Assignment 8 Mass-Spring System
CS180 Spring 2019

Professor: Lingqi Yan
University of California, Santa Barbara

Assigned on May 30, 2019 (Thursday)
Due at 23:59 on June 6, 2019 (Thursday)

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 Rope constructor. It should create a new Rope object starting at start and ending at end, containing num_nodes nodes. That is, something along the following diagram:

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

0-----------0-----------0
\^          ^
start       end
```

The nodes at the indices specified by pinned_nodes should have their pinned attribute set to true. Make sure to pass along the node_mass constant into the Mass constructor, and k into the Spring constructor. You should be creating a new Mass object for each node in the rope, and link together adjacent pairs of Mass objects with Spring objects.

Run ./ropesim. You should see the rope drawn to the screen.

1.2 Explicit Euler

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

\[ f_{a\rightarrow b} = -k_s \frac{b - a}{||b - a||^2} (||b - a|| - l) \]

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
```

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 3 value for the number of nodes to a higher constant like 16 for both ropes.
Run ./ropesim -s 32 to set the simulation to use a different number of simulation steps per frame. Try small values and large values (default is 64). What happens?

### 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:

\[ x(t+1) = 2 \times x(t) - x(t-1) + a(t) \times dt \times dt \]

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 Euler. In a spring-damper system, an additional damping force is applied (springs in real life don’t continue bouncing forever - energy is lost to friction). The damping force is proportional to the relative velocity of the two particles. Use a damping constant of 0.5.

\[ x(t+1) = (2 - \text{damping}\_\text{factor}) \times x(t) - x(t-1) + a(t) \times dt \times dt \]

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

The only dependency of the start code is CMake. It should be able to run on your own machine now. Please download the project’s skeleton code, and build the project by using the following commands:

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

After this, you should be able to run the given code by using `./ropesim`. There are several updates of the code base. Some general introductions are:

3 Grading and Submission

Grading:

- Linking the rope constraints (5 points)
- Explicit Euler (5 points)
- Explicit Verlet (10 points)
- Damping (5 points)

Submission:
After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, whether you modified it or not. Add a short README.md 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 ”Lei.zip”. Submit the .zip file on GauchoSpace.