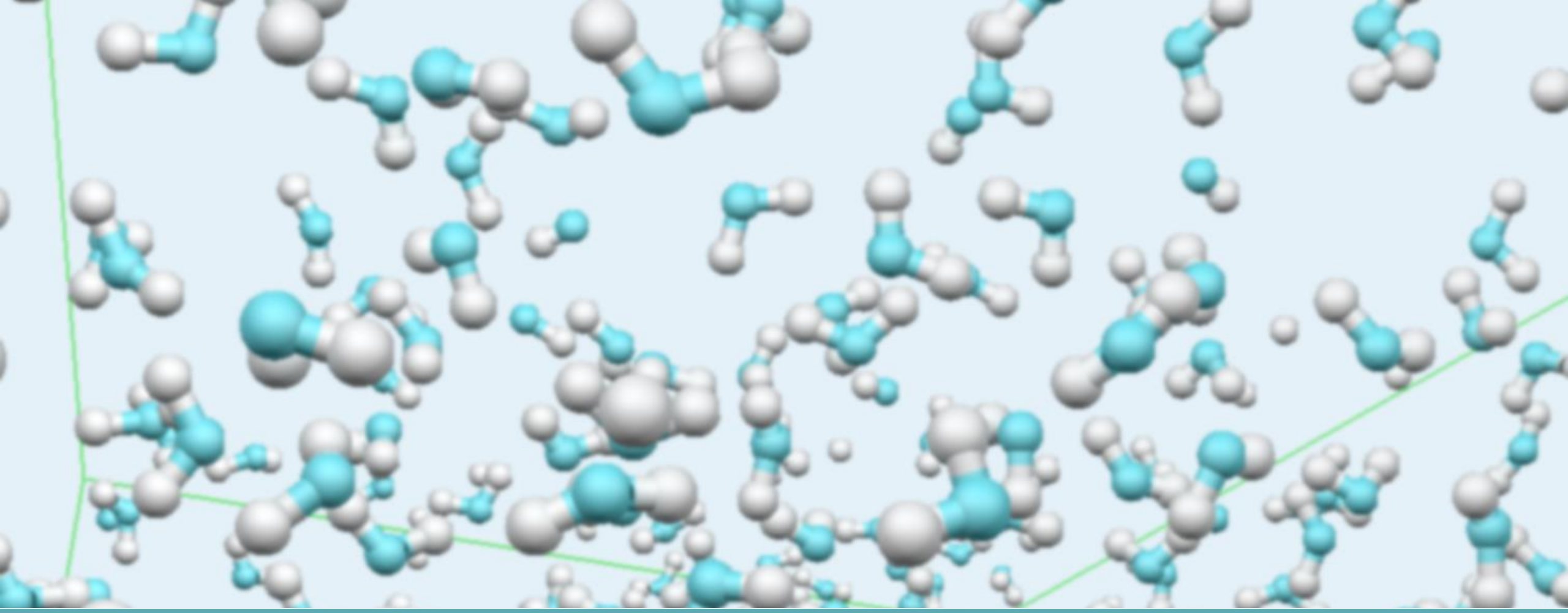


TaichiMD

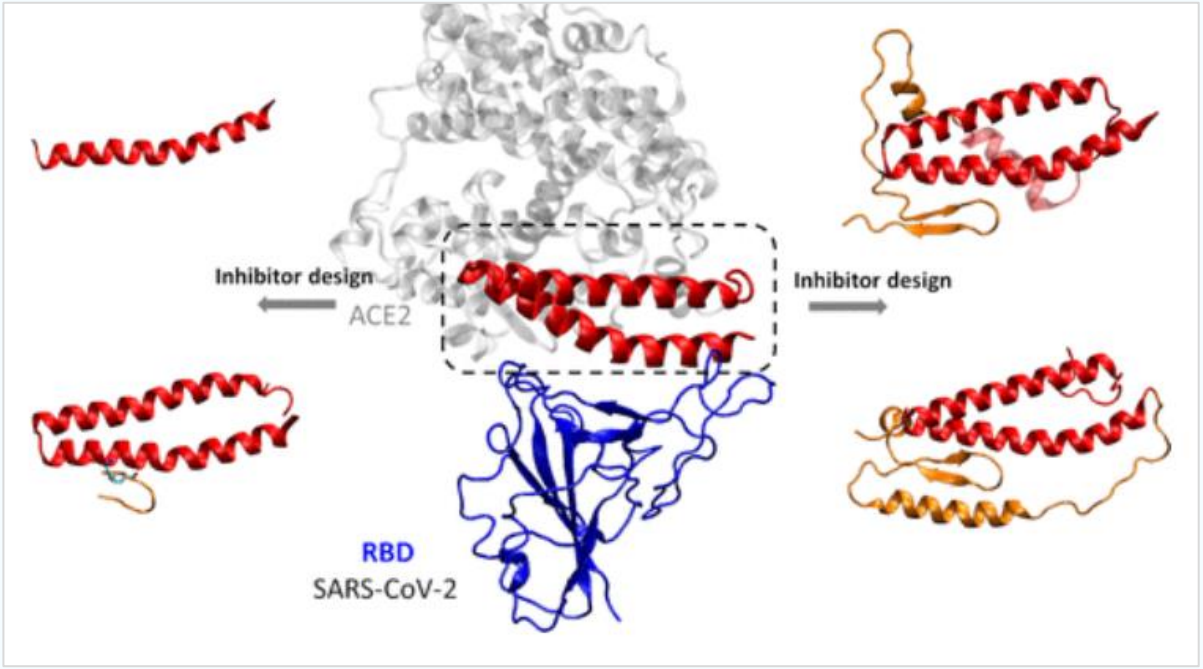
Interactive, GPU-accelerated Molecular Dynamics using the Taichi programming language

Yangzesheng Sun
Department of Chemistry & Department of Computer Science and Engineering,
University of Minnesota

Background

Computer simulations of microscopic systems is one of the key approaches to scientific discovery in physics, chemistry, biology, and materials science.

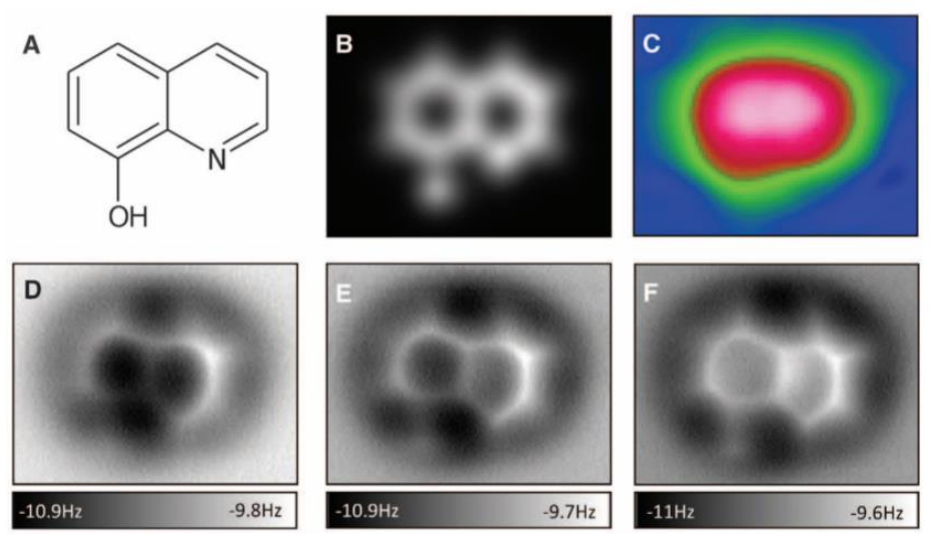
Molecular dynamics (MD) simulations are employed to design ACE2 inhibitors which prevents SARS-CoV-2 (COVID-19 virus) from infecting human cells.



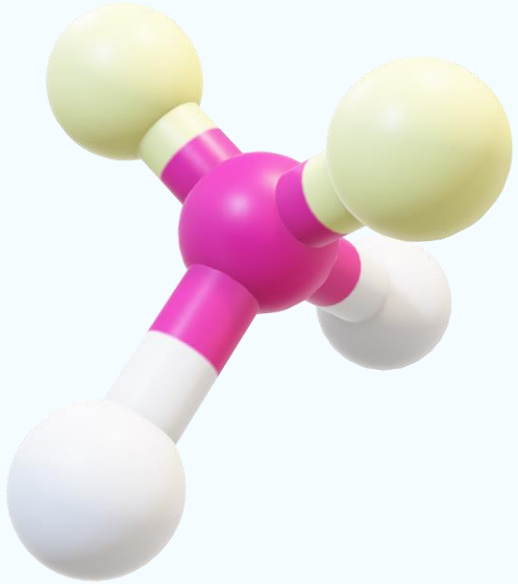
Y. Han *et al*, Computational Design of ACE2-Based Peptide Inhibitors of SARS-CoV-2, *ACS Nano* **2020**, 14, 4, 5143–5147

Background

Developing visualization and interactive techniques can improve our understanding of microscopic structures and processes



A – Molecular structure illustration
B – Imaging of the molecule generated by simulations
C – Visualization using scanning tunnel microscopy (STM)
D, E, F – Visualization using atomic force microscopy (AFM)



3D structure model of dichloromethane (CH_2Cl_2)

J. Zhang *et al*, Real-Space Identification of Intermolecular Bonding with Atomic Force Microscopy, *Science* **2013**, 342, 611–614

Background

The Taichi programming language provides new possibilities into molecular simulations compared with traditional molecular simulation packages written in Fortran or C.

- Algorithms and data structure in molecular simulations are highly relevant to computer graphics, such as Monte Carlo, time integrators, and *N*-body dynamics
- Differentiable or machine-learning-based molecular simulations has become a hot research topic in computational physics and chemicstry

Approach	Forward Time	Backward Time	Total Time	# Essential LoC
PyTorch (CPU, f32)	405 ms	328 ms	733 ms (13.8×)	74
PyTorch (GPU, f32)	254 ms	457 ms	711 ms (13.4×)	74
Autograd (CPU, f64)	307 ms	1197 ms	1504 ms (28.4×)	51
JAX (GPU, f32)	24 ms	75 ms	99 ms (1.9×)	90
DiffTaichi (CPU, f32)	66 ms	132 ms	198 ms (3.7×)	75
DiffTaichi (GPU, f32)	24 ms	29 ms	53 ms (1.0×)	75

Taichi is an order of magnitude faster in physical simulations compared with PyTorch and Autograd (Eulerian smoke simulation shown as example)

Y. Hu *et al*, DiffTaichi: Differentiable Programming for Physical Simulation, *ICLR 2020*.

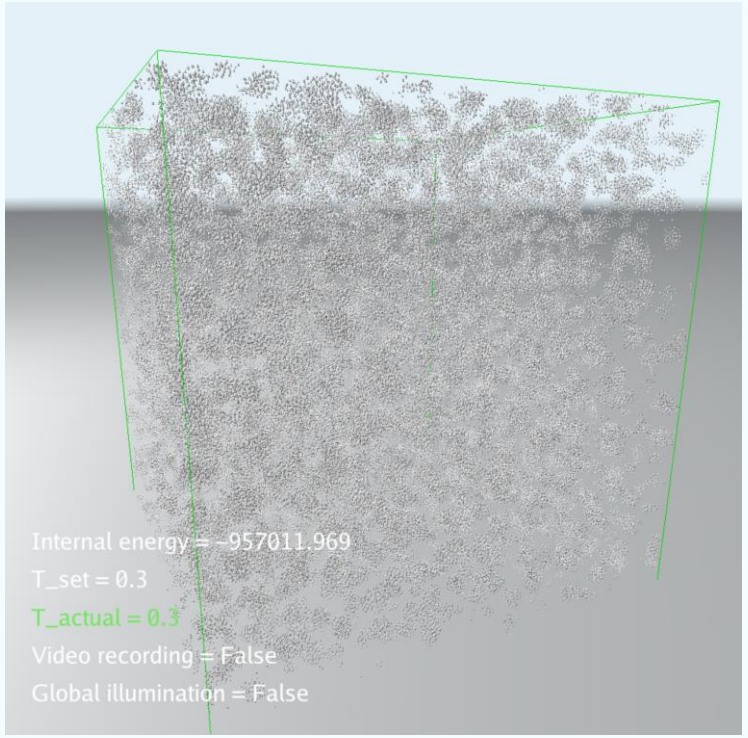
Introduction

TaichiMD is a real-time interactive molecular simulation package based on the Taichi programming language. Its features and goals include:

- Visually pleasing and interactive particle simulations accelerated by GPU
- A platform for rapid implementation of new simulation algorithms and differentiable simulations

TaichiMD is capable of performing various microscopic and macroscopic particle-based simulations with neighbor grid acceleration for large systems, including monoatomic molecules, polyatomic molecules, and particle-in-cell methods.

The figure shows a fluid simulation of 260,000 Lennard-Jones particles in TaichiMD at semi-interactive frame rates (~15 FPS). Typical simulations Lennard-Jones systems do not exceed 10,000 particles on a single computer.

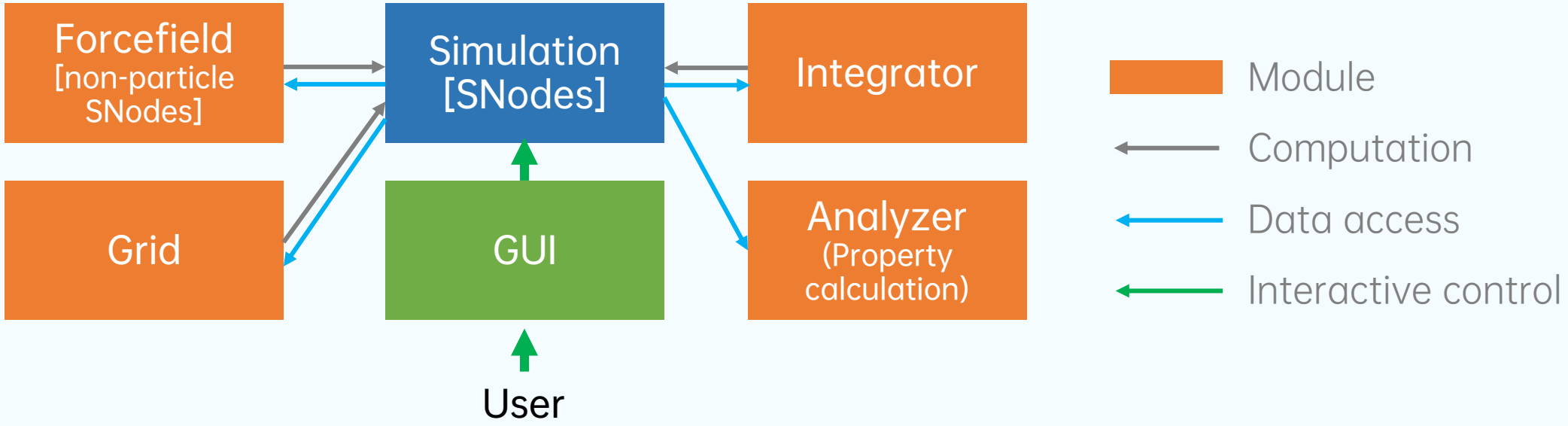


[1] Lennard-Jones potential, Wikipedia, https://en.wikipedia.org/wiki/Lennard-Jones_potential
[2] Q. P. Chen *et al*, Using the k-d Tree Data Structure to Accelerate Monte Carlo Simulations, *J. Chem. Theory Comput.* **2017**, 13, 4, 1556–1565

System

TaichiMD extensively uses **the objective data-oriented programming (ODOP) feature of the Taichi programming language.**

- Data-oriented: Data relevant to the simulation system in the taichi scope, such as particle position, velocity, and grid attributes, comprise the core data structure of TaichiMD (the `Simulation` class and its subclasses)
- Object-oriented: Components of TaichiMD is highly modular, with the integrator, grid, forcefield all being `Module` subclasses. This decouples the computation from data structure which enables rapid implementation of new algorithms through new subclasses



System

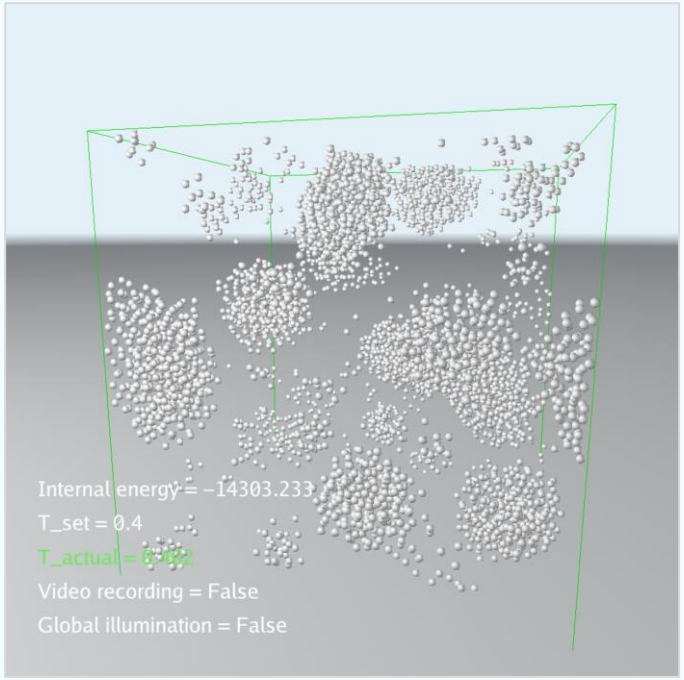
TaichiMD **includes a high-performance implicit surface renderer for particle systems with global illumination in pure Taichi**

- Spherical particles and chemical bonds are directly rendered using particle positions without triangle meshes
- Physically-based shading with Cook-Torrance BRDF[1] model
- Approximate global illumination algorithm based on radiance transfer[1] obtaining noise-free analytical solutions
- Real-time performance Rendered in for large systems of 260,000 particles when simulation is paused (> 30 fps, NVIDIA RTX 2080 Super)

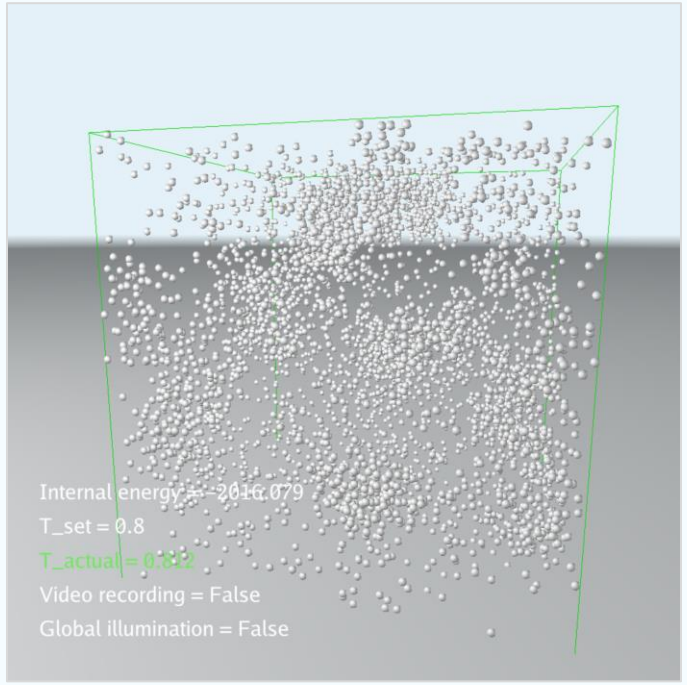
[1] PBR: Theory，LearnOpenGL.com, https://learnopengl.com/PBR/Theory
[2] Dynamic Ambient Occlusion and Indirect Lighting, *GPU Gems 2*, Addison-Wesley, 2005.

Visualization results

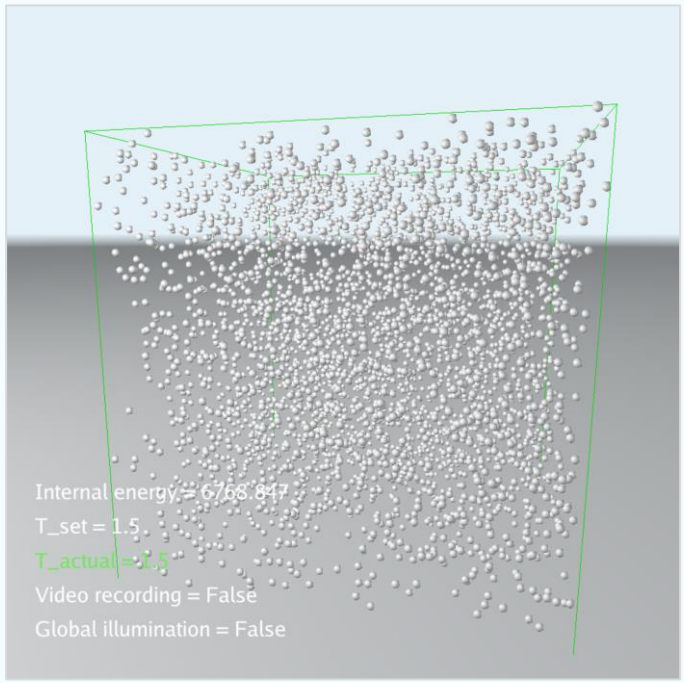
Evolution of microscopic systems through time or thermodynamic states can be directly visualized using interactive simulations of TaichiMD. For example, transition and coexistence between vapor, liquid, and solid phases can be observed in MD simulations in canonical (*NVT*) ensemble:



T = 0.4, **Vapor–solid coexsitence**



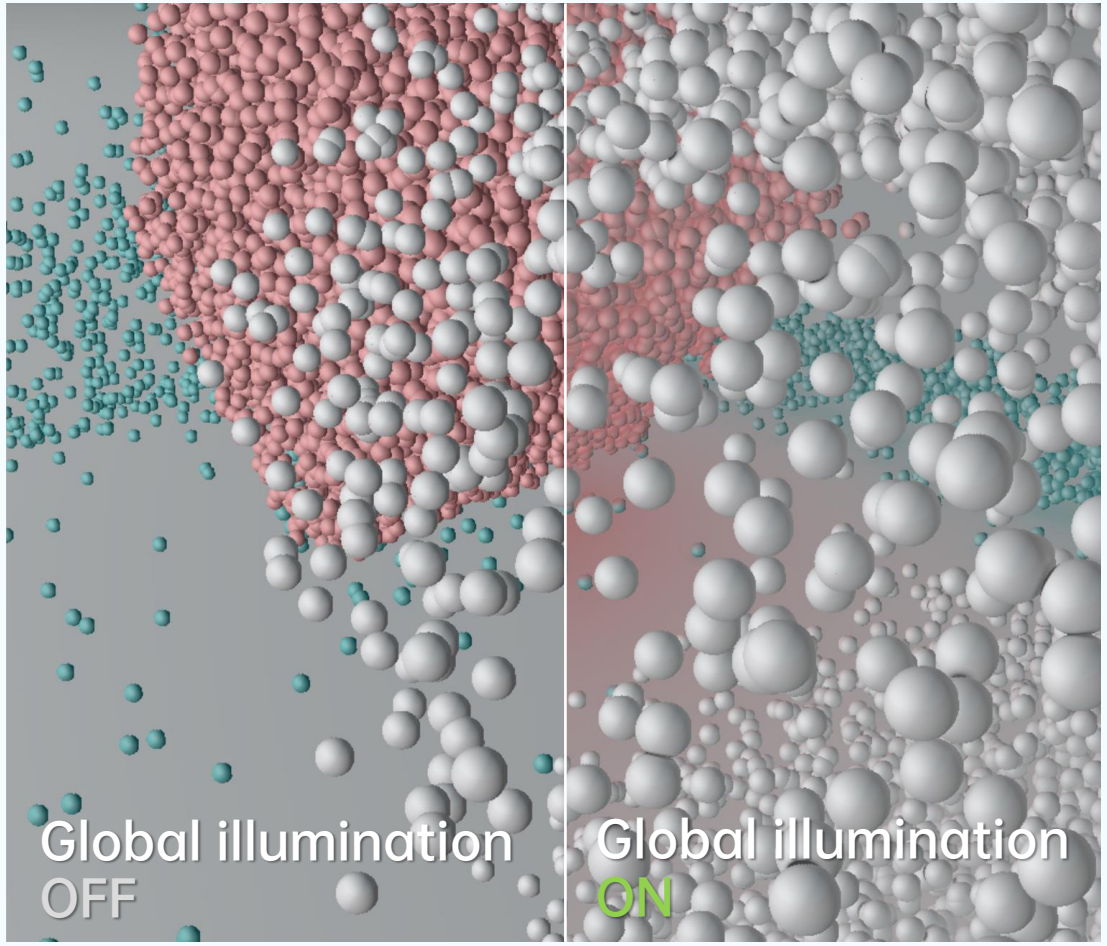
T = 0.8, **Vapor – liquid coexistence**

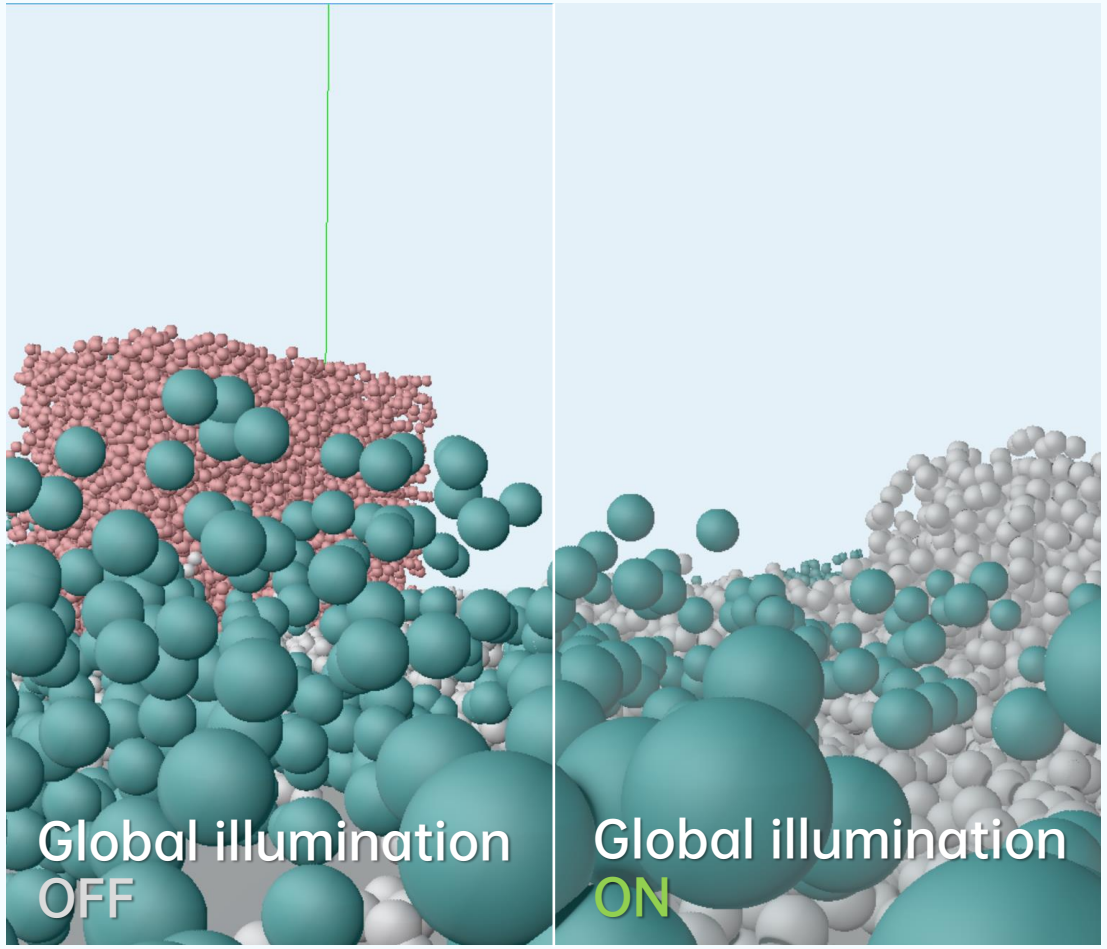


T = 1.5, **Vapor phase**

Visualization results

Rendering of TaichiMD systems





Code Example

Moving least squares material point method (MLS-MPM) readily implemented in TaichiMD:

1. MLS-MPM is based on the affine particle-in-cell (APIC) method. The MPMGrid class is defined to inherit the APIC grid class. The main difference between MLS-MPM and APIC is the calculation of the affine velocity field for each particle. Therefore, the `affine()` method is overridden for the MPMGrid class.

```
import taichi as ti
from taichimd import Simulation, MDRenderer, DIM
from taichimd.grid import APIC, QuadraticKernel
from taichimd.integrator import ForwardEulerIntegrator as FE
MDRenderer.radius = 0.2

class MPMGrid(APIC):
    @ti.func
    def affine(self, i):
        self.system.F[i] = (ti.Matrix.identity(ti.f32, DIM) \
        + self.system.dt * self.system.C[i]) @ self.system.F[i]
        ...
        ...
        ...
        stress = (-self.system.dt * self.vol * 4 \
             * self.inv_dx * self.inv_dx) * stress
        return stress + self.mass * self.system.C[i]
```

(see https://github.com/victoriacity/taichimd/blob/master/mpm48.py for full code)

Code Example

Moving least squares material point method (MLS-MPM) readily implemented in TaichiMD:

2. Initialize the program and set the materials parameters, then instantiates a Simulation object as the simulation system using forward Euler integration and the MPMGrid. Two additional attributes F (deformation gradient) and Jp (plastic deformation) are added to the existing attributes for the APIC method

```
ti.init(arch=ti.cuda, device_memory_GB=3)
dt = 2e-4
n_particles, n_grid = 32768, 64
p_vol, p_rho = (0.5 / n_grid)**2, 1
mpmgrid = MPMGrid(QuadraticKernel(), n_grid, mass=p_vol *\
          p_rho, gravity=50)
mpmgrid.vol = p_vol
E, nu = 0.1e4, 0.2 # Young's modulus and Poisson's ratio
mpmgrid.mu_0, mpmgrid.lambda_0 = E / (2 * (1 + nu)), \
        E * nu / ((1+nu) * (1 - 2 * nu)) # Lame parameters
sim = Simulation(n_particles, integrator=FE(dt, False),\
        grid=mpmgrid)
sim.add_attr("F", dims=(DIM, DIM), dtype=ti.f32)
sim.add_attr("Jp", dims=(), dtype=ti.f32)
sim.gui.set_colors([[6/255, 133/255, 135/255],\
   [237/255, 85/255, 89/255], [238/255, 238/255, 240/255]])
```

(see https://github.com/victoriacity/taichimd/blob/master/mpm48.py for full code)

Code Example

Moving least squares material point method (MLS-MPM) readily implemented in TaichiMD:

3. Invoke the `sim.build()` method to initialize the data structures for the simulation. Then the attribute values are initialized and `sim.run()` is invoked to start the simulation. `pause=True` indicates that the simulation will start after pressing the space button (and performs just-in-time compilation)

```
sim.init_random(center=(0.4, 0.15, 0.7), \
          length=0.2, start=0, end=n_particles//3, inittype=0)
sim.init_random(center=(0.4, 0.45, 0.7), \
          length=0.2, start=n_particles//3,
end=2*n_particles//3, inittype=1)
sim.init_random(center=(0.5, 0.75, 0.7), \
           length=0.2, start=2*n_particles//3, \
           end=n_particles, inittype=2)
sim.build()
sim.velocity.fill(0); sim.F.fill(((1,0,0),(0,1,0),(0,0,1)));
sim.C.fill(0); sim.Jp.fill(1)
sim.run(irender=10, pause=True)
```

(see https://github.com/victoriacity/taichimd/blob/master/mpm48.py for full code)

Future directions

- Machine-learned molecular simulation and intelligent agent simulations with reinforcement learning
- Acceleration methods for N-body dynamics (Fast multipole method, Ewald summation) and optimization-based integration
- Improving the realism and aesthetics of particle visualizations
- More interactive methods such as GUI or external potential fields

Acknowledgements

- Yuanming Hu (MIT CSAIL)
- Taichi forum https://forum.taichi.graphics/