Project 1: Particle systems

Due date: 25.05.2022

Overview

In this project you will implement a *particle system* with *constraints*. You must implement at least the required features, see below; you will also have to make a demo video showing your system in action. Additionally, you may implement some of the listed extensions (or invent your own !) for extra credit.

Skeleton Code

You have been provided with some *skeleton code* which you may use to jump-start your coding. The only thing the code does is move three particles around randomly, and draw some (nominal) constraints and spring between them. Further, the code does little more than implement basic window management and graphics.

Required Features

Your code must implement the following features:

- **Generalized force structure: 2 points.** This is described in the slides. (If you use the skeleton code, you should replace `delete_this_dummy_spring` by a `std::vector` of forces.) Then, you'll use your generalized force structure to implement two forces:
  - `GravityForce.` Acts like gravity...
  - `SpringForce.` A damped spring between two particles. (Skeleton code is already provided.)
- **Generalized constraint structure: 4 points.** This is also described in the slides. (If you use the skeleton code, replace `delete_this_dummy_rod` and `delete_this_wire` by a `std::vector` of forces.) You must implement at least the following two subclasses:
  - `RodConstraint.` Constraints two particles (x_1, y_1) and (x_2, y_2) to be a fixed distance r apart, i.e.,

$$C(x_1, y_1, x_2, y_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2 - r^2. \quad (1)$$

As a variation, implement also

$$C(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r. \tag{2}$$

and compare with the previous formulation; discuss possible problems. (Rendering code included in the skeleton.)

- `CircularWireConstraint`. Constraints a particle (x, y) to be at a fixed distance r from some point (x_c, y_c), i.e.,

$$C(x, y) = (x - x_c)^2 + (y - y_c)^2 - r^2. \tag{3}$$

- **Mouse interaction: 1 points.** When the user clicks and drags the mouse, a spring force should be applied between the mouse position and the given particle to make your system interactive.

- **Several numerical integration schemes: 3 points.** The integration scheme should be selectable at run-time with keystrokes or GLUT menus. You will find this easiest if you implement a pluggable integration architecture as described in the slides. The minimum integration schemes are:
  - Euler
  - Mid-point
  - Runge-Kutta 4.

  Experiment with each of these and make a comparison w.r.t. to breaking point when increasing the time step. Perform timings and discuss your findings.

- **2D Cloth: 3 points.** Create a (rectangular) network of particles with appropriate springs holding them together. For example, you can imagine a curtain which either slides along a line (holding it in place) or has two (or more) fixed points. Sliding or fixing points should be regarded as constraints, implemented using the generalized-constraints approach above. Further, experiment with spring configurations and constraints, and report your findings (which worked/didn't work, why, etc.?)

- **Simple interaction with the 2D cloth: 2 points.** Based on the above constraints and mouse interaction. Implement a simple interaction mechanism, by which you can apply a horizontal force to the cloth. Collisions of the cloth with a line (wall) should also be experimented with.

- **Angular Springs: 2 points.** Pulls a triplet of particles so that their subtending angle approaches some rest angle. Based on these, build a simple hair(-like) simulation. Some simple collision-handling scheme should also be experimented with.

- **Implicit integration: 3 points.** Implement implicit integration for the 2D cloth. Note that, code for solving linear systems by the Conjugate Gradient method is provided within the skeleton code.

Remarks:

– Your demo must be able to turn each of these features on and off individually, so that they can be verified.

– The number of points above represents the maximum you can get per requirement, assuming both correct implementation and reporting. Details can be found in the rubric of the assignment.

Optional Features

- **Verlet Integrator**. See http://en.wikipedia.org/wiki/Verlet_integration.
- **Leapfrog Integrator**. See https://en.wikipedia.org/wiki/Leapfrog_integration.
- **Collisions with other particles**. Particles bounce off each other.
- **Angular Constraints**. Similar to angular springs, but the angle is actually constrained.
- **Visualization**. Implement simple visualizations which allow you to visually inspect forces, velocities, etc. of your particles.
- **Hair with collisions**. Improve collision handling for the basic hair-like simulation. How can we implement this ?

Deliverables

- **Code.** On the due date, you must deliver the code of your project (use Canvas). Instructions about how to compile and run your project should also be included.
- **Demo video.** Show-off what and how you implemented. Delivered through Canvas. The skeleton code contains functionality to save snapshots from your code. The snapshots can be coalesced into a movie using various packages.
- **Paper.** The paper ($4 - 8$ double-column pages, in e.g. IEEE format) describes the results you obtained, and the methods you used (implemented). Snapshots, pseudo-code fragments, equations, timings, etc. should all be included. Additionally, make sure that you include an appendix in which you explain what and how each group member contributed to the project. Deliver the paper through Canvas.