ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

Physically-Based Simulation in CG

Exercise 1: Mass Points and Springs

(Assigned: September 27; Due: October 11)

How to submit the exercise

Send your version of the file Exercise.cpp by email to Byungsoo Kim (kimby@inf.ethz.ch), along with [PBS17] header, your names, ID numbers, and theoretical answers, by October 11. An answer to any of the questions should not be more than two sentences.

Framework

We provide a Microsoft Visual C++ project that initializes simulation settings, performs the simulation loop, and uses OpenGL/GLUT for visualization. If you wish, you can use the same source files on a different platform.

The exercise consists of computing one simulation step of various scenes with different settings. The routines for simulation time steps are included in the file Exercise.cpp. Do all additions to the code inside Exercise.cpp. We will test the exercise by compiling and running your version of Exercise.cpp, therefore do not include external libraries, or the code will not compile in our platform.

Resources

We provide classes for 2D vectors and matrices that should help with 2D simulations. These classes implement basic operations such as addition, subtraction, multiplication, cross and dot product, etc. Check the file Vec2.h. If you need further functionality, you should code it inside the file Exercise.cpp itself.

Default Settings

The default settings for the simulations are: mass of each point m = 0.1 Kg, stiffness of each spring k = 10.0 N/m, damping γ = 0.01 N*s/m, time step dt = 0.003 sec, integration method ‘Euler’. You may modify these settings in the initialization in the file Scene.cpp, or through the command line when starting the executable. Test the behavior with different settings!

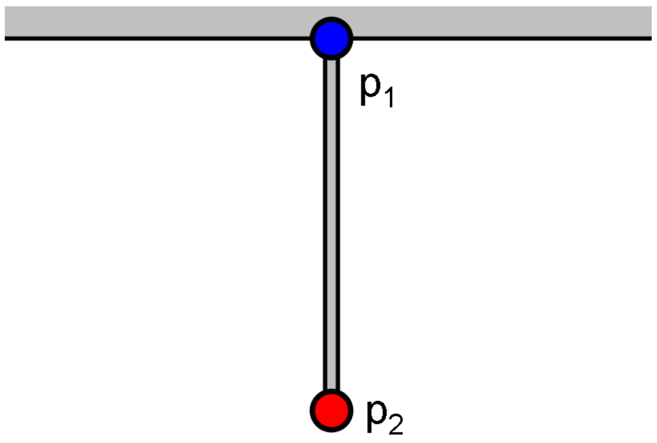
Problem statement

In this exercise, a couple of simple mass-spring systems will be simulated. The mass points *p_i* all have the mass *m*. The springs are characterized by their stiffness *k*, initial length *L*, and damping *d*. These parameters are identical for all springs and mass points, and are provided as function arguments. For damping, use a point-based damping force linear in the velocity.

The scenes are simulated in the XY plane, and gravity should be applied in the –Y direction (downwards). p_1 is at coordinate $(0,0)$, p_2 is therefore coordinate $(0,-L)$

Problem 1: Mass point hanging from the ceiling (33.3%)

This exercise consists of simulating a 1-dimensional spring. One end-point is attached to the ceiling, and the other one falls due to gravity.



FUNCTION INTERFACE:

```
void AdvanceTimeStep1(float k, float m, float d, float L, float dt,
      int method, float p1, float v1, float& p2, float& v2);
```

REQUIREMENTS: Program the update per-time-step of the position p2 and velocity v2 of the bottom point for the following integration methods:

- explicit Euler (method = 1) (8.3%)
- symplectic Euler (method = 2) (8.3%)
- explicit midpoint (method = 3) (8.3%)
- semi-implicit Euler (method = 4) (8.3%)

Write the updated position and velocity to p2 and v2.

Use the following command line arguments to use the different methods:

- `-method euler`
- `-method symplectic_euler`
- `-method midpoint`
- `-method backwards_euler`

Problem 2: Analytic solution and results analysis (33.3%)

This exercise requires the following:

2.1. **Analytic solution**:
Find an analytic solution that describes the 1D behavior from problem 1, and implement it as another method in the same function implemented in problem 1 (method = 5). Use the command line argument `–method analytic` to test your code.
Guide: the solution is of the form

$$y(t) = c_1 e^{\alpha t}\cos(\beta t) + c_2 e^{\alpha t}\sin(\beta t) - L - \frac{mg}{k},$$

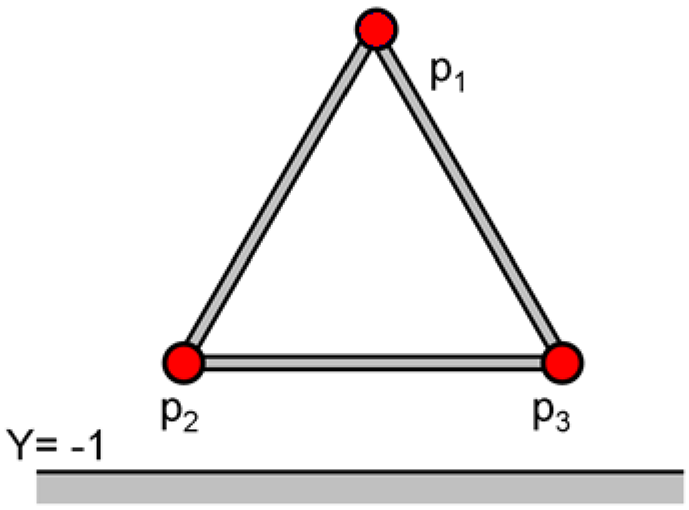
with $\alpha = -\frac{\gamma}{2m}$, $\beta = \frac{\sqrt{4km-\gamma^2}}{2m}$. You are required to find only the constants c_1, c_2. This should be done by using the rest state initial conditions (no energy in the system, except gravitational energy at $t = 0$). (11.1%)

2.2. **Error convergence analysis**:
Use the flag `-testcase error_measurement`, and set damping to zero (`–damp 0`). In this test case a single step is simulated (from $t = 0.1$ to $t = 0.1 + h$) using all 5 methods, and resulting displacements and velocity changes are reported to the console output. This procedure is repeated 10 times and in each experiment, the step size is halved.
You are required to compute the displacement error for each of the numerical methods, compared to the analytic one. Denoting an error in iteration i as e_i, you are further required to compute the error convergence $\frac{e_i}{e_{i+1}}$, and report its average for each method.
Comment on these results in one or two sentences: how do you explain them and what do you learn of the different methods' accuracy?
Repeat the experiment with damping: do the results change your conclusions significantly? Recommended (starting) step size is 0.5. (11.1%)

2.3. **Stability analysis**:
Use the flag `–testcase stability_measurement` and set damping to zero (`–damp 0`). In this test case the simulation is performed using all 5 methods, from $t = 0$ to $t = 10$, and the maximum amplitude throughout this simulation is reported to the console output.
This procedure is repeated 10 times and in each experiment, the step size is doubled.
Comment on these results in one or two sentences: what do you learn of the different methods' stability? What, in your opinion, is the correlation between stability and accuracy? Repeat the experiment with a damping coefficient of 0.5: do the results change your conclusions significantly? Recommended (starting) step size is 10^{-4}. (11.1%)

Problem 3: Triangle colliding with the ground (33.3%)

This exercise consists of simulating a triangle in 2D that is dropped on the floor (located at Y = -1). Each pair of vertices is connected by a spring, and the vertices are not fixed. Collision response should be applied with the penalty-based method, i.e. a repulsive spring force in the ground, with $k_R = 100$. This will result in an elastic ground response.



FUNCTION INTERFACE:

```
void AdvanceTimeStep3(float k, float m, float d, float L, float dt,
      Vec2& p1, Vec2& v1, Vec2& p2, Vec2& v2, Vec2& p3, Vec2& v3);
```

REQUIREMENTS: Program the update per time step of the positions and velocities (p1, p2, p3, v1, v2, v3) of all three vertices, with the symplectic euler method.

Use the command line argument `-testcase falling` to select the correct test case.