How to easily simulate a structure made of many springs and masses

Hod Lipson

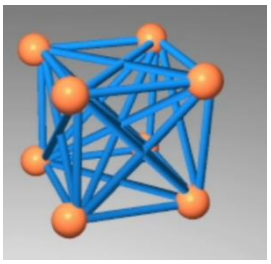
These instructions explain how to create a basic spring-mass physics simulation. You can implement this in any language. You will need to be able to plot some 3D graphics (lines and points). Assume that ***x,y*** plane is horizontal and ***z*** axis is pointing upwards.

First, let's implement a bouncing cube, like this:

https://1drv.ms/v/s!Ap6pJEdQihYkhZAzoncgCWfIjUSQRw

Implementation Steps

1. Create data structures for masses and springs
    a. Create two lists (arrays): One for springs and one for masses.
    b. Each mass needs the following attributes: ***m*** (mass, [kg]); ***p*** (3D position vector [meter]), ***v*** (3D velocity vector, meter/s), ***a*** (3D acceleration vector, meters/s^2)
    c. Each spring has the following properties: ***k*** (spring constraint, [N/m]), $\boldsymbol{L_0}$ (original rest length, [meters]), and the indices of the two masses it connects: $\boldsymbol{m_1}$, $\boldsymbol{m_2}$
2. Populate data structure with an initial test structure – a 3D cube
    a. Create an initial cube sized 0.1x0,1x01m, total weight 0.8 Kg.
    b. Initialize corner masses with 0.1Kg masses at all 8 corners
    c. Create 28 springs connecting all pairs of masses: 12 edges, 12 short diagonals and 4 long diagonals. Set ***k* =10,000 (or 1000)**, for example.
    d. Set the rest length of all springs to their initial Euclidean length and the velocities and accelerations of each mass to zero
    e. Plot the cube in 3D to make sure it looks ok. (you will need some 3D graphics library that can draw lines, like OpenGL)



3. Set Global variables of the simulation:
    a. Set Gravity constant **g=(0,0,-9.81)** m/s^2
    b. Set Simulation time-step **dt =0.001** or some other small number. You can increase or decrease this as needed
4. Simulate – the fun begins
    a. Loop over all masses. For each mass, sum all the forces applied to it by all the springs that are connected to it, and any external force like gravity and collision with the grounds. Here is how to do it:
        1. Tally a vector sum of all the forces that each spring connected to the mass induces on the mass. Each spring produces a force $\boldsymbol{F}=k(L-L_0)$, where L is its current length and L_0 is the original rest length. The force can be compression or tension depending on whether L is greater than or less than the rest length L_0. Make sure you get the sign correct.
        2. Add gravity force F=mg downwards
        3. Add any external forces, F_e. Initially these are zero.
        4. If the mass is below ground (negative z), add a restoration force F_c upwards proportional to how much below zero the mass is $\mathbf{F_c} = \mathbf{(0,0,k_c z^2)}$, k_c=10,000
        5. Add up all these forces into a single vector **F** for each mass

b. Loop over all masses again. Update the position of each mass based on the total force being applied to it.
    1. Use the summed force **F** to calculate acceleration of the mass ***a=F/m***
    2. Update the velocity of the mass using ***v=v+a*dt***
    3. Update the position of the mass using ***p=p+v*dt***
  ii. Draw all the strings and masses in their new positions (you will need some 3D graphics library that can draw lines, like OpenGL)

c. Update ***T = T + dt***

d. Repeat until masses stop moving or some fixed time elapses

5. The cube should be seen bouncing on the ground plane. You may need to adjust *dt* and *k* until you get smooth motion.

General tips:

1. Once you get it to work, you can easily parallelize the loop in 4a and 4b to use all your CPU cores. For example, in C++ use OMP. If you are really ambitious, you can even use GPU/CUDA.
2. You don't need to draw the cube every time step. For example, you can have a time-step of *dt=0.001sec* but draw the cube only every 100 time-steps
3. Plot the total energy of the cube as function of time (kinetic energy, potential energy due to gravity, potential energy in the springs, as well as the energy related to the ground reaction force). The sum should be nearly constant. If it is not constant, you have some bug
4. Be sure to use double precision numbers for all your calculations.

Advanced features

Once you have a basic running simulator you can start making things more interesting

1. **Fix some nodes**. Masses can be fixed (immovable) simply by skipping the update for that mass. For example, you can make your cube hang like a pendulum by simply fixing one corner.
2. **Add friction.** When a mass is at or below the ground, you can simulate friction and sliding. Calculate the horizontal force $F_H=\text{sqrt}(F_x^2+f_y^2)$ and the vertical force F_v. If $F_H<F_V*\mu$ (where μ is a coefficient of friction) then do not update the position of the mass.
3. **Add dampening**. To make things more realistic, you can add some dampening. For example, to add velocity dampening just multiply the velocity V by 0.999 each time step.
4. **Add interactivity.** You can "nudge" the cube by applying an external force to one or more of its masses. This can be done interactively using keyboard keys. For example, when the left arrow key is pressed, a force in the +X is applied to the topmost mass, etc. This is a good way to debug the code.
5. **Create actuators.** You can create a "muscle" simply by changing the rest length L_0 of one or more of the springs while the simulation is running. For example, if you change the rest lengths of the cube's springs slowly in a sine wave, the cube will appear to "breathe".
6. **Create larger structures**. You can create more complex structures by adding more masses and springs. For example, you can create three attached cubes. Or you can add masses randomly. Just make sure that each new mass is connected to at least three other masses if you want a solid structure. For example, see video with three cubes: https://1drv.ms/u/s!Ap6pJEdQihYkg5AEPPLCV-pl3K3eWg

6. **Create linkage mechanisms**. If you connect to cubes along a single edge, you can create a hinge. Other connectivity topologies will give you other types of joints.
7. **Create robots**. Combine linkage mechanisms and actuators to create a walking robot
8. **Create a solid appearance**. You can make the robot look solid by shading (filling in) all the exterior triangles of the robot.