Advanced Computer Graphics
Exercise 6 - Particle System Part 2

Handout date: 3.11.2015
Submission deadline: Monday, 16.11.2015, 11:00pm

Note

Undeclared copying of code or images (either from other students or from external sources) is strictly prohibited! Any violation of this rule will lead to expulsion from the class. No late submission allowed.

What to Hand In

Solve the exercise in groups of 3. Hand in a compressed zip file renamed to `Exercise6-firstname_lastname-firstname2_lastname2-firstname3_lastname3.zip` that contains the following files:

- `Mass_spring_viewer.cpp`
- A readme file (.txt or .pdf) containing the following: a description of your solution, how much time you needed, encountered problems (if any), and answers to the questions.

This is the second part of the two-part particle system exercises. You'll keep working with framework from last week. All your implementation tasks are again in `Mass_spring_viewer.cpp`.

6.1 Midpoint Integration (10 points)

Implement the `Midpoint` case in the method `time_integration`. You can use the `i`-button during the simulation to cycle through the different integration schemes (by default the Euler integration is active).

Note that the class `Particle` has a boolean member `locked`. If it is set to true, you should neither change the velocity nor the position of the particle (you should also adapt that in the explicit Euler integration from last week if you haven't already done that).

6.2 Velocity Verlet Integration (10 points)

Implement the `Verlet` case in the method `time_integration` according to the course slides on Velocity Verlet.

6.3 Triangle-area Forces (10 points)

Implement area preserving forces for triangles inside the `compute_forces` method. The details are explained in the course slides. You can test your code with scenes `4` and `5`. Use the member `area_stiffness_`. All required information about the triangles in the scene is given in the vector `body_.triangles`.

6.4 Implicit Euler Integration (60 points)

The code you are provided with for this exercise comes with the Eigen linear algebra library (`http://eigen.tuxfamily.org`) - you can make use of it for your matrix computations. The linear system solver is already implemented in the class `ImplicitSolver`, which comes with a convenience method `addElementToJacobian(row, col, value)` that should be used to fill the `2N x 2N` Jacobian matrix (N = number of particles). Please write your solution code in the `compute_jacobians` method, where you should fill up the system jacobian matrix for each of the forces present in the scenes:

- mouse spring
- damped springs
- force-based collisions
- gravitation
- center force
- triangle-area forces.

Please notice in the case of the implicit integrator the damping forces are not used, as the integration method damps the system implicitly.

6.5 Impulse-based Collision Response (10 points)

The problem with the force-based collision response you implemented in the last part of the exercise is that particles often overshoot the boundary. The collision forces make the colliding particles' velocity change smoothly over time. To avoid particles going through the boundary too far, we must use a collision response that changes the velocity instantly.
Implement the `impulse_based_collisions` method. You can switch between the the two collision algorithms by pressing `c`. By default, collision forces are active.

6.6 Equilibrium Force (10 points)

Design an inter-particle force that will let a large set of free particles converge into a uniform distribution (i.e., after several iterations the particles should stop moving and all distances between neighboring particles should be roughly the same). Test it on scene `2` (with Verlet integration and impulse-based collisions - no need to derive the Jacobians). Describe your force in the readme file. Also add a keyboard switch to toggle the force on and off.

Report

Please submit a short report along with the code for this assignment. The report should analyze the following topics (by briefly explaining the theoretical background and showing some experiments ran with your code if necessary):

- Any implementation details you feel are important.
- Derivation of the Jacobians for all the forces present in the scenes.
- Force-based collision response and Impulse-based collision response.
- Different integration methods and their stability. Observe how the different integrators behave with smaller/larger time steps and the different types of forces. Compute the total kinetic energy of the system at the end of the `compute_forces` method and display it for each iteration. Accumulate these values in a graph showing how the total kinetic energy decays over multiple iterations with the different integration methods. Briefly interpret the results.