direct/diagonal neighbors

- We hardcoded membrane to work with neighboring relation along x and z axes, instead of x and y
- **Reason:** the gravity works along y axis and assigning the pulling force also along y axis visualizes the membrane structure much better in the simulation, plus its the last iteration, so hardcoding doesn’t affect future iterations

- Neighbor particle coordinates are calculated by making use of division and modulo operations. After validity checks for simulation range is done, the neighbor particles are set with their ids to the ‘direct/diagonalNeighbors’ int vectors



Simulation of a Membrane

LinkedCellParticleContainer and Simulation Classes

- In the ‘Simulation’ class membrane and non-membrane simulations are differentiated and force calculation is done accordingly. Again simulation parameters are adjusted so that cuboid orientation lies on the x-z plane flatly and correct particle setup for pulling force application is done.

- ‘applyToAllPairsOnceMembrane’ iterates through all unique particle pairs analogous to the non-membrane cases, with the only difference being that instead of Lennard-Jones potential the harmonic potential is applied on the particle pairs if they are neighbors of each other (which is checked by the id attribute)

- If they are not neighbors, then Lennard-Jones potential truncated at 2^(1/6) * σ is applied on the particle pairs, to avoid self-penetration of the membrane

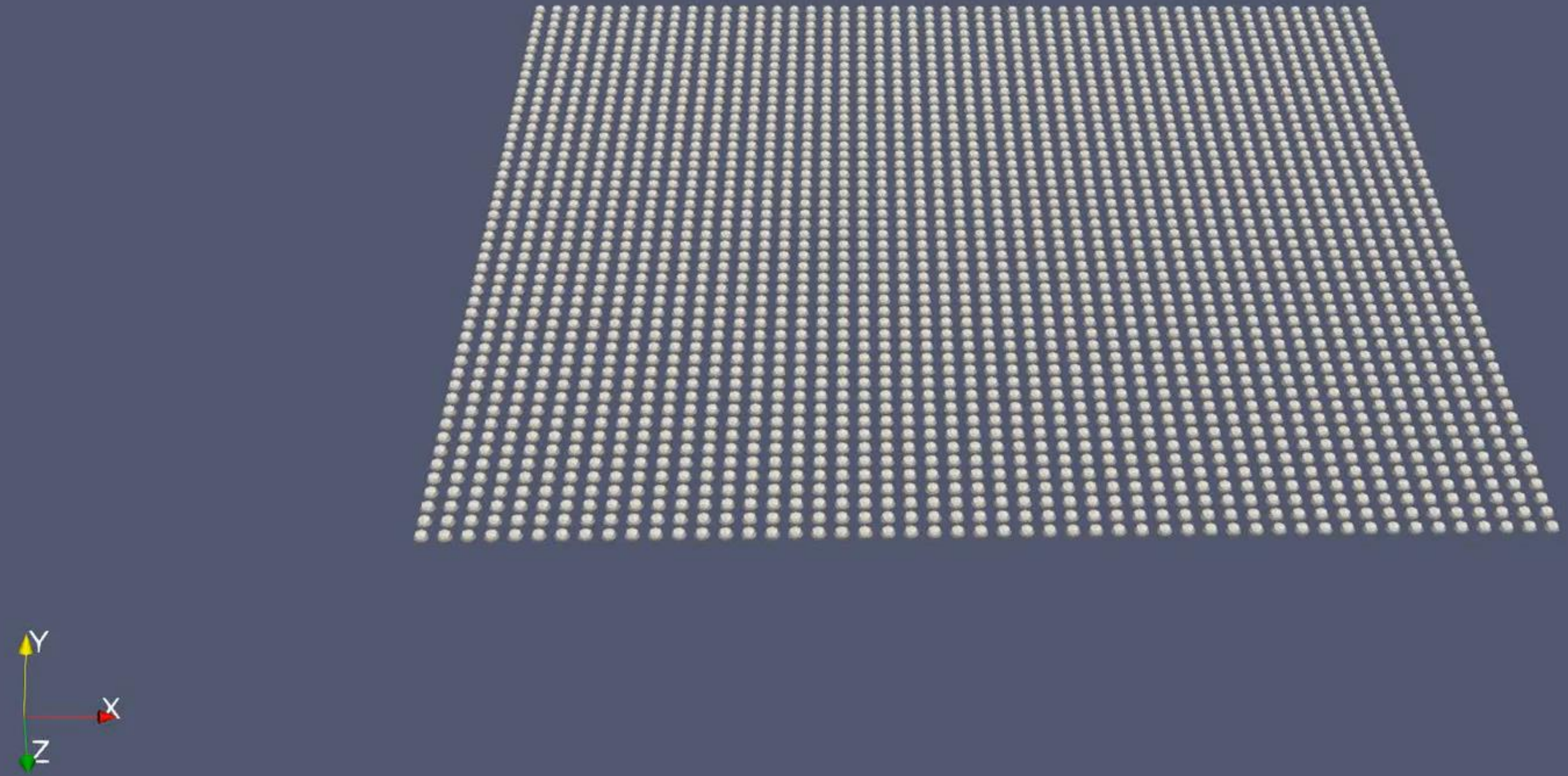
$$F(x_i, x_j) = k \cdot (||x_i - x_j||_2 - r_0) \cdot \frac{(x_j - x_i)}{||x_i - x_j||_2}$$

Directly neighboring particles

$$F(x_i, x_j) = k \cdot (||x_i - x_j||_2 - \sqrt{2} r_0) \cdot \frac{(x_j - x_i)}{||x_i - x_j||_2}$$

Diagonally neighboring particles

Membrane Simulation

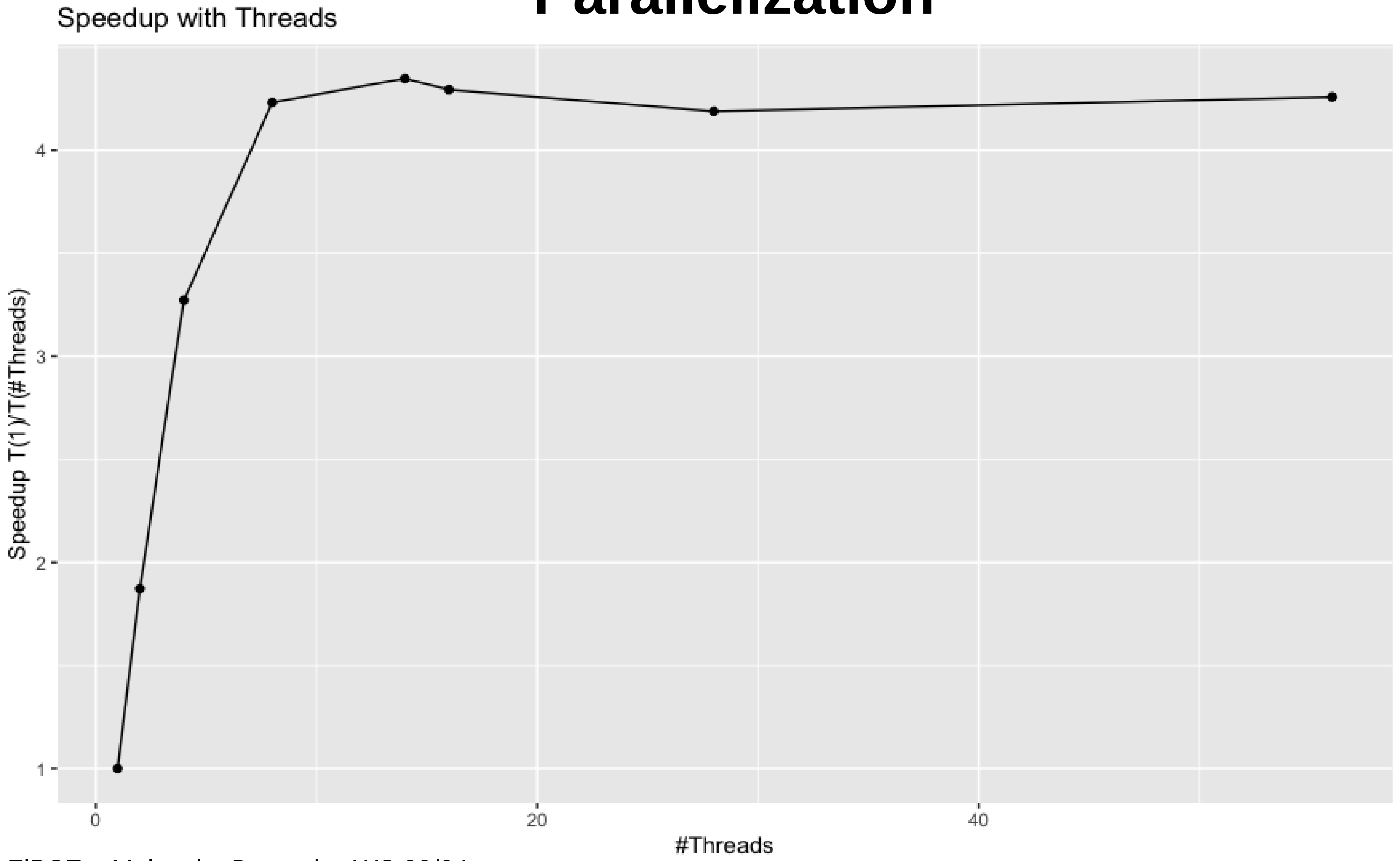


Parallelization

- One of our parallelization ideas was to parallelize the cell and particle iteration in the ‘applyAllPairsOnce’ method for pairwise force calculation and the results showed that it enhanced the performance significantly

- We also tried other approaches for parallelization, e.g. parallelizing also ‘applyToAll’ method etc. However all of these approaches caused one of the two following problems: either the overhead resulting from threads was so high (especially because of many critical regions) or the result wasn’t deterministic, as critical regioning wasn’t applied correctly. In all of the cases this dilemma forced us to actually not employ the respective parallelization approach

Parallelization



Rayleigh-Taylor Instability in 3D

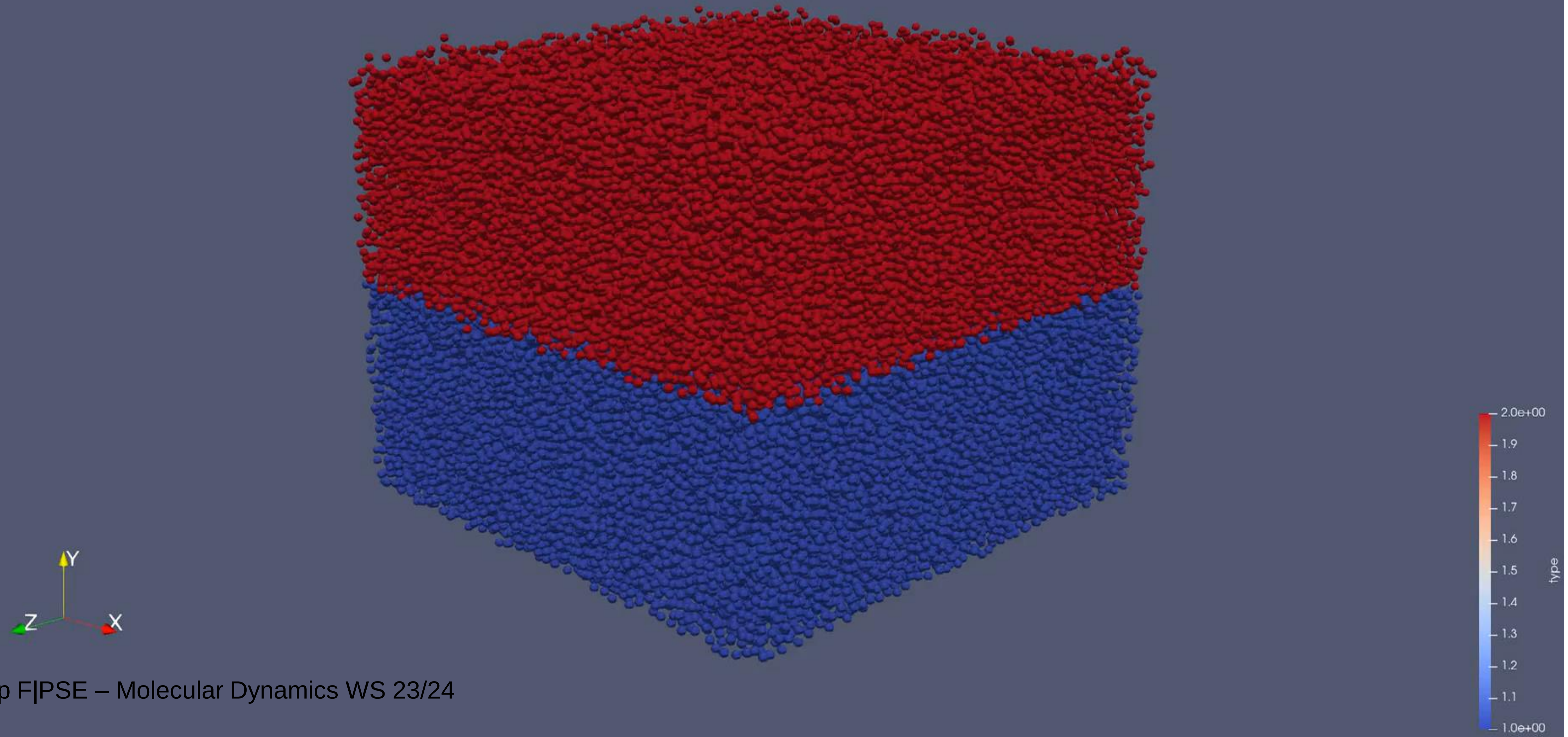
- **Initial Assumption:** Our 2D implementation was assumed to suffice for the 3D case, as we «designed it that way»

- **Identified Issue:** Problem arose in handling halo particle creation, when more than one axis is periodic, as there were forgotten cases for halo cell creation for corner boundary cells

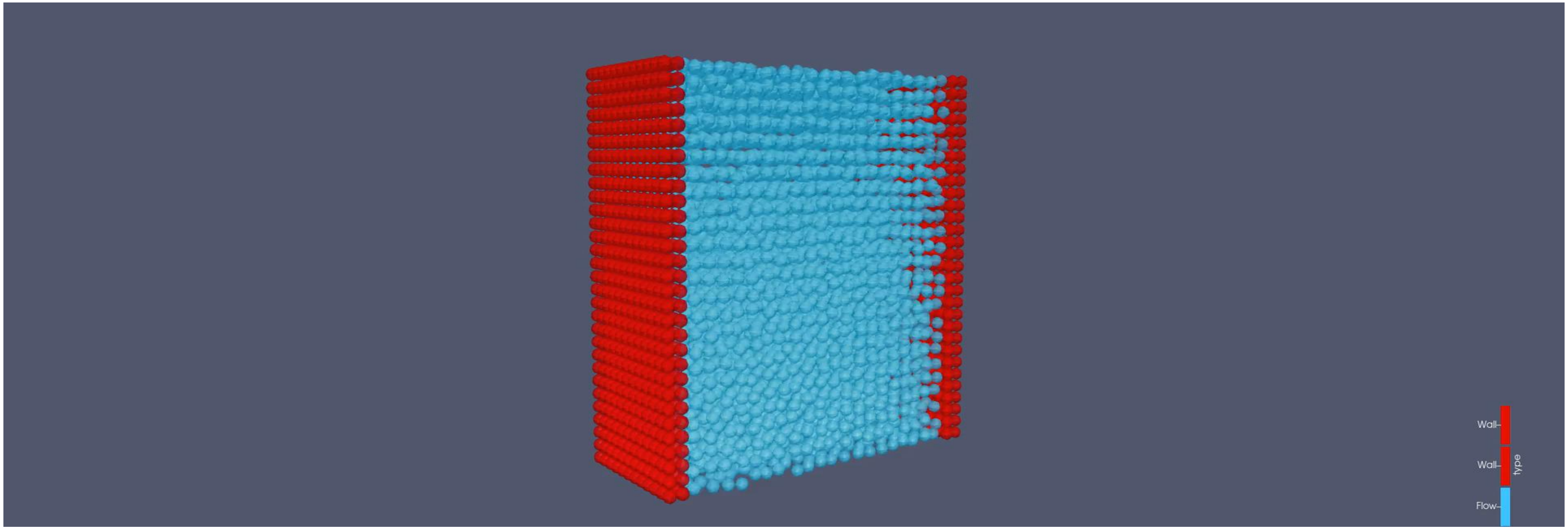
- **Resolution Steps:** Revised the logic for halo particle copying. Introduced three methods: ‘handleBoundariesOneAxis’, ‘handleBoundariesTwoAxes’, and ‘handleBoundariesThreeAxes’ for periodicity handling.

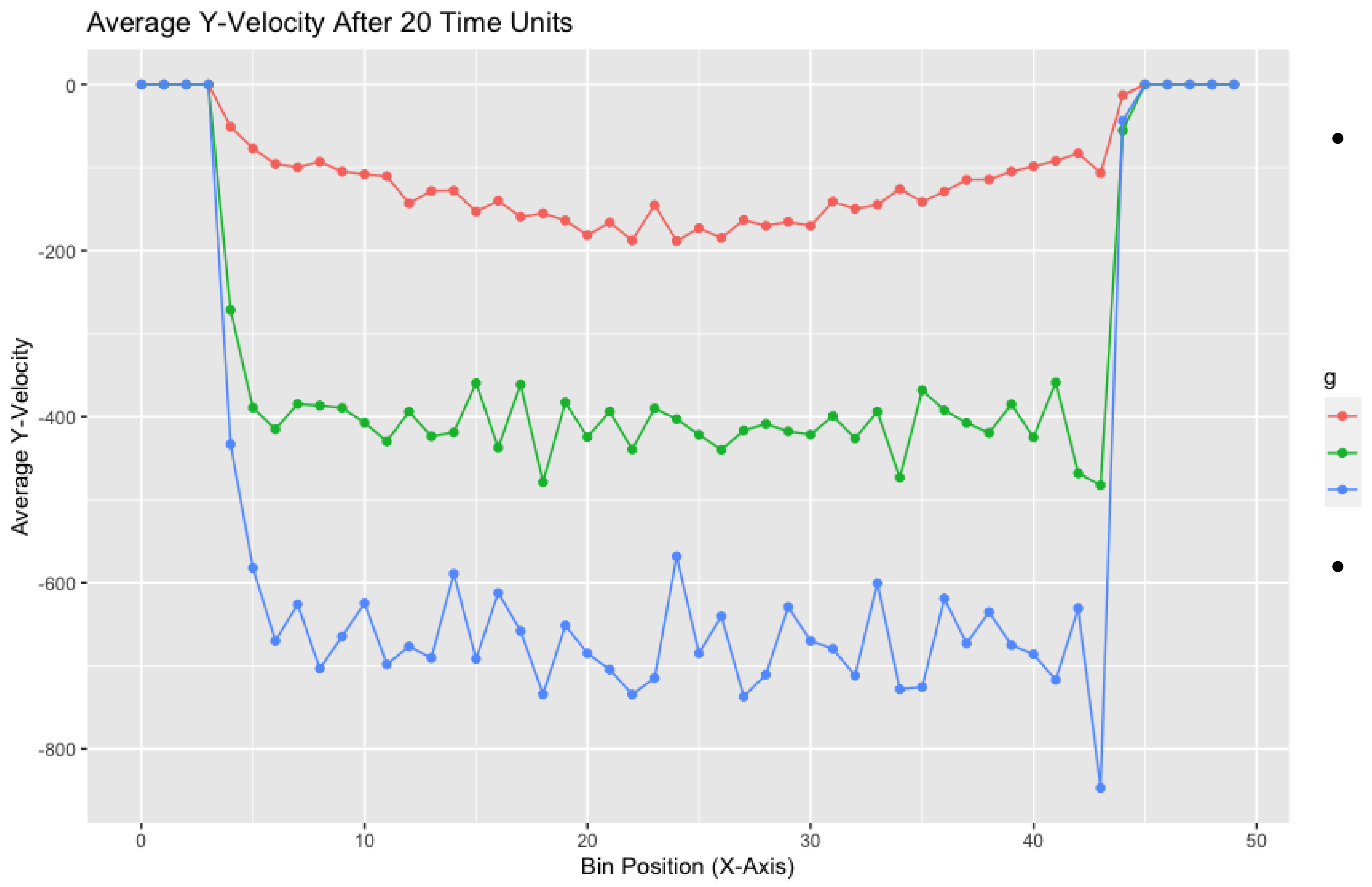
- These methods, integrated into the general method for updating halo cells, ensure accurate periodic behavior across all axes during simulation updates.

Rayleigh-Taylor Instability in 3D



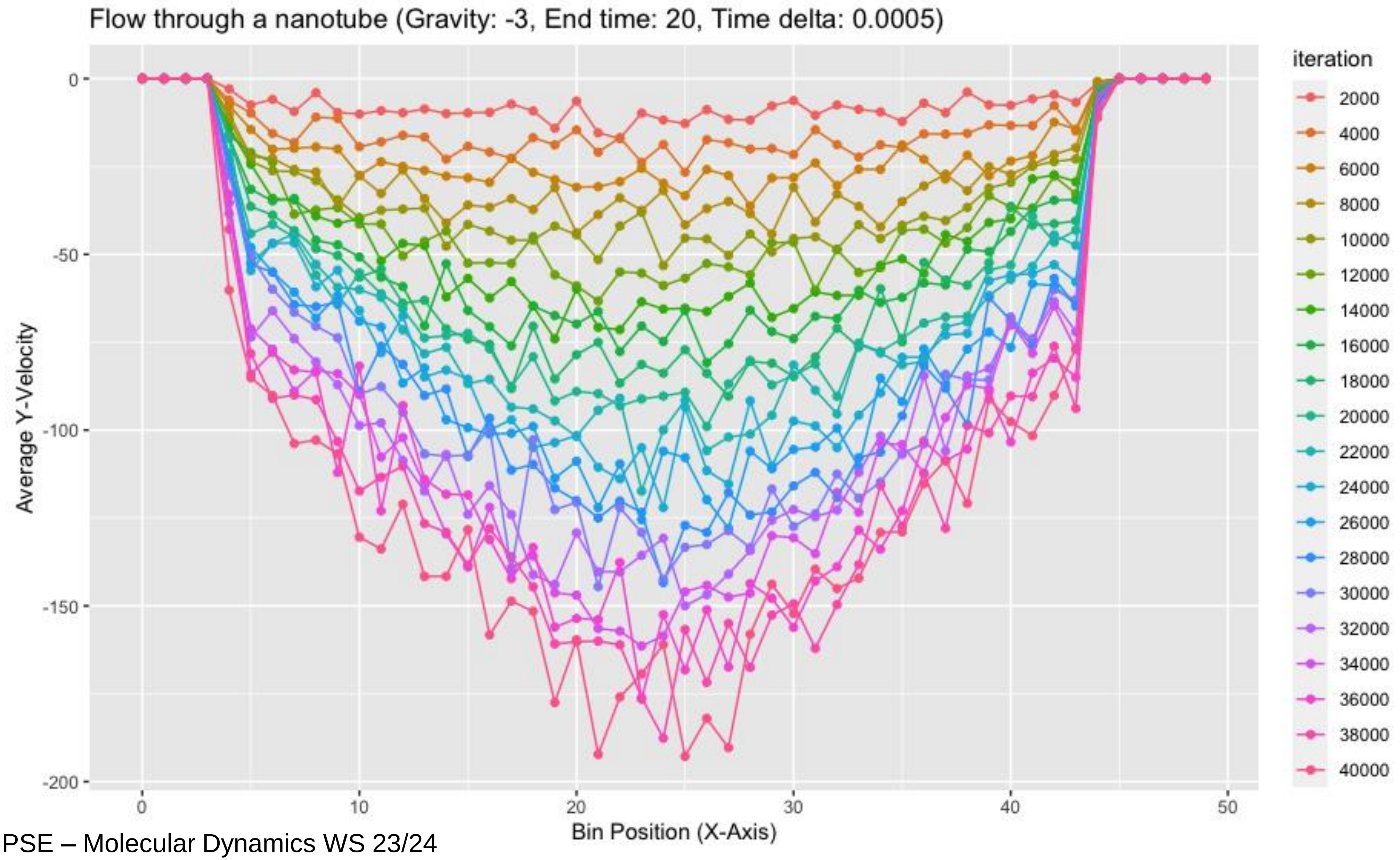
Nanotube Flow Simulation

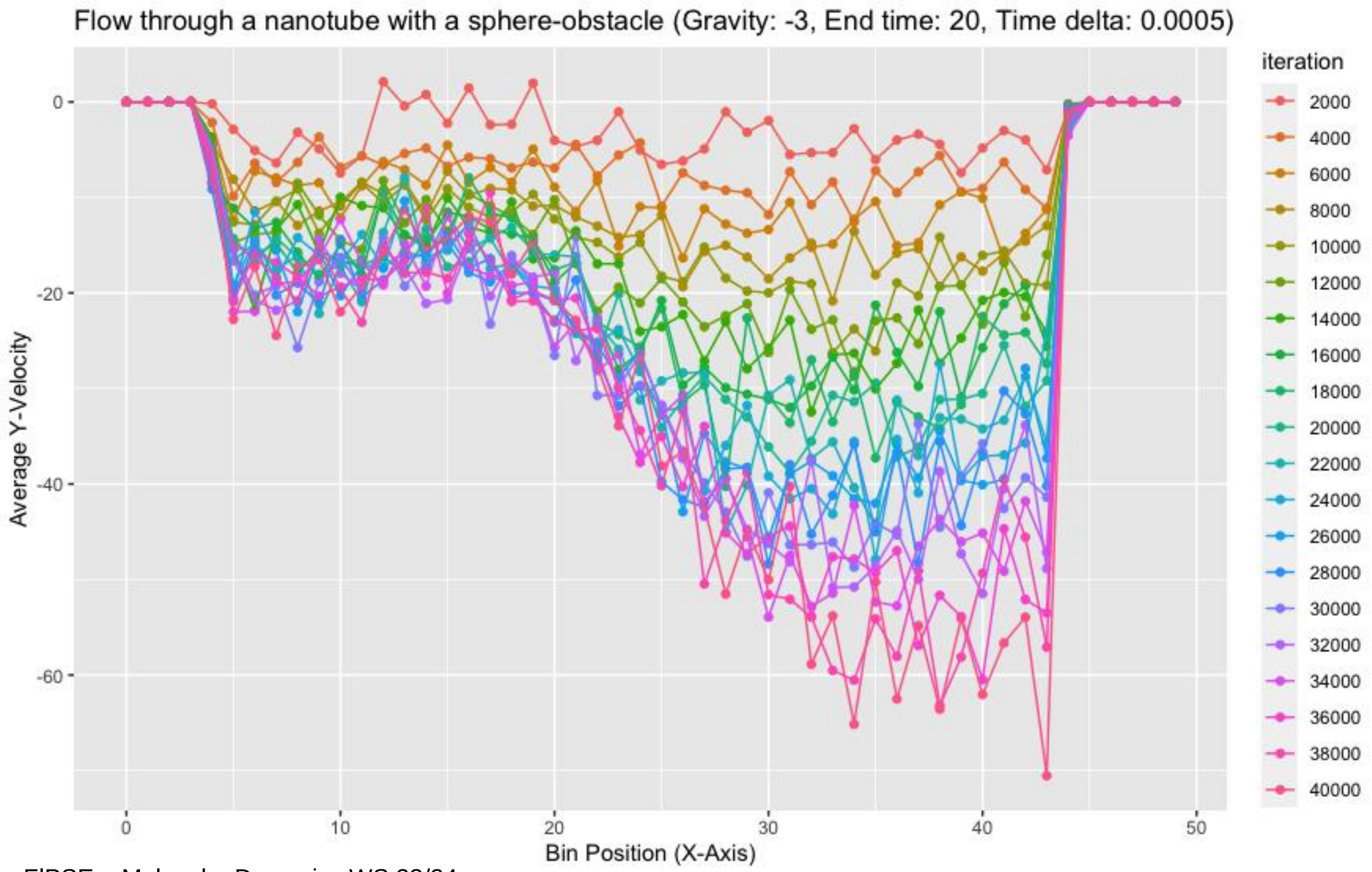




- ‘fixed’ bool as a Particle attribute to differentiate between flowing and wall particles
- Thermostat chooses dynamically between 2 methods for velocity scaling

- **Density and Velocity Profiling in Nanotube Flow Simulation With Respect To Proximity To Walls**

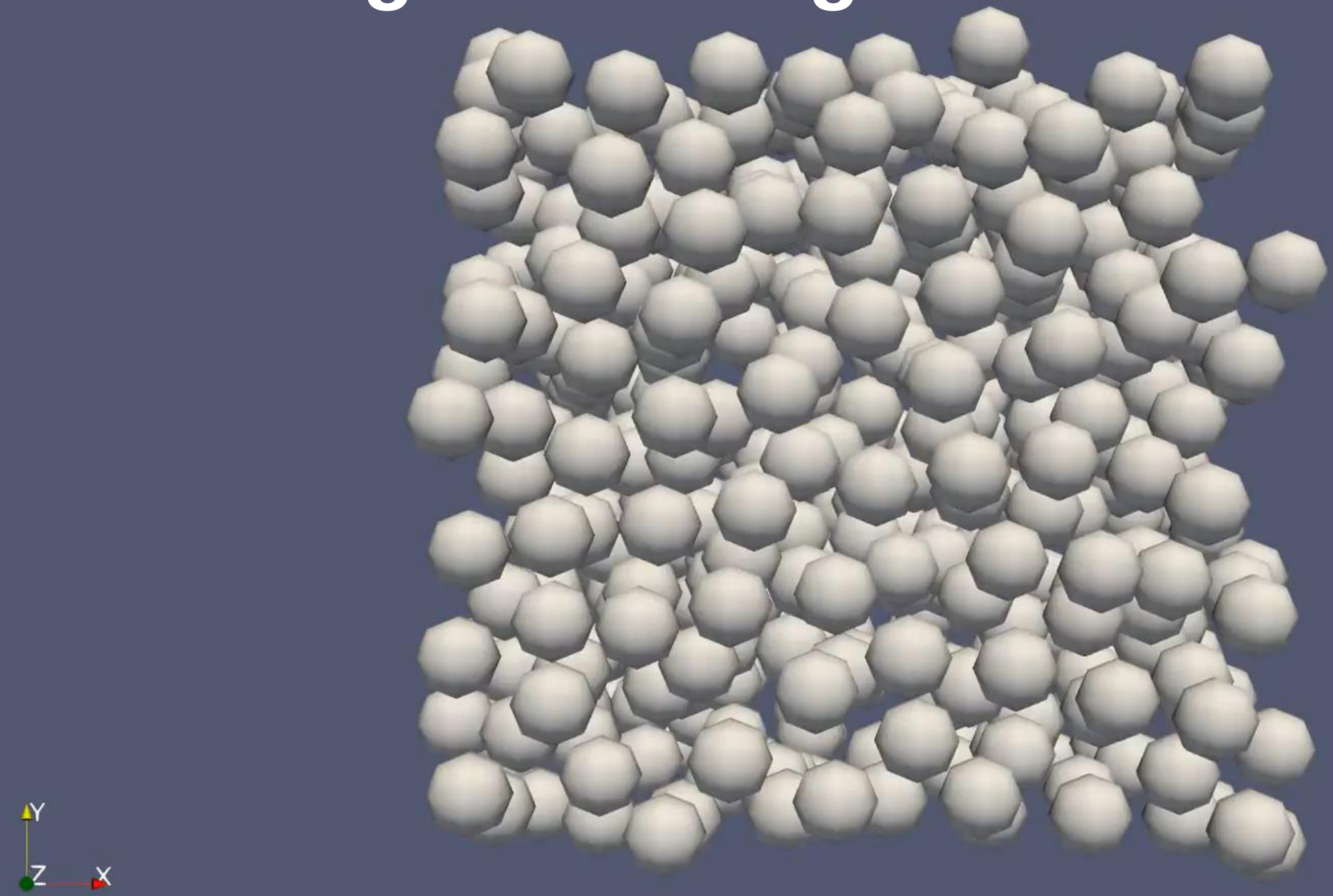



Flow through a nanotube with a sphere-obstacle (Gravity: -3, End time: 20, Time delta: 0.0005)
Average Y-Velocity
Bin Position (X-Axis)
0
-20
-40
-60
0
10
20
30
40
50
iteration
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
22000
24000
26000
28000
30000
32000
34000
36000
38000
40000

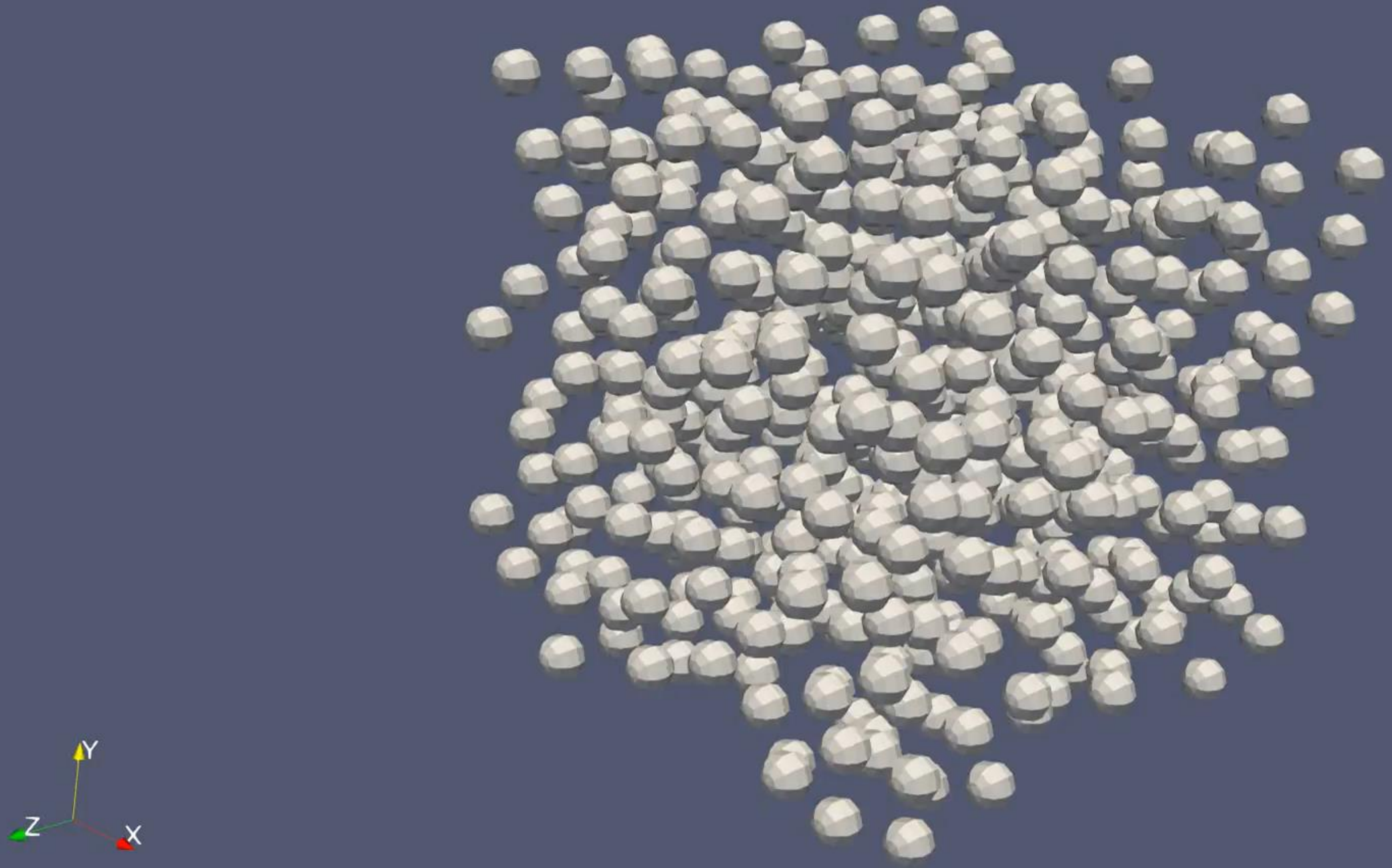
Argon Cooling Simulations

- As a bonus, we also did the Argon simulations to observe how cooling processes occur in accordance with our molecule simulation logic, the outcomes were satisfying

Argon Cooling Simulation



Argon Super Cooling Simulation



References

All images are generated by us using Microsoft Bing AI, so no copyright