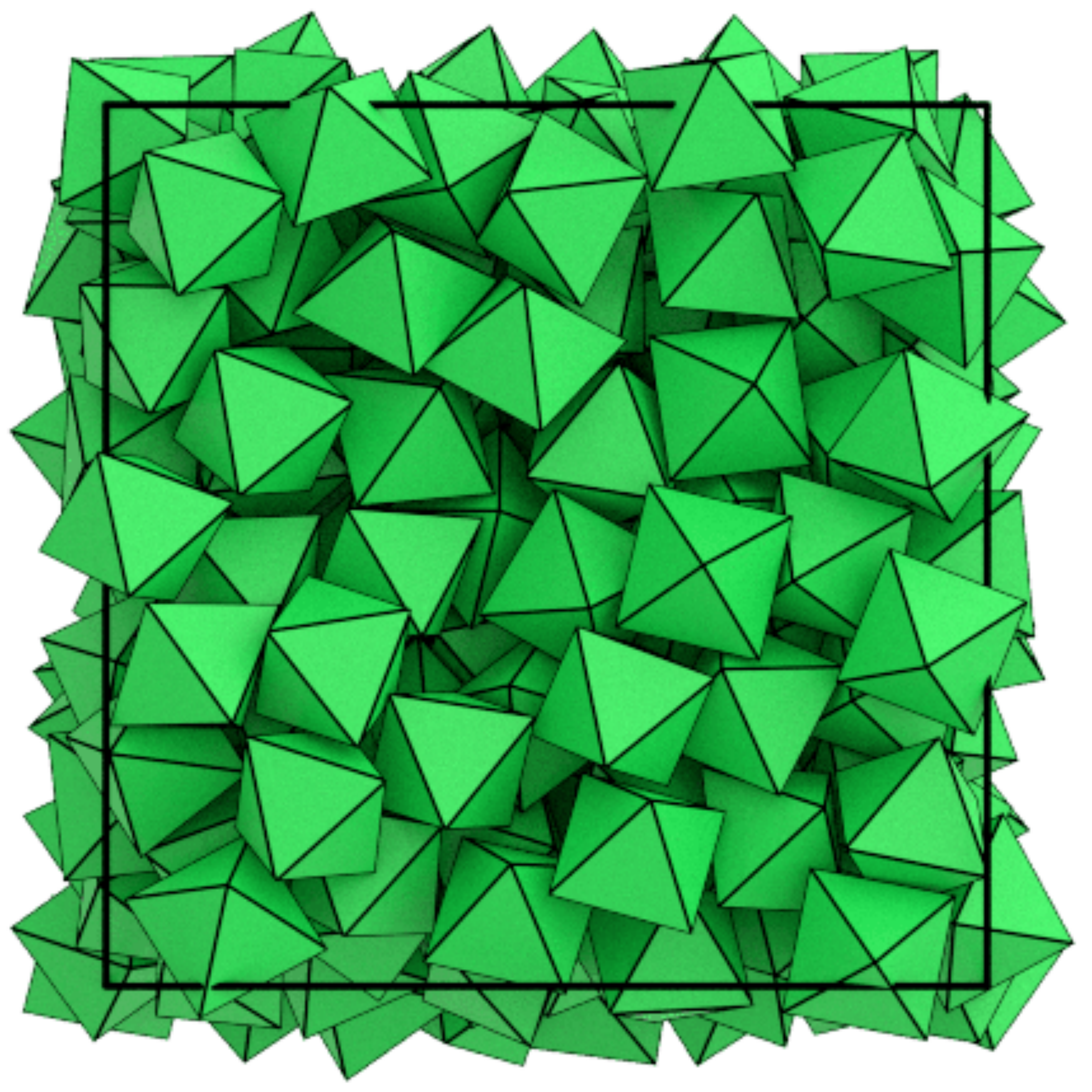
Self Assembly of Hockey stick

End-term course project presentation by
Aman Anand (14807)
For Molecular Simulation course (PH 322)
Taught by: Prof. Prabal Maity
& Prof. Anand Srivastava



https : //github .com/glotzerlab/hoomd − examples

Self Assembly of Hockey stick

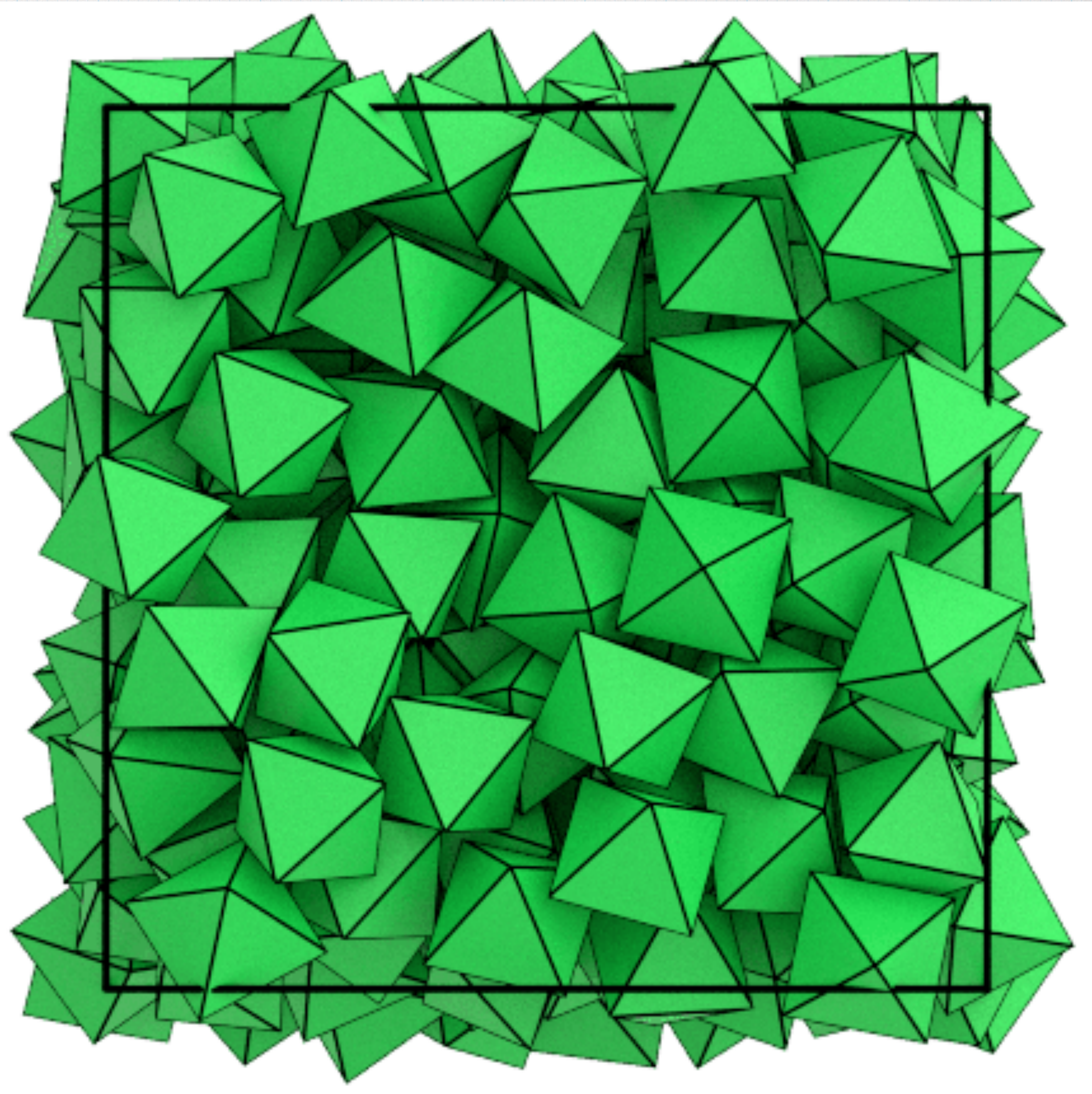
End-term course project presentation by

Aman Anand (14807)

For Molecular Simulation course (PH 322)

Taught by: Prof. Prabal Maity

& Prof. Anand Srivastava



https://github.com/glotzerlab/hoomd-examples

What is Self-Assembly?

* Self-assembly is the process of association of individual units of a material into highly arranged/ordered structures/patterns.
* $F = E - TS$ (Free energy)
* In physics we see energy-minimisation comes into play to order things at low temperatures and disorder at high-temperature.
* With no interaction and at high density, we observe the emergence of quasi-order. This happens to increase the number of available microstates.

* https://youtu.be/JW1L_vZ6K1M

Why do self assembly??

* The potential of 'bottom-up' manufacturing to improve the economics and performance of certain technologies.
* It draws from the enormous wealth of examples in biology for inspiration: self-assembly is one of the most important strategies used in biology for the development of complex, functional structures
* It tends to produce structures that are relatively defect-free and self-healing because it requires that the target structures be the thermodynamically most stable ones open to the system.

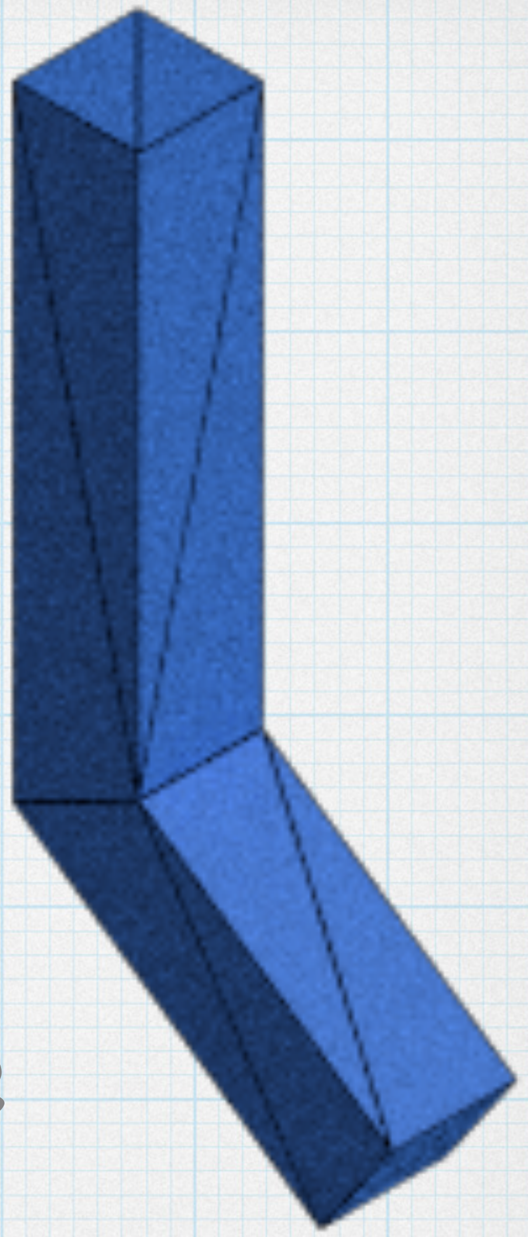
https://www.sciencedirect.com/science/article/pii/B9780323512565000046

Hoomd-blue

* Software developed by Glotzer group, UMich.
* It is a python package that run simulations of particle systems on CPUs and GPUs.
* It performs hard particle Monte Carlo simulation of a variety of shape classes and molecular dynamics simulations of particles with a range of pair, bond, angle, and other potentials.
* The group has also created other software which are used for compiling of hoomd files. Like **FREUD**, **FRESNEL** and **GSD**.
* **GSD** - **GSD** files store trajectories of the HOOMD-blue system state in a binary file with efficient random access to frames.
* **FREUD** - The **FREUD** Python library provides a simple, flexible, powerful set of tools for analyzing trajectories obtained from molecular dynamics or Monte Carlo simulations.
* **FRESNEL** - **FRESNEL** is a python library for path tracing publication quality images of soft matter simulations in real time.

https : //github . com/glotzerlab

FRESNEL : making hockey sticks



* The FRESNEL provides Geometry feature to build certain geometries: sphere, cylinder, convex polyhedra, mesh, polygon and box.
* The convex polyhedra option takes vertex coordinates and creates the convex hull of the vertices. Our hockey stick is not a convex polyhedron. So, the only option left is to make the figure out of triangular meshes.

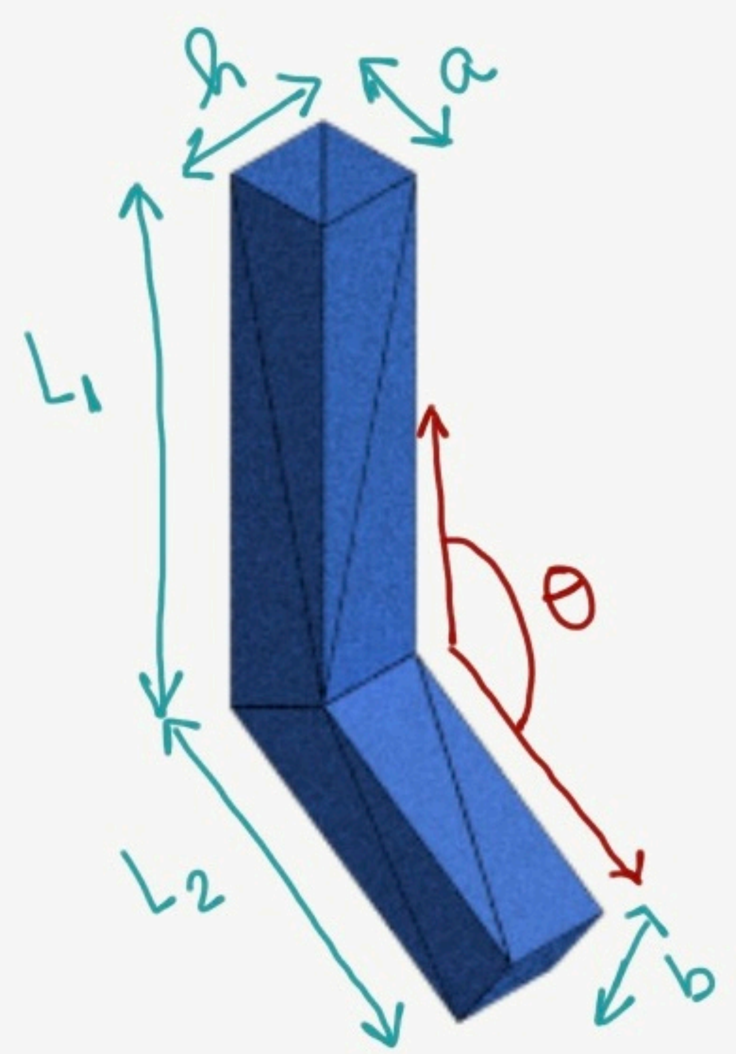
```
In [1]: import fresnel
        import numpy
```

```
In [2]: verts = numpy.load('hstick_mesh.npy')
        scene1 = fresnel.Scene()
        hstick = fresnel.geometry.Mesh(scene1,vertices=verts)
        hstick.material = fresnel.material.Material(color=fresnel.color.linear([0.25,0.5,0.9]), roughness=0.6)
        scene1.camera = fresnel.camera.Orthographic.fit(scene1,margin=0)
        scene1.lights = fresnel.light.cloudy()
        #scene1.background_alpha = 1.0
        #scene1.background_color = fresnel.color.linear([0, 0, 0])
        hstick.outline_width = 0.015
        fresnel.pathtrace(scene1, samples=200)
```

Creating a triangular mesh data file (.npy) for the hockey stick

In [2]:

```
import numpy as np
L1=5
L2=3
theta_degree=125
h=1
a=1
b=1
theta_radian = theta_degree*np.pi/180
cost = np.cos(theta_radian)
sint = np.sin(theta_radian)
#finding the internal edge lengths-l1,l2
l1 = L1 - b/sint - a*(cost/sint)
l2 = L2 - a/sint - b*(cost/sint)
#using simple trigo finding the vertices of the hockey stick and storing in v
v = ["null",[0,0,0],[0,0,h],[L2*sint,L2*cost,0],[L2*sint,L2*cost,h],[0,L1,0],[0,L1,h],
     [a+l2*sint,L1-l1+l2*cost,0],[a+l2*sint,L1-l1+l2*cost,h],[a,L1,0],[a,L1,h],[a,L1-l1,0],[a,L1-l1,h]]
#writing the 20triangle vertices number, like 1,4,2 are the vertices of first triangle and so on
x = [1,4,2,1,3,4,1,3,11,3,7,11,3,7,4,7,8,4,7,8,11,11,8,12,4,8,12,2,4,12,2,12,6,12,10,6,12,11,9,12,
     9,10,9,5,10,5,6,10,1,2,6,6,5,1,1,5,9,1,9,11]
#creating the mesh array which will store triangle vertices coordinates according to x
mesh = ["null"]*len(x)
j=0
for i in x:
    mesh[j]=v[i]
    j = j+1
#saving the mesh data as .npy file which we can load at our convinience
np.save('hstick_mesh.npy', mesh)
```



Initialising the state

In [18]:

```
import math
import hoomd
import os
import warnings
import fresnel
import IPython
import packaging.version

device = fresnel.Device()
tracer = fresnel.tracer.Path(device=device, w=300, h=300)

def render2(snapshot):

    L = snapshot.configuration.box[0]
    verts = numpy.load('hstick_mesh.npy')
    scene = fresnel.Scene(device)
    hstick = fresnel.geometry.Mesh(scene,vertices=verts, N=snapshot.particles.N)
    hstick.material = fresnel.material.Material(color=fresnel.color.linear([0.25
                      ,0.5,0.9]), roughness=0.6)

    hstick.position[:] = snapshot.particles.position[:]
    hstick.orientation[:] = snapshot.particles.orientation[:]
    hstick.outline_width = 0.01
    box = fresnel.geometry.Box(scene, [L, L, L, 0, 0, 0], box_radius=.02)

    scene.lights = [
        fresnel.light.Light(direction=(0, 0, 1),
                            color=(0.8, 0.8, 0.8),
                            theta=math.pi),
        fresnel.light.Light(direction=(1, 1, 1),
                            color=(1.1, 1.1, 1.1),
                            theta=math.pi / 3)
    ]
    scene.camera = fresnel.camera.Orthographic(position=(L * 2, L, L * 2),
                                               look_at=(0, 0, 0),
                                               up=(0, 1, 0),
                                               height=L * 1.4 + 1)
    scene.background_color = (1, 1, 1)
    return IPython.display.Image(tracer.sample(scene, samples=500)._repr_png_())
```

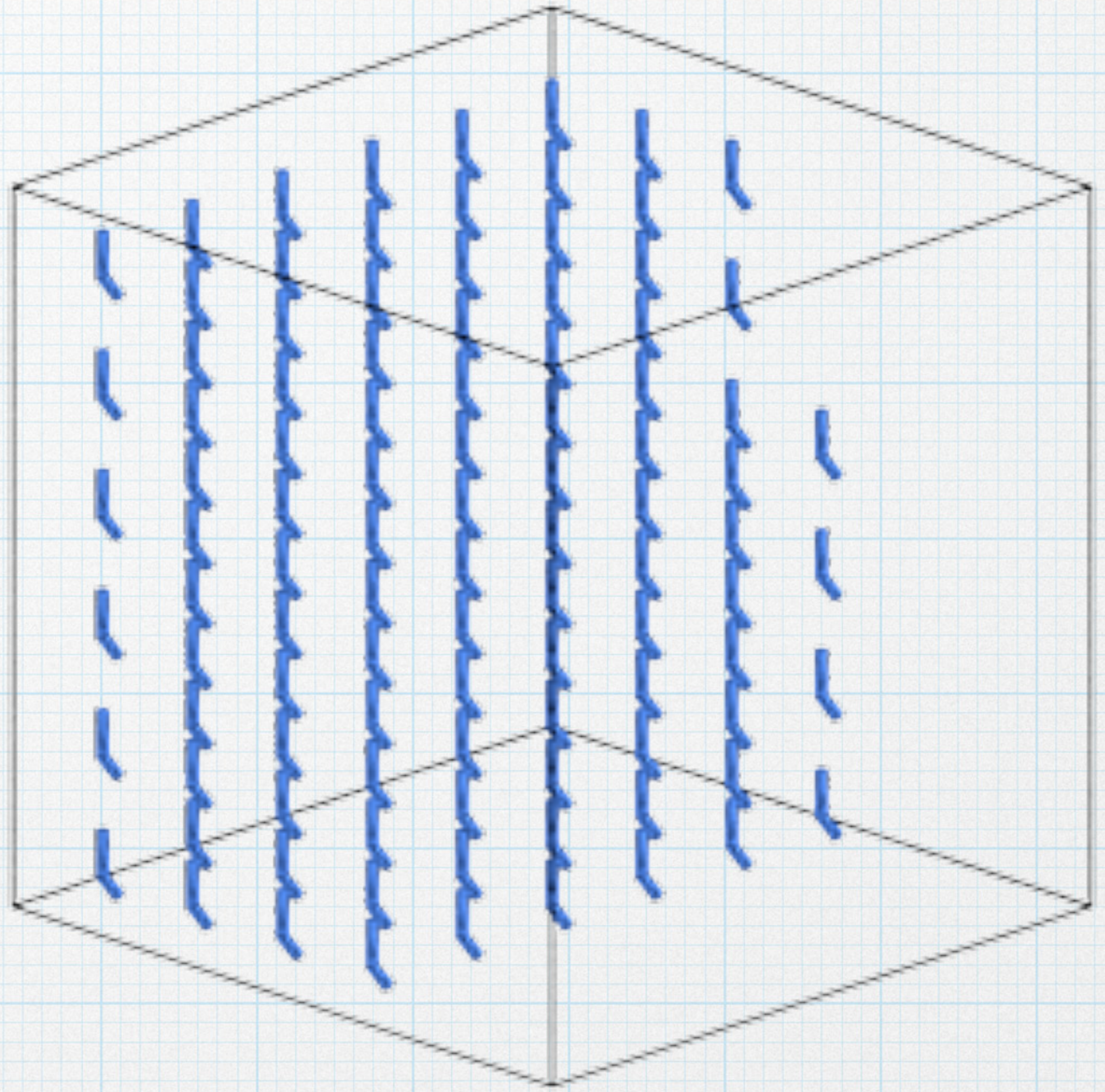
Initialising the state

```
In [6]: import gsd.hoomd
        import os

        fn = os.path.join(os.getcwd(), 'lattice.gsd')
        ![ -e "$fn" ] && rm "$fn"

        snapshot = gsd.hoomd.Snapshot()
        snapshot.particles.N = N_particles
        snapshot.particles.position = position
        snapshot.particles.orientation = orientation

        snapshot.particles.typeid = [0] * N_particles
        snapshot.particles.types = ['hstick']
        snapshot.configuration.box = [L, L, L, 0, 0, 0]
        with gsd.hoomd.open(name='lattice.gsd', mode='xb') as f:
            f.append(snapshot)
```

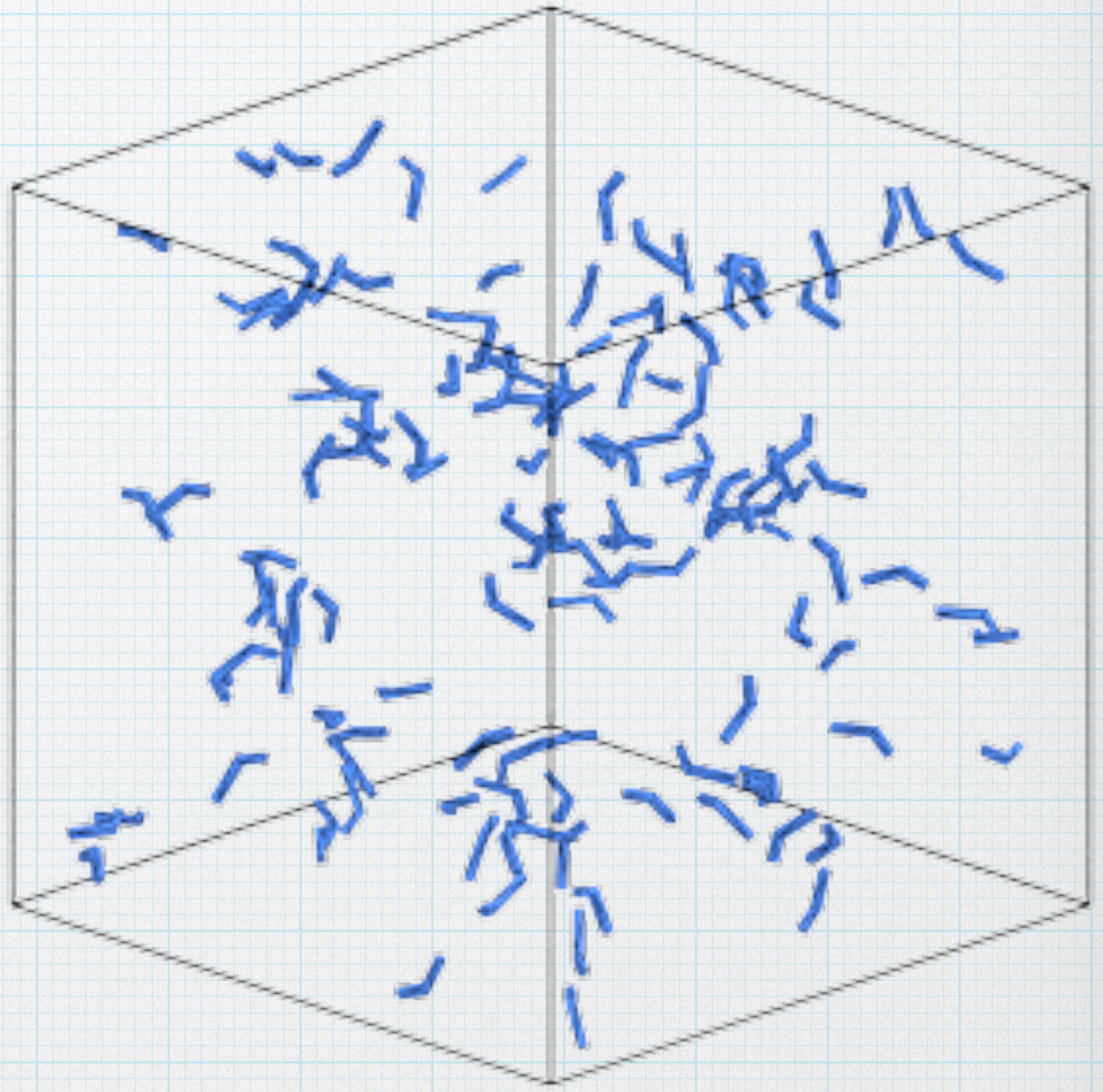


```
In [*]: m = 4
        N_particles = 2* m**3
        spacing = 1.2
        K = math.ceil(N_particles**(1 / 3))
        L = K * spacing
        x = numpy.linspace(-L / 2, L / 2, K, endpoint=False)
        position = list(itertools.product(x, repeat=3))
        position = position[0:N_particles]
        orientation = [(1, 0, 0, 0)] * N_particles
```

Randomising the initialised state

In [8]:
```
import gsd.hoomd
cpu = hoomd.device.CPU()
sim = hoomd.Simulation(device=cpu, seed=2)
mc = hoomd.hpmc.integrate.Polyhedron(0.15,0.2,0.4,2)
vertices1 = numpy.load('hstick_vertices.npy')
faces1=numpy.load('hstick_faces.npy')
mc.shape["hstick"] = dict(vertices=vertices1 ,faces=faces1)
sim.operations.integrator = mc
sim.create_state_from_gsd(filename='lattice.gsd')
```

In [20]:
```
initial_snapshot = sim.state.get_snapshot()
sim.run(10e3)
final_snapshot = sim.state.get_snapshot()
```



Compressing the random state

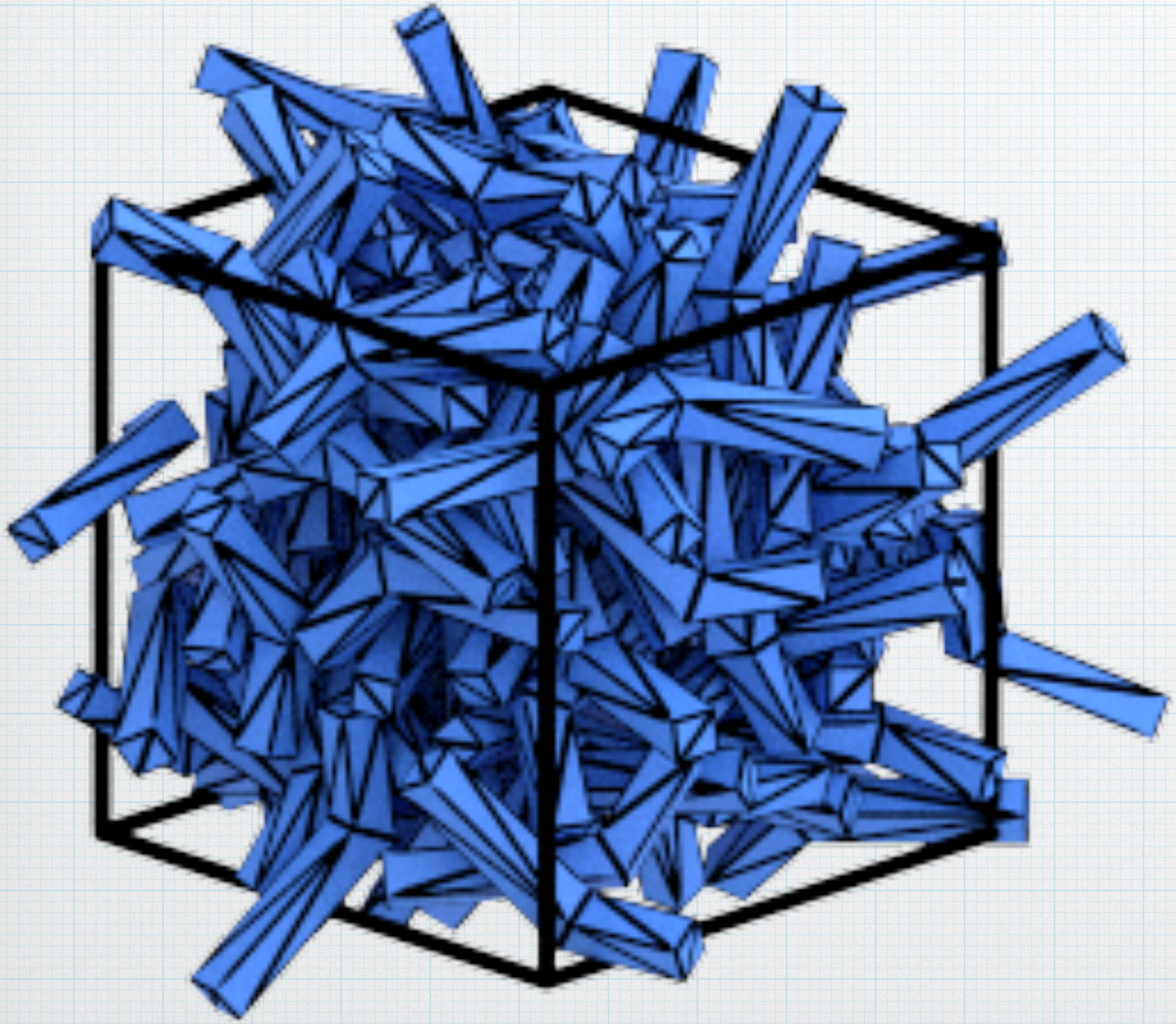
In [37]:
```
initial_box = sim.state.box
final_box = hoomd.Box.from_box(initial_box)
final_packing_fraction = 0.57
final_box.volume = sim.state.N_particles * V_particle / final_packing_fraction
compress = hoomd.hpmc.update.QuickCompress(trigger=hoomd.trigger.Periodic(10),
                                           target_box=final_box)
```

In [38]:
```
sim.operations.updaters.append(compress)
```

In [39]:
```
periodic = hoomd.trigger.Periodic(10)
tune = hoomd.hpmc.tune.MoveSize.scale_solver(moves=['a', 'd'],
                                             target=0.2,
                                             trigger=periodic,
                                             max_translation_move=0.2,
                                             max_rotation_move=0.2)
sim.operations.tuners.append(tune)
```

In [*]:
```
while not compress.complete and sim.timestep < 1e6:
    sim.run(1000)
```

Equilibrating the compressed state




1
2
3

Future directions

* Test with the standard results. I change just the mesh file so that the mesh now gives a tetrahedron.
* Look at the self-assembly for different edge-lengths and angles.
* Will the system equilibrate? If yes, then at what critical densities do we get ordered structures?
* Check the self-assembly for experimentally possible parameters.
* Effect of smoothening of hockey stick on self-assembly.