MECE 4510: Evolutionary Computation and Design

Project - Phase A: Physics Simulator

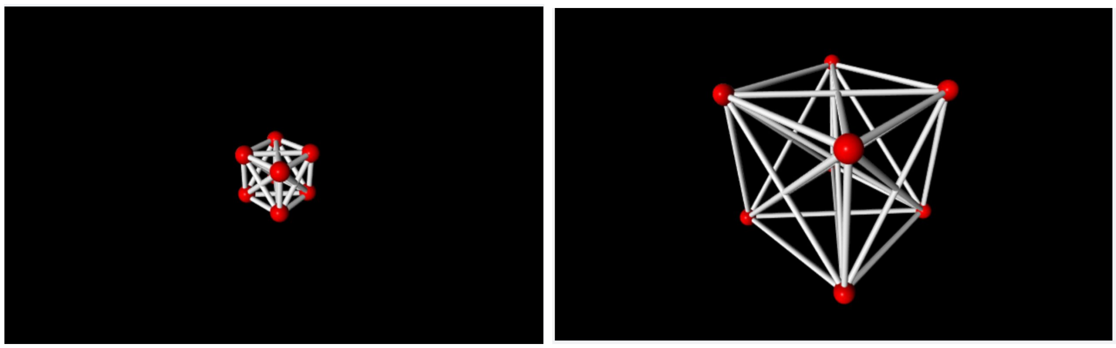
Jerry Zhang | UNI: jz2966 | Grace Hours Remaining: 33 + 29
Zhengyang Du | UNI: zd2219 | Grace Hours Remaining: 28 + 29

Instructor: Professor Hod Lipson
Date Submitted: 11/01/2018

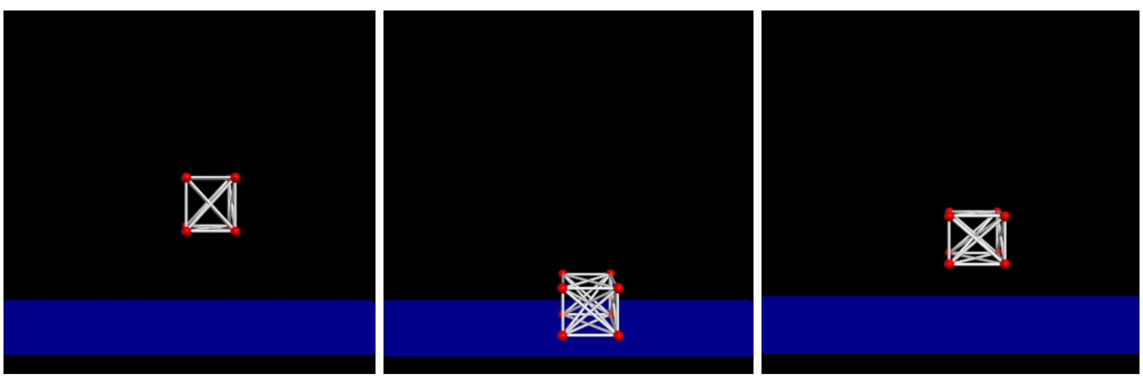
RESULTS SUMMARY

FULL VIDEO LINK: https://www.youtube.com/watch?v=nNQc2xbe9XU

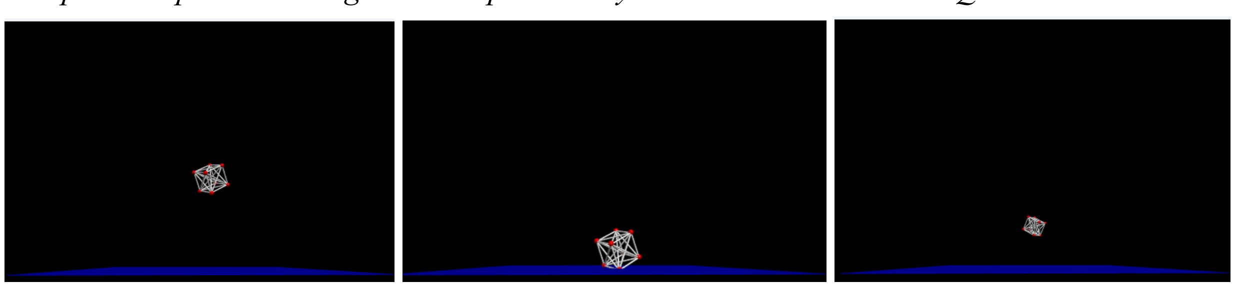
Breathing Cube: https://www.youtube.com/watch?v=nNQc2xbe9XU



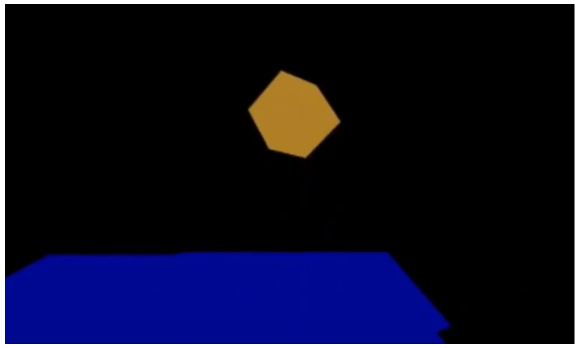
Bouncing Cube: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=0m14s
Damped Bouncing Cube : https://www.youtube.com/watch?v=nNQc2xbe9XU#t=0m27s



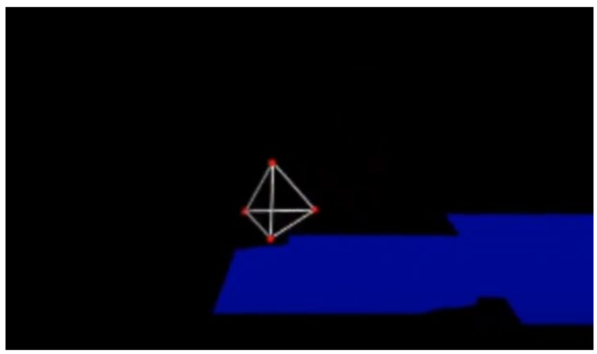
Complex Bouncing Cube: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=0m42s
Damped Complex Bouncing Cube: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=0m59s



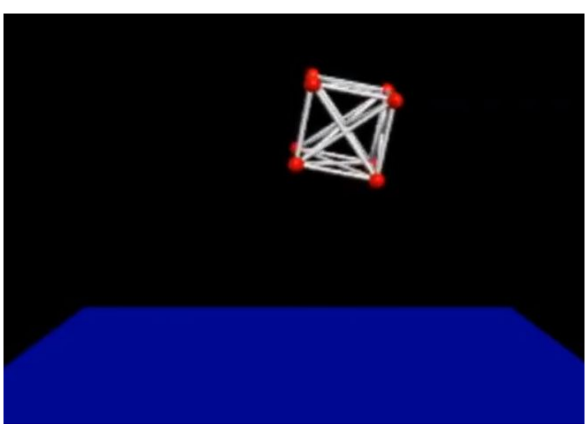
Triangles Shaded: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=1m17s

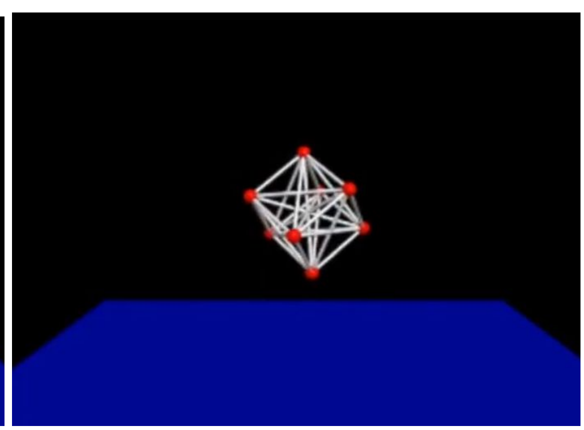


Tetrahedron Debug Test: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=1m36s



Grounded Node: https://www.youtube.com/watch?v=nNQc2xbe9XU#t=1m54s





METHODS

For phase A of this assignment, a cube was created in vpython with 8 balls (masses) and 28 springs to connect each individual mass. The cube is able to bounce once after dropping from any distance from the floor. Friction was added to the simulation and 8 cases have were simulated: the breathing cube, bouncing cube without a slight spin (damped + undamped), bouncing cube with a slight spin (damped + undamped), bouncing cube w/ triangles shaded, tetrahedron debugging test, and grounded node case. These cases could be evaluated both with damping and without dampening. The mass of each ball was 0.1 kg, or 0.8 kg for the entire cube. The gravity was specified to be 9.81 m/s^2, the spring constant for the springs in the cube was adjusted to 2000 N/m and the spring constant for the ground (rubber) was set to 10000 N/m. The dt set to 0.004 seconds to best fit the cube movement.

To simulate the various cases, force vectors that acted on all of the balls had to first be calculated. Each ball mass would experience eight forces from the other masses, (including itself, which was set 0) that came from the springs that it connected to. Subsequently, the force vectors were stored in an 8 x 8 matrix. Each time after the matrix updated, the force vectors in each column would be summed into one vector: the combined force vector that acting one each ball. Then once the force vectors were obtained, gravity was added into all force vectors. Using the force vectors, the acceleration vectors were calculated and updated into velocity vectors using kinematic equation vf = vi+a*t and then by using the new velocity vectors, the position of the balls were updated in relation with the forces. Once the ball positions were updated, the simulated spring positions updated with respect to the new position of the balls. If the ball position was below the floor position, a force vector from the floor (F = kc*delta(ball position - floor position)) would react on the ball force vectors to make the cube bounce back. Therefore, using a while loop allowed the process to keep updating for multiple bounces.

For the breathing cube, the cube would just stay at its original place without adding neither gravity nor floor. The rest length L0 of the spring would be change in an order of a sine wave with respect to time. It was achieved by multiplying the rest length L0 by a breathing factor 1 + 0.5*sin(5*t), where t =0.01 and it would be updated += 0.01 each time in the cycle of the while loop, producing the breathing effect.

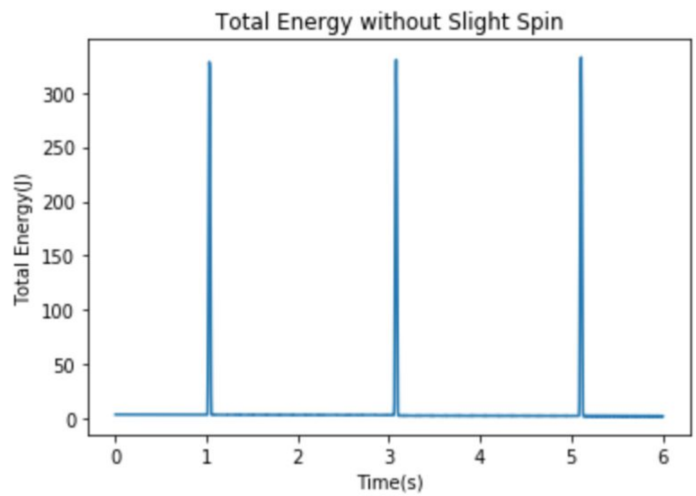
To add a damping effect to the cube, a dampening of 0.9 was multiplied by the velocity updates at the end of the loop and the damping would be updated each time so in the end the velocity would decay to zero.

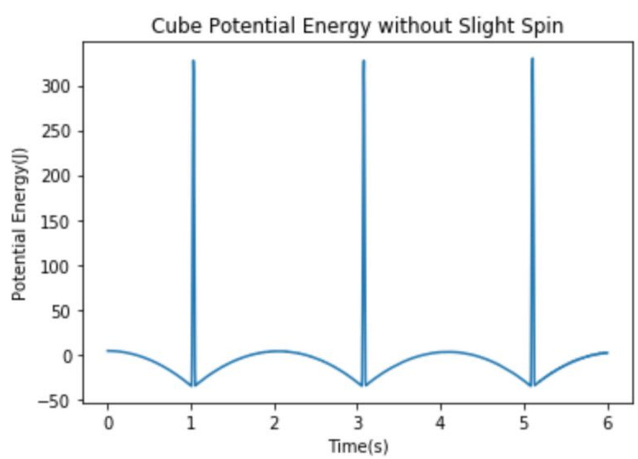
The energy calculations for the total energy (all of cube energy and ground energy), potential energy (gravity, spring, ground), kinetic energy (mass velocity) were obtained by using K_g = m*g*h, K_k = ½*m*v^2 and K_s = ½*k*delta(L)^2 and K_g = ½*kc*delta(y)^2. After calculating the total energy (cube and ground), we can see in the plot below that the total cube energy stayed constant until it hit the ground and the energy transformation appeared in the plot which is a pulse that the ground energy added in. The potential energy and the kinetic energy plot were also reasonable that the changes of the energy were corresponding to the cube motion.

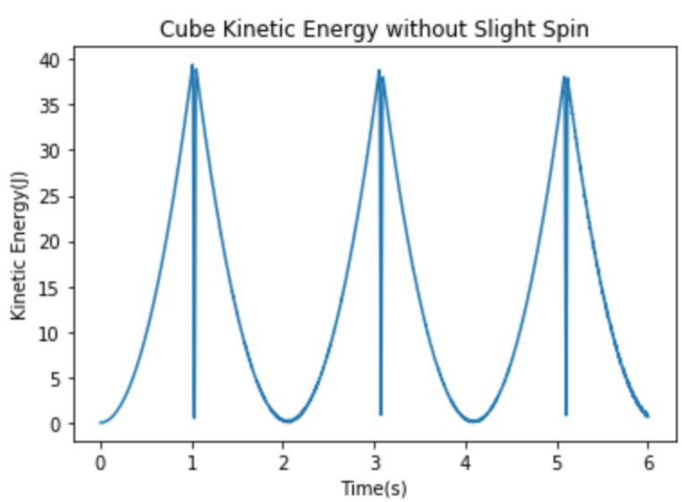
The triangle shading case was created by simulating triangles at every combination of 3 mass positions. The tetrahedron debugging test utilized 4 masses and 6 springs, which simplified the code, and proved uniformity. The grounded node case was created by only updating positions of 7 out of 8 masses and keeping one mass fixed.

PERFORMANCE PLOTS (ENERGY CURVES

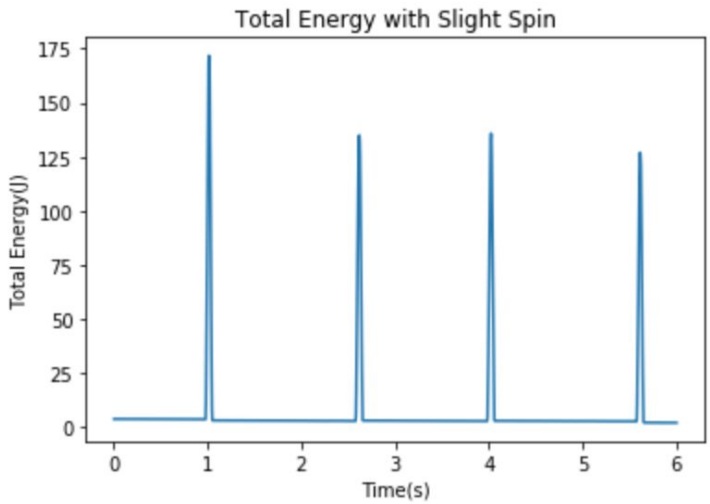
Bouncing Cube (no spin):

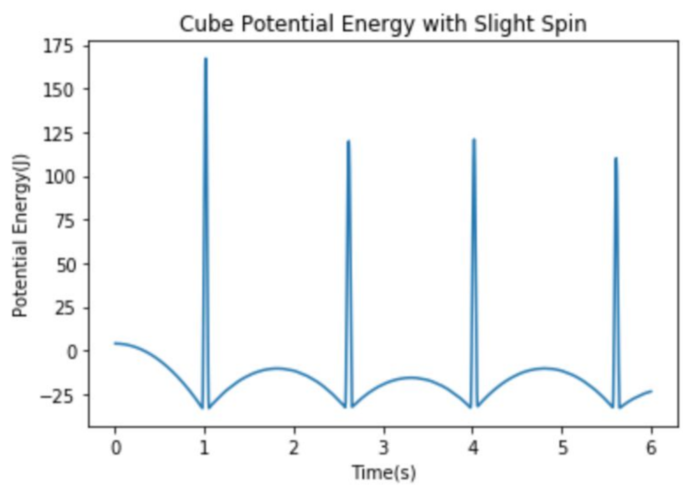


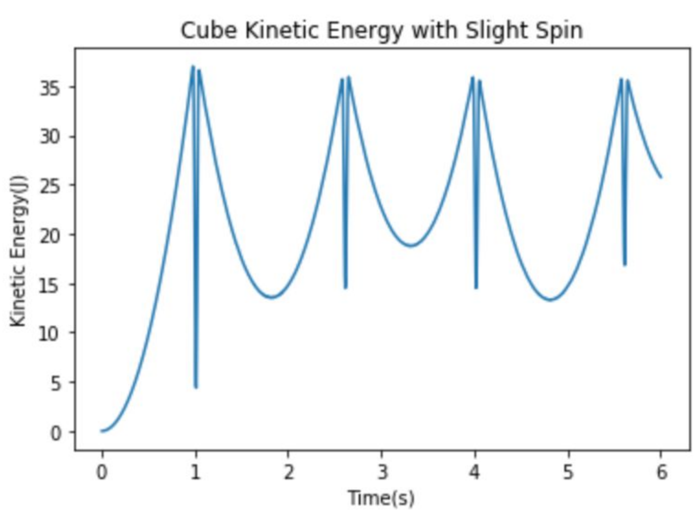




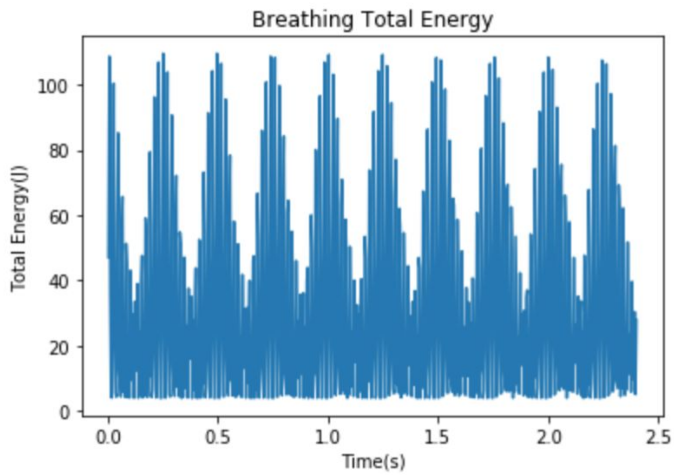
Complex Bounce:



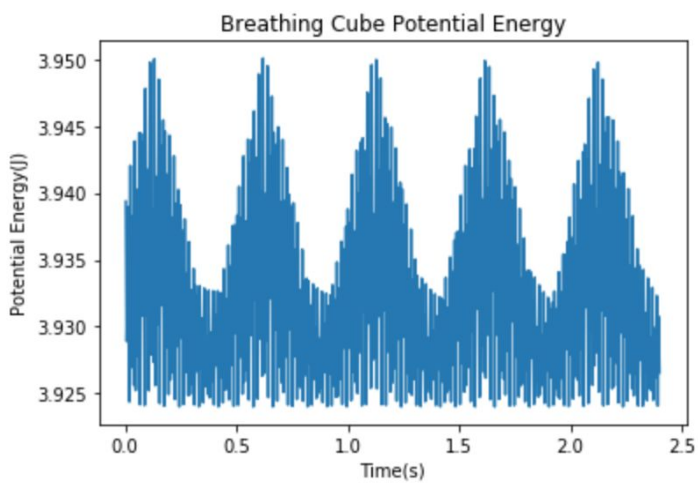




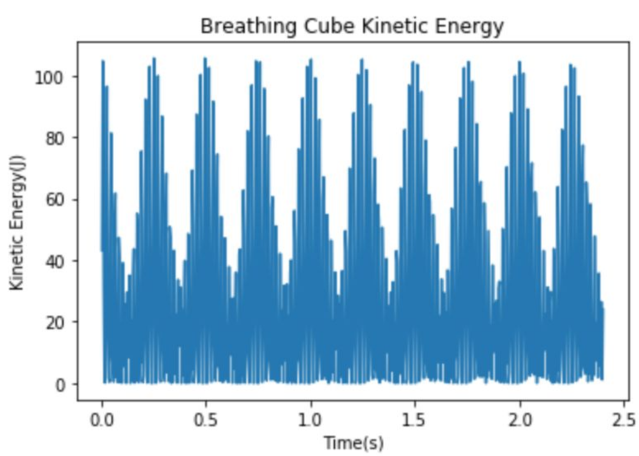
Breathing Cube:



Breathing Total Energy
Total Energy(J)
Time(s)
0
20
40
60
80
100
0.0
0.5
1.0
1.5
2.0
2.5



Breathing Cube Potential Energy
Potential Energy(J)
Time(s)
3.925
3.930
3.935
3.940
3.945
3.950
0.0
0.5
1.0
1.5
2.0
2.5



Breathing Cube Kinetic Energy
Kinetic Energy(J)
Time(s)
0
20
40
60
80
100
0.0
0.5
1.0
1.5
2.0
2.5

APPENDIX

BREATHING CUBE

```
import vpython as vp
import itertools
import numpy as np
from math import *
import matplotlib.pyplot as plt

scene = vp.canvas()
# vp.display(width=100, height=100)
# floor = vp.box(pos=vp.vector(0,-3,0), length=10, height=0.001, width=10, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0),
vp.vector(1, 1, 0),
              vp.vector(0, 1, 1),
              vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

# for i in range(8):
#     ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,1,1))


springvecs = []
for i in range(len(ballname)):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red,
velocity=vp.vector(0, 0, 0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)

spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13',
's14', 's15', 's16', 's17',
         's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']

for i in range(28):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03,
                           color=vp.color.white)

v = 0
dt = 0.003
mass = 0.1
g = 9.81
# k_sp = []
k1 = 2000


g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0, 0, 0)

F_c = vp.vector(0, 1000, 0)

L0 = np.zeros((8, 8))
```

```
for i in range(8):
   for j in range(8):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)

# F = np.zeros((8,3))
t = 0.01
t1 = 0
Time = []
E_H = []
E_S = []
E_K = []
while 1:
   vp.rate(100)
   k_sp = 0.5 * sin(5 * t) + 1
   t += 0.01

   for i in range(8):
       ballvectors[i] = ballname[i].pos
   springvecs = []
   for z in itertools.combinations(ballvectors, 2):
       springvecs.append(z)
   for i in range(28):
       position = springvecs[i][1] - springvecs[i][0]  # - L0[i]
       spring[i].pos = springvecs[i][0]
       spring[i].axis = position
       spring[i].length = vp.mag(position)

   dampening = 0.9

   F_mat = np.zeros((8, 8))
   F_vec = []
   F_v = []
   a = np.array(np.zeros((8, 8)))
   E_s = []
   for i in range(8):
       for k in range(8):
           if k == i:
               L = 0
               F_mat[i][k] = 0
               F_vec.append(vp.vector(0, 0, 0))
           else:
               L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i] * k_sp
               E_s.append(1 / 2 * k_sp * L ** 2)
               F_mat[i][k] = L * k1
               pf0 = ballname[k].pos - ballname[i].pos
               # a[i,k] = vp.norm(pf0)*L*k_sp
               F_vec.append(vp.norm(pf0) * L * k1)
   E_S.append(sum(E_s) / 2)
   a = np.array(F_vec).reshape(8, 8)
   F = a.sum(axis=0)
#     for i in range(8):
#         F[i] = F[i] + mass*g_vector
#         if ballname[i].pos.y < floor.pos.y:
```

```
    #              F[i].y = F[i].y - dampening*((floor.pos.y-ballname[i].pos.y)**2)*10000
    E_k = []
    E_h = []
    for i in range(8):
        ballname[i].velocity -= F[i] / mass * dt
        E_k.append(1 / 2 * mass * (vp.mag(ballname[i].velocity)) ** 2)
        ballname[i].pos += ballname[i].velocity * dt
        E_h.append(mass * 9.81 * (ballname[i].pos.y))
    E_K.append(sum(E_k))
    E_H.append(sum(E_h))

    t1 += 0.004
    Time.append(t1)
    if t > 6:
        break

E_total = []
E_potential = []
for i in range(len(E_H)):
    E_total.append(E_K[i] + E_H[i] + E_S[i])
for i in range(len(E_H)):
    E_potential.append(E_H[i] + E_S[i])

plt.plot(Time, E_total)
plt.xlabel('Time(s)')
plt.ylabel('Total Energy(J)')
plt.title('Breathing Total Energy')
plt.show()

plt.plot(Time, E_potential)
plt.xlabel('Time(s)')
plt.ylabel('Potential Energy(J)')
plt.title('Breathing Cube Potential Energy')
plt.show()

plt.plot(Time, E_K)
plt.xlabel('Time(s)')
plt.ylabel('Kinetic Energy(J)')
plt.title('Breathing Cube Kinetic Energy')
plt.show()

#          if ballname[i].pos.y < floor.pos.y:
#              ballname[i].velocity = dampening*-1*(ballname[i].velocity)+ F_c*(floor.pos.y - ballname[i].pos.y)*(floor.pos.y - ballname[i].pos.y)
```

BOUNCING CUBE WITHOUT A SLIGHT SPIN

```python
import vpython as vp
import itertools
import numpy as np
from math import *
import matplotlib.pyplot as plt

scene = vp.canvas()
# vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0, -5, 0), length=10, height=0.001, width=10, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0),
vp.vector(1, 1, 0),
              vp.vector(0, 1, 1),
              vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

# for i in range(8):
#     ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,1,1))


springvecs = []
for i in range(len(ballname)):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red,
velocity=vp.vector(0, 0, 0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)

spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13',
's14', 's15', 's16', 's17',
         's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']

for i in range(28):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03,
                           color=vp.color.white)

v = 0
dt = 0.004
mass = 0.1
g = 9.81
k_sp = 2000
g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0, 0, 0)

F_c = vp.vector(0, 10000, 0)

L0 = np.zeros((8, 8))
for i in range(8):
   for j in range(8):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)
```

```python
Time = []
t = 0
E_K = []
E_G = []
E_H = []
E_S = []
while 1:
   vp.rate(100)

   for i in range(8):
       ballvectors[i] = ballname[i].pos
   springvecs = []
   for z in itertools.combinations(ballvectors, 2):
       springvecs.append(z)

   for i in range(28):
       position = springvecs[i][1] - springvecs[i][0]  # - L0[i]
       spring[i].pos = springvecs[i][0]
       spring[i].axis = position
       spring[i].length = vp.mag(position)

   dampening = 1  # adjust damping here

   F_mat = np.zeros((8, 8))
   F_vec = []
   F_v = []
   a = np.array(np.zeros((8, 8)))
   E_s = []
   for i in range(8):
       for k in range(8):
           if k == i:
               L = 0
               F_mat[i][k] = 0
               F_vec.append(vp.vector(0, 0, 0))
           else:
               L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i]
               E_s.append(1 / 2 * k_sp * L ** 2)
               pf0 = ballname[k].pos - ballname[i].pos
               # a[i,k] = vp.norm(pf0)*L*k_sp
               F_vec.append(vp.norm(pf0) * L * k_sp)
   E_S.append(sum(E_s) / 2)
   a = np.array(F_vec).reshape(8, 8)
   F = a.sum(axis=0)
   E_g = []

   for i in range(8):
       F[i] = F[i] + g_vector * mass
       if ballname[i].pos.y < floor.pos.y:
           F_N = ((floor.pos.y - ballname[i].pos.y) ** 2) * 10000
           F[i].y = F[i].y - F_N
           E_g.append(1 / 2 * F_N)
           mu = 1
           F_st = mu * F_N
           F_horiz = (F[i].x * 2 + F[i].z * 2) ** 0.5
           v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
           vx = ballname[i].velocity.x / v_xz
           vz = ballname[i].velocity.z / v_xz
```

```
            if F_st < F_horiz:
                F[i].x += F_horiz * vx - F_N * vx
                F[i].z += F_horiz * vz - F_N * vz
            else:
                F[i].x = F_horiz * vx
                F[i].z = F_horiz * vz
                ballname[i].velocity.x = 0
                ballname[i].velocity.z = 0
    E_G.append(sum(E_g))

    E_k = []
    E_h = []
    for i in range(8):
        ballname[i].velocity -= (F[i] / mass * dt) * dampening
        E_k.append(1 / 2 * mass * (vp.mag(ballname[i].velocity)) ** 2)
        ballname[i].pos += ballname[i].velocity * dt
        E_h.append(mass * 9.81 * (ballname[i].pos.y))
    E_H.append(sum(E_h))
    E_K.append(sum(E_k))
    t += 0.004
    Time.append(t)
    if t > 6:
        break


E_total = []
E_potential = []
for i in range(len(E_H)):
    E_total.append(E_K[i] + E_H[i] + E_S[i] + E_G[i])
for i in range(len(E_H)):
    E_potential.append(E_G[i] + E_H[i] + E_S[i])

plt.plot(Time, E_total)
plt.xlabel('Time(s)')
plt.ylabel('Total Energy(J)')
plt.title('Total Energy without Slight Spin')
plt.show()

plt.plot(Time, E_potential)
plt.xlabel('Time(s)')
plt.ylabel('Potential Energy(J)')
plt.title('Cube Potential Energy without Slight Spin')
plt.show()

plt.plot(Time, E_K)
plt.xlabel('Time(s)')
plt.ylabel('Kinetic Energy(J)')
plt.title('Cube Kinetic Energy without Slight Spin')
plt.show()
```

BOUNCING CUBE WITH A SLIGHT SPIN CODE:

```python
import vpython as vp
import itertools
import numpy as np
from math import *
import matplotlib.pyplot as plt

scene = vp.canvas()
# vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0, -5, 0), length=10, height=0.001, width=10, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1), vp.vector(1, 0, 0),
vp.vector(1, 1, 0),
              vp.vector(0, 1, 1),
              vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

for i in range(8):
   ballvectors[i] = ballvectors[i].rotate(angle=3.14 / 4, axis=vp.vector(1, 1, 1))

springvecs = []
for i in range(len(ballname)):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red,
velocity=vp.vector(0, 0, 0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)

spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13',
's14', 's15', 's16', 's17',
         's18', 's19', 's20', 's21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']

for i in range(28):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03,
                           color=vp.color.white)

v = 0
dt = 0.004
mass = 0.1
g = 9.81
k_sp = 2000
g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0, 0, 0)

F_c = vp.vector(0, 10000, 0)

L0 = np.zeros((8, 8))
for i in range(8):
   for j in range(8):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)

Time = []
```

```
t = 0
E_K = []
E_G = []
E_H = []
E_S = []
while 1:
   vp.rate(100)

   for i in range(8):
       ballvectors[i] = ballname[i].pos
   springvecs = []
   for z in itertools.combinations(ballvectors, 2):
       springvecs.append(z)

   for i in range(28):
       position = springvecs[i][1] - springvecs[i][0]  # - L0[i]
       spring[i].pos = springvecs[i][0]
       spring[i].axis = position
       spring[i].length = vp.mag(position)

   dampening = 1  # adjust damping here

   F_mat = np.zeros((8, 8))
   F_vec = []
   F_v = []
   a = np.array(np.zeros((8, 8)))
   E_s = []
   for i in range(8):
       for k in range(8):
           if k == i:
               L = 0
               F_mat[i][k] = 0
               F_vec.append(vp.vector(0, 0, 0))
           else:
               L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i]
               E_s.append(1 / 2 * k_sp * L ** 2)
               pf0 = ballname[k].pos - ballname[i].pos
               # a[i,k] = vp.norm(pf0)*L*k_sp
               F_vec.append(vp.norm(pf0) * L * k_sp)
   E_S.append(sum(E_s) / 2)
   a = np.array(F_vec).reshape(8, 8)
   F = a.sum(axis=0)
   E_g = []

   for i in range(8):
       F[i] = F[i] + g_vector * mass
       if ballname[i].pos.y < floor.pos.y:
           F_N = ((floor.pos.y - ballname[i].pos.y) ** 2) * 10000
           F[i].y = F[i].y - F_N
           E_g.append(1 / 2 * F_N)
           mu = 1
           F_st = mu * F_N
           F_horiz = (F[i].x * 2 + F[i].z * 2) ** 0.5
           v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
           vx = ballname[i].velocity.x / v_xz
           vz = ballname[i].velocity.z / v_xz
           if F_st < F_horiz:
```

```
                F[i].x += F_horiz * vx - F_N * vx
                F[i].z += F_horiz * vz - F_N * vz
            else:
                F[i].x = F_horiz * vx
                F[i].z = F_horiz * vz
                ballname[i].velocity.x = 0
                ballname[i].velocity.z = 0
    E_G.append(sum(E_g))

    E_k = []
    E_h = []
    for i in range(8):
        ballname[i].velocity -= (F[i] / mass * dt) * dampening
        E_k.append(1 / 2 * mass * (vp.mag(ballname[i].velocity)) ** 2)
        ballname[i].pos += ballname[i].velocity * dt
        E_h.append(mass * 9.81 * (ballname[i].pos.y))
    E_H.append(sum(E_h))
    E_K.append(sum(E_k))
    t += 0.004
    Time.append(t)
    if t > 6:
        break

#     for i in range(8):
#         F[i] = F[i] + mass*g_vector
#         if ballname[i].pos.y < floor.pos.y:
#             F[i].y = dampening*(F[i].y - ((floor.pos.y-ballname[i].pos.y)**2)*10000)
#             E_g.append(1/2*10000*(floor.pos.y-ballname[i].pos.y)**2)
#     E_G.append(sum(E_g))

#     E_k = []
#     E_h = []
#     for i in range(8):
#         ballname[i].velocity -= F[i]/mass*dt
#         E_k.append(1/2*mass*(vp.mag(ballname[i].velocity))**2)
#         ballname[i].pos += ballname[i].velocity*dt
#         E_h.append(mass*9.81*(ballname[i].pos.y))
#         if ballname[i].pos.y < floor.pos.y:
#             ballname[i].velocity = dampening*((ballname[i].velocity)+ F_c*(floor.pos.y -
ballname[i].pos.y)*(floor.pos.y - ballname[i].pos.y))
#     E_H.append(sum(E_h))
#     E_K.append(sum(E_k))
#     t += 0.004
#     Time.append(t)
#     if t > 6:
#         break


E_total = []
E_potential = []
for i in range(len(E_H)):
    E_total.append(E_K[i] + E_H[i] + E_S[i] + E_G[i])
for i in range(len(E_H)):
    E_potential.append(E_G[i] + E_H[i] + E_S[i])

plt.plot(Time, E_total)
plt.xlabel('Time(s)')
```

```
plt.ylabel('Total Energy(J)')
plt.title('Total Energy with Slight Spin')
plt.show()

plt.plot(Time, E_potential)
plt.xlabel('Time(s)')
plt.ylabel('Potential Energy(J)')
plt.title('Cube Potential Energy with Slight Spin')
plt.show()

plt.plot(Time, E_K)
plt.xlabel('Time(s)')
plt.ylabel('Kinetic Energy(J)')
plt.title('Cube Kinetic Energy with Slight Spin')
plt.show()
```

TRIANGLES SHADED CODE:

```
import vpython as vp
import itertools
import numpy as np

scene = vp.canvas(title = 'Box', width = 600, height = 400, center = vp.vector(0,0,0))
floor = vp.box(pos=vp.vector(0,-3,0), length=8, height=0.01, width=8, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1),  vp.vector(1, 0, 0),
vp.vector(1, 1, 0), vp.vector(0, 1, 1),
              vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

for i in range(8):
   ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,0,1))
triangles = []
for z in itertools.combinations(ballvectors, 3):
   triangles.append(z)

T = list(range(56))
for i in range(len(triangles)):
   T[i] = vp.triangle(v0= vp.vertex(pos=triangles[i][0]), v1=vp.vertex(pos=triangles[i][1] ),
v2=vp.vertex(pos=triangles[i][2]), texture = "https://i.imgur.com/eQueRtf.jpg")
springvecs = []
for i in range(len(ballname)):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.001, color=vp.color.red, f_k=
vp.vector(0,0,0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)


spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10','s11', 's12', 's13', 's14',
's15', 's16', 's17', 's18', 's19', 's20','s21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']
for i in range(28):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.001, color=vp.color.white)

v = 0
dt = 0.003
mass = 0.1
g = 9.81
k_sp = 1000
g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0,0,0)

F_c = vp.vector(0, 1000, 0)

L0= np.zeros((8,8))
for i in range(8):
   for j in range(8):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)
```

```
while 1:
    vp.rate(200)
    floor = vp.box(pos=vp.vector(ballvectors[1].x, -3, ballvectors[1].z), length=5, height=0.01,
width=5, color=vp.color.blue)
    scene.center.x = ballvectors[1].x
    scene.center.z = ballvectors[1].z

    for i in range(8):
        ballvectors[i] = ballname[i].pos
    springvecs = []
    for z in itertools.combinations(ballvectors, 2):
        springvecs.append(z)
    for i in range(28):
        position = springvecs[i][1] - springvecs[i][0] #- L0[i]
        spring[i].pos = springvecs[i][0]
        spring[i].axis = position
        spring[i].length = vp.mag(position)

    triangles = []
    for z in itertools.combinations(ballvectors, 3):
        triangles.append(z)
    for i in range(len(triangles)):
        T[i].v0.pos = triangles[i][0]
        T[i].v1.pos = triangles[i][1]
        T[i].v2.pos = triangles[i][2]

    dampening = 0.9
    F_mat = np.zeros((8,8))
    F_vec = []
    F_v = []
    a = np.array(np.zeros((8,8)))
    for i in range(8):
        for k in range(8):
            if k == i:
                L = 0
                F_mat[i][k] = 0
                F_vec.append(vp.vector(0, 0, 0))
            else:
                L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i]
                F_mat[i][k] = L*k_sp
                pf0 = ballname[k].pos - ballname[i].pos
                #a[i,k] = vp.norm(pf0)*L*k_sp
                F_vec.append(vp.norm(pf0)*L*k_sp)
    a = np.array(F_vec).reshape(8, 8)
    F = a.sum(axis=0)
    for i in range(8):
        F[i] = F[i] + g_vector*mass
        if ballname[i].pos.y < floor.pos.y:
            F_N = ((floor.pos.y-ballname[i].pos.y)**2)*10000
            F[i].y = 0.99*(F[i].y - F_N)
            mu = 1
            F_st = mu*F_N
            F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
            v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
            vx = ballname[i].velocity.x / v_xz
```

```
            vz = ballname[i].velocity.z / v_xz
            if F_st < F_horiz:
                F[i].x += F_horiz*vx - F_N*vx
                F[i].z += F_horiz*vz - F_N*vz
            else:
                F[i].x = F_horiz*vx
                F[i].z = F_horiz*vz
                ballname[i].velocity.x = 0
                ballname[i].velocity.z = 0

    for i in range(8):
        ballname[i].velocity -= F[i]/mass*dt
        ballname[i].pos += ballname[i].velocity*dt
```

TETRAHEDRON CODE:

```
import vpython as vp
import itertools
import numpy as np

scene = vp.canvas(title = 'Tetrahedron', width = 600, height = 400, center = vp.vector(0,0,0))

#vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0,-7,0), length=10, height=0.01, width=10, color=vp.color.blue)

ballname = ['b1', 'b2', 'b3', 'b4'] #, 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(-1, 0, -1/(2)**0.5), vp.vector(0, 1, 1/(2)**0.5), vp.vector(1, 0,
-1/(2)**0.5),  vp.vector(0, -1, 1/(2)**0.5)] #, vp.vector(1, 1, 0), vp.vector(0, 1, 1),
             # vp.vector(1, 0, 1), vp.vector(1, 1 1)]
for i in range(4):
   ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,1,1))

springvecs = []
for i in range(4):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red, f_k=
vp.vector(0,0,0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)
spring = ['s1', 's2', 's3', 's4', 's5', 's6'] #, 's7', 's8', 's9', 's10','s11', 's12', 's13',
's14', 's15', 's16', 's17', 's18', 's19', 's20','s21', 's22', 's23', 's24', 's25', 's26', 's27',
's28']

for i in range(6):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03, color=vp.color.white)
v = 0
dt = 0.003
mass = 0.1
g = 9.81
k_sp = 1000
g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0,0,0)

F_c = vp.vector(0, 1000, 0)

L0= np.zeros((4, 4))
for i in range(4):
   for j in range(4):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)
#F = np.zeros((8,3))
while 1:
   vp.rate(150)
   floor = vp.box(pos=vp.vector(ballvectors[1].x, -7, ballvectors[1].z), length=5, height=0.01,
width=5, color=vp.color.blue)
   scene.center.x = ballvectors[1].x
   scene.center.z = ballvectors[1].z
   for i in range(4):
```

```
        ballvectors[i] = ballname[i].pos
    springvecs = []
    for z in itertools.combinations(ballvectors, 2):
        springvecs.append(z)
    for i in range(6):
        position = springvecs[i][1] - springvecs[i][0] #- L0[i]
        spring[i].pos = springvecs[i][0]
        spring[i].axis = position
        spring[i].length = vp.mag(position)
    dampening = 0.98
    F_mat = np.zeros((4,4))
    F_vec = []
    F_v = []
    a = np.array(np.zeros((4,4)))
    for i in range(4):
        for k in range(4):
            if k == i:
                L = 0
                F_mat[i][k] = 0
                F_vec.append(vp.vector(0, 0, 0))
            else:
                L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i]
                F_mat[i][k] = L*k_sp
                pf0 = ballname[k].pos - ballname[i].pos
                #a[i,k] = vp.norm(pf0)*L*k_sp
                F_vec.append(vp.norm(pf0)*L*k_sp)
    a = np.array(F_vec).reshape(4, 4)
    F = a.sum(axis=0)
    for i in range(4):
        F[i] = F[i] + g_vector*mass
        if ballname[i].pos.y < floor.pos.y:
            F_N = ((floor.pos.y-ballname[i].pos.y)**2)*10000
            F[i].y = 0.99*(F[i].y - F_N)
            mu = 1
            F_st = mu*F_N
            F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
            v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
            vx = ballname[i].velocity.x / v_xz
            vz = ballname[i].velocity.z / v_xz
            if F_st < F_horiz:
                F[i].x += F_horiz*vx - F_N*vx
                F[i].z += F_horiz*vz - F_N*vz
            else:
                F[i].x = F_horiz*vx
                F[i].z = F_horiz*vz
                ballname[i].velocity.x = 0
                ballname[i].velocity.z = 0

            # print(F[i].x, F[i].z)
    for i in range(4):
        ballname[i].velocity -= F[i]/mass*dt
        ballname[i].pos += ballname[i].velocity*dt
        # if ballname[i].pos.y < floor.pos.y:
        #     ballname[i].velocity = dampening*-1*(ballname[i].velocity)+ F_c*(floor.pos.y -
ballname[i].pos.y)*(floor.pos.y - ballname[i].pos.y)
```

GROUNDED NODE CODE:

```
import vpython as vp
import itertools
import numpy as np

scene = vp.canvas(width = 600, height = 400, center = vp.vector(0,0,0))
#vp.display(width=100, height=100)
floor = vp.box(pos=vp.vector(0,-5,0), length=8, height=0.01, width=8, color=vp.color.blue)

# -0.35 y for grounded node
ballname = ['b1', 'b2', 'b3', 'b4', 'b5', 'b6', 'b7', 'b8']
ballvectors = [vp.vector(0, 0, 0), vp.vector(0, 1, 0), vp.vector(0, 0, 1),  vp.vector(1, 0, 0),
vp.vector(1, 1, 0), vp.vector(0, 1, 1),
              vp.vector(1, 0, 1), vp.vector(1, 1, 1)]

for i in range(8):
   ballvectors[i] = ballvectors[i].rotate(angle=3.14/4, axis =vp.vector(1,0,1))


springvecs = []
for i in range(len(ballname)):
   ballname[i] = vp.sphere(pos=ballvectors[i], radius=0.1, color=vp.color.red, f_k=
vp.vector(0,0,0))
for z in itertools.combinations(ballvectors, 2):
   springvecs.append(z)


spring = ['s1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10','s11', 's12', 's13', 's14',
's15', 's16', 's17', 's18', 's19', 's20','s21', 's22', 's23', 's24', 's25', 's26', 's27', 's28']

for i in range(28):
   position = springvecs[i][1] - springvecs[i][0]
   spring[i] = vp.cylinder(pos=springvecs[i][0], axis=position, length=vp.mag(position),
radius=.03, color=vp.color.white)

v = 0
dt = 0.003
mass = 0.1
g = 9.81
k_sp = 1000
g_vector = vp.vector(0, 9.81, 0)
for i in range(len(ballname)):
   ballname[i].velocity = vp.vector(0,0,0)

F_c = vp.vector(0, 1000, 0)

L0= np.zeros((8,8))

dL = list(range(8))
for i in range(8):
   for j in range(8):
       if i == j:
           L0[j][i] = 0
       else:
           position = ballname[j].pos - ballname[i].pos
           L0[j][i] = vp.mag(position)
   dL[i] = (floor.pos.y - ballname[i].pos.y)
```

```
dt1 = dt
asdf = 100
#F = np.zeros((8,3))
while 1:
   vp.rate(200)
   floor = vp.box(pos=vp.vector(ballvectors[1].x, -5, ballvectors[1].z), length=5, height=0.01,
width=5, color=vp.color.blue)
   #scene.center.x = ballvectors[1].x
   dL[i] += (floor.pos.y - ballname[i].pos.y)
   dt1 += .05
   for i in range(8):
       ballvectors[i] = ballname[i].pos
   springvecs = []
   for z in itertools.combinations(ballvectors, 2):
       springvecs.append(z)
   for i in range(28):
       position = springvecs[i][1] - springvecs[i][0] #- L0[i]
       spring[i].pos = springvecs[i][0]
       spring[i].axis = position
       spring[i].length = vp.mag(position)


   dampening = 0.9
   F_mat = np.zeros((8,8))
   F_vec = []
   F_v = []
   a = np.array(np.zeros((8,8)))
   for i in range(8):
       for k in range(8):
           if k == i:
               L = 0
               F_mat[i][k] = 0
               F_vec.append(vp.vector(0, 0, 0))
           else:
               L = vp.mag(ballname[k].pos - ballname[i].pos) - L0[k][i]
               F_mat[i][k] = L*k_sp
               pf0 = ballname[k].pos - ballname[i].pos
               #a[i,k] = vp.norm(pf0)*L*k_sp
               F_vec.append(vp.norm(pf0)*L*k_sp)
   a = np.array(F_vec).reshape(8, 8)
   F = a.sum(axis=0)
   #print(F)
   for i in range(1,8):
       F[i].y = F[i].y + g_vector.y*mass #- dampening*(dL[i])/dt1

       if ballname[i].pos.y < floor.pos.y:
           F_N = ((floor.pos.y-ballname[i].pos.y)**2)*10000

           F[i].y = (F[i].y - F_N) - dampening*(floor.pos.y-ballname[i].pos.y)

           mu = 1
           F_st = mu*F_N
           F_horiz = (F[i].x ** 2 + F[i].z ** 2) ** 0.5
           v_xz = (ballname[i].velocity.x ** 2 + ballname[i].velocity.z ** 2) ** 0.5
           vx = ballname[i].velocity.x / v_xz
           vz = ballname[i].velocity.z / v_xz
           if F_st < F_horiz:
```

```
            F[i].x += F_horiz*vx - F_N*vx
            F[i].z += F_horiz*vz - F_N*vz
        else:
            F[i].x = F_horiz*vx* dampening
            F[i].z = F_horiz*vz* dampening
            ballname[i].velocity.x = 0
            ballname[i].velocity.z = 0

        print(F[i].x, F[i].z)
for i in range(1,8):
    ballname[i].velocity = ballname[i].velocity*0.999 - F[i]/mass*dt
    ballname[i].pos += ballname[i].velocity*dt
```