

Simulation of Elastic Textile as System of Springs

Lynn Serizawa

serizawa.lynn@nyu.edu

Contents

Introduction	3
<i>Description of the Problem</i>	3
<i>Approach to the Problem</i>	3
<i>Background Information</i>	5
Relevant Equations	7
<i>Motion</i>	7
<i>Hooke's Law for Springs</i>	7
<i>Newton's Second Law of Motion</i>	7
<i>Trigonometric</i>	8
<i>Norm</i>	11
<i>Unit Vector</i>	12
Numerical Method	14
<i>Forward Euler's Method</i>	14
Results	15
<i>Control Case</i>	15
<i>Heavier Ball</i>	16
<i>Off-center</i>	16
<i>Off-Center and Bigger Radius</i>	17
<i>More Rows and Columns</i>	18
<i>Less Stiff Spring</i>	18
<i>Stiffer Spring</i>	19
Conclusion	20
References	21
<i>YouTube Links</i>	21
<i>References</i>	21
Appendix	23

Introduction

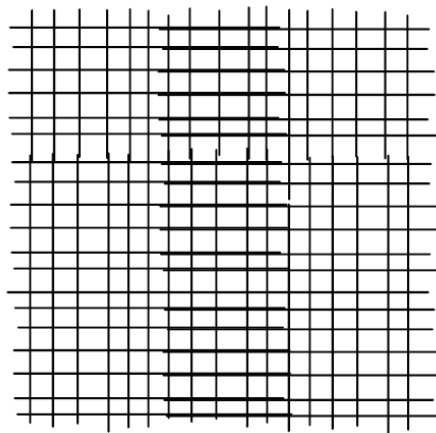
This report will describe the project in detail. First, I will describe the problem and how it was addressed. Then, I will state the relevant equations I used in creating a solution. Next, I will state the numerical method I used for the solution. Then, I will state my results and findings. Lastly, I will conclude the report and summarize my findings.

Description of the Problem

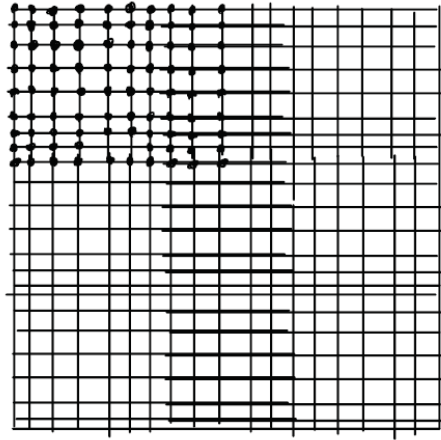
Here, I will describe the problem. The problem that I addressed is the simulation of a trampoline or net. A trampoline is an elastic fabric whose perimeter is held in place. The trampoline is woven using elastic strings (“Trampoline”). A net is similar, but there is more space between each elastic string (“Net”).

Approach to the Problem

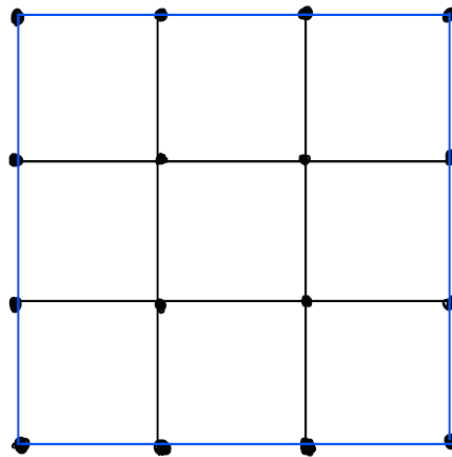
The strategy that I used to approach the problem was to visualize a net as a system of springs. As stated above, a net is a fabric woven using elastic strings. Below is a sketch of a net:



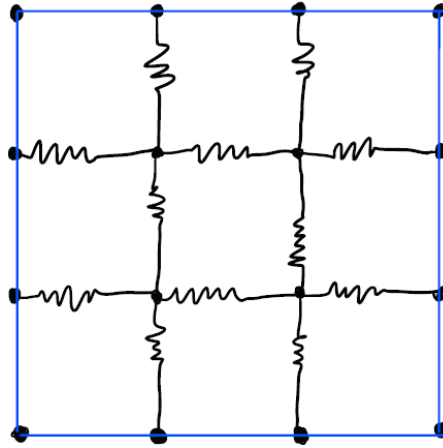
The strings are arranged in rows and columns. All of the rows are perpendicular to the columns. Since the row and column strings are perpendicular to each other, they intersect. Below is a sketch of a net with some of the intersections labeled with dots:



These intersections will be called nodes. To simplify the model, the nodes can be placed at regular intervals on the net. As stated previously, a net has its perimeter held in place. The blue lines in the sketch below represent the secured perimeter. The black lines represent the elastic strings:



Each segment of the elastic strings between each node can be individually represented as a spring:



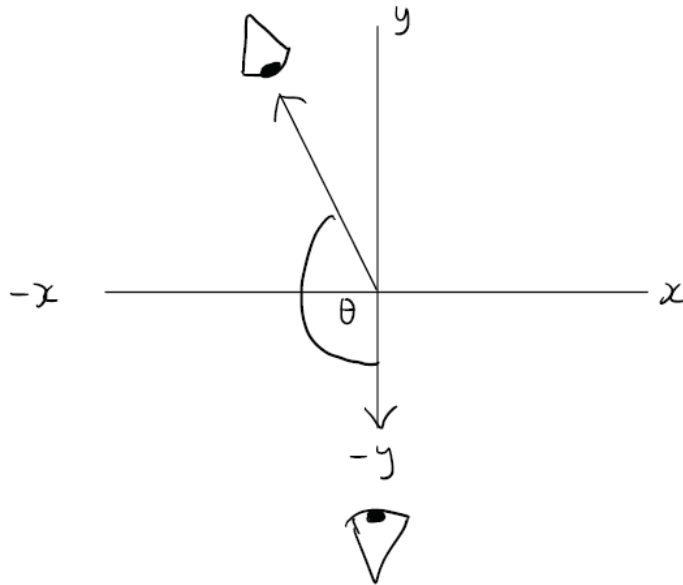
Therefore, one can represent a net or trampoline as a system of springs.

Now, I will describe the overall workflow of the script. The points representing the ball will be generated. Then, the net will be represented. Multiple matrices with the number of rows and columns as its dimension will be created. Then, the matrices will be used to store the position, velocity, and acceleration of each node in the x, y, and z directions. Per timestep, the script determines whether the net is in contact with the ball for each node in the net except the perimeter. If that is the case, the force exerted on the net by the ball is calculated. The force exerted on the ball by the net is also calculated. Then, the tension exerted on the node by the strings connected to its surrounding nodes is calculated. Then, the change in the position of both the ball and the net nodes is calculated. The cycle continues until all of the nodes are analyzed. The code continues this cycle until the simulation's duration is reached. Lastly, the code outputs an animation of the ball.

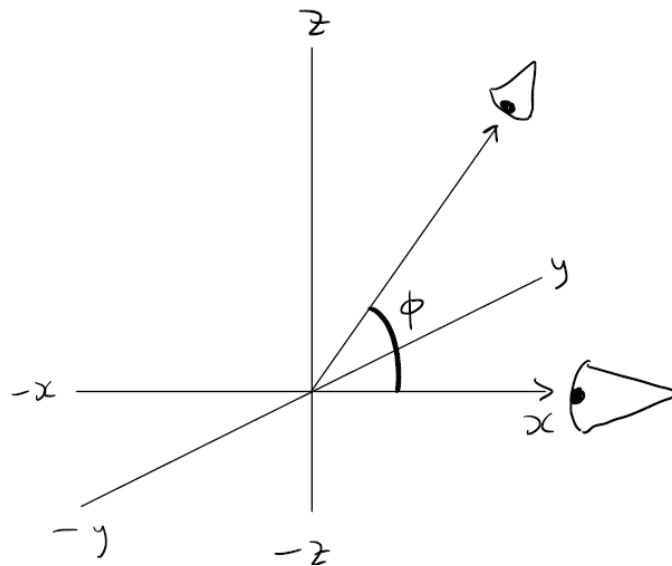
For the simulation, I will assume that the net has no mass. Furthermore, I will assume that the link between the nodes will not affect the ball. I will also assume that there is no friction or air resistance.

Background Information

Before I begin, I will define some terms that will be used in this report. The azimuth is the angle of the line of sight in the xy-plane from the negative y-axis. Below is a sketch showing the azimuth (θ):



The elevation is the angle of the line of sight between the xy-plane and the z-axis. Below is a sketch showing the elevation (ϕ) (“View”):



Relevant Equations

Here, I will state the equations I used for this project.

Motion

I used the following motion equations:

$$x_f = x_0 + vt + \frac{1}{2}at^2$$

Here, x_f is the final position of the object. x_0 is the initial position. t represents time. v is the velocity of the object at time t . a is the acceleration of the object at time t . The derivative of this equation with respect to t results in the velocity equation:

$$v_f = v_0 + at$$

Here, v_0 is the initial velocity. v_f is the final velocity. The other variables are the same as the above. The above equations were used to calculate the position of the ball. The equations were also used to calculate the kinematics of the nodes of the net (Pelcovits).

Hooke's Law for Springs

I also used Hooke's law, which calculates the spring force (tension):

$$F_S = -k \cdot (x_f - x_0)$$

The equation shows that the tension due to a spring is its stiffness multiplied by the change in its length. Here, F_S is the force of the spring. k is the spring coefficient. The spring coefficient represents how stiff the spring is. x_f represents the current length of the spring. x_0 represents the original length of the spring. This equation was used to calculate the forces exerted by each spring in the surface (Pelcovits).

Newton's Second Law of Motion

I also used Newton's second law of motion to calculate acceleration from force:

$$F = ma$$

Here, F represents the force. m represents the mass of the object. a represents the acceleration of the object. Suppose we know an object's force and mass. To find the acceleration of an object, one can divide both sides by m :

$$\frac{F}{m} = a$$

I used this equation to derive the acceleration from the force exerted on the nodes in the surface and the ball (“Pelcovits”).

Trigonometric

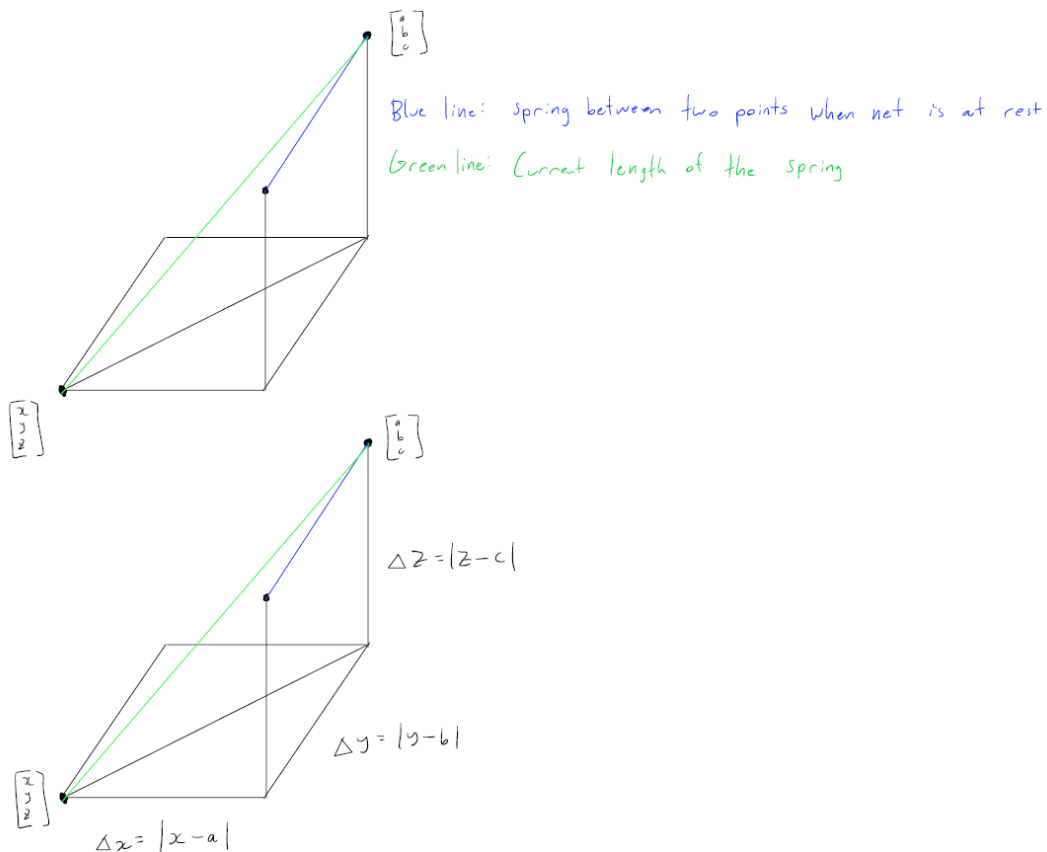
I also used basic trigonometric definitions to find the components of vectors.

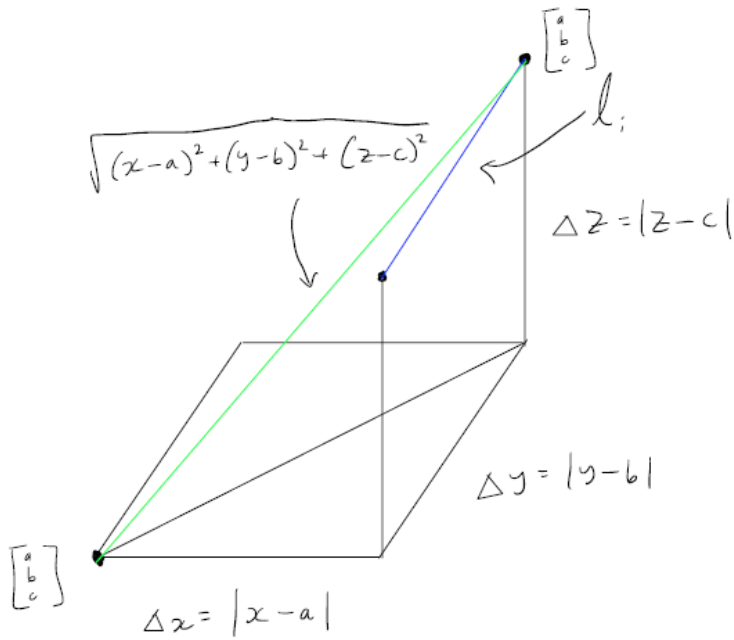
$$\tan \theta = \frac{\text{Opposite}}{\text{Adjacent}} \quad \sin \theta = \frac{\text{Opposite}}{\text{Hypotenuse}} \quad \cos \theta = \frac{\text{Adjacent}}{\text{Hypotenuse}}$$

I also used the Pythagorean theorem to determine the length of a spring between two nodes (Pelcovits):

$$a^2 + b^2 = c^2$$

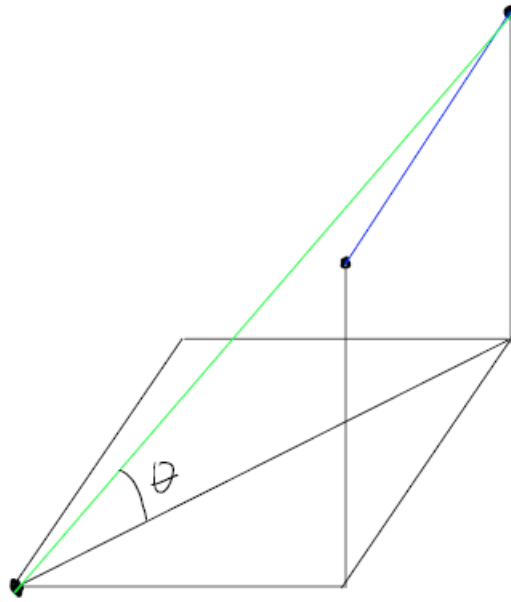
Combined with the aforementioned equation for tension, these equations were used to calculate the x, y, and z components of the tension exerted on a node by its surrounding nodes. Below is a sketch of the process:





Change in spring length: $\text{Final length} - \text{original length}$
 $\sqrt{(x-a)^2 + (y-b)^2 + (z-c)^2} - l_i = \Delta l$

$$T = kx = k(\Delta l)$$



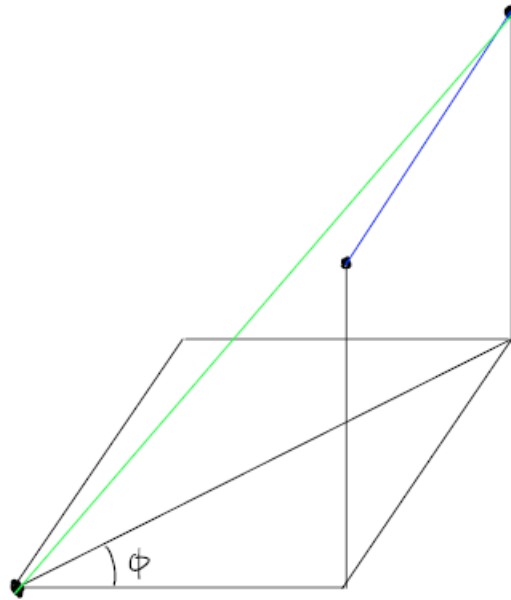
$$\tan \theta = \frac{\Delta z}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}$$

$$\theta = \tan^{-1} \left(\frac{\Delta z}{\sqrt{(\Delta x)^2 + (\Delta y)^2}} \right)$$

$$\sin \theta = \frac{T_z}{T}$$

$$T \sin \theta = T_z$$

$$T \cos \theta = T_{xy}$$



$$\tan \phi = \frac{\Delta y}{\Delta x}$$

$$\phi = \tan^{-1} \frac{\Delta y}{\Delta x}$$

$$\cos \phi = \frac{T_x}{T_{xy}}$$

$$T_{xy} \cos \phi = T_x$$

$$T_{xy} \sin \phi = T_y$$

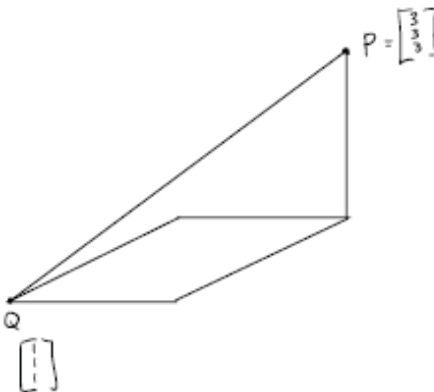
Norm

I used the `norm()` command in MATLAB to calculate the distance between two points in \mathbb{R}^3 . Suppose x_1, x_2, x_3 are the x , y , and z components of a point P . Suppose y_1, y_2, y_3 are the x , y , and z components of a point Q . `norm(P-Q)` does the following:

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}$$

This calculates the Euclidean distance between the two points. Below is a sketch of the calculation (“Norm”):

$$\text{norm}(P-Q)$$



$Q \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
 $\text{norm}(P-Q)$
 $\text{norm}\left(\begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right)$
 $\text{norm}\left(\begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}\right)$

$$\sqrt{2^2 + 2^2 + 2^2}$$

$$\sqrt{4 + 4 + 4}$$

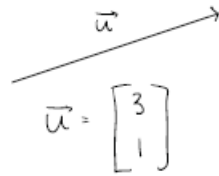
$$\sqrt{12}$$

Unit Vector

I used the formula for a unit vector to calculate the direction of the force the ball exerts on the net and vice versa. Vectors are a combination of magnitude and direction. Finding the force the net exerts on the ball can be found through Hooke's law. A different equation must be used to determine the direction of the force.

$$\frac{\vec{u}}{\|\vec{u}\|}$$

Below is a sketch of the calculation (Strang 14):



$$\vec{u} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$\|\vec{u}\| = \sqrt{3^2 + 1^2} = \sqrt{9 + 1} = \sqrt{10}$$

$$\frac{\vec{u}}{\|\vec{u}\|} = \begin{bmatrix} 3/\sqrt{10} \\ 1/\sqrt{10} \end{bmatrix}$$

Numerical Method

Here, I will describe the numerical method I utilized for this project.

Forward Euler's Method

Now I will explain forward Euler's method. The forward Euler's method is a numerical method that estimates the output of a function after a small change in the function's inputs. The forward Euler's method estimates the output of a function at a small step in time in the future. The method is used for solving ordinary differential equations (ODEs) when there is a known initial point ("Euler method"). The following is an equation for forward Euler's method (Peskin):

$$x_{i+1} = x_i + x'(t) dt$$

Here, x_i is an initial output of a function. x_{i+1} is the output after the initial output. $x'(t) dt$ is the change in the function with respect to a variable t multiplied by a timestep dt . Therefore, Euler's method estimates the next output by adding a small change to the initial point.

The equation above looks similar to the motion equations I described in the previous section. Therefore, I believe that the forward Euler method is a good method for this project.

Results

Here, I will discuss my findings from the simulation.

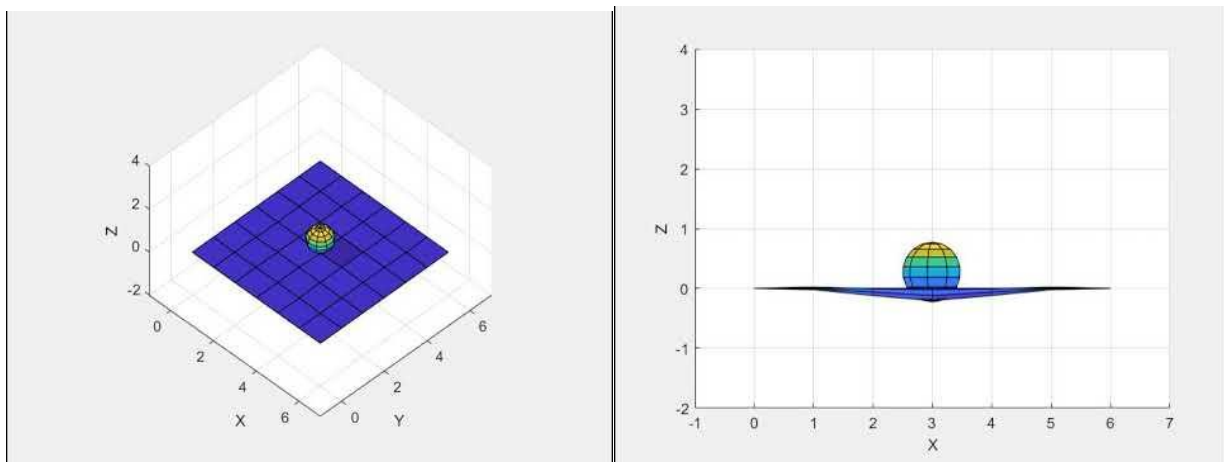
Control Case

Here, I will state what the control case's parameters are:

1. Mass: 1 kg
2. Radius: 0.5 m
3. Gravity: -9.8 m/s^2
4. Initial coordinates of the ball: (3,3,1.5) m
5. Initial velocity, acceleration of the ball: None
6. Net dimensions: 7 by 7
7. Spring Coefficient: 999
8. Timestep: 0.001 seconds
9. Duration: 5 seconds

Below is a video of the simulation. The video on the left was filmed with azimuth and an elevation of 45. The video on the right was filmed with azimuth and elevation of 0.

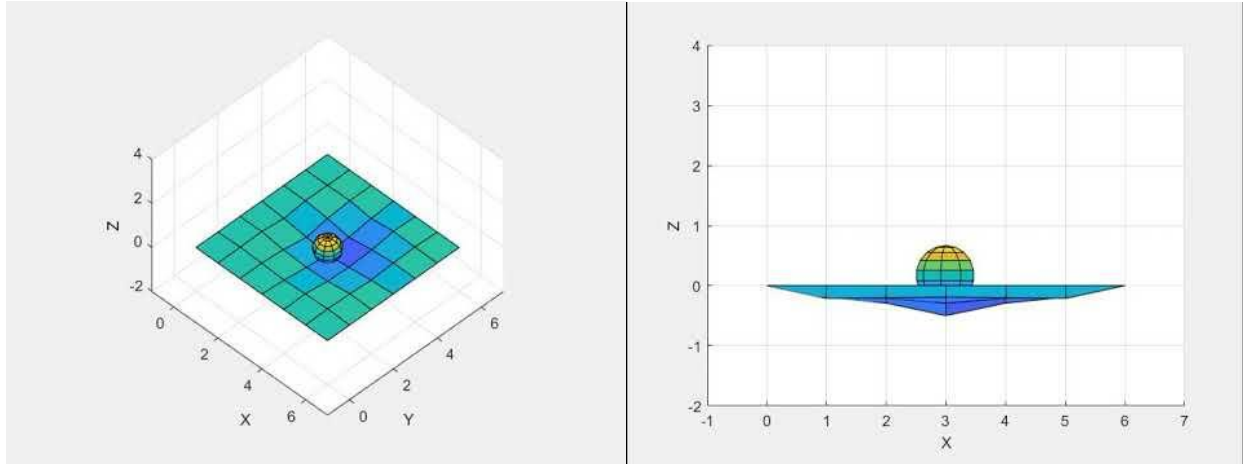
For every experiment described below, the order of the videos will be the same. (Animations can be viewed on YouTube. Links will be placed in the References section.)



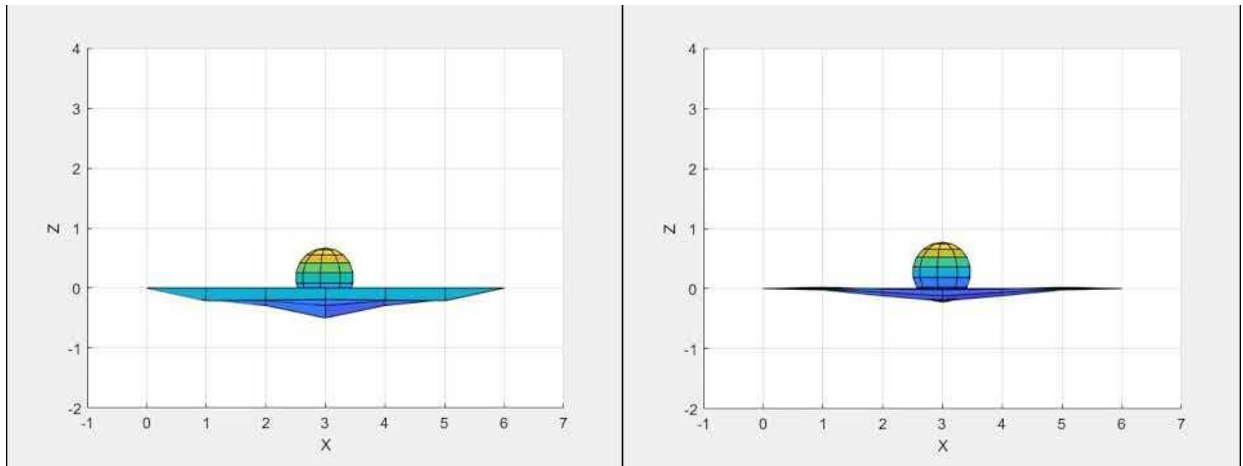
The videos are slowed down to ten seconds so that the effects on the net can be seen clearly. The following are some experiments I did with my simulation.

Heavier Ball

In this experiment, I changed the mass of the ball to 5 kilograms. My expectation was that the net would stretch further in the $-z$ direction than the control case.



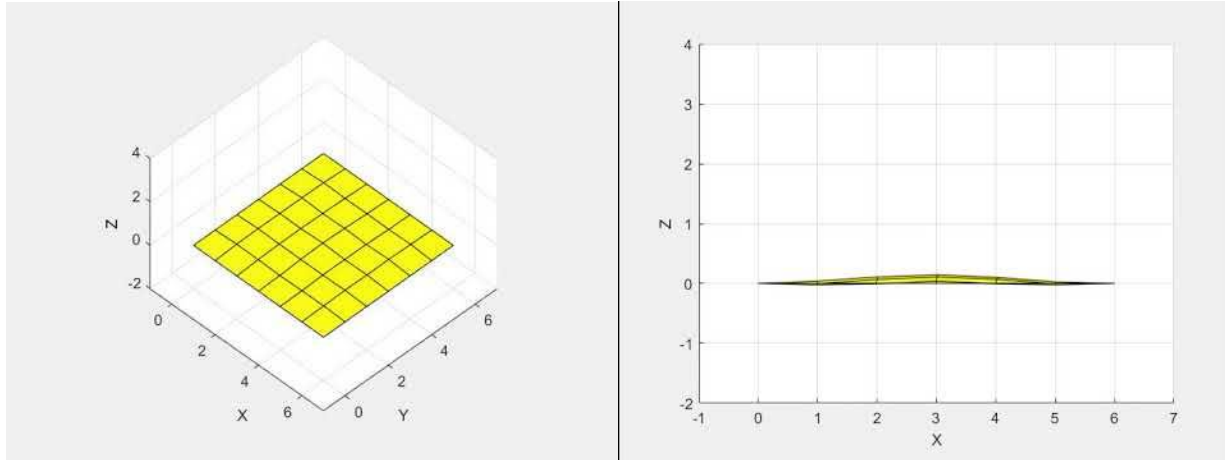
Below is a comparison with the control animation. The control is on the right.



Based on the animations, it is clear that the heavier ball will stretch the net further in the $-z$ direction. Furthermore, increasing the ball's mass increased the error between the net and the ball's position. This implies that for the most accurate measurements, a small timestep is better. However, making the timestep small may increase the runtime of the simulation.

Off-center

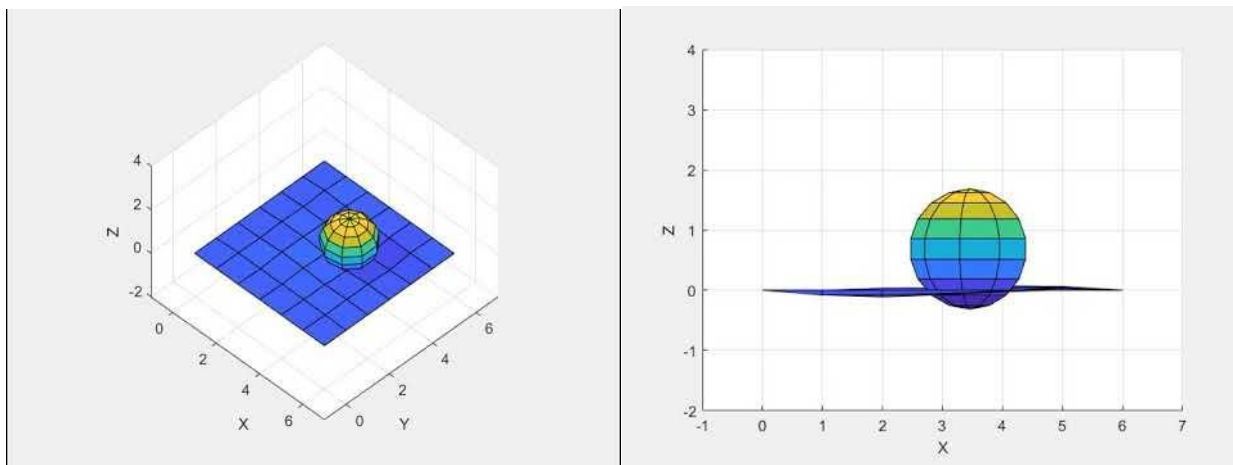
Here, the ball's initial point was moved 0.1 m in both the x and y direction so that the ball would not land perfectly on a node.



Since the ball was not centered directly on a node, the ball bounced off in the positive xy direction. Also, since the ball's radius was small enough, it could fit through the gaps in the nodes and fall out of the net.

Off-Center and Bigger Radius

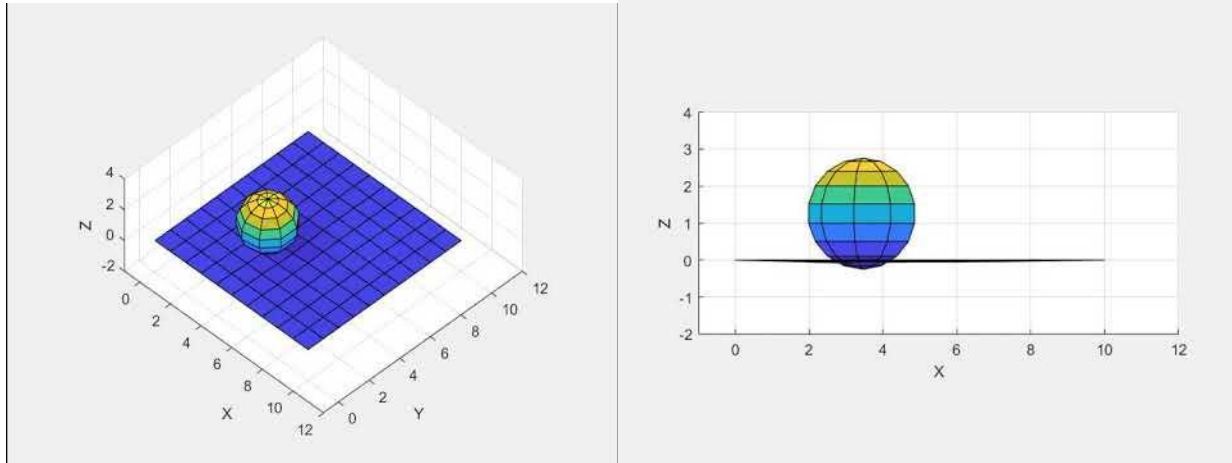
Seeing that the ball could fall through the gaps between the nodes, I ran another experiment where the ball's radius was large enough to not fall through the gaps and that it would not perfectly land on a node.



The ball moves in the positive xy direction because the ball landed on a node at an angle. However, since the ball could not fit through the gaps, the ball moved around in the gap between four nodes. Thus, depending on the initial energy and position of the ball, as long as the ball's diameter is greater than the space between each node, the ball will get stuck in some position in the net surrounded by four nodes. The ball may also fall out of the net if its initial energy is very high.

More Rows and Columns

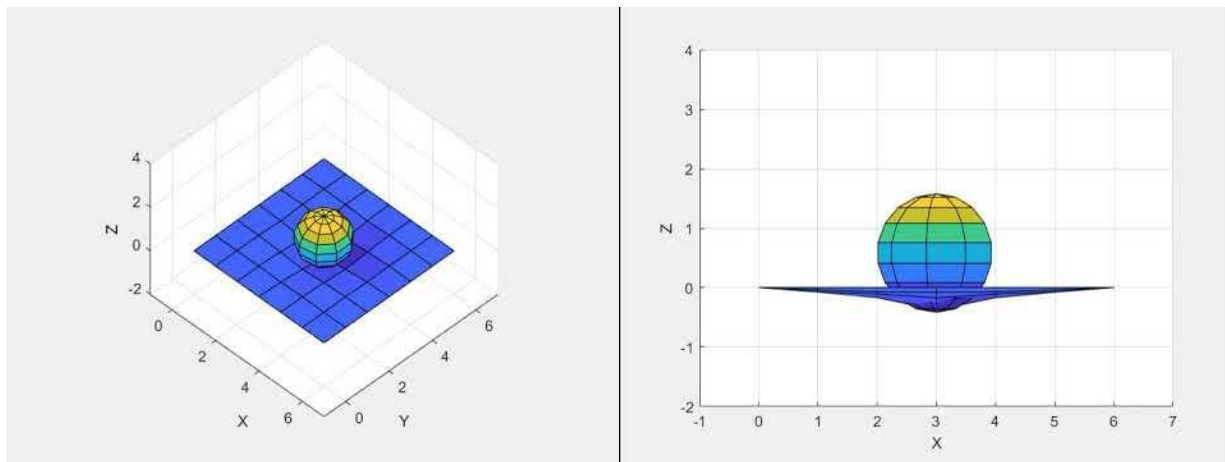
Here, the net's dimensions were increased to 11 by 11.



The ball moves in the positive xy direction. The ball moves toward the center. The ball's movement shows that being away from the center of the net causes the nodes to push the ball toward the center. However, like the previous experiment, the ball gets stuck in between four nodes.

Less Stiff Spring

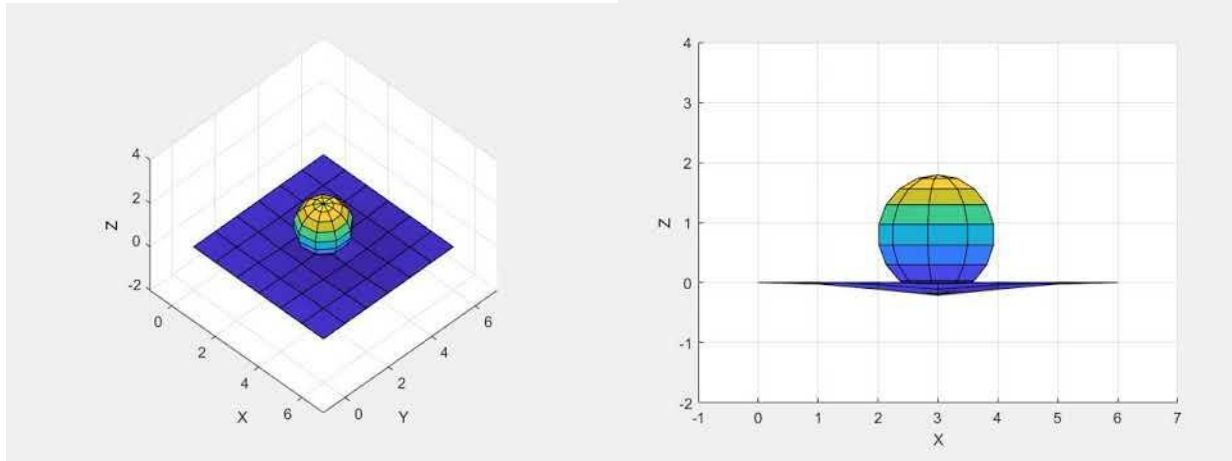
Here, the springs connecting each node were relaxed to a stiffness coefficient of 450.



A smaller spring coefficient will lead to a decrease in the tension in each string. Therefore, as the animation shows, the net stretched more than the net in the control simulation. This is expected based on Hooke's law.

Stiffer Spring

Here, the springs connecting each node were tightened to a stiffness coefficient of 1250.



An increase in the spring coefficient leads to an increase in the tension in each spring. Again, this is expected based on Hooke's law.

Conclusion

I used the forward Euler method to simulate a ball falling onto a net or trampoline in this project. I started the project by simplifying trampolines and nets as a system of nodes connected by springs. I used equations such as the motion equations, Hooke's law, trigonometry, and linear algebra equations. The script that I wrote calculates the ball's motion, the force exerted by the ball onto the net, the force exerted by the net on the ball, the position of all the nodes on the net, and the tension exerted on each node by its neighboring nodes. The script uses the forward Euler's method to calculate the kinematic properties of the net and the ball. I ran several experiments, such as modifying the ball's mass, position, radius, the net's dimensions, and spring stiffness. Through these experiments, I found that an increase in the mass of the ball increases the error between the ball and the net. I also found that if the ball is not at the center, the net tends to push the ball toward the center. Furthermore, Hooke's law and motion properties were observed when the spring constant and the ball's mass were altered.

References

YouTube Links

The following are YouTube links for the animations. The first link will be the animation viewed from an azimuth and elevation of 45 degrees.

1. Control
 - a. <https://www.youtube.com/watch?v=4z2btjaUs14>
 - b. https://www.youtube.com/watch?v=CQxAkOS_sU4
2. Heavier Ball
 - a. https://www.youtube.com/watch?v=e9OJ_9tO_oo
 - b. <https://www.youtube.com/watch?v=1BFqFzZgRis>
3. Off-center
 - a. <https://www.youtube.com/watch?v=MvHjKUm80A0>
 - b. <https://www.youtube.com/watch?v=zWHSQOHTXBs>
4. Off-center and Bigger Radius
 - a. <https://www.youtube.com/watch?v=etnX4DX2qzI>
 - b. <https://www.youtube.com/watch?v=rF9OnTBntVk>
5. More Rows and Columns
 - a. <https://www.youtube.com/watch?v=8K7OdEPBaMg>
 - b. <https://www.youtube.com/watch?v=Z4sg3pBylxU>
6. Less Stiff Spring
 - a. https://www.youtube.com/watch?v=VoF8j_NEFAg
 - b. https://www.youtube.com/watch?v=F5w8oJY_LvE
7. Stiffer String
 - a. https://www.youtube.com/watch?v=mNSu_iMlqgA
 - b. <https://www.youtube.com/watch?v=pqX3I67wJZQ>

References

1. Euler method. *Encyclopedia of Mathematics*. URL:
http://encyclopediaofmath.org/index.php?title=Euler_method&oldid=46859

2. “Net.” *Encyclopædia Britannica*, Encyclopædia Britannica, Inc.,
<https://www.britannica.com/technology/net>.
3. “Norm.” *Help Center*, <https://www.mathworks.com/help/matlab/ref/norm.html>.
4. Peskin, Charles. “Dynamics of Structures: Networks of Springs and Dashpots with Point Masses at the Nodes” *New York University, MATH-UA 395*,
https://www.math.nyu.edu/~peskin/modsim_lecture_notes/structural_mechanics.pdf,
accessed 5 Nov. 2021.
5. Pelcovits, Robert A., and Joshua Farkas. *Barron’s AP Physics C*. 4th ed., Barron’s Educational Series, 2016.
6. Strang, Gilbert. *Introduction to Linear Algebra*. Cambridge Press, 2016.
7. “Trampoline.” *Encyclopædia Britannica*, Encyclopædia Britannica, Inc.,
<https://www.britannica.com/sports/trampoline>.
8. “View.” *Help Center*, MathWorks,
<https://www.mathworks.com/help/matlab/ref/view.html>.

Appendix

The following is the code for this project.

```
%all or nothing 2D

%%Define ball
%mass of the ball
mass=1;

%Number of ball faces (n*n)
ball_resolution=9;

%radius
radius=1;

%gravity
gravity=-9.8;

%Initial Position of the center of the ball
ball_pos_xi=3.1;
ball_pos_yi=3.1;
ball_pos_zi=1.5;
ball_pos_center=[ball_pos_xi,ball_pos_yi,ball_pos_zi];

%Initial velocity of the ball
ball_vel_xi=0;
ball_vel_yi=0;
ball_vel_zi=0;
ball_vel_center=[ball_vel_xi,ball_vel_yi,ball_vel_zi];

%Initial acceleration of the ball
ball_acc_xi=0;
ball_acc_yi=0;
ball_acc_zi=gravity+0;
ball_acc_center=[ball_acc_xi,ball_acc_yi,ball_acc_zi];

%Collection of points representing the ball
[ball_pos_x,ball_pos_y,ball_pos_z]=sphere(ball_resolution);

%Adjust ball points to the specified initial points and radius
ball_pos_x=(ball_pos_x*radius)+ball_pos_xi;
ball_pos_y=(ball_pos_y*radius)+ball_pos_yi;
ball_pos_z=(ball_pos_z*radius)+ball_pos_zi;

%Build arrays to store the acceleration and velocity for each point
sz=size(ball_pos_x);
ball_vel_x=zeros(sz(1),sz(2));
ball_vel_y=zeros(sz(1),sz(2));
ball_vel_z=zeros(sz(1),sz(2));
ball_acc_x=zeros(sz(1),sz(2));
ball_acc_y=zeros(sz(1),sz(2));
ball_acc_z=zeros(sz(1),sz(2));
```



```
%define the net
%Initial length of each string in the net
initial_length=.95;

%Dimensions of the net
rows=7;
columns=7;

%spring stiffness
k=999;

%Build arrays for the positions, velocities, and acceleration of the nodes
net_pos_x=zeros(columns,rows);
net_pos_y=zeros(columns,rows);
net_pos_z=zeros(columns,rows);
net_vel_x=zeros(columns,rows);
net_vel_y=zeros(columns,rows);
net_vel_z=zeros(columns,rows);
net_acc_x=zeros(columns,rows);
net_acc_y=zeros(columns,rows);
net_acc_z=zeros(columns,rows);

%Iterate through the rows to dictate their initial positions
%We assume that the net starts at z=0
%If someone wants to place the net at some other z, the initial height of
%the ball can be adjusted instead.
counter_x=0;
counter_y=0;
for i=1:rows
    for j=1:columns
        net_pos_x(i,j)=(counter_x);
        net_pos_y(i,j)=(counter_y);
        net_pos_z(i,j)=0;
        counter_x=counter_x+1;
        if counter_x==columns
            counter_x=0;
            counter_y=counter_y+1;
        else
            continue
        end
    end
end

%Code for movie/visuals
my_figure=figure(1);

axis tight
axis equal
```

```
grid on
v=VideoWriter("filename.mp4");
%video fps
v.FrameRate=500;
open(v)

%%Here, one can alter these variables to alter output of the code

%Timestep
timestep=.001;

%Duration of the simulation
seconds=5;

%The angle of viewing in the xy-plane
azimuth=0;

%Angle of viewing between the z-axis and the xy-plane
elevation=0;

for time=0:timestep:seconds
    %%Plot sphere, net at current dt
    time
    dt=timestep;
    %Clear plot
    cla()

    %Turn on the gridlines
    grid on
    axis equal
    %Dictate angle of viewing
    view(azimuth,elevation)

    %Plot the ball
    surf(ball_pos_x,ball_pos_y,ball_pos_z)

    %Keep the current plot
    hold on

    %Set the limits of the axes
    axis([-1 7 -1 7 -2 4])
    %surf(net_pos_x,net_pos_y,net_pos_z,'FaceAlpha',0.25)
    surf(net_pos_x,net_pos_y,net_pos_z)

    xlabel("X")
    ylabel("Y")
    zlabel("Z")

    %%Calculate the next position of the ball
```

```

%%Determine forces created due to the interaction between the ball and
%%the net

net_ball_vector=[0,0,0];
oppo_vec=[0,0,0];
for i=2:(rows-1)
    for j=2:(columns-1)
        net_pos_vec=[net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j)];
        if (radius-norm(ball_pos_center-net_pos_vec))>0
            %%Force from the ball
            L=norm(ball_pos_center-net_pos_vec);
            vector_mag=(k)*(radius-L);
            vector_direction=-(net_pos_vec-ball_pos_center)/L;
            vector=vector_mag*vector_direction;
            net_ball_vector=net_ball_vector+vector;
            oppo_vec=oppo_vec+(((k)*(radius-L)*(net_pos_vec-ball_pos_center))/L);

            %%Force from the other nodes
            net_acc_x(i,j)=oppo_vec(1,1)/mass;
            net_acc_y(i,j)=oppo_vec(1,2)/mass;
            net_acc_z(i,j)=oppo_vec(1,3)/mass;
            Tension_up=tension(net_pos_x(i-1,j),net_pos_y(i-1,j),net_pos_z(i-1,
j),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
            Tension_down=tension(net_pos_x(i+1,j),net_pos_y(i+1,j),net_pos_z(i+1,
j),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
            Tension_left=tension(net_pos_x(i,j-1),net_pos_y(i,j-1),net_pos_z(i,j-
1),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
            Tension_right=tension(net_pos_x(i,j+1),net_pos_y(i,j+1),net_pos_z(i,
j+1),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
            Tension=((Tension_up+Tension_down+Tension_left+Tension_right)/mass);

            %%Euler Method on node
            net_acc_x(i,j)=net_acc_x(i,j)+Tension(1,1);
            net_acc_y(i,j)=net_acc_y(i,j)+Tension(1,2);
            net_acc_z(i,j)=net_acc_z(i,j)+Tension(1,3);
            net_vel_x(i,j)=net_vel_x(i,j)+(net_acc_x(i,j)*dt);
            net_vel_y(i,j)=net_vel_y(i,j)+(net_acc_y(i,j)*dt);
            net_vel_z(i,j)=net_vel_z(i,j)+(net_acc_z(i,j)*dt);
            net_pos_x(i,j)=net_pos_x(i,j)+(net_vel_x(i,j)*dt)+(0.5*(dt^2)
*net_acc_x(i,j));
            net_pos_y(i,j)=net_pos_y(i,j)+(net_vel_y(i,j)*dt)+(0.5*(dt^2)
*net_acc_y(i,j));
            net_pos_z(i,j)=net_pos_z(i,j)+(net_vel_z(i,j)*dt)+(0.5*(dt^2)
*net_acc_z(i,j));

        else
            %%Calculate net tension exerted on node
            Tension_up=tension(net_pos_x(i-1,j),net_pos_y(i-1,j),net_pos_z(i-1,
j),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);

```

```

        Tension_down=tension(net_pos_x(i+1,j),net_pos_y(i+1,j),net_pos_z(i+1,
j),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
        Tension_left=tension(net_pos_x(i,j-1),net_pos_y(i,j-1),net_pos_z(i,j-
1),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
        Tension_right=tension(net_pos_x(i,j+1),net_pos_y(i,j+1),net_pos_z(i,
j+1),net_pos_x(i,j),net_pos_y(i,j),net_pos_z(i,j),k,initial_length);
        Tension=((Tension_up+Tension_down+Tension_left+Tension_right)/mass);

        %Euler method on net
        net_acc_x(i,j)=Tension(1,1);
        net_acc_y(i,j)=Tension(1,2);
        net_acc_z(i,j)=Tension(1,3);
        net_vel_x(i,j)=net_vel_x(i,j)+(net_acc_x(i,j)*dt);
        net_vel_y(i,j)=net_vel_y(i,j)+(net_acc_y(i,j)*dt);
        net_vel_z(i,j)=net_vel_z(i,j)+(net_acc_z(i,j)*dt);
        net_pos_x(i,j)=net_pos_x(i,j)+(net_vel_x(i,j)*dt)+(0.5*(dt^2)
*net_acc_x(i,j));
        net_pos_y(i,j)=net_pos_y(i,j)+(net_vel_y(i,j)*dt)+(0.5*(dt^2)
*net_acc_y(i,j));
        net_pos_z(i,j)=net_pos_z(i,j)+(net_vel_z(i,j)*dt)+(0.5*(dt^2)
*net_acc_z(i,j));
        end
    end
end

%%Calculate the new position,velocity of the ball
ball_acc_x=net_ball_vector(1,1)./mass;
ball_acc_y=net_ball_vector(1,2)./mass;
ball_acc_z=gravity+(net_ball_vector(1,3)./mass);
ball_acc_center=[ball_acc_x,ball_acc_y,ball_acc_z];

ball_vel_x=ball_vel_x+(ball_acc_x*dt);
ball_vel_y=ball_vel_y+(ball_acc_y*dt);
ball_vel_z=ball_vel_z+(ball_acc_z*dt);
ball_vel_center=ball_vel_center+(dt*ball_acc_center);

ball_pos_x=ball_pos_x+(ball_vel_x*dt)+(.5*(dt^2)*ball_acc_x);
ball_pos_y=ball_pos_y+(ball_vel_y*dt)+(.5*(dt^2)*ball_acc_y);
ball_pos_z=ball_pos_z+(ball_vel_z*dt)+(.5*(dt^2)*ball_acc_z);
ball_pos_center=ball_pos_center+(ball_vel_center*dt)+(.5*(dt^2)*ball_acc_center);

%Code for visuals
Mv=getframe(gcf);
writeVideo(v,Mv)

end

close(v)

```

```

%Function to determine direction of the tension vector
function [direction_vec] = direction(x_pulling,y_pulling,x_node,y_node)
if (x_pulling-x_node>0) && (y_pulling-y_node)>0
    direction_vec=[-1 -1];
elseif (x_pulling-x_node)>0 && (y_pulling-y_node)==0
    direction_vec=[-1 0];
elseif x_pulling-x_node>0 && y_pulling-y_node<0
    direction_vec=[-1 1];
elseif x_pulling-x_node==0 && y_pulling-y_node>0
    direction_vec=[0 -1];
elseif x_pulling-x_node==0 && y_pulling-y_node==0
    direction_vec=[0 0];
elseif x_pulling-x_node==0 && y_pulling-y_node<0
    direction_vec=[0 1];
elseif x_pulling-x_node<0 && y_pulling-y_node>0
    direction_vec=[1 -1];
elseif x_pulling-x_node<0 && y_pulling-y_node==0
    direction_vec=[1 0];
elseif x_pulling-x_node<0 && y_pulling-y_node<0
    direction_vec=[1 1];
end
end

%Function to calculate tension
function [tension_vector] = tension(x_pulling,y_pulling,z_pulling,np_x,np_y,np_z,k, initial_length)
pull=[x_pulling,y_pulling,z_pulling];
node=[np_x,np_y,np_z];
change_x=abs(pull(1,1)-node(1,1));
change_y=abs(pull(1,2)-node(1,2));
change_z=(pull(1,3)-node(1,3));
net_change_spring=initial_length-sqrt((change_x^2)+(change_y^2)+(change_z^2));
T=k*net_change_spring;
phi=atan(change_z/sqrt((change_x^2)+(change_y^2)));
Txy=T*cos(phi);
Tz=T*sin(phi);
theta=atan(change_y/change_x);
Tx=Txy*cos(theta);
Ty=Txy*sin(theta);
direction_vec=direction(pull(1,1),pull(1,2),node(1,1),node(1,2));
tension_vector=[direction_vec(1,1)*Tx,direction_vec(1,2)*Ty,-Tz];
end

```