Assignment 3

Nattapong Ounanong

UNI: no2313

Course: MECS 4510

Instructor: Hod Lipson

Date Submitted: October 31, 2018

Grace Hours: 0 used / 52 accumulated / 104 remain

Results

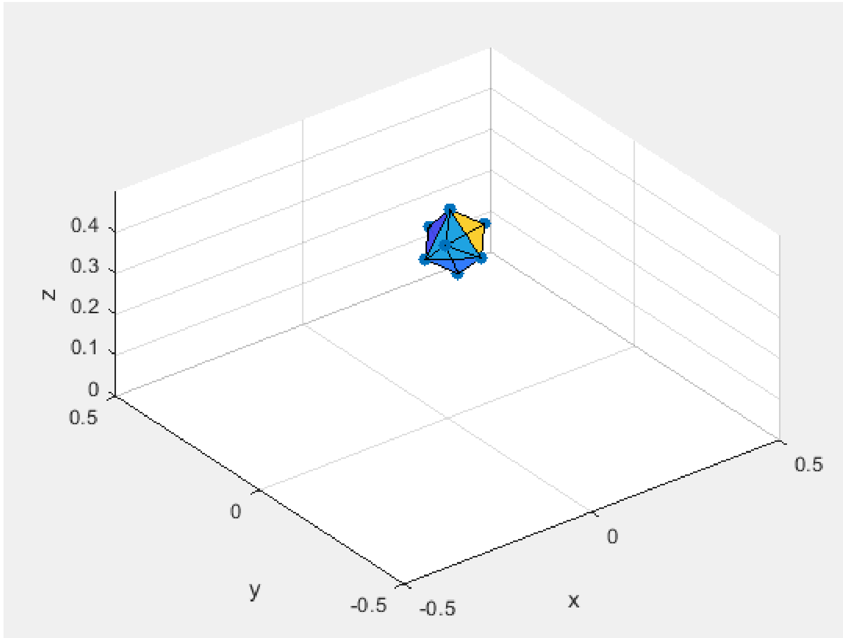

Figure 1: Image of bouncing cube

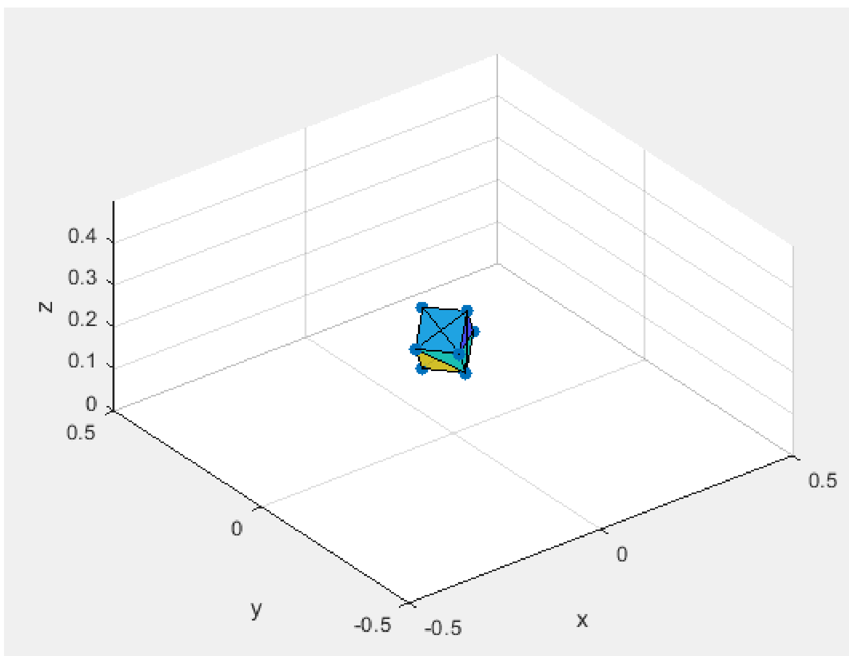

Figure 2: Image of breathing cube

Note that this video link has cube which bounced cube with and without damping, breathing, triangle shaded, grounded node implemented simpler problem (tetrahedron), and cube launched with slight spin.

Link: https://www.youtube.com/watch?v=W2coHvtRHzk&feature=youtu.be

Method

The mass-spring simulator is constructed by using 8 mass located at each vertices of the cube and having 28 springs attached to each mass which connected mass together. Spring force was estimated using hook's law and implemented by using 10,000 N/m spring constant. Initially, the cube was launched that the height of 0.2-0.3 meter. When the mass hit the ground and, further, underground, the reaction force was determined by using spring equation $F_{ground} = -kz$, where k is a spring constant, which in this case equal to 10,000 N/m, and z is the distance of the mass from the ground. Spring force is calculated at mass index 1 and spring force that connected to this mass would be the inversed value. For example, if the spring force that connected to mass 1 and 2 is positive, spring force that exert on mass 1 would be positive, while mass 2 would become negative. Once spring forces was determined, the equivalent spring forces was determine using Newton 2nd law at as force vector then utilized to calculated net force acted on the mass. In order to determine velocity and position of each mass on the cube, acceleration is utilized for integration by multiplying the acceleration to dt = 0.0001 s then add this value to the initial velocity of the cube. Likewise, position of the mass is also utilized the same method. The runtime for this simulation is 2 seconds which yielded 10k spring evaluation per seconds.

In order to make the simulator more realistic, damping was implemented using various value from 0.99 to 0.999 which yielded significant different as 0.99 allowed the cube to be bounced for 3 times. For breathing mechanism, the rest length of the spring is varied using initial length plus 0.001sin(1000T), where T is the number of iterations. This would make every spring of the cube to be expanded and shrink at the same rate.

Performance Plot

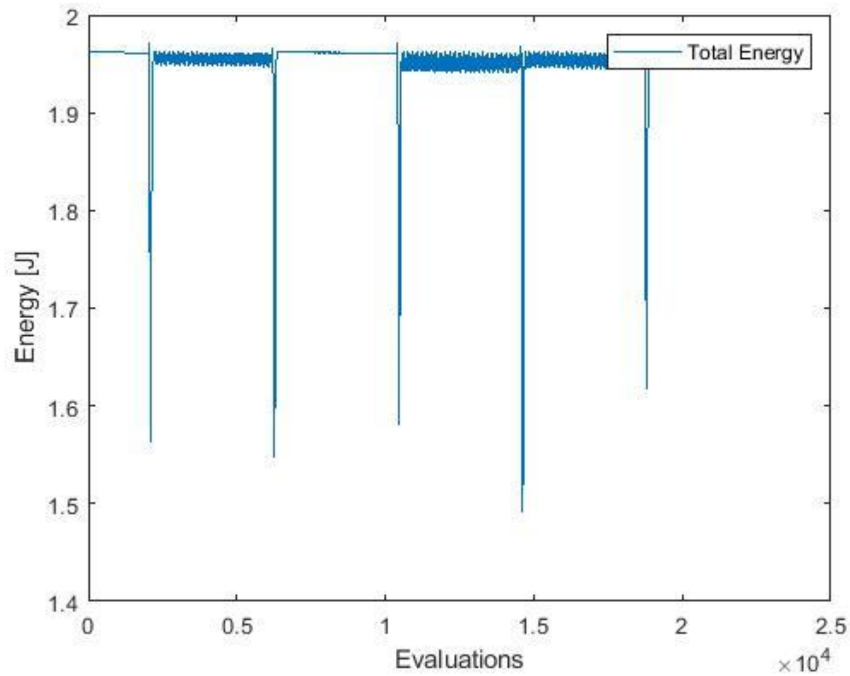

Figure 3: Total energy plot; potential (spring, potential and ground) and kinetic energy) for 20k evolutions

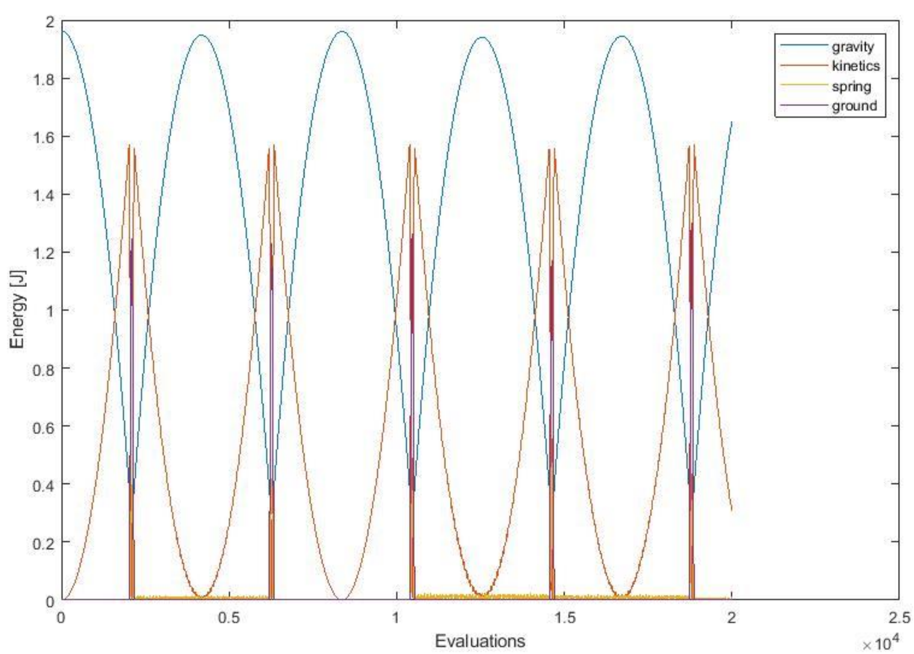

Figure 4: Comparison of potential energy (spring gravity and ground) and kinetics energy (velocity)

Appendix (Matlab Code)

```
clear;clc;close all;
% initiallized constant
initPos = 0.2; k = 1e4;g = [0,0,-9.81];dt=0.0001;z = 0;T = 0;iterate = 1;
Fc = zeros(1,8);
% dt = 0.1;
%     create mass matrix in which tha the 1st column is mass of vertex in
%     kg, column 2-4 are 3D position vector, column 5-7 are velocity vector
%     and column 8-10 are acceleration vector
mass =cube();
initPos = 0.2;

% Create Spring array having spring constant in column one, 2nd column is
% rest length of spring and column 3 and 4 are mass that the spring
% connects to
% Loop start here

count  = 1;
for link = 1:size(mass,1)
    for linkto = (link+1):size(mass,1)
        %         determine length between mass
        Lxo(count) = mass(link,2) - mass(linkto,2);
        Lyo(count) = mass(link,3) - mass(linkto,3);
        Lzo(count) = mass(link,4) - mass(linkto,4);
        Lo(count) = sqrt(Lxo(count)^2+Lyo(count)^2+Lzo(count)^2);
        spring(count,:) = [k, Lo(count), link, linkto];
        count = count +1;
    end
end

sL = size(spring,1);
% These two variable might be useful later
mL = size(mass,1);sL = size(spring,1);
mg = mass(:,1).*g;

while T <= 2
    %     Update current spring length
    count  = 1;
    %     mass(1,[2 3 4]) = [0.1 0.1 0.3];
    %     Lo(randi(28)) = Lo(randi(28))+0.1*sin(10*T);
    if mod(iterate,500) == 0
        %         funF = -10;
    else
        funF = 0;
        %         mass(1,[2 3 4]) = 0.1;
    end
            Lo = breathNow(Lo,T);
    for link = 1:size(mass,1)
        for linkto = (link+1):size(mass,1)
            %         determine length of spring

            Lx(count) = (mass(linkto,2) - mass(link,2));
            Ly(count) = (mass(linkto,3) - mass(link,3));
            Lz(count) = (mass(linkto,4) - mass(link,4));
            L(count) = sqrt(Lx(count)^2+Ly(count)^2+Lz(count)^2);
            %             Unit vector would be use for calculating new orientation of
            %             the spring
            unitVector(count,:) = [Lx(count)/L(count),Ly(count)/L(count),Lz(count)/L(count)];
            count = count +1;
        end
    end

    %     Calculate spring force and Energy
    x = []; y =[]; z=[];
    for F = 1:sL
        sF(spring(F,3),F) = -k*(Lo(F)-L(F));
```

```
        sFx(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,1);
        sFx(spring(F,4),F) = -sFx(spring(F,3),F);
        sFy(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,2);
        sFy(spring(F,4),F) = -sFy(spring(F,3),F);
        sFz(spring(F,3),F) = sF(spring(F,3),F)*unitVector(F,3);
        sFz(spring(F,4),F) = -sFz(spring(F,3),F);

        x = [x;mass(spring(F,[3:4]),2)];
        y = [y;mass(spring(F,[3:4]),3)];
        z = [z;mass(spring(F,[3:4]),4)];

        %          Spring Energy
        sE(F) = k/2*(Lo(F)-L(F))^2;
    end
    spE(iterate) = sum(sE);

    % Sum all spring force acting on each mass
    sFxm = sum(sFx,2);
    sFym = sum(sFy,2);
    sFzm = sum(sFz,2);

    %          Sum reaction force
    if sum(Fc) == 0
        FcmG = 0;
        zSum = zeros(1,8);
        damping = 1;
    else
        proportionZ = sum(unitVector(:,3));
        for mG = 1:mL
            for Zforce = 1:size(sFz,2)
                FcmG(mG,Zforce) = Fc(mG) * unitVector(Zforce,3)/proportionZ;
            end
            zSum(mG) = sum(FcmG(mG,:));
        end
        damping = 0.9999;
    end
    statFz(iterate,:) = sFzm;
    statzRec(iterate,:) = zSum;
    for mG = 1:mL
        %          Caculate equivalent spring force

        %          summation of forces
        if mG == 1 && T == 0
%              sumFx = sFxm(mG)+mg(mG,1)-1;
%              sumFy = sFym(mG)+mg(mG,2)-2;
            sumFx = -100;
            sumFy = 200;
            sumFz = sFzm(mG)+mg(mG,3)+zSum(mG)+5000;
        else
            sumFx = sFxm(mG)+mg(mG,1)+funF;
            sumFy = sFym(mG)+mg(mG,2)+funF;
            sumFz = sFzm(mG)+mg(mG,3)+Fc(mG);
        end
        %          acceleration
        mass(mG,8) = sumFx/mass(mG,1);
        mass(mG,9) = sumFy/mass(mG,1);
        mass(mG,10) = sumFz/mass(mG,1);
        %          velocity
        mass(mG,5) = (mass(mG,5)+dt*mass(mG,8))*damping;
        mass(mG,6) = (mass(mG,6)+dt*mass(mG,9))*damping;
        mass(mG,7) = (mass(mG,7)+dt*mass(mG,10))*damping;
        %          *damping
        %          position
        mass(mG,2) = mass(mG,2)+dt*mass(mG,5);
        mass(mG,3) = mass(mG,3)+dt*mass(mG,6);
        mass(mG,4) = mass(mG,4)+dt*mass(mG,7);

%         return
        %          Energy (gravity)
        mgH(mG,iterate) = mass(mG,4)*-mg(mG,3);
```

```
        KE(mG,iterate) =
mass(mG,1)/2*mass(mG,5)^2+mass(mG,6)^2*mass(mG,1)/2+mass(mG,7)^2*mass(mG,1)/2;


        %         make sure the cube would not go below ground
        %         Bounce the ball if individual mass is below ground level
        if mass(mG,4) < 0
            Fc(1,mG) = -k*mass(mG,4);
            Eg(mG,iterate) = k/2*mass(mG,4)^2;
        else
            Fc(1,mG) = 0;
            Eg(mG,iterate) = 0;
        end
    end

    if mod(iterate,100)==0
%         plot3(x,y,z)
        DT = delaunayTriangulation(mass(:,[2:4]));
        tetramesh(DT);
        hold on
        axis ([-0.5 0.5 -0.5 0.5 0 0.5]);
        xlabel('x');ylabel('y');zlabel('z');
        grid on
        scatter3(mass(:,2),mass(:,3),mass(:,4),'filled');
        drawnow
        pause(0.0001)
        grid on
        hold off
    end

    massH(iterate) = sum(mgH(:,iterate),1);
    massKE(iterate) = sum(KE(:,iterate),1);
    massEg(iterate) = sum(Eg(:,iterate),1);

    allE(iterate) = massH(iterate)+massKE(iterate)+massEg(iterate);
    T = T+dt;
    % +spE(iterate)+sum(dampE(:,iterate),1)+sum(Eg(:,iterate),1);
    iterate = iterate + 1;
    % if iterate >3500
    %     allE(end)
    %     massH(end)
    %     massKE(end)
    %     spE(end)
    %     massEg(end)
    %
    %     break
    % end
    % if allE(end) <1.9
    %     allE(end)
    %     massH(end)
    %     massKE(end)
    %     spE(end)
    %     massEg(end)
    %     break
    % end
end
figure(2)
plot([1:iterate-1],allE)
xlabel('Evaluations');ylabel('Energy [J]')
legend('Total Energy')
figure(3)
plot([1:iterate-1],massH)
hold on
plot([1:iterate-1],massKE)
plot([1:iterate-1],spE)
plot([1:iterate-1],massEg)
xlabel('Evaluations');ylabel('Energy [J]')
legend('gravity','kinetics','spring','ground')
% figure(4)
% plot([1:iterate-1],statFz(:,1))
% hold on
```

```
% plot([1:iterate-1],statzRec(:,1));
% xlabel('Evaluations');ylabel('Force [N]')
% legend('spring','ground')

function [rLength] = breathNow(rLength,time)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
noSpring = 28;
for k = 1:noSpring
             rLength(k) = rLength(k)-0.0001*sin(100*time);
end
end


function [mass] = cube
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
initPos = 0.2;
mass = [0.1, 0.1 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.2 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.1 ,0.2 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.2 ,0.2 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.1 ,0.1 ,initPos+0.1 , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.2 ,0.1 ,initPos+0.1 , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.1 ,0.2 ,initPos+0.1  , 0 ,0 ,0 ,0 ,0 ,0;...
        0.1, 0.2 ,0.2 ,initPos+0.1  , 0 ,0 ,0 ,0 ,0 ,0;];

end


function [mass] =  tetrahedron
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
initPos = 0.2;
mass = [0.1, 0.1 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.2 ,0.1 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.1 ,0.2 ,initPos , 0 ,0 ,0 ,0 ,0 ,0;
        0.1, 0.2 ,0.2 ,initPos+0.2, 0 ,0 ,0 ,0 ,0 ,0;];
end
```