Assignment 8 Mass-Spring System
CS180 Winter 2021

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Mar 5, 2021 (Friday)

Due at 23:59 on Mar 12, 2020 (Friday)

Notes:

- Be sure to read the Programming and Collaboration Policy on course website.
- Any updates or correction will be posted on Piazza, so check there occasionally.
- You must do your own work independently.
- Read through this document carefully before posting questions on Piazza.
- Submit your assignment on GauchoSpace before the due date.

---

1 Overview

1.1 Linking the rope constraints

In `rope.cpp`, implement the constructor of the class `Rope`. It should create a new Rope object starting at start and ending at end, containing `num_nodes` nodes. That is, something like the following diagram:

```
Rope(start, end, 3, ...)

  O-----------O------------O
  ^                        ^
start                     end
```

Each node is a **mass**, and each line between them is a **spring**. By creating a set of springs and masses, you create an object that behaves like a spring.

The nodes at the indices specified by **pinned_nodes** should have their pinned attribute set to true (those will be stationary). For each node, you should create a Mass object, and set the mass and spring constant **k** into the Mass constructor (PLEASE check the code/header files to check what exactly you should pass to the constructor). You also should create a spring between consecutive masses, again check the signatures of the constructors to see what you should pass as arguments to it.

Run `./ropesim`. You should see the rope drawn to the screen, although it won't be moving for now.

1.2 Explicit/Implicit Euler

Hooke's law states that the force on two points along a spring is proportional to their distance. That is,

$$f_{b \to a} = -k_s \frac{b-a}{||b-a||}(||b-a|| - l)$$

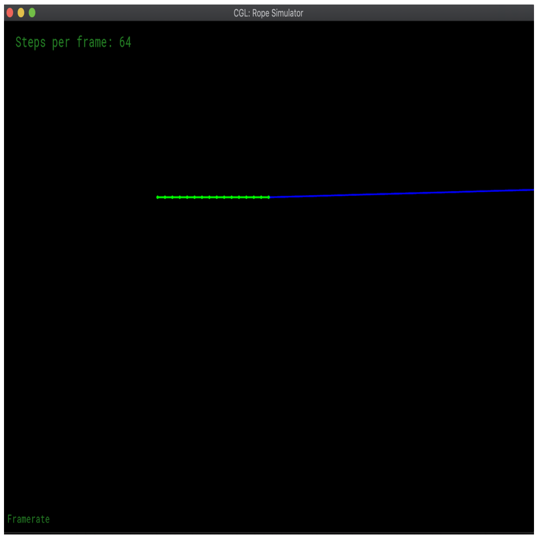
In `Rope::simulateEuler`, first implement Hooke's law. Iterate over all the springs and apply the correct spring force to the mass on either end of the spring. Ensure that the force is pointing in the correct direction! Accumulate all forces due to springs in the forces attribute of each Mass.

Once all the spring forces have been computed, apply the laws of physics to each particle:

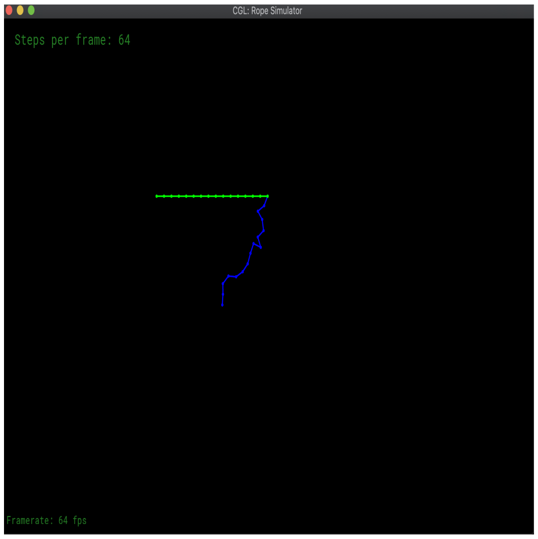
```
F = ma
v(t+1) = v(t) + a(t) * dt
x(t+1) = x(t) + v(t) * dt  // For explicit Euler method
x(t+1) = x(t) + v(t+1)* dt // For implicit Euler method
```

Run `./ropesim`. Your simulation should start running, but as it only has 3 nodes, it doesn't look like much. At the top of `application.cpp`, you should see where the Euler and Verlet ropes are defined. Change the value 3 for the number of nodes to a higher constant like 16 for both ropes.

Run `./ropesim -s <step-count>` to set the simulation to use a different number of simulation steps per frame. Try small values and large values (default is 64).



```
Steps per frame: 64
Framerate
```



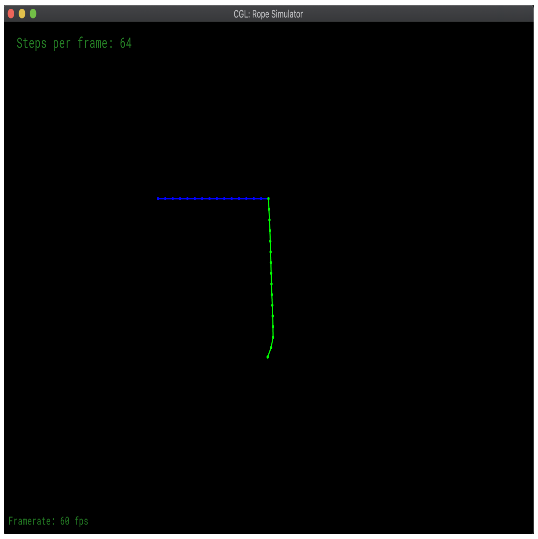
```
Steps per frame: 64
Framerate: 64 fps
```

1.3 Explicit Verlet

Verlet is a different way of ensuring that all constraints are dealt with accurately. The benefit to this approach is that the simulation is handled entirely through the positions of the vertices in the simulation, and it remains fourth-order accurate! Unlike Euler, Verlet integration follows the following rule to calculate the next position in the simulation:

```
x(t+1) = x(t) + [x(t) - x(t-1)] + a(t)*dt*dt
```

In addition, we can now emulate springs with an infinite spring constant. Instead of bothering with spring forces, we simply move each mass's position such that the springs are set back to their rest length. The correction vector should be proportional to the displacement between the two masses and in the direction between one mass and the other. Each mass should move by half the displacement.

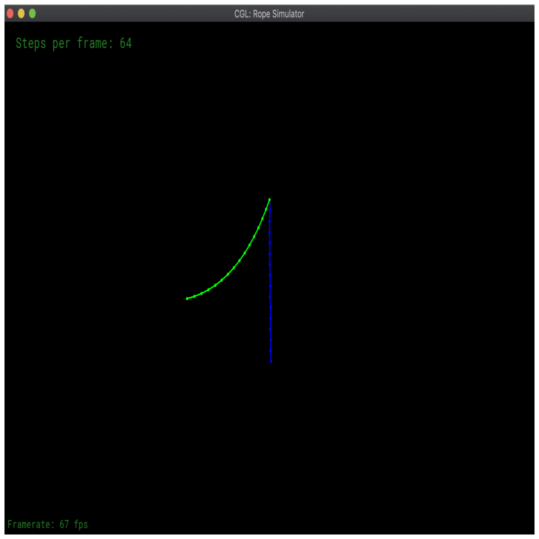


As long as we do this for every pair of springs, the simulation should approach stability. Additional rounds of simulations may be necessary to make the motion smoother.

1.4 Damping

Add damping to Hooke's law in explicit Verlet. Springs in real life don't continue bouncing forever - energy is lost to friction. Use damp factor 0.00005. The damping is applied by:

```
x(t+1) = x(t) + (1 - damping_factor) * [x(t) - x(t-1)] + a(t) * dt * dt
```



1.5

The functions you should modify are:

- `Rope::Rope(...)` in rope.cpp
- `void Rope::simulateEuler(...)` in rope.cpp
- `void Rope::simulateVerlet(...)` in rope.cpp

2 Getting started

You need to have OpenGL libraries installed on your system to compile and run. Otherwise, you can use the VM provided. Download the project's skeleton code, and build the project by using the following commands:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

After this, you should be able to run the given code by using **./ropesim**.

3 Grading and Submission

Grading:

- `[5 points] Submission is in the correct format, includes all necessary files. Code can compile and run.`
- `[5 points] Linking the rope constraints, correctly constructing the rope.`
- `[5 points] Implicit Euler`
- `[5 points] Explicit Euler`
- `[10 points] Explicit Verlet`
- `[5 points] Damping`

Submission:
After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, whether you modified it or not. **Submit your code including the implicit Euler and explicit Verlet with damping, comment out the explicit Euler lines.** Add a short report as a .pdf file in the directory, write down your full name and perm number inside it. **Briefly describe what you have implemented in each function.** Then compress the entire folder and rename it with your name, like "Kalyan.zip". Submit the .zip file on GauchoSpace.