Dartmouth COSC 89.18/189.02 Computational Methods for Physical Systems, Winter 2019

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

Assignment 1: Mass-Spring Model

Reading

In this assignment, we will learn how to build a mass-spring soft body simulation system using C++11 and Eigen. Before starting the task, please read through the course slides (Lecture 2) 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 assignment one from the course GitLab: https://gitlab.com/boolzhu/dartmouth-phys-comp-starter. You may either use Git to pull the codebase or download the zip package from the link provided on the project webpage. If this is your first time to work with the starter code, please read the tutorial posted on the project page carefully to learn how to compile the source files and the OpenGL viewer (see https://gitlab.com/boolzhu/dartmouth-phys-comp-starter/blob/master/README.md). We highly recommend you to follow every step described in the tutorial to set up your coding and compiling environment (We use the same file structure for all the assignments, and if you want, also for your final project). These steps include installing CMake, compiling the test project in proj/a0_hello_world and run it (make sure to see the correct output), and compiling the OpenGL viewer (optional, you may also use the executable we uploaded in GitLab).

The starter code for Assignment 1 is in proj/a1_mass_spring, including three C++ files -- main.cpp, MassSpring.h, and MassSpringDriver.h. The building and compiling process is the same as the hello_world project, except modifying the path for the source code and the build binaries in CMake. (specifically, set source code as [Your path]/dartmouth-phys-comp-starter/proj/a0_mass_spring and build binaries as [Your path]/dartmouth-phys-comp-starter/build/a0_mass_spring). After you generate the project files (e.g., the .sln in Windows or makefile in Linux), you can compile the code within your IDE and generate the executable.

Requirements

You are expected to implement your own mass-spring simulator for this assignment. There are three-level requirements including an explicit simulator, an implicit simulator, and a technical paper implementation, corresponding to 1, 2, and 3 points (remember you need 4 points for the full 40% assignment credits).

Level-One (10%):

(Deadline Jan 24)

1) Implement spring force calculation (including both the elastic force and the damping force, 4%), force accumulation on particles (2%), and the explicit Euler time integration scheme (2%) for the mass-spring simulation model. Setup a scene for your own mass-spring simulation (2%).

Level-Two (20%):

(Deadline Jan 31)

1) Implement spring force calculation (including both the elastic force and the damping force, 4%), force accumulation on particles (2%), and the explicit Euler time integration scheme (2%) for the mass-spring simulation model.

2) Implement the implicit Euler time integration scheme, including the stiffness matrix assembling (4%), the Jacobian matrix calculation for the elastic force (2%) and the damping force (2%).

3) Design your own mass-spring simulation scene to exhibit the power of implicit time integrator in modeling very stiff systems (4%).

Level-Three (30%):

(Deadline Feb 21)

Implement the major part of one of the following classical technical papers or any original paper related to mass-spring soft body simulation (you need to get approval from the instructor and the TA if you want to implement a paper out of the provided list). You may take advantage of the starter code to help your paper implementation. You may also start from complete scratch. Only do this when you believe the assignment's topic is highly related to your perspective final project. Talk to the instructor and the TA before you get started.

Implementation Tips

Level-One:

You need to implement three necessary 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:

-- 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.

-- 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:

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).

Level-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 below).

Submission

For Level-One and Level-Two assignments, you need to submit your source code (including SoftBodyMassSpring.h, MassSpringDriver.h and any other pieces of code you created for the assignment), a README file for any specific issue, and a video for your own mass-spring simulation. For Level-Three assignment, you need to submit your source code package, your data and executable, and schedule a time with the instructor and the TA to present your implementation briefly.