Dartmouth COSC 89/189 Computational Methods for Physical Systems, Fall 2019

Instructor: Bo Zhu (bo.zhu@dartmouth.edu)

Assignment 1: Mass-Spring Model

Reading

In this assignment, you will learn how to build a mass-spring soft body simulator using C++11 and Eigen. Before starting the programming tasks, please read through the course slides, course notes, and the recommended reading material ("Real-Time Physics Course Notes," Chapter 3.1 and 3.2) to review the mathematical and numerical backgrounds of a mass-spring system.

Starter Code

Check out the latest starter code for this assignment from the course GitLab. The starter code for Assignment 1 is in proj/a1_mass_spring, including three C++ files -- main.cpp, SoftBodyMassSpring.h, and MassSpringInteractiveDriver.h. The building and compiling process is the same as Assignment 0.

Requirements

You are expected to implement your own mass-spring simulator for this assignment. This assignment includes two parts: explicit simulation and implicit simulation.

Part One (Explicit Mass Spring):

1) Implement spring force calculation (including both the elastic force and the damping force, 40%), force accumulation on particles (20%), and the explicit Euler time integration scheme (20%) for the mass-spring simulation model. Setup a scene for your own mass-spring simulation (20%).

Part Two (Implicit Mass Spring):

1) Implement the implicit Euler time integration scheme, including the stiffness matrix assembling (40%), the Jacobian matrix calculation for the elastic force (20%) and the damping force (20%).

2) Design your own mass-spring simulation scene to exhibit the power of implicit time integrator in modeling very stiff systems (20%).

Implementation Tips

Part One:

You need to implement three steps to build a mass-spring simulator: spring force calculation, particle force accumulation, and time integration. To this end, you are expected to implement the functions Spring_Force_Calculation, Particle_Force_Accumulation, and Advance_Explicit_Euler.

Spring_Force_Calculation:

1) Spring and particle indices

First, you are asked to compute the spring force for a spring connecting two particles. Each spring is represented by a 2D integer vector (i,j) specifying the indices of its two connecting particles. This 2D vector can be read from an array named springs.

2) Particle attributes

To calculate a spring force, you need to read the particles' attributes such as position, velocity, and mass. The i'th particle's attributes can be accessed by calling particles.X(i) (or particles.V(i), particles.F(i), particles.M(i), etc.) These functions return a reference to the value stored in an array. You may operate it the same way you operate an array element. For example,

If you want to read an attribute, e.g., the position of particle i:

```
VectorD pos=particles.X(i);
```

If you want to write:

```
particles.X(i)=VectorD::Ones();
```

Compute the elastic force, the damping force, and add them together to get the spring force. The function will return a Vector for f_ij (the spring force applied on particle i, accordingly, the force on particle j will be –f_ij).

Particle_Force_Accumulation:

After implementing the code to compute a force for each spring, now you want to apply the forces from all springs to their connected particles. Insert your code in Particle_Force_Accumulation() to update particles.F(i). Be careful about the "+" and "-" of the force vector when it is applied onto the two connected particles.

Advance_Explicit_Euler:

To build your explicit Euler integrator, you need to update the velocity and position of each particle based on its force and the time interval dt. After finishing these three steps, you should have an explicit mass-spring simulator. Test your code with the three test cases (rod, cloth, and beam) by running the executable with the argument –test 1 (or 2, 3).

Part Two:

You are asked to build the implicit Euler solver for a mass-spring system. This task consists of three subtasks: constructing the elastic Jacobian matrix, constructing the damping matrix, assembling the whole stiffness matrix K, and assembling the right-hand side b. The stiffness matrix is initialized with the correct nonzero indices the function of Initialize_Implicit_K_And_b. The value for each element is by default zero, and you are expected to correctly initialize these values according to the mathematics we have derived in class. Keep in mind that the stiffness matrix is sparse, symmetric, and block-based. We have already implemented the time integration scheme for you in the starter code.

To test your code, change time_integration to ImplicitEuler (line 28) and run the system with the same arguments. We use a much stiffer k_s to test our implicit solver. You are expected to see a stable and highly damped simulator if everything works correctly (See the videos in GitLab).

Grading

We will have the grading session for Part One in the X-hour of Week 3 and the grading session for Part Two in the X-hour of Week 4. You are expected to run the three examples (rod, cloth, soft body) in the grading session and answer questions from the TA. You also need to submit your source code to Canvas.